FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Návrh a zavedení kešovacího mechanismu pro e-shopovou platformu wpjshop
Student:	Bc. Leoš Tobolka
Vedoucí:	Ing. Josef Kejzlar
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Účelem této práce je umožnit kešování HTML obsahu, který poskytuje webový server e-shopu. Tímto způsobem nebudou klienti, kteří přistupují na e-shop, nuceni čekat na vygenerování celého obsahu při každém požadavku. Společný obsah pro více klientů jim bude poskytnut z mezipaměti.

Vzhledem k vysoké dynamičnosti e-shopu není možné jednoduše sdílet HTML obsah mezi více uživateli. Poskytované HTML stránky obsahují velké množství personalizovaného obsahu, jako jsou produkty v košíku, měření Google Tag Manager (GTM), oblíbené produkty, měna, jazyk a další. Předpokládá se, že součástí práce bude i úprava způsobu poskytování těchto personalizovaných dat jednotlivým uživatelům, aby se umožnilo kešovat celý HTML obsah, nebo jeho převážná část.

Zavedením kešovacího mechanismu je také očekáváno zlepšení odolnosti systému vůči možným útokům typu DDOS, které by mohly zahlcovat webový server nežádoucími požadavky.

1. Proveďte rešerši a vyberte nejvhodnější přístup pro účel kešování HTML obsahu pro danou platformu.
2. Navrhněte postup implementace a implementujte výsledný návrh.
3. Otestujete implementaci.
4. Zhodnoťte výsledné řešení a navrhněte úpravy do budoucna.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 12. prosince 2023 v Praze.

Diplomová práce

NÁVRH A ZAVEDENÍ KEŠOVACÍHO MECHANISMU PRO E-SHOPOVOU PLATFORMU WPJSHOP

Bc. Leoš Tobolka

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Josef Kejzlar
9. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval Ing. Josefu Kejzlarovi za nabídnutí tohoto tématu diplomové práce a za jeho vedení. Děkuji mu také za všechny cenné rady, které mi v uplynulých sedmi let dal; jeho zkušenosti mi poskytly mnoho užitečných poznatků. Dále bych chtěl poděkovat své rodině za neustálou podporu a povzbuzení ve všem, co dělám. Velké díky patří také všem ostatním, kteří mě podporovali během celého mého studijního období.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 9. května 2024

Abstrakt

Tato práce se zabývá návrhem a implementací kešovacího mechanismu pro e-commerce platformu WPJshop. Cílem je zrychlit načítání HTML obsahu multi-page aplikací, které jsou postaveny na této platformě. V teoretické části práce jsou analyzovány problémy spojené s kešováním a poskytováním personalizovaného obsahu. Dále jsou představeny technologie vhodné pro kešování a podrobně je popsán výsledný návrh řešení. Praktická část obsahuje implementaci návrhu. Řešení využívá sdílenou HTTP mezipaměť pro ukládání celých HTML odpovědí. Zvolený přístup umožňuje obsluhu požadavků přímo z mezipaměti, aniž by bylo nutné při každém požadavku navazovat spojení se zdrojovým serverem. HTML odpovědi jsou upraveny tak, aby neobsahovaly žádná personalizovaná data, což umožňuje jejich ukládání ve sdílené mezipaměti. Veškerá personalizovaná data uživatelů jsou získávána dodatečně prostřednictvím front-endové JS aplikace skrze GraphQL API.

Klíčová slova e-shop, multi-page aplikace, kešování, HTTP mezipaměť, RFC 9111, HTML, reverzní proxy server, React

Abstract

This thesis focuses on the design and implementation of a caching mechanism for the e-commerce platform WPJshop. The goal is to accelerate the loading of HTML content for multi-page applications built on this platform. The theoretical part of the thesis analyzes problems associated with caching and providing personalized content. It also introduces technologies suitable for caching and describes in detail the final design of the solution. The practical part includes the implementation of the design. The solution utilizes a shared HTTP cache to store whole HTML responses. This approach allows requests to be handled directly from the cache, without the need to establish a connection with the origin server for each request. The HTML responses are modified so that they do not contain any personalized data, which allows their storage in the shared cache. All personalized user data are subsequently acquired through a front-end JS application via the GraphQL API.

Keywords e-shop, multi-page application, caching, HTTP cache, RFC 9111, HTML, reverse proxy server, React

Seznam zkratek

ACL	Access Control List
API	Application Programming Interface
CDN	Content Delivery Network
CMS	Content Management System
CSR	Client Side Sendering
DOM	Document Object Model
ESI	Edge Side Includes
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
LTS	Long Time Support
MPA	Multi page application
MVC	Model-view-controller
PDA	Personal Digital Assistant
PHP	PHP Hypertext Preprocessor
PHP	PHP Hypertext Preprocessor
RDBMS	Relational database Management System
REST	REpresentational State Transfer
RPS	Reverzní proxy server
SaaS	Software as a Service
SLA	Service-level Agreement
SPA	Single Page Application
SQL	Structured Query Language
SSR	Server Side Rendering
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VCL	Varnish Configuration Language
XHR	XMLHttpRequest

Úvod

V současné době se e-commerce sektor neustále rozvíjí a s tím roste i počet uživatelů přistupujících k webovým stránkám e-shopů. To klade velké nároky na výkon a rychlost odpovědí serverů, které musí zpracovávat velké množství požadavků v co nejkratším čase. Jedním z klíčových aspektů, který může výrazně ovlivnit uživatelskou zkušenost, je doba načítání webových stránek. Dlouhé doby načítání mohou vést k tomu, že uživatelé opustí web, což má přímý dopad na prodej a úspěšnost e-shopů.

Existuje mnoho faktorů, které mohou přispět k pomalé reakci serveru. Jedním z běžných důvodů může být překročení limitu maximální kapacity požadavků, které server dokáže zpracovat v daném časovém úseku. Dalším faktorem, který hraje roli, je postupný nárůst objemu dat uchovávaných v databázi. Se zvyšováním množství dat se zvyšují i nároky na jejich zpracování, což může také významně prodloužit dobu odpovědi serveru. Také postupné rozšiřování funkcionality aplikace může způsobit zpomalení. Nově přidané funkce často vyžadují dodatečné zpracování dat a výpočetní výkon. Všechny tyto důvody, a mnoho podobných dalších, mohou více či méně negativně ovlivňovat uživatelskou zkušenost.

Tato práce se zabývá hledáním řešení pro platformu WPJshop, které umožní zvýšit rychlost webových stránek prostřednictvím kešování HTML obsahu. Základní myšlenkou je využití mezipaměti pro ukládání často požadovaného HTML obsahu, čímž se eliminuje potřeba jeho opětovného generování při každém požadavku klienta. Tento přístup má potenciál výrazně snížit dobu čekání koncových uživatelů z řádů tisíců milisekund na pouhé desítky.

Vysoká míra personalizace obsahu e-shopů pro jednotlivé uživatele však představuje významnou výzvu při hledání efektivního způsobu kešování. Personalizované prvky, jako jsou produkty v košíku, doporučené produkty, oblíbené produkty, zvolená měna, jazyk stránky a další podobné prvky, komplikují využití mezipaměti. Kešovací mechanismus vyžaduje promyšlený přístup, který v mezipaměti umožní ukládat často požadovaný obsah a zároveň nenaruší uživatelskou zkušenost.

Cíle práce

Hlavním cílem práce je, aby veškerý hlavní HTML obsah vracený v odpovědích z e-shopů (založených na platformě WPJshop) bylo možné uložit v mezipaměti, která slouží k obsloužení následujících ekvivalentních požadavků uživatelů. Hlavním obsahem jsou myšleny všechny často navštěvované stránky – úvodní stránka, katalogy produktů, detaily produktů, stránky blogu a ostatní obsahové stránky.

Cíleno je primárně na využití sdílené mezipaměti, která je sdílena mezi všemi uživateli. To umožní svůj obsah poskytnout všem požadavkům bez ohledu na to jestli již konkrétní uživatel na e-shop v minulosti přistoupil nebo ne. Sdílná mezipaměť umožňuje uložit odpověď vrácenou jednomu uživateli a poskytnout ji jinému. V této mezipaměti však nebude ukládán jakýkoliv personalizovaný obsah, který je určen konkrétnímu uživateli, aby nedošlo k porušení bezpečnosti. Mimo sdílenou mezipaměť lze využít i soukromou mezipaměť (mezipaměť prohlížeče), která přísluší pouze jednotlivým uživatelům.

Navazujícím cílem je umožnit obsloužení častých požadavků bez nutnosti navázání spojení se zdrojovým serverem. Toto souvisí se sdílenou mezipamětí, která dokáže uložit celou HTTP odpověď a ve stejné formě ji vrátit uživateli bez toho, aby se musela dotázat zdrojového serveru na jakýkoliv obsah.

Veškerá stávající funkcionalita e-shopů musí být zachována – není uvažováno odebírání jakýchkoliv stávajících prvků. Je však možné jejich funkčnost upravit tak, aby bylo možné naplnit všechny ostatní cíle. Z uživatelského hlediska zůstanou všechny prvky bez změny; jediná pozorovatelná změna bude rychlost celého systému.

V teoretické části práce bude vyčteno, jaké technologie se nabízí pro požadovanou doménu a jaký postup je zvolen k zavedení zvolených technologií při naplnění všech cílů a požadavků. V praktické části budou popsány nutné úpravy ve stávajícím řešení a implementované nové části systému. Poslední částí práce je ukázka výsledného řešení včetně naměřených statistik, které znázorňují efektivitu výsledného řešení.

Kapitola 1
Platforma WPJshop

Prezentace platformy WPJshop a analýza technologií použitých při jejím vývoji. Tato práce je zaměřena primárně na vývoj kešovacího systému pro tuto platformu, což vyžaduje prozkoumání využívaných technologií. Tento přehled umožní zohlednit tyto technologie při následném návrhu řešení.

1.1 O platformě

WPJshop je SaaS platforma specializující se na e-commerce od české společnosti Wpj s.r.o. Tato společnost vznikla v roce 2007, má své hlavní sídlo ve městě Vrchlabí a má svojí pobočku v Ústí nad Labem. Historicky se společnost zaměřovala na tvorbu webových stránek, ale změnila své zaměření na oblast e-commerce. Při realizaci projektů je dán důraz na komunikaci se zákazníky a tvorbu řešení přímo na míru. [1]

Aplikace se skládá z jádra a nemalého množství rozšiřujících modulů, které zprostředkovávají další funkce, které klientské společnosti mohou potřebovat. Mezi tyto moduly patří napojení na různé ERP systémy (např. SAP, Pohoda, Abra), učetní systémy (např. Money S3), platební brány (např. GoPay, ThePay, Twisto), marketingové nástroje (Google Analytics), expediční služby (např. Balíkobot) a další. Kromě napojení na jiné systémy, moduly přidávají další interní funkcionality, jako je např. automatizace objednávek, skladový systém, správa prodejen a jejich skladu, slevové kupóny, podporu více jazyků atp. [2, 3]

Tuto platformu využívá již více než 170 klientů napříč celou Evropou, přičemž nejvíce klientů je z České republiky a Slovenské republiky. [1]

1.2 Technický popis

WPJshop funguje jako webová aplikace postavená v programovacím jazyce PHP. Jádrem této platformy je Symfony framework, přičemž data jsou uložena v relační databázi MariaDB. Pro dočasné ukládání dat aplikační vrstvy je využívána Redis databáze. Kromě toho je nabízené GraphQL API, které poskytuje možnost integrace s externími nebo doplňujícími aplikacemi. Pro vývoj některých částí platformy je využíván React framework. Pro modularizaci a optimalizaci souborů JS a CSS je použit nástroj Webpack.

Symfony část aplikace obsahuje tisíce automatizovaných jednotlivých PHPUnit testů, které udržují spolehlivost aplikace v průběhu stálého vývoje. Dále jsou využity i automatizované front-end testy v nástoji Cypress.

1.2.1 PHP

PHP, což je zkratka pro „PHP: Hypertext Preprocessor“, je široce používaný open-source skriptovací jazyk pro všeobecné použití, který je obzvláště vhodný pro vývoj webových stránek a umožňuje dynamické generování HTML obsahu. Jeho syntaxe vychází z jazyků C, Java a Perl a je snadná na naučení. Hlavním cílem jazyka je umožnit vývojářům webových stránek rychle psát dynamicky generované webové stránky. [4]

WPJshop je realizován formou webové aplikace v programovacím jazyce PHP. Aktuálně se používá verze 8.2, přičemž je kladen důraz na udržování aktuální verze.

1.2.2 Symfony Framework

Symfony je sada znovupoužitelných PHP komponent a PHP framework pro vývoj webových aplikací, API a mikroslužeb. Byl zveřejněn jako svobodný software v roce 2005 a je vydán pod licencí MIT. [5]

Symfony si klade za cíl urychlit tvorbu a údržbu webových aplikací a nahradit opakující se úlohy kódování. Je také zaměřen na vytváření robustních aplikací v podnikovém kontextu a jeho cílem je poskytnout vývojářům plnou kontrolu nad konfigurací: od adresářové struktury až po cizí knihovny, téměř vše lze přizpůsobit. Aby odpovídal podnikovým směrnicím vývoje, je Symfony dodáván s dalšími nástroji, které pomáhají vývojářům testovat, ladit a dokumentovat projekty. [5, 6]

Jednou z vlastností tohoto frameworku je možnost vytvářet tzv. Symfony „bundles“. Bundle je kolekce souborů a složek uspořádaných do určité struktury. Obecně by bundles měly být modelovány tak, aby je bylo možné opakovaně používat ve více aplikacích. Daly by se přirovnat principu pluginů v jiných softwarech. Základní funkce Symfony frameworku jsou také implementovány pomocí bundles (např. FrameworkBundle, SecurityBundle, DebugBundle). [7, 8]

WPJshop používá ve svém jádru Symfony Framework ve verzi 6.4 LTS. Každý modul v platformě WPJshop je sám o sobě Symfony bundle. Jejich využitelnost mimo platformu WPJshop je však značně limitována, jelikož mají moduly častokrát mezi sebou závislosti, anebo využívají kódu, který je dostupný pouze v jádře WPJshopu. [3]

1.2.3 MariaDB

MariaDB Server je univerzální open-source systém pro správu relačních databází. Je to jeden z nejoblíbenějších databázových serverů na světě, mezi jehož významné uživatele patří například Wikipedia, WordPress.com a Google. Server MariaDB je vydán pod licencí GPLv2 open-source a je zaručeno, že zůstane open-source. [9]

Když předchůdce MariaDB Serveru, MySQL, koupila v roce 2009 společnost Oracle, zakladatel MySQL Michael Widenius projekt rozvětvil kvůli obavám ze správcovství společnosti Oracle a nový projekt pojmenoval MariaDB. Většina původních vývojářů se připojila k novému projektu a MariaDB Server se od té doby dále rychle vyvíjí. [9]

WPJshop pro persistenci dat využívá právě tento RDBMS. Momentálně využívaná verze MariaDB je 10.11 LTS, avšak v budoucnu je plánován upgrade na verzi 11. Tato nová verze přináší rozsáhlé změny v optimalizátoru dotazů, který by mohl vyřešit některé stávající nedostatky ve verzi 10.11. WPJshop je silně závislý na efektivitě mnoha rozsáhlých SQL dotazů, které implementuje; přechod na novou verzi není tedy triviální, vzhledem k tomu, že optimalizátor dotazů je obecně rozsáhlá a na změny citlivá komponenta všech DBMS.

Pro abstrakci nad MariaDB lze využít knihovnu Doctrine DBAL. Tato knihovna nabízí objektově orientované rozhraní a mnoho dalších funkcí, jako je introspekce databázových schémat a manipulace s nimi. [10]

WPJshop pro komunikaci s databází nevyužívá ORM, ale spoléhá na takzvaný SQL Query Builder, který je poskytován knihovnou Doctrine DBAL. SQL Query Builder umožňuje vytvářet

SQL dotazy v PHP s využitím objektově orientovaného návrhového vzoru „builder“, což umožňuje generování SQL dotazů. Na rozdíl od ORM, které po zaslání dotazu do databáze vrací instance entit odpovídající tabulkám v databázi, SQL Query Builder produkuje výsledek ve formě asociativního pole.

1.2.4 Redis

Redis je open-source, v paměti uchovaná, key-value databáze, která nabízí vysokou rychlost a flexibilitu při práci s daty. Jako in-memory databáze ukládá Redis data přímo do operační paměti, což umožňuje rychlý přístup a manipulaci s daty. Redis podporuje různé datové struktury jako jsou řetězce, hashovací tabulky, seznamy, množiny a další. [11]

Redis, ale i ostatní key-value databáze, jsou často využívány jako doplněk k tradičním relačním databázím. Tento přístup umožňuje kombinovat pevnou strukturu, integritu dat a durabilitu, kterou nabízejí relační databáze, s vysokou rychlostí a flexibilitou, které poskytují key-value databáze.

WPJshop využívá Redis primárně pro ukládání uživatelských relací a pro ukládání mezivýsledků pomocí volání z PHP kódu. Díky tomu lze tato data sdílet napříč více instancemi WPJshopu. To umožňuje WPJshop horizontálně škálovat.

1.2.5 Smarty

Jak v administraci, tak v prostředí e-shopu se pro generování HTML používá šablonovací jazyk Smarty. Ten umožňuje snadnější oddělení prezentační vrstvy od aplikační logiky systému. Zároveň přináší i snadnější podporu pro znovupoužitelnost HTML kódu a možnost dědění šablon mezi sebou.

Pro nové projekty psané v Symfony je mnohdy výhodnější použít šablonovací jazyk Twig, jelikož má svůj Symfony bundle a je i Symfony vyvíjen. Nicméně v době, kdy začal vývoj wpjshopu, nebyl jako základní kámen využíván framework Symfony a Smarty byl zvolen jako vhodný jazyk pro tvorbu šablon. Z tohoto hlediska nedává smysl v platformě kombinovat vícero různých šablonovacích jazyků a je nadále využíván Smarty. [3]

1.2.6 React

React je knihovna pro vývoj uživatelských rozhraní, která je spravována společností Meta Platforms a komunitou open-source vývojářů. React umožňuje vývojářům stavět velké webové aplikace, které mohou dynamicky měnit zobrazená data bez nutnosti načítat celou stránku. Klíčovým konceptem Reactu je komponenta, což je izolovaná jednotka kódu, která obsahuje vlastní logiku a stavy. Komponenta může být znovu použita v rámci aplikace, čímž se zvyšuje efektivita a udržitelnost kódu. React také zavádí virtuální DOM, což je „odlehčená kopie“ skutečného DOM ve webovém prohlížeči. To umožňuje efektivní aktualizaci uživatelského rozhraní tím, že je minimalizována manipulace s DOM, jehož aktualizace je obvykle náročnější na výkon. Tato knihovna se stala základem mnoha moderních webových a mobilních aplikací díky své flexibilitě, výkonu a velké komunitě vývojářů, která za ní stojí. [12, 13]

WPJshop využívá React na několika projektech, je v něm postavený jejich vlastní CMS systém pro editaci obsahu článků, aplikace pro mobilní čtečky pro jejich skladový systém Shipped a aplikace pro pokladní systém. React je také využíván v mnoha mikro aplikacích napříč celým WPJshop systémem.

1.2.7 GraphQL

GraphQL je dotazovací jazyk pro API a běhové prostředí pro provádění těchto dotazů. GraphQL není vázané na konkrétní databázi a je v podstatě považováno za flexibilnější alternativu k REST. Bylo vyvinuto společností Facebook v roce 2012 a v roce 2015 bylo uvolněno jako open-source. Na rozdíl od tradičních REST API, která vyžadují načítání z více URL pro získání požadovaných dat, GraphQL umožňuje klientům formulovat požadavky na získání přesně těch dat, která potřebují, a to vše z jediného endpointu. To může vést k rychlejším načítacím časům aplikací a efektivnějšímu využití datového přenosu. Klienti mohou specifikovat strukturu požadované odpovědi, což umožňuje aplikacím získávat data v přesně takové formě, jakou potřebují, bez nadbytečných informací. Díky své vysoké efektivitě a flexibilitě si GraphQL rychle získalo popularitu mezi vývojáři a je široce používáno mnoha organizacemi po celém světě. [14]

WPJshop poskytuje tato GraphQL rozhraní: veřejné API přístupné bez omezení a chráněné API pro úpravu dat, které využívá pokladní systém a také je zpřístupněno administrátorům e-shopu. Na ukázce 1.1 lze vidět admin GraphQL API dotaz na skladovost produktů s kódem „6390“ a „2293“. Odpověď na tento dotaz ve formátu JSON je na ukázce 1.2.

```
query {
  products(filter: {code: ["6390", "2293"]}) {
    items {
      code
      inStore
    }
  }
}
```

Výpis kódu 1.1 Ukázka GraphQL dotazu na skladovost produktů

```
{
  "data": {
    "products": {
      "items": [
        {
          "code": "6390",
          "inStore": 10
        },
        {
          "code": "2293",
          "inStore": 24
        }
      ]
    }
  }
}
```

Výpis kódu 1.2 Ukázka odpovědi na GraphQL dotaz 1.1 – výpis skladovosti produktů

Rozhraní je implementováno pomocí PHP knihovny GraphQLite[1], která umožňuje snadnou

[1] `https://graphqlite.thecodingmachine.io`

integraci GraphQL do Symfony frameworku. Pomocí anotací v PHP třídách je možné definovat, jaké objekty a metody budou dostupné prostřednictvím GraphQL API. Knihovna se stará o vytvoření GraphQL schématu z označených objektů a metod, což výrazně zjednodušuje celkový vývoj.

1.2.8 Google Tag Manager

Google Tag Manager (GTM) je nástroj od Google, který umožňuje snadnou správu měřicích a marketingových kódů (značek) na webových stránkách. GTM funguje jako prostředník mezi webem a analytickými či reklamními službami, což umožňuje přidávat, upravovat nebo měnit všechny tyto značky bez nutnosti zásahu do kódu stránky. Tím se výrazně snižuje závislost na vývojářích. Základním prvkem v GTM je datová vrstva (DataLayer), která slouží jako úložiště informací z webových stránek. Informace v datové vrstvě mohou zahrnovat údaje o interakcích uživatelů, transakcích, nebo jiných událostech, které jsou poté využívány značkami pro analýzu a marketing. [15, 16]

WPJshop do datové vrstvy přidává události jako jsou kliknutí na produkt, přidání do košíku, odebrání z košíku, provedení nákupu, informace o uživateli atp. Příklad vložení událostí do datové vrstvy lze vidět na ukázce 1.3; v JavaScriptu je do pole `dataLayer` vložena událost `user` s informacemi o uživateli a stavu jeho košíku. Všechny události/objekty v datové vrstvě jsou po vložení zpracovány nasazenými značkami.

```
dataLayer.push({
  "event": "userInfo",
  "user": {
    "sessionId": "9q4oj9e1ujruo9to41hapnk82a",
    "cartId": "kblonasl7l13rprt9bumh1oavd",
    "type": "anonymous",
    "cartValue": 1699,
    "cartValueWithVat": 1699,
    "cartValueWithoutVat": 1404,
    "cartItems": 1
  }
})
```

Výpis kódu 1.3 Příklad vložení dat do GTM dataLayer

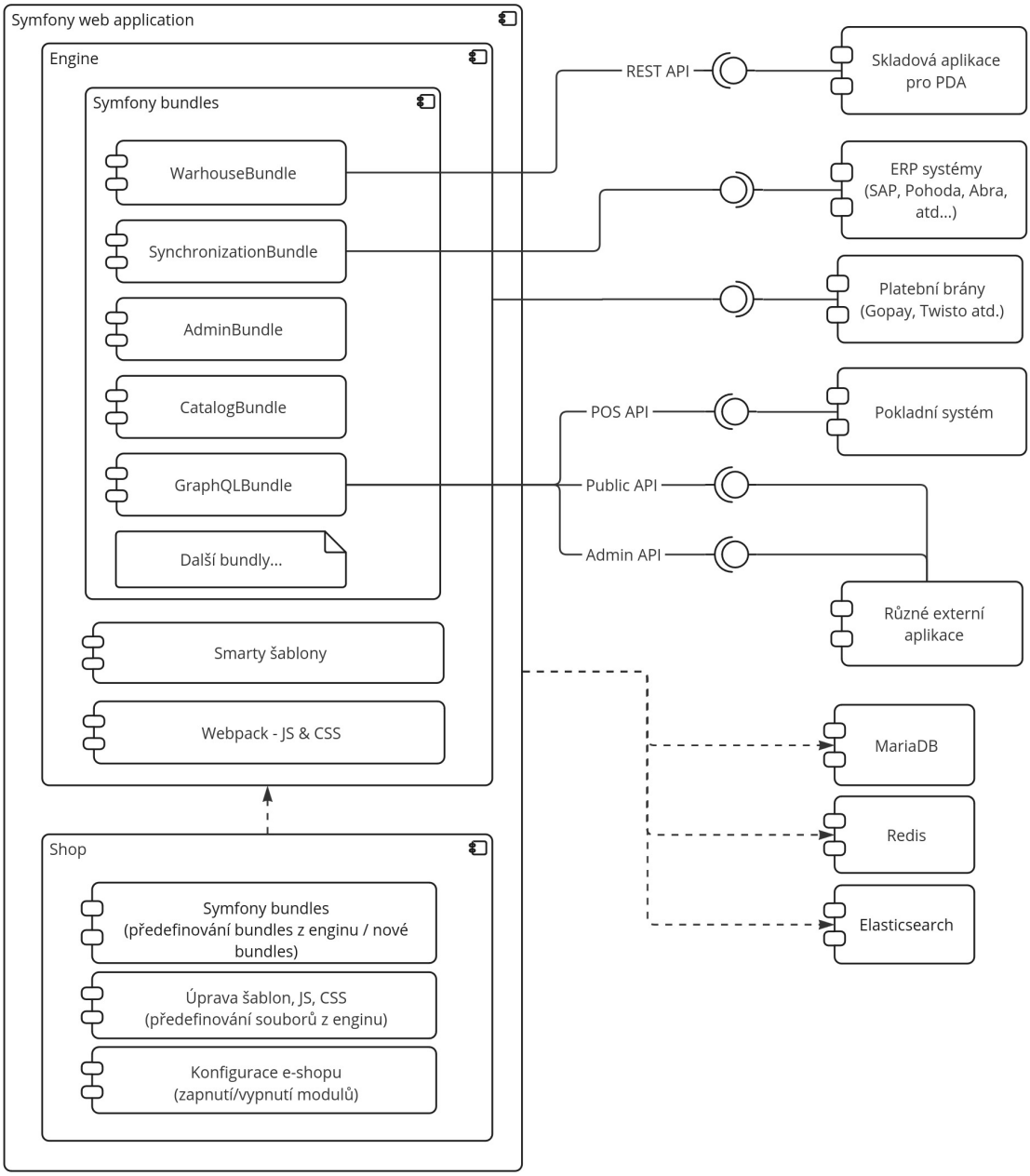
1.2.9 Architektura WPJshopu

WPJshop je rozdělen na dvě hlavní části. **První část** představuje společné jádro, označované interně jako „engine“. V tom jsou umístěny všechny sdílené komponenty a moduly, jejichž kód je k dispozici ve všech instancí/e-shopech WPJshopu. Některé moduly jsou placené a v rámci jednotlivých e-shopů lze tyto moduly libovolně aktivovat nebo deaktivovat. V jádře je implementována administrační část, logika prezentace katalogů/zboží, skladová část, GraphQL API a mnoho dalších modulů a funkcionalit, které mohou být využity pro více e-shopů. Skladová část (WarehouseBundle) je modul / Symfony bundle implementující logiku pro evidenci skladových zásob a existuje k němu i aplikace pro PDA čtečky, která je na tomto modulu závislá. Pokladní systém je také samostatná aplikace, která komunikuje s WPJshopem přes vlastní GraphQL rozhraní (POS API), jehož implementace je obsažena v Symfony bundlu. Veřejné a administrační GraphQL API je využíváno externími aplikacemi. WPJshop obsahuje celkově již přes 90 bundlů/modulů, které postupem času vznikly k naplnění všech potřebných funkcionalit požadovaných zákazníky.

Druhá část obsahuje konfiguraci a přizpůsobení pro jednotlivé e-shopy (interně „shop“). Ta obsahuje různé konfigurační soubory, kterými lze upravit vlastnosti jednotlivých e-shopů. Kromě konfiguračních souborů lze i předefinovat jednotlivé třídy / Symfony služby, Smarty šablony, CSS/JS soubory atp. Dále může shop obsahovat i zcela nové vlastní moduly / Symfony bundly či skripty, které přísluší jednotlivým e-shopům a není důvod je sdílet v jádru.

Tento popis architektury je pouze na úrovni znázornění základního konceptu, nelze tedy říci že diagram vystihuje všechny komponenty a jejich závislosti, které WPJshop obsahuje. Pro účel této práce však není třeba zacházet do detailů, které nejsou relevantní.

Popisovanou architekturu lze vidět na diagramu komponent 1.1. Symfony aplikace je rozdělena na engine a shop. Engine obsahuje zmiňované desítky Symfony bundlů, definici šablon, soubory JS a CSS zastřešené Webpackem atp. Shop obsahuje konfigurační soubory, předefinované části enginu a rozšiřující logiku pro konkrétní e-shop. WPJshop je závislý na relační databázi MariaDB, key-value databázi Redis a fulltextovém vyhledávači Elasticsearch. Vystavené GraphQL API se dělí na 3 části – veřejnou, administrační a část pro pokladní systém. Skladová aplikace pro PDA zařízení komunikuje se Symfony aplikací prostředníctvým REST API, které vzniklo ještě před před samotným GraphQL API. Nechybí ani znázornění napojení na ERP systémy a platební brány.



Obrázek 1.1 Architektura WPJshopu – diagram komponent

Kapitola 2
Analýza problematiky

Analýza základních pojmů a technologií, které souvisí s návrhem a zavedením kešovacího mechanismu. Rozbor klíčových otázek pro návrh takového mechanismu a analýza současného stavu WPJshopu.

2.1 Základní pojmy

Doména, kterou se tato práce zabývá, se soustředí na komunikaci mezi uživatelem a serverem, kdy uživatel pomocí webového prohlížeče přistupuje ke zdrojům na serveru. Webové prohlížeče využívají standardizovaný HTTP protokol, prostřednictvím kterého probíhá komunikace s webovými servery.

Obsah vrácený ze serveru může být v mnoha formátech, např. HTML, JSON, XML atp. Obecně je princip kešování takového obsahu jednoduchý, stačí pouze uložit výsledek vrácený ze serveru do mezipaměti a pro opakované požadavky tento obsah z ní vrátit, bez nutnosti jeho opakovaného vygenerování. Toho ale ne vždy jednoduše bez hlubšího rozmyslu dosáhnout, odpověď ze serveru mnohdy obsahuje personalizovaná data, která jsou určena pouze jednotlivým uživatelům či určeným skupinám uživatelů.

Pro bližší popis této problematiky a všech jeho částí je nejdříve nutné blíže popsat protokol HTTP, jazyk HTML a přístupy k architektuře webové architektuře, aby bylo zřejmé, jak webový server funguje a s jakým kontextem při řešení této problematiky je třeba pracovat.

2.1.1 Protokol HTTP

HTTP je protokol pro model klient-server. Klientem může být právě například webový prohlížeč, zatímco serverem může být proces nazvaný webový server, který běží na počítači hostujícím jednu nebo více webových stránek. Klient odešle serveru zprávu s HTTP požadavkem. Server, který poskytuje zdroje, jako jsou soubory HTML, JSON a další, nebo vykonává jiné funkce jménem klienta, vrátí klientovi odpověď, která obsahuje informace o stavu dokončení požadavku a může také obsahovat požadovaný obsah v těle zprávy. [17]

Pro přesnou lokaci zdrojů na serverech se používá URL. Tento identifikátor obsahuje informaci o umístění zdroje v internetu a způsob jak se k tomuto zdroji dostat. Příkladem takového lokátoru může být adresa `https://www.example.org/products/?limit=10`, který indikuje použití protokolu `https`, doménu `www.example.org`, hierarchickou identifikaci zdroje `products` uvnitř serveru a query parametr `limit` s hodnotou 5. I bez bližší znalosti obsahu zdroje, který by mohl být touto URL adresou identifikován, lze říci, že se by se mohlo jednat o výpis produktů, který je limitován parametrem.

HTTP Protokol je navržen tak, aby umožnil zprostředkujícím síťovým prvkům zlepšit komunikaci mezi klientem a serverem. Webové stránky s velkým provozem často využívají webové cache servery, které dodávají obsah jménem zdrojových serverů, aby se zlepšila doba odezvy. [17]

HTTP je označován jako bezestavový protokol, neexistuje žádná vazba mezi dvěma požadavky, které jsou postupně prováděny na stejném spojení. To má za následek problémy pro uživatele, kteří se snaží souvisle komunikovat s určitými stránkami, například při používání nákupních košíků v elektronických obchodech. Ale zatímco samotné jádro protokolu HTTP je bezstavové, soubory HTTP cookies umožňují používat stavové relace (v anglické terminologii nazýváno „sessions“). Pomocí rozšiřitelnosti HTTP hlaviček se do zpráv přidávají HTTP cookies, které umožňují, aby server při zpracování každého HTTP požadavku sdílel stejný kontext nebo stejný stav. [18]

Soubory cookie jsou malé soubory informací, které webový server vytváří a odesílá do webového prohlížeče. Webové prohlížeče ukládají přijaté soubory cookie po předem stanovenou dobu. Příslušné soubory cookie se připojují ke všem budoucím požadavkům, které uživatel na webový server vznese. [19]

Jednoduchý příklad HTTP požadavku na server a odpovědi z něho lze vidět na ukázce 2.1 a 2.2. Požadavek je na zdroj `/produkty` pro doménu `www.example.org` s indikací požadovaného návratového typu obsahu `text/html`, indikací preferovaného jazyku vráceného obsahu `en-US` a hlavičkou `Cookie` s proměnnou `PHPSESSID`, která obsahuje klíč k uživatelské relaci. Tento klíč slouží právě k identifikaci uživatelské relace, která je uložena na serveru. Odpověď na tento požadavek obsahuje také hlavičky s metadaty o obsahu (viz. `date`, `server`, `content-type` atp.) a samotnou odpověď ve svém těle.

```
GET /produkty/ HTTP/2
Host: www.example.org
Accept: text/html
Accept-Language: en-US,en
Cookie: PHPSESSID=b1pmta7b1399auce497jc
```

Výpis kódu 2.1 Ukázka HTTP požadavku

```
HTTP/2 200
date: Sun, 18 Feb 2024 15:51:54 GMT
server: nginx/1.19.3
content-type: text/html; charset=UTF-8
content-length: 49028

<!DOCTYPE html>... (obsah délky 49028 bytů)
```

Výpis kódu 2.2 Ukázka HTTP odpovědi

`PHPSESSID` je výchozí název pro cookie identifikátor klíče relace ve skriptovacím jazyce PHP. Tento název jde změnit dle libosti v případě potřeby. Pro různé webové servery může mít tento identifikátor jiný název, např. `JSESSIONID` pro server implementovaný v Javě.

V případě, že klient nemá přiřazenou žádnou relaci, server může relaci pro libovolný požadavek vytvořit a v odpovědi na požadavek klíč k této relaci vrátit v hlavičce `Set-Cookie`. Tato hlavička nastaví hodnotu cookie proměnné na požadovanou hodnotu. Ukázku takové odpovědi, která obsahuje `Set-Cookie` hlavičku, lze vidět v ukázce 2.3.

```
HTTP/2 200
date: Wed, 21 Feb 2024 11:57:55 GMT
server: nginx/1.19.3
content-type: text/html; charset=UTF-8
set-cookie: PHPSESSID=2vuctfbbj7vn28cptbqlhtkjr0;
    expires=Sat, 24-Feb-2024 11:57:55 GMT; Max-Age=259200;
    path=/; secure; httponly; samesite=lax
content-length: 40054

<!DOCTYPE html>... (obsah délky 40054 bytů)
```

Výpis kódu 2.3 Ukázka HTTP odpovědi – nastavení cookie ze strany serveru

Už z tohoto úvodního přehledu HTTP protokolu a funkce HTTP cookies je patrné, že pokud by kešovací mechanismus fungoval tak, že by se odpovědi ze serveru ukládaly včetně hlaviček do mezipaměti a tyto uložené odpovědi by byly dostupné více uživatelům, pak by přítomnost `Set-Cookie` hlavičky v uložené odpovědi mohla představovat bezpečnostní riziko. Konkrétně by hlavička určená pro jednoho klienta mohla být omylem sdílena s jinými, což by jim umožnilo přistupovat k relacím, ke kterým nemají mít přístup.

2.1.2 Jazyk HTML

Jazyk HTML (HyperText Markup Language) je nejrozšířenější způsob jak definovat význam a strukturu webového obsahu. K popisu vzhledu/chování tohoto formátu se používají i další technologie (CSS a JavaScript). Pojem „Hypertext“ se vztahuje k odkazům, které propojují webové stránky mezi sebou, ať už uvnitř jednoho webu nebo mezi různými weby. [20]

Jazyk HTML používá „tagy“ k anotaci struktury dokumentu. Názvy jednotlivých tagů se uzavírají mezi ostré závorky „<“ a „>“. Příklady tagů jsou např. `<head>`, `<title>`, `<body>`, `<p>`, `<div>` a mnoho dalších. HTML „element“ je text, který začíná otevíracím tagem a končí uzavíracím tagem. Element může také obsahovat jiné vnořené elementy; tímto způsobem vzniká stromová struktura, která reprezentuje obsah stránky. [20]

HTML, jako formát pro webové prezentace, může být z jednoho zdroje / stejné URL vracen i s různým obsahem. Důvody pro to mohou být např. personalizace nebo jazyk textu. Pokud by se tento typ obsahu ukládal do mezipaměti a následně sdílel s více uživateli současně, může dojít k situaci, kdy uživatelé uvidí obsah v jiném jazyce, nebo se jim zobrazí cizí soukromé informace. Jedno z možných řešení, jak spojit kešování HTML obsahu a dynamičnost stránky, je ukládat do mezipaměti pouze určité neměnné části HTML dokumentu (elementy) a určité personalizované části dokumentu vždy doplnit dle uživatelovy relace. Koncept tohoto principu by mohl být založen např. na oddělení takovýchto částí pomocí speciálních HTML tagů, které by určovaly které části se mají ukládat do mezipaměti a které se mají vždy mají načíst ze zdrojového serveru. Avšak HTML, ani HTTP takovéto rozdělování pro účel kešování nativně nespecifikují. K takovému účelu je potřeba specializovaný middleware přímo pro tento účel.

2.1.3 Architektura webové aplikace

Dvě hlavní možnosti architektury pro vytváření webových aplikací jsou jednostránkové aplikace (SPA) a vícestránkové aplikace (MPA). Ty se mohou ještě dělit na principu renderování obsahu, který může být na straně serveru (SSR), či na straně klienta (CSR). Návrh kešovacího mechanismu závisí na těchto principech, protože dodání obsahu ze serveru se může lišit v závislosti na zvolené architektuře.

▶ **Definice 2.1.** ***Úvodní požadavek*** *(neboli* ***prvotní požadavek****) je v této práci první požadavek na zdroj webového serveru, který získává základní HTML obsah stránky a v elementu* `<head>` *specifikuje další statické zdroje, jako jsou JS a CSS. Tyto požadavky jsou běžně vyvolány po vložení URL do prohlížeče, po kliknutí na URL odkaz, nebo po aktualizaci stránky.*

▶ **Definice 2.2.** ***Prvotní HTML stránka*** *je obsah odpovědi na prvotní požadavek.*

Jednostránková aplikace (SPA) je webová aplikace nebo web, který používá jedinou HTML stránku. Tato stránka dynamicky aktualizuje obsah pomocí JavaScriptu, který komunikuje se serverem na pozadí, místo aby načítala nové stránky celé. Tento přístup snižuje množství dat přenášených mezi klientem a serverem, protože není nutné při každém požadavku znovu načítat společné prvky uživatelského rozhraní, jako jsou navigace, záhlaví a zápatí. Díky tomu lze lépe oddělit front-end a back-end logiku, což zjednodušuje vývoj a údržbu aplikace. SPA umožňuje využití pokročilých JavaScriptových frameworků a knihoven, které podporují sdílení stavu aplikace a reaktivní aktualizace uživatelského rozhraní v reálném čase. Efektivní využití mezipaměti prohlížeče zvyšuje výkon aplikace a zlepšuje uživatelskou zkušenost i v případě slabého internetového připojení nebo výpadků. Správný návrh SPA také umožňuje uživatelům pokračovat v práci s určitými funkcemi aplikace i při dočasném odpojení od internetu. [21, 22]

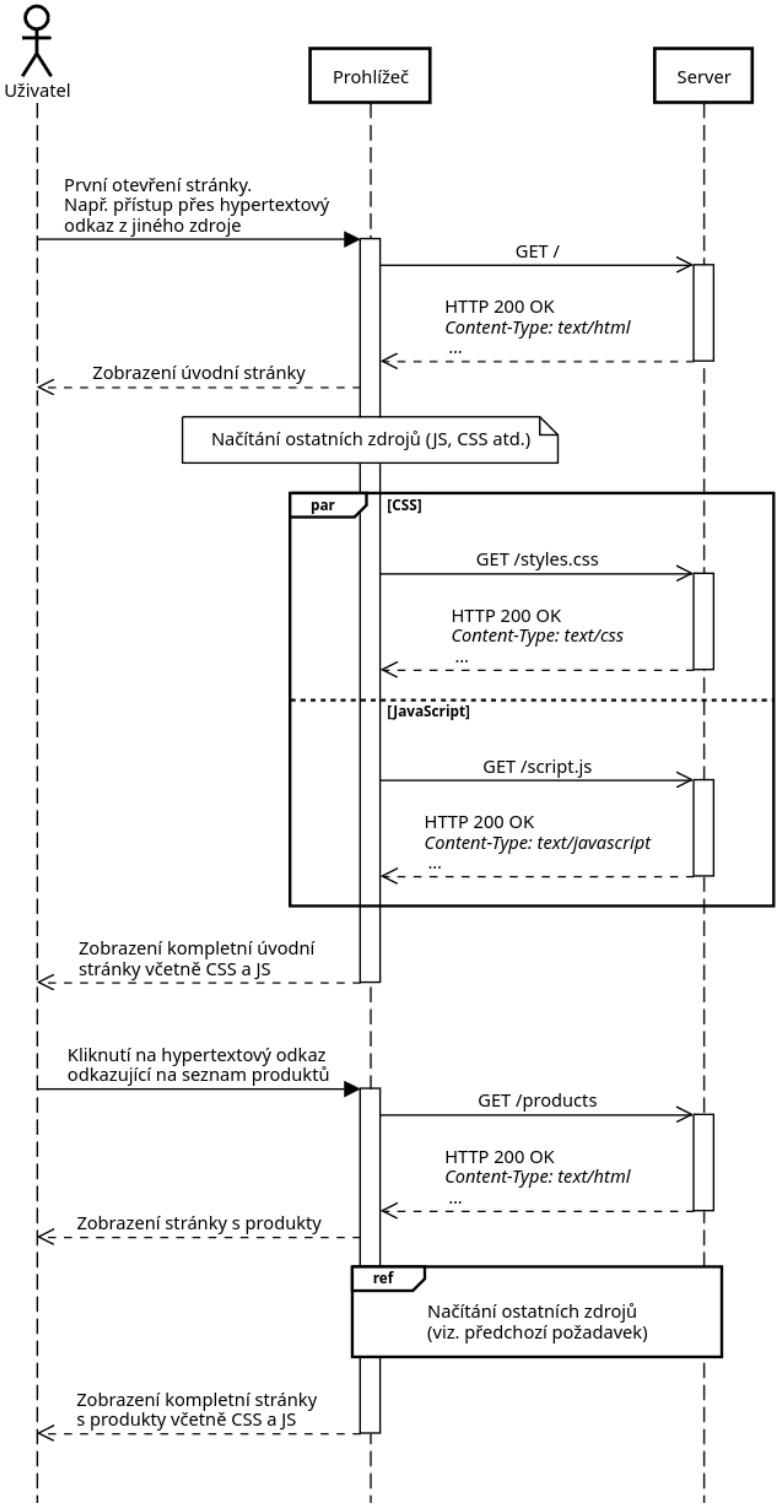
Tato architektura přináší určité výzvy, např. v oblastech podpory klientů a optimalizace pro vyhledávače (SEO). Existuje možnost, že někteří uživatelé mohou mít JavaScript vypnutý nebo ho jejich prohlížeče nemusí vůbec podporovat, což může omezit funkčnost aplikace. Co se týče SEO, dynamická povaha SPA znamená, že URL se často nemění, což může vést k menšímu množství stránek, jež lze indexovat vyhledávači. Také prvotní HTML stránka, který vyhledávací stoje používají k indexaci, nemusí obsahovat celý obsah, což snižuje dohledatelnost. Tento problém je možné řešit, ale vyžaduje to dodatečná opatření, jako je třeba pre-rendering. [22, 23]

Vícestránková aplikace (MPA) je typ webové aplikace, která má několik samostatných stránek s různými URL, mezi kterými se naviguje pomocí hypertextových odkazů uvnitř obsahu. Tento typ aplikací je považován za tradiční přístup, vzhledem k tomu, že jedná o původní koncept webových aplikací. Tato architektura je jednodušší v tom smyslu, že prvotní HTML stránka již obsahuje všechen důležitý obsah, tedy z hlediska SEO ji lze jednodušeji indexovat. Na druhou stranu, oproti SPA, je potřeba přenést více obsahu ze serveru při procházení několika stránek za sebou. Vždy při načtení nové stánky se musí stáhnout celý HTML obsah včetně záhlaví, zápatí, navigace atp. [22]

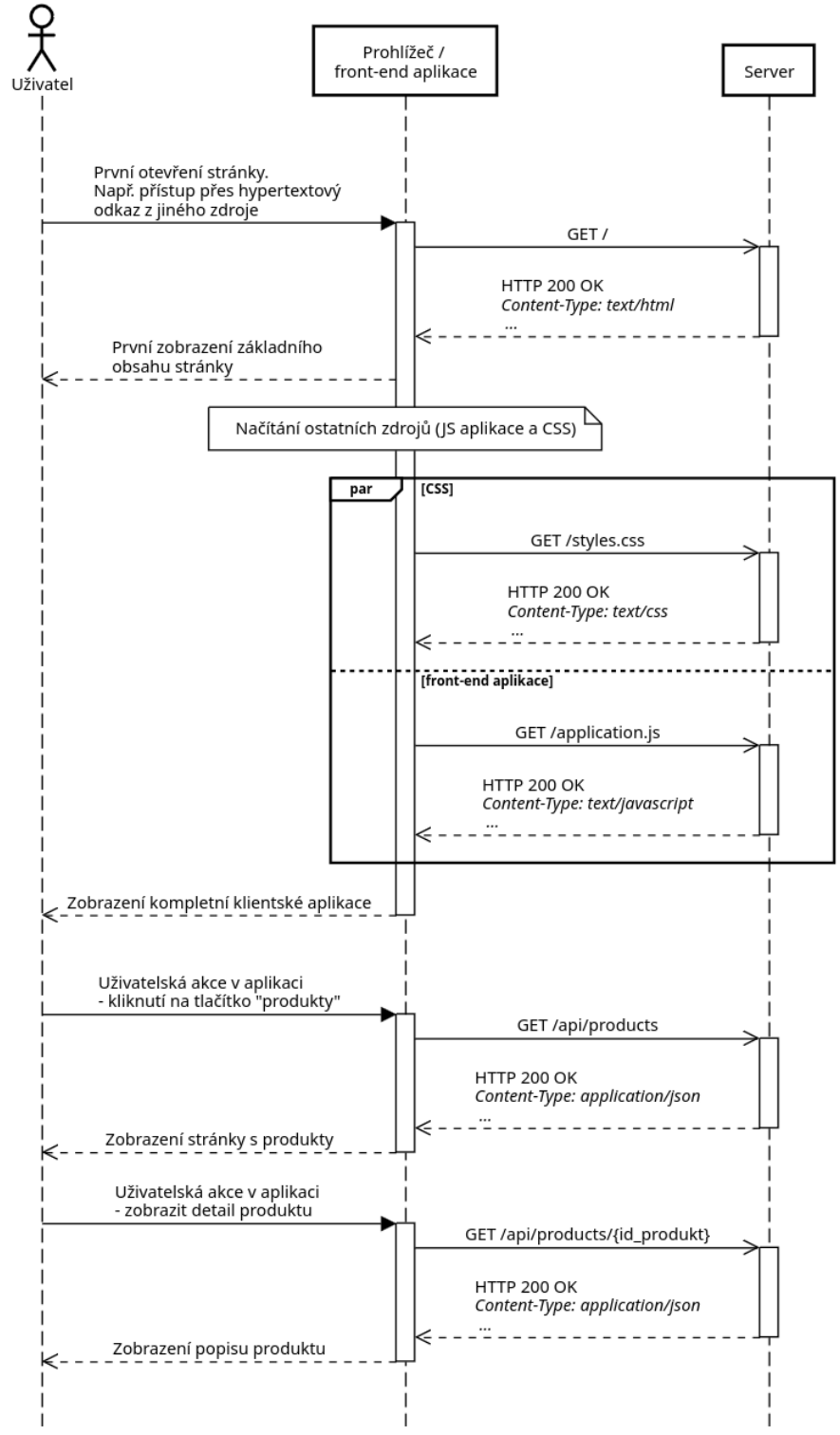
Mimo rozdělení architektury aplikace na SPA a MPA lze definovat i pojmy SSR (server-side-rendering) a CSR (client-side-rendering), které se také vztahují k principu doručení obsahu ke klientovi. Pojem CSR, tedy vykreslování na straně klienta, je technika, při níž proces vykreslování probíhá na straně klienta, obvykle ve webovém prohlížeči uživatele. V tomto případě server odesílá klientovi data v některém z kompaktních formátu (např. JSON) a aplikace na straně klienta tento obsah vykreslí. Na druhé straně je pojem SSR, tedy vykreslování na straně serveru, u kterého renderování obsahu probíhá již na serveru a klientovi je dodáván již kompletní strukturovaný HTML obsah. V tomto případě se klient nemusí starat o strukturizaci přijatého obsahu a stačí mu přijatý obsah pouze vložit do stránky. [24]

Nelze říci, že každá aplikace se dá rozdělit buď na MPA a SPA, nebo SSR a CSR. Ve většině případů aplikace kombinují jednotlivé principy k dosažení potřebné funkcionality. Pro MPA aplikace, vzhledem ke svému principu, je přirozenější využití SSR. SPA aplikace na druhou stranu mohou volně volit mezi SSR i CSR.

Na sekvenčních diagramech lze vidět příklad komunikace mezi klientem a serverem v případě použití architektury MPA + SSR (obr. 2.1) a v případě použití SPA + CSR (obr. 2.2).



Obrázek 2.1 Komunikace klient-server pro vícestránkové aplikace (MPA + SSR) – sekvenční diagram



Obrázek 2.2 Komunikace klient-server pro jednostránkové aplikace (SPA + CSR) – sekvenční diagram

2.2 Kešování obsahu

V rámci webových aplikací, kešování obsahu – ukládání obsahu získaného ze serveru do mezipaměti pro jeho možné opětovné použití, pokud by tentýž obsah byl znovu vyžádán – je technika, která pomáhá snižovat zátěž serveru a zkracovat dobu reakce. Při vytváření kešovacího mechanismu pro určitý typ obsahu je důležité zvážit několik klíčových bodů, dle kterých lze získat představu, jak má takový mechanismus vypadat, aby bylo zajištěno efektivní a vhodné kešování. Zmiňované body, které zvážit, mohou být následující:

- **Druh kešovaného obsahu**

  Obsah může být buď statický nebo dynamický.

- **Frekvence změny obsahu**

  Je důležité zvážit frekvenci s jakou se obsah mění a zda lze tuto změnu předpovědět.

- **Důležitost zobrazení aktuálního obsahu**

  Důležitost zobrazení aktuálního obsahu uživatelům se liší v závislosti na kontextu. V určitých situacích je nezbytné zajistit, aby uživatelé vždy obdrželi nejnovější informace.

- **Požadavky na konzistenci obsahu**

  Je nutné si položit otázku, zda je důležité zajistit konzistenci informací napříč všemi stránkami. Může dojít k situaci, kdy jedna stránka načtená z mezipaměti obsahuje zastaralé informace, zatímco jiná stránka zobrazuje stejné informace v aktuálnější podobě. Například skladová dostupnost jednoho produktu se v důsledku kešování může lišit na různých stránkách téhož webu.

- **Lokalizace obsahu**

  Je třeba zvážit geografické faktory, jako je potřeba lokalizovaného obsahu pro uživatele v různých regionech.

- **Kapacita serveru a infrastruktury**

  Jaké jsou k dispozici mechanismy pro ukládání do mezipaměti v současné infrastruktuře a zda je možnost využití nějakých externích služeb pro ukládání do mezipaměti.

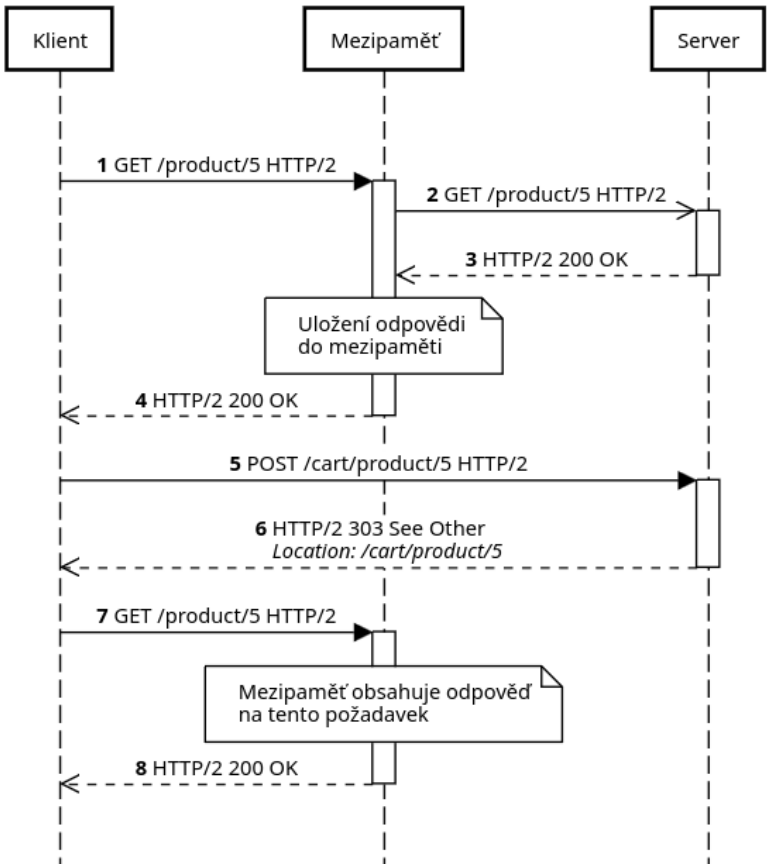
- **Důsledky pro bezpečnost a soukromí uživatelů**

  Pro určité typy obsahu mohou platit právní předpisy týkající se jeho ukládání. To zahrnuje například citlivé uživatelské údaje, na které se mohou vztahovat specifické zákony.

2.2.1 Kešování a REST

Již bylo uvedeno, z jakého důvodu lze chtít odpovědi ze serveru kešovat a co vše je třeba zvážit při návrhu kešovacího mechanismu. Komunikace mezi klientem a serverem pomocí HTTP protokolu však může vypadat různě. V některých případech, pokud se architektura nedrží alespoň základních REST pravidel, může být návrh takového mechanismu komplikovaný. Díky softwarové architektuře REST je však možné popsat i obecný princip kešování, který může být použit pro téměř libovolnou aplikaci využívající tento standard.

REST standardizuje použití HTTP metod. HTTP metoda GET je považována za bezpečnou (anglicky „safe“) a slouží pouze ke čtení. Tato skutečnost dělá právě z GET odpovědi ideálního kandidáta pro kešování. O ostatních „unsafe“ metodách, jako jsou POST, PUT a DELETE, nelze říci, že by byly ideálními kandidáty ke kešování, vzhledem k tomu, že účel jejich použití je mít efekt na serveru. Tedy je žádoucí, aby požadavek s touto metodou byl vždy doručen na server.



Obrázek 2.3 Příklad kešování HTTP pro RESTful server

Názorný příklad lze vidět na obrázku 2.3, kde je znázorněna komunikace mezi klientem a RESTful serverem. První požadavek klienta je zobrazení stránky produktu s identifikátorem 5. Vzhledem k tomu, že je to první požadavek, mezipaměť ještě neobsahuje obsah odpovědi na tento požadavek. Požadavek je tedy delegován na zdrojový server, který odpověď musí vygenerovat a odeslat zpět. Vrácená odpověď je uložena do mezipaměti a přeposlána klientovi. Na základě získaných dat o produktu klient provede další akci: přidání produktu do košíku. Tato operace je provedena pomocí HTTP POST metody na URL `/cart/product/5`. POST požadavek neinteraguje s mezipamětí, je nutné aby požadavek prošel až na server a není tedy žádoucí ani kešovat jeho odpověď. Přesměrování, které je v odpovědi na POST požadavek, odkazuje zpět na URL produktu. Klient tedy opět přistupuje na `/product/5`. Pro tento požadavek již mezipaměť má odpověď uloženou, není nutné, aby tento požadavek musel server zpracovat znovu.

V mezipaměti hrají roli i další vlastnosti, mezi které patří expirace, revalidace, mazání a případně další. Jaké všechny další vlastnosti může mezipaměť mít, záleží na její implementaci. Podrobný popis těchto vlastností je popsán v kapitole 3.1.

2.3 Analýza WPJshopu: Kešování

WPJshop je příkladem multi-page aplikace (MPA), která pro zlepšení uživatelského zážitku navíc využívá AJAX. Tato technologie se uplatňuje především při uživatelských akcích, jako je filtrace produktů v katalogu, načítání různých variant produktů, nebo úpravy obsahu nákupního košíku. Díky tomu je WPJshop schopen nabídnout uživatelům plynulé a intuitivní prostředí. Veškerý

uživatelský obsah je vygenerován na straně serveru (SSR) a vrácen ve většině případů v prvotní HTML odpovědi.

E-shopy platformy zahrnují personalizované prvky, jako je právě košík, doporučené produkty, naposledy navštívené produkty a rozdílná skladovost/cena produktů podle aktivního uživatelského účtu. Některé WPJshop e-shopy také automaticky přizpůsobují jazyk a obsah stránek podle preferencí uživatele nebo jeho geografické polohy. Všechny tyto aspekty podtrhují složitost a dynamickou povahu WPJshopu v kontextu kešování.

2.3.1 Stávající metody kešování WPJshopu

Jak již bylo řečeno v popisu WPJshopu 1.1, WPJshop už používá přes 170 klientů napříč celou Evropou. Nelze tedy říci, že by platforma ve své stávající formě již nějaké kešovací prvky neobsahovala.

WPJshop aktuálně využívá Redis k ukládání mezivýsledků aplikační vrstvy, což umožňuje snížit zátěž relační databáze, která by jinak musela obsloužit velké množství dotazů při každém požadavku. To by vedlo k přehlcení databáze potenciálně zbytečnými a redundantními dotazy. Příklad využití této key-value databáze je např. kešování dostupných doprav a plateb pro objednávky nebo načítání stromu sekcí – z povahy těchto dat je zřejmé, že není vždy potřeba dotazovat aktuální data relační databáze a lze je ukládat do této mezipaměti.

Dále je využívána i tzv. „Symfony Reverse Proxy“, což je Symfony komponenta sloužící ke kešování HTTP obsahu právě na úrovni Symfony aplikace. Jedná se o implementaci reverzní kešovací proxy v jazyce PHP, která může sloužit jako jednoduchý startovací bod pro nenáročné kešovací účely. Její síla spočívá v jednoduché integraci do Symfony aplikací. Mimo implementaci standardu RFC 9111 poskytuje i podporu značkovacího jazyku „Edge Side Includes“, který umožňuje mít součástí HTML obsahu i speciální tagy, které dovolují oddělit obsah na jednotlivé části. Tyto části se potom mohou lišit ve strategii kešování a mohou mít rozdílnou dobu platnosti v mezipaměti, nebo některé části nemusí být kešované vůbec. Popis technologie Edge Side Includes a standardu RFC 9111 je detailně popsán, společně s dalšími technologiemi, v následující kapitole 3.

WPJshop tuto Symfony komponentu používá právě ke kešování některých zdrojů, které vyžadují odlehčit od potřeby generování celého HTML obsahu. Každý požadavek je však stále potřeba obsloužit zdrojovým serverem, ten se zároveň stará o správu mezipaměti a generování obsahu. Do mezipaměti jsou ukládány statické části a dynamické části jsou generovány při každém požadavku.

Vzhledem ke své implementaci v jazyce PHP, nedosahuje Symfony Reverse Proxy stejné efektivity jako některé ostatní reverzní proxy. Tento fakt plyne přímo z dokumentace Symfony, kde lze nalézt následující text:

> *Symfony Reverse Proxy je skvělý nástroj k použití při vývoji vašeho webu nebo při nasazení na sdíleném hostingu, kde nemůžete instalovat nic kromě PHP kódu. Nicméně, protože je napsaná v PHP, nemůže být tak rychlá jako proxy napsaná v C. Naštěstí, vzhledem k tomu, že všechny reverzní proxy jsou v podstatě stejné, měli byste být schopni přejít na něco robustnějšího.* [25]

Z jistého pohledu lze říci, že samotná dokumentace Symfony frameworku vyzývá vývojáře, že mohou využít i efektivnější implementace kešovacích proxy.

2.3.2 Prostor pro zlepšení

I přes to, že WPJshop již využívá některé kešovací mechanismy, stále se oproti plnému potenciálu jedná pouze o mezistupeň, který vznikl cestou nejmenšího odporu pro naplnění potřebných nároků v průběhu let. Tyto mechanismy, ačkoli poskytují značné zlepšení výkonu a pomohly

vystát mnoho Black Friday akcí, představují pouze část toho, co by bylo možné dosáhnout s plně optimalizovaným systémem.

Jednou z oblastí pro zlepšení je využití úložiště prohlížeče, ať už k ukládání celých HTTP odpovědí, nebo jen částí dat prostřednictvím API prohlížeče pomocí JavaScriptu. Všechna data, která se nemění s každým požadavkem, jsou nyní stahována ze zdrojového serveru při každé interakci s webem, což vede k nepotřebnému zatížení serveru a zpomalení zpracování požadavků. Využitím úložiště prohlížeče by se dala tato zbytečná zátěž výrazně snížit a zkrátit tím dobu načítání stránek.

Společně se zapojením JavaScriptu a paměti prohlížeče lze uvažovat o redukování rozsahu prvotních HTML odpovědí ze serveru, které nyní obsahují veškerý obsah. Některé části obsahu lze získat prostřednictvím API až v případě, kdy bude obsah potřeba a vyrenderovat ho na straně klienta. K tomuto se přímo nabízí například obsah košíku, nebo podobná personalizovaná data.

Dalším výrazným aspektem pro zlepšení je umožnění obsluhy některých požadavků z mezipaměti bez potřeby kontaktovat zdrojový server. Tímto by se snížil počet požadavků, který zdrojový server musí zpracovat a zároveň by se výrazně snížila doba čekání na odpověď požadavku. S tímto souvisí i uvažované nahrazení Symfony Reverse Proxy komponenty za efektivnější implementaci, čímž by se nejen zvedla rychlost obsluhy HTTP mezipaměti, ale mohla by být právě provozována jako samostatná aplikace mimo Symfony framework.

Kapitola 3
Dostupné technologie

V této kapitole je popis existujících technologií, které lze použít pro kešování obsahu – standard HTTP protokolu pro kešování, typy kešovacích middlewarů a další možné technologie, které slouží ke snížení latence a potřebného výkonu webového serveru.

3.1 Kešování HTTP

Protokol HTTP s sebou přináší i standard **RFC 9111**. Ten specifikuje aspekty HTTP, které souvisí s kešováním a přepoužitím odpovědí ze serveru.

Mezipaměť HTTP je úložiště zpráv s odpovědí a subsystém, který řídí ukládání, načítání a mazání zpráv v ní. Mezipaměť ukládá odpovědi, u kterých je to povoleno, aby se snížila doba odezvy a spotřeba šířky pásma sítě při budoucích ekvivalentních požadavcích. [26]

Mezipaměť považuje uloženou odpověď za „čerstvou“ (anglicky „fresh“), pokud ji lze znovu použít bez ”validace”. Čerstvá odpověď tak může snížit latenci i síťovou režii při každém opakovaném použití mezipaměti. Pokud odpověď uložená v mezipaměti není čerstvá (zastaralá – anglicky „stale“), může být přepoužita, pokud je revalidována, nebo pokud je zdrojový server nedostupný. [26]

3.1.1 Typy HTTP mezipamětí

Existují dva hlavní typy HTTP mezipamětí, které se liší podle místa uložení kešovaných dat a způsobu jejich správy. Celá tato kapitola (3.1.1) vychází ze zdrojů [26, 27].

- **Soukromá mezipaměť**

  Soukromá mezipaměť (anglicky „**private cache**“) je vázaná na konkrétního klienta – obvykle HTTP mezipaměť prohlížeče. Vzhledem k tomu, že uložená odpověď není sdílena s ostatními klienty, může soukromá mezipaměť ukládat personalizovanou odpověď pro daného uživatele.

- **Sdílená mezipaměť**

  Sdílená mezipaměť (anglicky „**public cache**“) je umístěna mezi klientem a serverem a může uchovávat odpovědi, které mohou být sdíleny mezi uživateli. Pokud je personalizovaný obsah uložen ve sdílené mezipaměti, ostatní uživatelé mají možnost tento obsah načíst, což může způsobit nežádoucí únik informací.

  Nejčastěji využívané sdílené mezipaměti pro účel kešování HTTP komunikace jsou tzv. „spravované mezipaměti“. Ty jsou nasazovány přímo vývojáři služeb, aby odlehčili zdrojovému serveru a efektivně doručovali obsah. Příkladem jsou reverzní proxy servery a CDN. Jejich přesné charakteristiky se však liší na základě jejich poskytovatele.

3.1.2 HTTP Cache-Control

Hodnota HTTP hlavičky `Cache-Control` se používá k uvedení direktiv pro mezipaměť v řetězci požadavek/odpověď. Direktivy pro mezipaměť jsou jednosměrné, tzn. že přítomnost direktivy v požadavku neznamená, že stejná direktiva je přítomna nebo zkopírována v odpovědi.

Proxy server, ať už implementuje mezipaměť nebo ne, musí předávat všechny kešovací direktivy do přeposlaných zpráv. Tyto direktivy mohou být využity všemi příjemci v řetězci požadavek/odpověď. Není možné zacílit kešovací direktivu pouze na určitou mezipaměť.

Celá tato kapitola (3.1.2) vychází ze zdroje [26], kde jsou detailně popsány všechny kešovací direktivy pro požadavky a odpovědi.

Direktivy odpovědi

Direktivy definované standardem pro odpovědi řídí chování mezipaměti při směru komunikace ze serveru na klienta. HTTP mezipaměti **jsou** povinny tyto direktivy implementovat.

Server může nastavit do hlavičky `Cache-Control` téměř libovolnou kombinaci direktiv, která společně definují pravidla pro kešování. Ne všechny kombinace direktiv však dávají smysl, význam některých direktiv se může navzájem vylučovat.

- **max-age** udává, po jaké době uložená odpověď bude považována za zastaralou, pokud její stáří přesáhne zadaný počet sekund.
- **s-maxage** udává, jak dlouho zůstane odpověď ve sdílené mezipaměti čerstvá. Direktiva je ignorována soukromými mezipaměťmi a nahrazuje hodnotu určenou direktivou `max-age`, pokud je přítomna.
- **private** udává, že sdílená mezipaměť nesmí ukládat odpověď (tj. odpověď je určena pro jednoho uživatele). Rovněž naznačuje, že soukromá mezipaměť může uložit odpověď.
- **public** explicitně udává, že odpověď může být uložena ve sdílené nebo soukromé mezipaměti i za okolností, které by normálně vedly k restrikci kešování. Například povoluje využití sdílené mezipaměti pro opětovné použití odpovědí, které obsahují autorizační hlavičky.
- **no-store** udává, že mezipaměť nesmí ukládat žádnou část požadavku ani odpovědi a nesmí použít odpověď k uspokojení jiného požadavku. Direktiva se vztahuje k soukromé i sdílené mezipaměti.
- **no-cache** udává, že odpověď může být uložena do mezipaměti, ale před každým opětovným použitím musí být úspěšně validována zdrojovým serverem. Použít lze tuto hlavičku v případě, pokud je povoleno opakovaně použít uložený obsah, ale je potřeba obsah vždy před jeho použitím validovat.
- **must-revalidate** udává, že jakmile se uložená odpověď stane zastaralou, mezipaměť nesmí tuto odpověď znovu použít k uspokojení jiného požadavku do té doby, dokud není úspěšně revalidována. Tuto direktivu by měly servery používat pouze tehdy, pokud by neprovedení validace požadavku mohlo způsobit nesprávnou operaci, například v tichosti neprovedenou finanční transakci.
- **proxy-revalidate** funguje analogicky jako direktiva `must-revalidate`, pouze s tím rozdílem, že tato direktiva se neaplikuje pro soukromé mezipaměti.

Direktivy požadavku

Direktivy definované standardem pro požadavky řídí chování mezipaměti při směru komunikace z klienta na server. Mezipaměti nejsou povinné je implementovat, slouží pouze jako doporučení pro jejich chování.

Jedním z mnoha příkladů použití může být vyžádání aktuálního obsahu, nebo jeho revalidace. Výčet základních direktiv požadavku `Cache-Control` hlavičky je následující:

- **max-age** udává, že klient preferuje odpověď, jejíž stáří je menší nebo rovno zadanému počtu sekund.
- **max-stale** udává, že klient přijme odpověď, která překročila dobu čerstvosti. Pokud je uvedena hodnota, pak je klient ochoten přijmout odpověď, která překročila dobu čerstvosti maximálně o zadaný počet sekund.
- **no-cache** udává, že klient preferuje, aby nebyla použita uložená odpověď k obsloužení požadavku bez úspěšného ověření na zdrojovém serveru.
- **no-store** udává, že mezipaměť nesmí ukládat žádnou část tohoto požadavku, ani žádnou odpověď na něj. Tato direktiva platí pro soukromé i sdílené mezipaměti.
- **only-if-cached** udává, že klient si přeje získat pouze uloženou odpověď. Mezipaměti, které tuto direktivu požadavku respektují, musí po jejím přijetí odpovědět buď uloženou odpovědí v souladu s ostatními omezeními požadavku, nebo stavovým kódem HTTP 504.

3.1.3 Revalidace obsahu

Zastaralé odpovědi nejsou okamžitě smazány z mezipaměti. Protokol HTTP má mechanismus, který umožňuje přeměnit zastaralou odpověď na čerstvou tak, že požádá zdrojový server o informaci, zda je uložená odpověď stále aktuální. Tomu se říká validace, nebo někdy revalidace. Validace se provádí pomocí podmíněného požadavku, který obsahuje „precondition“ hlavičku `If-Modified-Since` nebo `If-None-Match`. [27]

V případě, že je potřeba uloženou odpověď z konkrétního zdroje revalidovat, musí být do požadavku na zdrojový server přidána precondition hlavička, jejíž obsah je běžně složen z hlaviček předchozí uložené odpovědi. Obohacený požadavek je poté odeslán na zdrojový server, který podle precondition hlavičky rozhodne, zda obsah mezipaměti revalidovat, nebo vygenerovat nový obsah. Pro účel revalidace server obsah znovu generovat nemusí, stačí pouze vrátit HTTP status 304 (Not Modified), který indikuje, že se obsah nezměnil a stále je možné využít obsah z mezipaměti. Pokud je vrácena odpověď s obsahem, znamená to, že obsah v mezipaměti již není aktuální a mezipaměť musí použít odpověď ze zdrojového serveru k naplnění požadavku a může aktualizovat svůj obsah z této odpovědi. [26, 27]

Jednou z hlaviček umožňujících revalidaci je `Last-Modified`, která obecně slouží jako datum toho, kdy byl obsah naposledy upraven. Pokud uložená odpověď tuto hlavičku obsahuje, pro podmíněný požadavek může být použita precondition hlavička `If-Modified-Since` s datumem z `Last-Modified`. Na základě této hlavičky může zdrojový server determinovat, zda mezipaměť obsahuje aktuální obsah.

Další hlavička umožňující revalidaci je `Etag`, jejíž obsah je textový řetězec složený z ASCII znaků umístěný mezi uvozovky. Typicky je `Etag` hash vytvořený z obsahu, ale může to být například i číslo verze obsahu. Do podmíněného požadavku poté může být přidána `If-None-Match` hlavička, analogicky jako je tomu u `Last-Modified` a `If-Modified-Since`.

3.1.4 Kešovací klíč

Kešovací klíč je informace, kterou mezipaměť používá k výběru odpovědi z úložiště – skládá se minimálně z HTTP metody (GET, POST, atd.) a cílového URI. Mnoho dnes běžně používaných HTTP mezipamětí však ukládá do mezipaměti pouze odpovědi GET metody, a proto jako klíč mezipaměti používají pouze samotnou URI. [26]

Obsah odpovědi pro jednu URI vždy nemusí být stejný. Typ obsahu může záviset na hodnotách hlaviček požadavku `Accept`, `Accept-Language` atp. Pro takové případy může být kešovací

klíč rozšířen pomocí hlavičky `Vary` v odpovědi, která přizpůsobí kešovací klíč tak, aby obsahoval i hodnoty hlaviček, které jsou ve `Vary` specifikovány. [26]

V určitých situacích je možné kešovací klíč rozšířit pomocí specifické hodnoty z `Cookie` hlavičky. Pokud jsou cookies uchovávány ve formátu klíč-hodnota, což je standardem, může být pro některé typy mezipamětí specifikováno, že by měly brát v úvahu pouze hodnotu z určitého klíče cookie. Tato specifická hodnota může pak být využita pro rozšíření kešovacího klíče. Nicméně tato funkcionalita není standardem a je podporována jen některými typy sdílených mezipamětí.

3.2 Edge Side Includes

Edge Side Includes (ESI) je značkovací jazyk navržený pro sestavování dynamického webového obsahu na úrovni edge serverů, což jsou servery umístěné na hranici mezi uživatelem a poskytovatelem obsahu. Může se jednat o CDN nebo reverzní proxy; ESI však může využívat i libovolný HTTP klient, pokud ho podporuje. Tento jazyk umožňuje webovým aplikacím označit části webových stránek jako oddělené fragmenty, které mohou být kešovány nezávisle. Edge servery musí být schopny rozumět jazyku ESI, aby mohly zpracovávat pokyny dané pomocí ESI tagů, které jsou umístěny v rámci HTML a instruují, co je třeba udělat pro dokončení sestavení stránky. [28, 29]

Příklad využití ESI tagu `include` je vidět na ukázce 3.1. Pro ukázku je uvažován zdrojový webový server poskytující HTML obsah a edge server podporující ESI. Do HTML dokumentu, který je vracen ze zdroje `/category`, jsou vkládány dva `esi:include` tagy; první odkazuje na zdroj poskytující navigaci stránky a druhý odkazuje na zdroj poskytující výpis produktů. Toto umožňuje uložit do mezipaměti edge serveru celou odpověď. Než však takováto odpověď může být vrácena uživateli, musí edge server vyslat požadavky na zdrojový server pro získání zdroje `/nav` a `/products`, z jejichž odpovědí je složen celý obsah. Každý jeden zdroj může mít navíc rozdílnou strategii kešování, která může být řízena hlavičkou `Cache-Control`. Tímto způsobem lze např. kešovat zdroj `/category` na 10 minut, zdroj `/products` na 3 minuty a zdroji `/nav` kešování neumožnit. V takovém případě při požadavku na zdroj `/category`, musí být navigace požadována ze zdrojového serveru vždy, zatímco všechny ostatní části mohou být poskytnuty z mezipaměti, pokud jsou čerstvé.

Mimo tag `esi:include` jsou k dispozici i další tagy, například `esi:choose`, `esi:when` a `otherwise`, umožňují definici podmínek na základě hlaviček požadavku, podobně jako je princip `if, else` u programovacích jazyků. [29]

```
<!DOCTYPE html>
<html>
   <head><!-- Metadata, JS, CSS --></head>
   <body>
      <nav>
          <esi:include src="/nav" />  <!-- navigace -->
      </nav>
      <div>
          <esi:include src="/products" /> <!-- výpis produktů -->
      <div>
      <footer><!-- Zápatí stránky --></footer>
    </body>
</html>
```

Výpis kódu 3.1 Ukázka použití Edge Side Includes

3.3 Reverzní proxy server

V počítačových sítích je reverzní proxy server (RPS) aplikace, která se nachází před koncovými servery a předává požadavky klienta těmto serverům, namísto toho, aby klient komunikoval přímo se servery. Reverzní proxy servery pomáhají zvýšit škálovatelnost, výkon, odolnost a zabezpečení. Zdroje vrácené klientovi se tváří, jako by pocházely ze samotného webového serveru. [30]

Na rozdíl od dopředných proxy serverů, jsou reverzní proxy servery běžně nasazovány a spravovány poskytovateli webových služeb. Jednou z jejich funkcionalit je právě možnost ukládání odpovědí ze zdrojového serveru do mezipaměti, čímž slouží jako sdílená mezipaměť a spadají přímo do kategorie spravovaných mezipamětí, jak již bylo zmíněno v sekci 3.1.1.

Existuje mnoho open-source řešení pro provoz reverzního proxy serveru. Příkladem takových řešení mohou být Apache, Nginx, HAProxy, Varnish a monoho dalších. Komerční řešení proxy serverů nabízí např. Cloudflare, Microsoft Azure CDN, Fastly, Nginx Plus, Bunny.net a opět mnoho dalších.

Všechny běžné reverzní proxy servery implementují standard RFC 9111. Pokud se zdrojový webový server také drží tohoto standardu, není vázán na žádnou konkrétní implementaci reverzního proxy serveru a lze přejít z jedné implementace na druhou. Problém při změně implementace RPS může nastat v případě, že je využívána nějaká nadstandardní funkcionalita, kterou jiné implementace neposkytují.

3.3.1 Nginx

Nginx je HTTP server, reverzní proxy server, e-mailový proxy server a univerzální TCP/UDP proxy server. Jedná se o open-source software, ale existuje i jeho komerční varianta Nginx Plus, která poskytuje dodatečné funkcionality, mezi které patří API pro správu mezipaměti, integrovaný monitoring, integrovaná key-value databáze atp. [31, 32]

Nginx se skládá z modulů, které jsou řízeny pomocí direktiv zadaných v konfiguračním souboru. Direktivy se dělí na jednoduché a blokové. Jednoduchá direktiva se skládá z názvu a parametrů oddělených mezerami a končí středníkem. Bloková direktiva má stejnou strukturu jako jednoduchá direktiva, ale místo středníku je zakončena sadou dalších instrukcí obklopených složenými závorkami. [33]

Příklad části konfiguračního souboru lze vidět na ukázce 3.2. Jedná se o jednoduchou konfiguraci kešovacího RPS, jehož zdrojový server je nastaven na doménu `http://example.org`. Takto nastavený RPS se bude řídit hlavičkami v HTTP zprávách dle standardu RFC 9111. Kešovací klíč je nastaven tak, aby se skládal pouze z URL požadavku, ale může být rozšířen i z libovolných jiných částí požadavku, jako je třeba konkrétní hodnota cookie.

```
http {
  proxy_cache_path  /var/www/cache keys_zone=my-cache:8m;
  proxy_temp_path /var/www/cache/tmp;

  server {
    location / {
      proxy_pass http://example.org; proxy_cache my-cache;
      proxy_cache_key "$scheme$host$request_uri";
    }
  }
}
```

Výpis kódu 3.2 Ukázka konfiguračního souboru Nginx – kešovací reverzní proxy server

3.3.2 Varnish

Varnish je reverzní kešovací proxy server, který je zaměřen přímo na kešování HTTP a už od svého vzniku byl designován tak, aby fungoval jako HTTP akcelerátor. Je dostupný ve dvou formách: open-source projekt Varnish Cache a komerční produkt Varnish Enterprise. Komerční verze nabízí vše co open-source verze a navíc k tomu další funkcionality včetně SLA. [34]

Ve výchozí konfiguraci bude Varnish respektovat HTTP direktivy `Cache-Control` a obsah ukládat do mezipaměti po dobu, kterou zdrojový server určí. Velká část síly technologie Varnish však spočívá v tom, že jeho chování lze konfigurovat, rozšiřovat a měnit mnoha způsoby. Existuje mnoho parametrů, které lze ladit. Způsob zpracování požadavků a ukládání do mezipaměti lze změnit nebo zcela předefinovat pomocí konfiguračního jazyka Varnish – VCL (Varnish Configuration Language). [34]

Varnish navíc podporuje funkcionality jako request coalescing, **ESI**, transformaci URL, ACL, pokročilé logování a mnoho dalších. Veškerá funkcionalita je popsána v oficiální knize [34], kterou lze zdarma získat na oficiálních stránkách.[1]

Na ukázce 3.3 lze vidět Varnish konfiguraci, která požadavkům s URL začínající na `/admin` neumožňuje kešování a ostatním požadavkům modifikuje cookie hlavičku tak, aby v ní zůstala pouze hodnota `language`, která je zároveň použita pro rozšíření kešovacího klíče.

```
vcl 4.1;
import cookie;
backend default {
    .host = "backend.example.com";
    .port = "80";
}
sub vcl_recv {
    if(req.url ~ "^/admin(/.*)?") {
        return(pass);
    }
    cookie.parse(req.http.cookie);
    cookie.keep("language");
    set req.http.cookie = cookie.get_string();
    return(hash);
}
sub vcl_hash {
    hash_data(cookie.get("language"));
}
```

Výpis kódu 3.3 Ukázka konfiguračního souboru VCL [34]

3.4 Content delivery network

Content delivery network (CDN) je síť propojených serverů, která urychluje načítání webových stránek pro aplikace s velkým množstvím dat. Když uživatel navštíví webovou stránku, data ze serveru této stránky musí „cestovat“ po internetu, aby dosáhla počítače uživatele. Pokud je uživatel umístěn daleko od tohoto serveru, načtení velkého souboru, jako je video nebo obrázek webové stránky, bude trvat dlouho. Proto lze statický obsah webových stránek uložit na serverech CDN, které jsou geograficky blíže uživatelům, a data z nich se ke k jejich počítačům dostanou rychleji. [35]

[1] `https://info.varnish-software.com/the-varnish-book`

Na rozdíl od reverzních proxy serverů, CDN jsou navrhnuty tak, aby fungovaly v globálním měřítku, pokročilé algoritmy dokáží vypočítat nejvhodnější edge server pro obsloužení požadavku na základě geolokace a jsou obecně lépe přizpůsobeny pro poskytování statického obsahu, jakým jsou videa a obrázky. Typicky poskytují i dodatečné funkce, jako je DDoS ochrana, optimalizace obsahu, správa SSL/TLS certifikátů a video streaming optimalizace. [35, 36]

Mezi nejznámější poskytovatele CDN lze zařadit např. Cloudflare, AWS, Fastly, Akamai a český CDN77.

WPJ pro některé své e-shopy využívá CDN **Bunny.net** pro zrychlení poskytnutí obrázků a videí. Bunny.net je komerční CDN platforma, která poskytuje optimalizaci obrázků, streamování videí a ochranu proti DDoS útokům. Mezi poskytované služby patří i kešování HTTP obsahu, tedy lze ji využívat také jako reverzní kešovací proxy server. Včetně podpory standardu RFC 9111 lze i konfigurovat dodatečnou funkcionalitu, jako je přepisování kešovacích hlaviček a konfigurace podoby kešovacího klíče. Lze nakonfigurovat, že součástí kešovacího klíče jsou specifické hodnoty cookie, nebo pouze specifické parametry query stringu z URL. [37]

3.5 HTTP mezipaměť prohlížeče

Jak již bylo naznačeno v obecném popisu HTTP mezipamětí 3.1.1, HTTP mezipaměť prohlížeče je typickým případem soukromé mezipaměti. Je to úložiště, které prohlížeče používají k ukládání stažených webových zdrojů, jako jsou HTML stránky, obrázky a soubory JavaScript, aby zrychlily následné požadavky na stejné zdroje. Toto ukládání do mezipaměti pomáhá snižovat zatížení serveru a latenci. Chování HTTP mezipamětí prohlížeče je řízeno hlavičkou `Cache-Control` v odpovědi ze serveru a odpovídá standardu RFC 9111. [27]

Na rozdíl od sdílených spravovaných mezipamětí, jakými jsou reverzní proxy servery a CDN, které často nabízejí rozmanité rozšiřující funkcionality, HTTP mezipaměť prohlížeče je v tomto ohledu omezenější. Drží se striktněji standardu a neposkytuje přímou správu této mezipaměti pomocí JavaScriptu, její chování je řízeno pomocí hlavičky `Cache-Control` v požadavku/odpovědi. [26, 38]

Možným způsobem, jak rozšířit kešovací klíč, kromě úpravy URL, je použití hlavičky `Vary` v odpovědi ze serveru. Při vývoji SPA je většina požadavků vyvolána JavaScriptem a tak lze jednoduše přidat libovolnou vlastní hlavičku s vlastní hodnotou do všech požadavků. Přidáním této vlastní hlavičky do `Vary` lze pak jednoduše rozšířit kešovací klíč o libovolnou hodnotu. U MPA je to však komplikovanější. Jednotlivé úvodní požadavky na zdroje jsou většinou vyvolány prohlížečem po kliknutí na hypertextový odkaz. Pro rozšíření požadavku je proto nutné JavaScriptem odchytnout tuto událost a přidat hlavičku do požadavku dodatečně.

Jedním z dalších možných způsobů jak modifikovat každý požadavek před jeho odesláním je registrace vlastního Service Workeru[2], který dokáže běžet na pozadí prohlížeče a odchytávat každý požadavek vznesený na zdrojový server.

3.6 Kešování na úrovni aplikace

Existuje mnoho efektivních řešení pro kešování v architektuře webových systémů. Nicméně tradiční řešení jsou často nasazena mimo hranice aplikace jako middleware služby (reverzní proxy, CDN), které operují nezávisle na aplikační logice a tedy rozhodnutí o ukládání do mezipaměti nejsou učiněna s přihlédnutím k zvláštnostem konkrétních webových aplikací. V důsledku toho mohou být tato řešení méně efektivní, zejména v případě, pokud webová aplikace zpracovává složitou logiku a personalizovaný obsah. Ukládání do mezipaměti na úrovni aplikace může doplnit kešovací middleware systémy a potenciálně zlepšit výkon a škálovatelnost webových aplikací. [39]

[2] `https://www.w3.org/TR/service-workers/`

Zavedení kešování na úrovni aplikace s sebou přináší i dodatečnou výzvu: implementaci samotné logiky kešování. Tato logika vyžaduje, aby aplikace mezipaměť spravovala, což zahrnuje manuální vkládání a získávání obsahu, generování kešovacích klíčů a zachování konzistence mezi mezipamětí a zdrojem dat. Implementace a údržba ukládání do mezipaměti přímo v aplikaci zvyšuje složitost a čas potřebný k údržbě. Celkově jsou tedy kladeny vyšší požadavky na vývojáře aplikací a zvyšuje se tím i náchylnost k chybám.

3.6.1 Kešování na straně serveru

Pro kešování na straně serveru efektivně slouží in-memory databáze, které se vyznačují svým rychlým přístupem k datům. Mezi ně lze zařadit např. Redis nebo Memcached. Mnoho webových aplikačních frameworků poskytuje abstrakci mezipaměti, která umožňuje konzistentní používání různých řešení mezipaměti s minimálním dopadem na kód. Ve frameworku Symfony existuje Cache Component[3], která poskytuje několik různých adaptérů pro jednotlivé typy úložišť – Redis Cache Adapter, Memcached Cache Adapter, APCu Cache Adapter atp.

Běžně jsou tyto mechanismy využívány pro uložení mezivýsledků časově náročných operací bussines vrstvy aplikace, nebo databázových dotazů, u kterých není potřeba vždy získat aktuální data. Frameworky pro objektově relační mapování (ORM) běžně poskytují ukládání do mezipaměti jako strategii pro zlepšení výkonu aplikace snížením počtu potřebných dotazů na databázi. To může výrazně zrychlit dobu načítání dat a snížit zatížení databázového serveru.

Vývojářům nic nebrání využít tyto technologie pro ukládání celých HTML stránek. Uložením celé stránky do mezipaměti umožní webovému serveru obsloužit požadavky pouze tím, že z in-memory databáze se získá hotový výsledek, který stačí pouze vrátit. Kromě ukládání celého HTML obsahu je možné ukládat pouze určité statické části, jako je záhlaví, zápatí, navigace a ostatní dynamické části vždy vygenerovat. Tento způsob však má nevýhodu, že stále celá logika je obsažena ve zdrojovém serveru. To je obecně pomalejší, než částečné přenesení logiky na middleware.

3.6.2 Lokální úložiště prohlížeče

Jako mezipaměť na aplikační úrovni lze využít i lokální úložiště prohlížeče (neplést s HTTP mezipamětí prohlížeče). V průběhu let se rozšířila různá API, která lze snadno použít pro ukládání dat prostřednictvím JavaScriptu. Mezi nejpoužívanější rozhraní lze považovat `Cookies`, `LocalStorage` a `SessionStorage`, avšak existuje i `IndexedDB storage`. Rozdíly mezi jednotlivými úložišti spočívají v jejich účelu, kapacitě, životnosti a synchronnosti/asynchronnosti.

- **Cookies**

  Jak již bylo naznačeno v kapitole 2.1.1, HTTP Cookie je malý kousek dat, který server odesílá do webového prohlížeče uživatele. Prohlížeč může soubor cookie uložit a při pozdějších požadavcích jej odeslat zpět na stejný server. Cookie však může být vytvořena i JavaScriptem na straně klienta. Soubory cookie jsou odesílány při každém požadavku, takže zvětšují velikost všech požadavků. Lze u nich nastavit dobu životnosti a maximální velikost jedné cookie je standardem RFC 6265 limitována na 4096 bytů. [40]

- **LocalStorage a SessionStorage**

  Web Storage API poskytuje mechanismy, pomocí kterých mohou prohlížeče ukládat dvojice klíč/hodnota více intuitivnějším způsobem než pomocí souborů cookie. SessionStorage je k dispozici po celou dobu trvání relace stránky (dokud je okno prohlížeče otevřeno, včetně opětovného načtení a obnovení stránky). Data se nepřenášejí na server a limit velikosti je stanoven na 5 MB. LocalStorage funguje stejně, pouze s tím rozdílem, že přetrvává i po

[3] `https://symfony.com/doc/7.1/components/cache.html`

zavření a opětovném otevření prohlížeče a stejná data jsou dostupná ve všech otevřených oknech. V obou případech jsou data ukládána bez expirace. [41]

- **IndexedDB storage**

  IndexedDB je nízkoúrovňové API pro ukládání velkého množství strukturovaných dat, včetně blob/souborů. Umožňuje vytváření, čtení, úpravy a mazání dat v databázi na straně klienta s použitím asynchronních operací. Zajištěná je i podpora pro transakce a indexy, což umožňuje komplexní manipulaci s daty s konzistencí, izolací a rychlostí vyhledávání. Toto API je ideální pro webové aplikace, které potřebují pracovat s velkými objemy dat offline, nebo vyžadují složité dotazy na uložená data. Je navrženo tak, aby bylo schopno zvládnout i velmi náročné požadavky na ukládání dat. Pro využití není potřeba žádná knihovna, IndexedDB je nativně podporována většinou moderních prohlížečů. [42]

Kapitola 4

Požadavky a návrh řešení

V této kapitole jsou definovány funkční a nefunkční požadavky pro požadované řešení. Dále je obsažen koncept a návrh kešovacího mechanismu se sekvenčními diagramy znázorňující komunikaci mezi klientem a serverem.

4.1 WPJshop kontext

V následujících částech práce je často využíván pojem „kontext“. Proto je nutné ho zde definovat.

▶ **Definice 4.1.** *V případě WPJshopu je pojem „kontext“ používán jako stav, který ovlivňuje typ obsahu stránek, které uživatel vidí. Jedná se například o jazyk, zemi, cenovou hladinu uživatele atp. Jeden uživatel může mít český kontext a druhý uživatel anglický kontext – jejich obsah stránek se liší v jazyku textu. Zároveň může být kontext s 5% cenovou hladinou a kontext s 10% cenovou hladinou, v závislosti na kterém je uživateli zobrazována rozdílná cena produktů. Každý uživatel má kombinaci kontextů, které determinují jaký typ obsahu mu je vracen.*

4.2 Specifikace požadavků

Specifikace požadavků stanovuje požadované vlastnosti řešení. Jedná se o mechanismus, který primárně umožní kešování většiny HTML obsahu vráceného z webového serveru a sníží požadovanou výpočetní kapacitu a klientovu průměrnou dobu čekání na odpověď požadavku. Vzhledem k tomu, že se jedná o úpravu existujícího řešení, je také kladen důraz na zachování stávající funkčnosti.

4.2.1 Funkční požadavky

F1 – Neukládání personalizovaného obsahu ve sdílené mezipaměti

Sdílená mezipaměť ukládá a sdílí mezi více uživateli pouze obsah, který je k tomu webovým serverem označen a je určen být sdílen mezi více klienty.

F2 – Dodání obsahu klientovi bez potřeby kontaktování origin server

Mezipaměť umí obsloužit požadavky na HTML obsah bez toho, aby přeposlala požadavek na zdrojový server, pokud má k dispozici aktuální požadovaný obsah. Výjimkou může být HTML obsah obsahující ESI tagy, avšak je preferováno, aby se vyhnulo závislosti na zdrojovém serveru při každém požadavku.

F3 – Respektování vypršení platnosti obsahu v mezipaměti a revalidace

Zastaralý obsah v mezipaměti není využit k obsloužení nově příchozích požadavků. V případě expirace platnosti je obsah znovu vyžádán, nebo revalidován ze zdrojového serveru. Webový server podporuje možnost revalidace pro jednotlivé typy stránek a logika revalidace je jednoduše konfigurovatelná prostřednictvím jednotného rozhraní na aplikační vrstvě.

F4 – Kešování obsahu košíku v prohlížeči

Obsah košíku uživatele není součástí HTML odpovědi. Není žádoucí, aby se vkládal do každé HTML odpovědi. Obsah košíku je vyžádán jen v případě, kdy prohlížeč tuto informaci v mezipaměti neobsahuje, nebo je aktualizován.

F5 – Zachování oblíbených produktů

Funkcionalita, která uživateli umožňuje označit produkty e-shopu a zařadit je mezi oblíbené, je zachována. Součástí této funkcionality je i zvýraznění těchto vybraných produktů v katalogu a na detailu produktu. Také počet oblíbených produktů je součástí uživatelské lišty.

F6 – Zachování srovnavače produktů

Funkcionalita, která uživateli umožňuje označit produkty e-shopu a porovnat je mezi sebou, je zachována. Tyto vybrané produkty jsou zvýrazněny v katalogu a na detailu produktu.

F7 – Zachování sběru personalizovaných dat skrze GTM

Javascriptová knihovna Google Tag Manager má stále k dispozici aktuální personalizovaná data o uživateli. Součástí těchto dat je identifikátor košíku, informace o uživateli, informace o košíku atd.

F8 – Zachování možnosti zobrazení různého obsahu na jedné URL podle kontextu uživatele

Je umožněno kešovat HTML obsah, avšak stále je možné poskytovat rozdílné obsahy z jedné URL – dereference jednoho zdroje na více rozdílných HTML reprezentací.

F9 – Rozpoznání uživatelova kontextu podle geolokace

Podle geolokace, ze které uživatel přistupuje, je možné uživateli nabídnout doporučený kontext. Například uživatelům přistupujícím z Německa lze nabídnout možnost přepnutí měny na eura.

4.2.2 Nefunkční požadavky

NF1 – Doba obsloužení z mezipaměti v řádu desítek milisekund

Sdílená mezipaměť umožňuje obsloužit požadavky v řádech desítek milisekund v případě, že je obsah v mezipaměti přítomen. Očekávaná doba odpovědi je v průměru alespoň 30 ms.

NF2 – Monitoring výkonnosti sdílené mezipaměti

Sdílená mezipaměť poskytuje informace o počtu požadavků obsloužených/neobsloužených z mezipaměti. Je umožněno tyto statistiky monitorovat a vizuálně promítnout.

NF3 – Snížení průměrné doby čekání na odpověď požadavku

Průměrná doba čekání klienta přistupujícího na web se sníží.

4.3 Výběr konceptu řešení

Při vývoji kešovacího mechanismu existuje více přístupů, které lze zvážit. Jelikož WPJshop je dynamicky se vyvíjející platforma, která se průběžně adaptuje na současné trendy v e-commerce, je klíčové zvolit takové řešení, které efektivně pokryje všechny současné potřeby a zároveň poskytne dostatečnou rozšiřitelnost a flexibilitu pro nadcházející výzvy.

Funkční požadavky kladou důraz na redukování závislosti každého požadavku na obsluze ze zdrojového webového serveru. Pro vysvětlení nového principu komunikace mezi klientem a serverem, je třeba definovat rozdělení dat na dvě skupiny:

▶ **Definice 4.2.** ***Dlouhodobá data*** – *Statická data, nebo taková, která mění svojí podobu zřídka. Příkladem takových dat je např. detail produktu, který obsahuje cenu, popis a parametry produktu. Tyto informace se mohou změnit při úpravě produktu administrátorem stránky, avšak to nastane pouze jednou za čas (denně/týdně/měsíčně).*

▶ **Definice 4.3.** ***Krátkodobá data*** – *Vysoce dynamická data, která frekventovaně mění svojí podobu. Typickým příkladem je obsah košíku, nebo jiná data, která buď závisí na uživatelských akcích, nebo se liší v každé odpovědi.*

Obě tyto skupiny dat se dále dají rozdělit na **personalizovaná** a **společná data** – mohou existovat dlouhodobá data, která jsou určena více uživatelům, nebo naopak jsou určena pouze pro konkrétního uživatele. Stejně tak je tomu i krátkodobých dat.

Současné řešení vrací všechny tyto typy dat vždy v jedné HTML odpovědi. S využitím ESI tagů odděluje tyto jednotlivé části obsahu od sebe, aby bylo umožněno ukládání dlouhodobých společných částí ve sdílené mezipaměti. Tento způsob však neredukuje závislost každého požadavku na zdrojovém serveru, z toho důvodu v tomto konceptu **není uvažováno využití ESI tagů** pro tento účel.

K docílení snížení závislosti všech požadavků na zdrojovém serveru je možné rozdělit i typy odpovědí ze serveru na dvě skupiny: **první skupina** jsou odpovědi, které obsahují dlouhodobá data, **druhá skupina** jsou odpovědi, které obsahují krátkodobá data. Tímto způsobem lze umožnit efektivní kešování první skupiny odpovědí ze serveru, vzhledem k tomu, že jejich data je možné využít opakovaně.

Rozdělení odpovědí ze serveru na tyto dvě skupiny umožňuje využití soukromé a sdílené mezipaměti. Webový prohlížeč, který v tomto případě reprezentuje soukromou mezipaměť, může kešovat celé HTTP odpovědi, které obsahují libovolná dlouhodobá data. Sdílená mezipaměť může ukládat celé HTTP odpovědi s dlouhodobými společnými daty. Výhoda soukromé mezipaměti oproti sdílené je, že umožňuje ukládat i personalizovaná data, je však žádoucí, pro maximalizaci efektivity, aby co nejvíce odpovědím bylo umožněno využít i sdílenou mezipaměť.

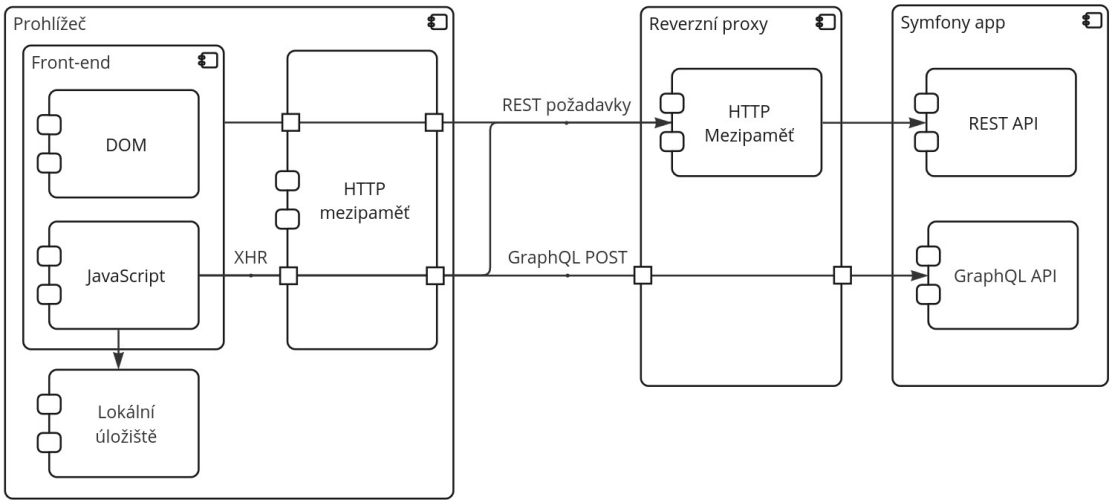
Mezi první skupinu odpovědí jsou zařazeny obzvláště odpovědi s HTML obsahem. Díky tomu je umožněna téměř okamžitá obsluha úvodního HTML požadavku, pokud je obsloužen z mezipaměti.

V důsledku redukce závislosti požadavku na zdrojovém aplikačním serveru je také vhodné přenést sdílenou mezipaměť mimo aplikační logiku WPJshopu. Tento účel může naplnit téměř libovolný reverzní kešovací proxy server, který zároveň, v případě že implementuje standard RFC 9111, zaručuje možnost revalidace a korektní obsluhu zastaralého obsahu. Tímto způsobem lze naplnit funkční požadavek F2 a F3.

Získávání krátkodobých a personalizovaných dat, které již v tomto konceptu nejsou součástí kešovatelných HTML odpovědí, je třeba přesunout do XHR požadavků. Ty může inicializovat JavaScript na straně klienta, který je součástí front-end aplikace. Lze využít GraphQL API, které WPJshop poskytuje, tím lze omezit velikost přenášených dat a vyrenderovat obsah po vzoru CSR.

Digram komponent 4.1 znázorňuje architekturu celého konceptu. Uživatel prostřednictvím prohlížeče přistupuje ke zdrojům webového serveru. HTML odpovědi, obsahující dlouhodobá

společná data, jsou kešovány na reverzním proxy serveru. HTTP mezipaměť prohlížeče navíc může kešovat i HTML odpovědi s dlouhodobými personalizovanými daty. Získání krátkodobých personalizovaných dat (např. obsah košíku) probíhá prostřednictvím XHR požadavků na GraphQL API. Tyto požadavky inicializuje JavaScriptová front-endová aplikace po načtění úvodního HTML obsahu. JS aplikace má přístup k lokálnímu úložišti prohlížeče, který může také ukládat některé data uživatele. Díky tomu není potřeba vysílat XHR požadavky po každém načtení stránky (naplnění požadavku F4). Data získaná XHR požadavky jsou vkládána do DOM stránky principem CSR (případně je možné využít i SSR).



Obrázek 4.1 Koncept řešení – diagram komponent

Souhrnem se tento koncept zaměřuje primárně na umožnění kešování celých HTML odpovědí včetně HTTP hlaviček na reverzním proxy serveru či v prohlížeči. Jeho výhodou je jeho flexibilita ve výběru kešovacího middlewaru, kterým může být téměř libovolný reverzní proxy server implementující HTTP standard pro kešování. I přes to, že existují komplexnější middlewary poskytující mnoho rozšiřujících funkcí (např. Varnish), které by umožnily širší možnosti v hledání řešení, je důležité zaměřit se na jádro problému a pokusit se ho vyřešit pomocí standardizovaných metod.

Zároveň tento koncept nevylučuje pozdější využití libovolných rozšíření, jako jsou právě ESI tagy, přejití na Varnish, výraznější zapojení in-memory databází na aplikační vrstvě atp.

4.4 Návrh řešení

Koncept sám o sobě není dostatečný pro přechod k implementaci. Nejprve je třeba zpracovat návrh, který detailněji specifikuje podobu konečného řešení. Je důležité určit, jakým způsobem budou splněny všechny funkční požadavky, což umožní posoudit, zda je realizace všech aspektů konceptu možná.

Návrh se zaměřuje na aktuální principy v systému, které komplikují splnění jednotlivých požadavků. K některým problémům existuje více možných řešení, která jsou diskutována, ale vždy je vybráno to řešení, které představuje nejlepší kompromis mezi dodržením standardů (jako jsou REST, RFC apod.), flexibilitou a jednoduchostí implementace.

Zmíněny jsou všechny potřebné úpravy stávajícího systému a případně nové části, které je třeba implementovat pro naplnění konceptu, například front-end aplikace. Přiložené sekvenční diagramy znázorňují nový princip komunikace mezi prohlížečem, reverzním proxy serverem a zdrojovým serverem.

4.4.1 Relace uživatele, cookies

Jak už bylo popsáno v kapitole 2.1.1, cookies umožňují v prohlížeči ukládat data, která mohou sloužit pro udržení stavu napříč více požadavky jednoho uživatele. To může být např. klíč k uživatelově relaci uložené na serveru. Problém, který je s použitím cookies spojený, je nemožnost ukládat do sdílené mezipaměti odpovědi, které obsahují hlavičku `Set-Cookie`. Hodnota cookie je také personalizovaný obsah, proto nadmíra jejich využití může značně limitovat schopnost kešování.

WPJshop nastavuje uživatelům cookie `cartID`, která umožňuje spojit požadavek s aktuálním obsahem košíku uživatele, který je uložen v databázi. Tato cookie je nastavena i v případě, že uživatel přistupuje na e-shop poprvé; tedy hned v HTML odpovědi na úvodní požadavek. To zabraňuje možnosti takovouto odpověď uložit celou do sdílené mezipaměti. Avšak, pokud uživatel přistupuje na server poprvé, jeho košík je vždy prázdný; dá se tedy uvažovat o tom, aby se `cartID` generovalo a přiřazovalo pouze v případech, kdy ho uživatel skutečně potřebuje, tedy pouze pokud košík uživatele skutečně obsahuje nějakou položku.

Dalším případem, kdy je vrácena hlavička `Set-Cookie` v odpovědi, je při vytvoření stavové relace. V takovém případě je nastavena cookie `PHPSESSID`. WPJshop využívá relaci např. pro zapamatování poslední navštívené kategorie produktů nebo aktivní měny/země či jazyku. Při první návštěvě kategorie produktů může tedy být také vrácena `Set-Cookie` hlavička v HTML odpovědi.

Jak plyne z konceptu řešení, je potřeba vracet hlavičku `Set-Cookie` pouze v nekešovatelných odpovědích – nemůže být obsažena v kešovatelných HTML odpovědích na úvodní požadavky, ani v jiných kešovatelných odpovědích s dlouhodobými daty. Možní kandidáti na vracení této hlavičky jsou nekešované odpovědi na XHR požadavky, nebo přesměrování po různých akcích (HTTP status kód 300–399). Tímto oddělením lze zaručit, aby vždy konkrétním uživatelům byla doručena jejich potřebná cookie a zároveň bylo umožněno efektivní kešování všech HTML odpovědí a jiných dlouhodobých dat.

Pokud by kešovatelný zdroj v některých případech vracel tuto cookie hlavičku, může se stát následující: mezipaměť uloží odpověď, která zrovna `Set-Cookie` neobsahuje a všechny následující požadavky na tento zdroj jsou obslouženy z této mezipaměti. Tedy i v případě, kdy by na požadavek bylo nutné vrátit v odpovědi `Set-Cookie` hlavičku, požadavek by byl obsloužen z mezipaměti a uživatel by cookie neobdržel.

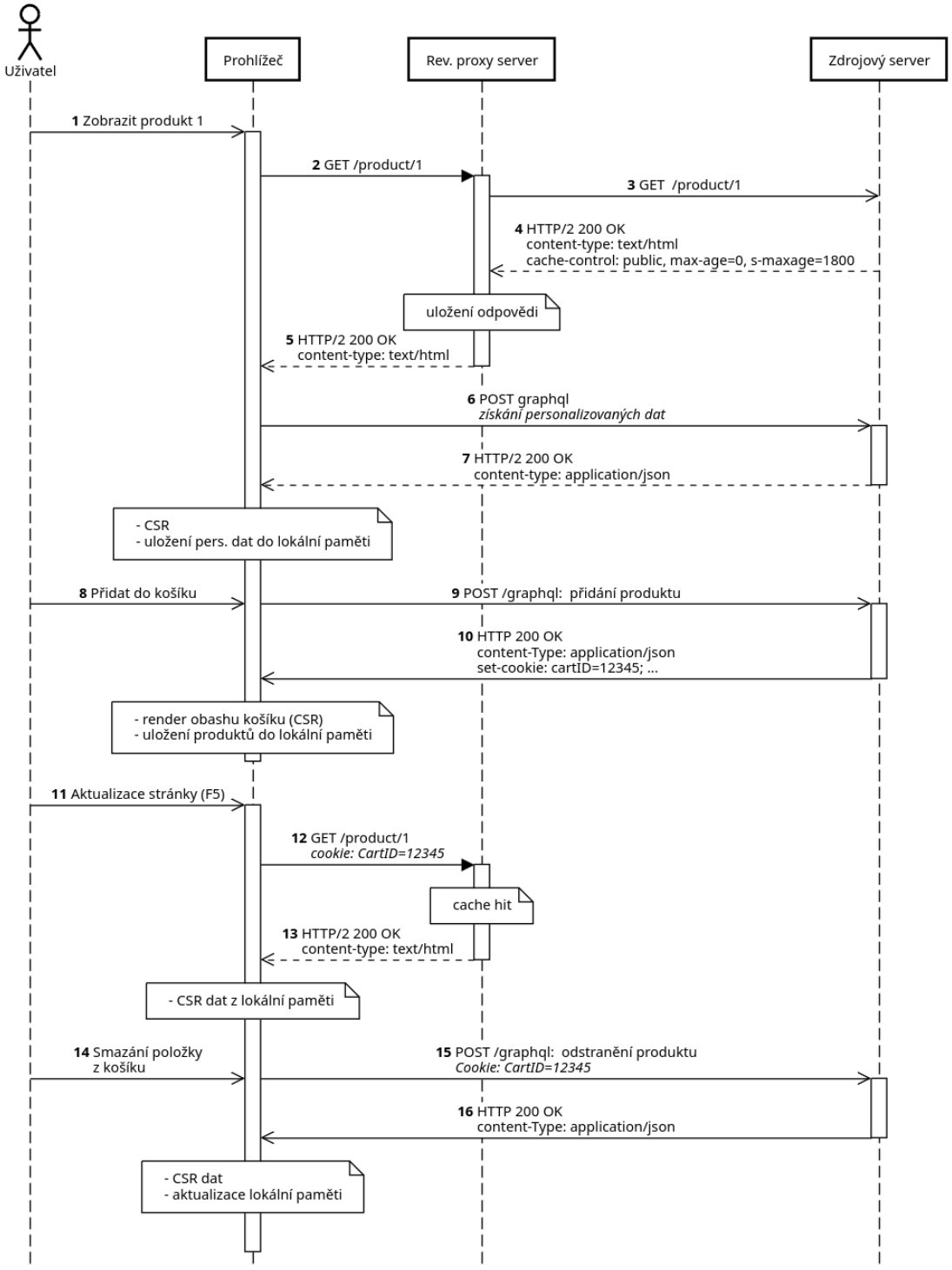
4.4.2 Košík

Obsah uživatelského košíku je ukládán v relační databázi a v současném řešení je získáván jako součást každé HTML odpovědi na úvodní požadavek. Důsledkem toho je vyžadováno spojení se zdrojovým serverem při každém požadavku a nelze uložit celou odpověď v HTTP mezipaměti.

Jak již je naznačeno v konceptu řešení, pro umožnění kešování prvotního HTML obsahu, je potřeba data týkající se košíku přenést mimo odpověď na úvodní požadavek. Jednou z možností je jeho získání pomocí XHR požadavku poté, co prohlížeč odpověď na úvodní požadavek zpracuje. Tento způsob vyžaduje využití JavaScriptu, který tento XHR požadavek provede, zpracuje a vloží do DOM stránky (principem CSR); to implikuje implementaci front-endové aplikace. Tato aplikace běžící na straně klienta může využít např. veřejné GraphQL API, které WPJshop poskytuje, a tak komunikovat se zdrojovým serverem pouze v případě, kdy potřebuje obsah košíku získat, nebo ho aktualizovat.

Obsah košíku se mění pouze v závislosti na uživatelských akcích, je proto možnost tato data i ukládat do lokální paměti prohlížeče. Díky tomu není potřeba jeho obsah stahovat při každém načtení stránky, ale pouze v případech, kdy není v paměti prohlížeče přítomen, čímž je naplněn funkční požadavek F4.

Příklad nového způsobu získávání košíku a jeho aktualizace je na znázorněn na sekvenčním diagramu 4.2. Znázorňuje uživatele, který provádí akce v prohlížeči a komunikaci mezi prohlížečem, reverzním proxy serverem (sdílená mezipaměť) a zdrojovým serverem v důsledku uživatelových akcí. Uživatel na e-shop přistupuje poprvé, v úložišti prohlížeče tedy nic není; další předpoklad v této ukázce je prázdná sdílená mezipaměť. Uživatel skrze prohlížeč vyslal požadavek na zobrazení detailu produktu (**1**), sdílená mezipaměť odpověď neobsahuje, požadavek je tedy přeposlán na zdrojový server. HTML odpověď ze serveru je uložena v mezipaměti a vrácena uživateli (**5**). Prohlížeč zpracuje odpověď a inicializuje JS aplikaci, která okamžitě vyšle XHR požadavek na GraphQL endpoint pro personalizovaná data (může se např. jednat o data závislé na geolokaci). Získaná data jsou vložena do stránky a uložena v paměti prohlížeče. Další akcí uživatele je kliknutí na tlačítko „přidat do košíku“(**8**), JS aplikace odešle XHR požadavek na přidání produktu do košíku. Odpověď ze serveru obsahuje `Set-Cookie` hlavičku a návratovou hodnotu odpovědi. Po přidání produktu do košíku je obsah košíku vykreslen do stránky (CSR) a obsah košíku je uložen do lokální paměti. Další uživatelskou akcí je aktualizace stránky(**11**); prohlížeč vyšle opět úvodní HTML požadavek, ten je v tomto případě už obsloužen z mezipaměti, protože odpověď již byla uložena při předchozím požadavku na zobrazení stránky. Po inicializaci JS aplikace jsou všechny personalizovaná data vzata z paměti prohlížeče a vložena do stránky. Aktualizace stránky tedy nevyžadovala navázání spojení se zdrojovým serverem. Poslední akcí uživatele je smazání produktu z košíku, ta proběhne analogicky jako přidání.



Obrázek 4.2 Princip nastavení cookies a manipulace s košíkem – sekvenční diagram

4.4.3 Různý HTML obsah na jedné URL

Jeden zdroj WPJshopu je možné v některých případech dereferencovat na několik možných HTML reprezentací. To slouží například k podpoře více měn, cenových hladin registrovaných uživatelů, nebo i rozdílné nabídky produktů pro určité skupiny uživatelů. Tato vlastnost komplikuje využití sdílené mezipaměti, která jako kešovací klíč běžně používá pouze URL a HTTP metodu požadavku. Pro správnou funkčnost sdílené mezipaměti je potřeba do kešovacího klíče zahrnout uživatelův aktivní kontext – aby aktivní měna, skupina atp. uživatele byla součástí kešovacího klíče a mezipaměť tak pro jednu URL dokázala vždy uživateli vrátit obsah odpovídající jeho kontextu.

Nejpřirozenější řešení je zahrnout potřebné informace o kontextu do všech URL. Příklad takové úpravy je transformace URL cesty `/products` na `/CZK/5/products`, kde `CZK` značí aktivní zemi uživatele a číslo `5` identifikátor aktivní cenové hladiny. Ačkoli se jedná o validní řešení, taková či podobná modifikace všech URL není vždy žádoucí vzhledem ke své výrazné viditelnosti nebo implementační náročnosti.

Alternativním řešením je kešovací klíč rozšířit. To je možné například udělat pomocí určité hodnoty cookie, která může v některých sdílených mezipamětích být nastavena k jeho rozšíření. V takovém případě URL mohou zachovat svojí podobu a hodnota `CZK/5` je obsažena ve speciálním klíči cookie, který je sdílená mezipaměť nakonfigurována využívat. Každý uživatel tedy bude mít vlastní cookie, která slouží jako identifikátor pro konkrétní obsah z množiny všech možných obsahů na jedné URL v mezipaměti.

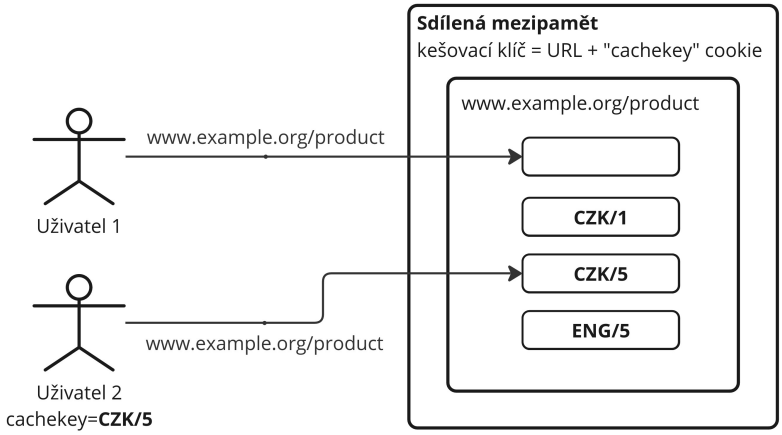
Nevýhody spojené s takovýmto využitím cookies jsou však následující. Reverzní proxy server musí umožňovat rozšíření kešovacího klíče pomocí konkrétní hodnoty cookie. Je tedy třeba vybrat takového poskytovatele, který toto podporuje. Zároveň prohlížeče neumožňují rozšíření kešovacího klíče HTTP mezipaměti pomocí konkrétní cookie. Pro soukromou mezipaměť lze pouze nastavit hlavičku `Cookie` do `Vary`, což rozšíří kešovací klíč o všechny hodnoty v cookies. Mezipaměť prohlížeče je tak méně efektivní, vzhledem k tomu, že umožňuje rozšířit kešovací klíč pouze o všechny cookies, nebo o žádnou.

Soukromá mezipaměť sice nepotřebuje uchovávat odpovědi kontextů pro všechny možné uživatele jako sdílená mezipaměť, ale stále může nastat situace, kdy se kontext uživatele změní, soukromá HTTP mezipaměť prohlížeče tuto změnu nerozliší a i nadále bude poskytovat obsah zastaralého kontextu.

Také je potřeba brát v potaz, že hlavička `Set-Cookie` nemůže být obsažena v kešovatelné odpovědi. Z tohoto důvodu je třeba mít definovaný výchozí kontext, pro který není třeba nastavit rozšíření kešovacího klíče do cookie. Klient při prvním přístupu na e-shop bude mít vždy výchozí kontext a pouze při změně kontextu na jiný než výchozí mu bude vrácena cookie pro rozšíření kešovacího klíče. Pokud by uživateli byla vrácena `Set-Cookie` na nastavení klíče hned při první návštěvě e-shopu, tato odpověď (prvotní HTML obsah) by nemohla být kešována (viz. 4.4.1).

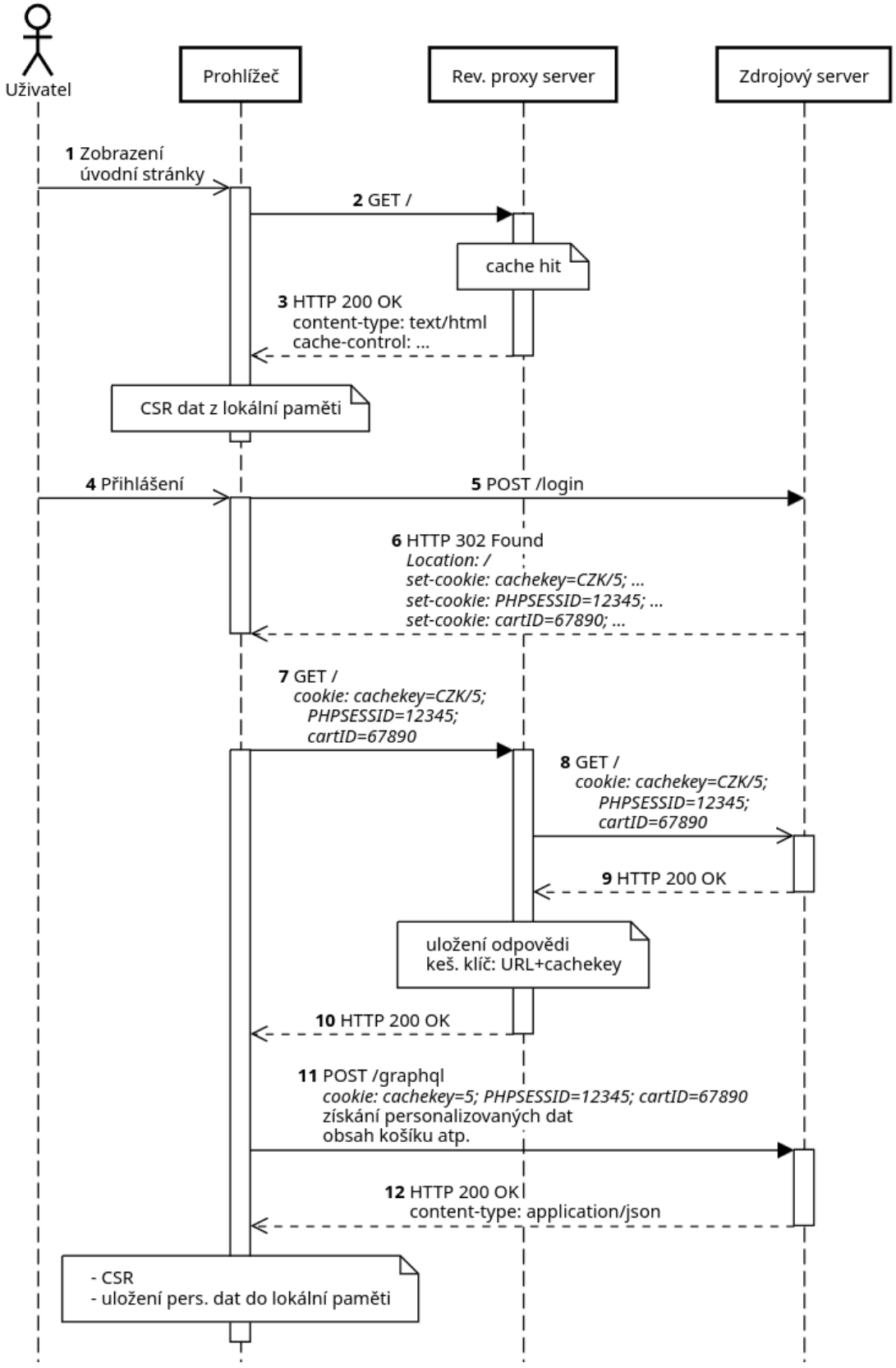
Oba zmiňované způsoby řešení naplňují funkční požadavek F8. Vybraný způsob, který bude implementován, je rozšíření kešovacího klíče pomocí cookie, vzhledem k jeho jednodušší implementaci. Z pohledu správné REST architektury je však vhodnější první způsob, který umožňuje plnohodnotné využití HTTP mezipaměti prohlížeče.

Přístup uživatelů k odpovědím ve sdílené mezipaměti, která využívá cookie hodnotu k rozšíření kešovacího klíče, je znázorněn na obrázku 4.3. Sdílená mezipaměť je nakonfigurována tak, aby kešovací klíč rozšiřovala pomocí hodnoty cookie „`cachekey`“, pro jednu URL je tak v mezipaměti několik uložených odpovědí. První uživatel žádnou cookie nemá a druhý uživatel má cookie `cachekey` s hodnotou `CZK/5`. Každému Uživateli je tak z mezipaměti poskytnut rozdílný obsah pro požadavek na stejnou URL.



Obrázek 4.3 Rozšíření kešovacího klíče sdílené mezipaměti pomocí cookie

Princip s rozšířením kešovacího klíče pomocí cookie je také znázorněn na sekvenčním diagramu 4.4. V tomto příkladu se jedná o uživatele, který již e-shop navštívil (tedy jeho prohlížeč obsahuje data v lokální paměti) a má nastavený výchozí kontext (nepotřebuje cookie na rozšíření mezipaměti). Uživatel se chce přihlásit do svého uživatelského účtu, přičemž jeho uživatelský účet má zvýhodněné ceny všech produktů (cenová hladina). První požadavek uživatele je zobrazení úvodní stránky (**1**), odpověď je vrácena ze sdílené mezipaměti a inicializovaný JS vyrenderuje data z lokální paměti prohlížeče. Uživatel se přihlásí přes dialogové okno na úvodní stránce, čímž se odešle POST požadavek (**5**). V odpovědi je přesměrování na úvodní stránku (**6**), včetně `Set-Cookie` hlaviček s identifikátorem relace, rozšiřujícím klíčem mezipaměti `cachekey` a identifikátorem košíku `cartID`. Uživatel má nyní nastavenou cookie na rozšíření kešovacího klíče, protože se změnil jeho kontext z výchozího na kontext s cenovou hladinou. Přesměrováním je opět vyslán požadavek na získání úvodní stránky (**7**), v tomto případě však požadavek není obsloužen ze sdílené mezipaměti, mezipaměť neobsahuje odpověď s tímto rozšiřujícím klíčem. Odpověď je tedy vrácena ze zdrojového serveru a uložena do mezipaměti (**10**). Zobrazená stránka nyní zobrazuje zlevněné produkty a po inicializaci JS je vyslán XHR požadavek na získání personalizovaných dat včetně obsahu košíku. Po zpracování odpovědi jsou data uložena do paměti prohlížeče a je zobrazen košík s produkty přiřazenými k uživatelskému účtu.



Obrázek 4.4 Různý HTML obsah na jedné URL – sekvenční diagram

4.4.4 Nastavení kontextu na základě geolokace

Některé e-shopy využívají funkcionalitu, která na základě uživatelovi geografické polohy dokáže rozpoznat správný kontext uživatele. Jeden z takových případů může být nastavení aktivní země e-shopu; muže se jednat například o země Německo a Rakousko, které sice sdílí jazyk, ale mají rozdílné sazby DPH. V takovém případě je potřeba pro uživatele zobrazovat rozdílné ceny produktů, vzhledem k zemi, ze které na e-shop přistupují.

WPJshop zjišťuje geolokaci uživatelů pomocí jejich IP adresy. Tímto principem lze spolehlivě získat skutečnou geolokaci uživatele, narozdíl od hodnoty hlavičky požadavku `Accept-Language`, která mnohdy neodpovídá jazyku země, ze které uživatel skutečně přistupuje. [43]

Problém, který přichází s určením geolokace podle IP adresy, je rozpoznání uživatelova kontextu v případě, že na webovou stránku přistupuje poprvé a obsah je mu vrácen ze sdílené mezipaměti. Výpočet kontextu uživatele se mnohdy liší na nastavení jednotlivých WPJshop e-shopů, z toho důvodu nelze spolehlivě zjistit kontext bez toho, aby požadavek doputoval na server.

Způsob, kterým lze nastavit uživatelův kontext správně a zároveň umožnit kešování HTML obsahu je přenesení zodpovědnosti rozpoznání kontextu uživatele podle geolokace z hlavního kešovatelného požadavku na XHR požadavek, který je vyvolán až JavaScriptem. Na základě odpovědi XHR požadavku je poté nabídnuto uživateli přesměrování na správný kontext, pokud se jeho současný výchozí kontext liší od kontextu, který mu je podle geolokace určen.

Nevýhodou tohoto řešení je nemožnost uživateli zobrazit okamžitě HTML obsah hlavního kešovatelného požadavku ve správném kontextu. Správný kontext je uživateli nabídnut až na základě odpovědi XHR požadavku. To vyžaduje, aby uživatel sám klikl na tlačítko, které ho přesměruje, ale i to lze považovat za validní řešení tohoto problému a naplňuje to požadavek F9.

4.4.5 Oblíbené produkty

WPJshop, stejně jako většina ostatních e-commerce platforem, nabízí funkcionalitu oblíbených produktů, díky které můžou uživatelé označit jednotlivé produkty jako oblíbené a poté se k nim jednoduše v budoucnu vrátit. Tato funkcionalita je v této platformě dostupná pouze pro registrované účty uživatelů, což umožňuje uživatelům přistupovat k oblíbeným produktům napříč více zařízeními a tedy je vyžadováno ukládat záznamy o oblíbených produktech do perzistentního úložiště – relační databáze s vazbou na uživatelský účet.

Problémem spjatým s touto funkcionalitou je, že oblíbené produkty jsou zvýrazněny na všech stránkách, které tyto produkty zobrazují (seznam produktů v kategorii a detail produktu). Sdílená mezipaměť tedy informaci o oblíbených produktech nemůže obsahovat, vzhledem k tomu, že se jedná o unikátní dynamickou informaci každého uživatele.

Jak už vyplývá z návrhu košíku, jehož data jsou téměř analogicky srovnatelná s principem oblíbených produktů, pro účel zachování této funkcionality a zároveň kešovatelnosti HTML obsahu je potřeba přenést logiku výměny informací o oblíbených produktech a jejich zvýraznění mimo HTML obsah. Toho lze opět dosáhnout pomocí XHR požadavků a front-endové JS aplikace, která tuto informaci asynchronně získá ze zdrojového serveru a principem CSR vloží do DOM HTML stránky. Zároveň lze informaci o oblíbených produktech ukládat do lokální mezipaměti a redukovat tím potřebu komunikace se zdrojovým serverem, pokud jsou informace o oblíbených produktech již v lokální paměti prohlížeče přítomny. Tímto způsobem lze naplnit funkční požadavek F5.

4.4.6 Srovnavač produktů

Zdánlivě podobná funkcionalita k oblíbeným produktům je porovnávání produktů mezi sebou. Ta slouží ke zvolení produktů, u kterých uživatel chce porovnat jejich parametry mezi sebou na jedné, k tomu určené stránce. Uživatel takovéto produkty označuje stejně jako u oblíbených produktů; tyto produkty jsou poté také specificky zvýrazněny na všech stránkách.

Účelem srovnavače však není uchovat vybrané produkty napříč více zařízeními, ale pouze porovnání jednotlivých produktů mezi sebou. Současné řešení uchovává porovnávané produkty v relaci uživatele – nejsou tedy perzistentně uložena. Vzhledem k účelu této funkcionality však není potřeba uchovávat tato data na straně serveru, ale pouze v lokálním úložišti prohlížeče. Přenesením veškeré této funkcionality do front-endové JS aplikace se redukuje potřeba komunikace se zdrojovým serverem, bez toho, aby se omezila stávající funkčnost a je tím naplněn funkční požadavek F6.

4.4.7 Personalizované GTM události

Personalizovaná GTM událost, kterou WPJshop do datové vrstvy vkládá, je událost typu `user`. JS objekt, který tuto událost reprezentuje, obsahuje identifikátor košíku `cartID`, klíč relace `PHPSESSID`, počet produktů v košíku uživatele, celkovou hodnotu těchto produktů a případný podobjekt s údaji přihlašeného uživatele. Veškerá logika vkládání do datové vrstvy včetně personalizovaných dat události, je obsažena v HTML odpovědi na úvodní požadavek a je do datové vrstvy vložena hned při zpracování tohoto HTML. Vzhledem k tomu, že úvodním HTML odpovědím bude umožněno být uloženo ve sdílené mezipaměti, je potřeba informace týkající se této události přesunout mimo sdílený obsah.

Možné řešení je `user` objekt získávat XHR požadavkem po načtení prvotního HTML, nebo také jako součást odpovědi na XHR požadavek pro přídání/odebrání produktů do košíku. Získaný objekt se uloží do lokálního úložiště prohlížeče, aby byl k dispozici vždy při načítání úvodní HTML stránky a nemusel být dotazován pokaždé. Tímto způsobem může být tato událost vložena do datové vrstvy hned z lokálního úložiště prohlížeče a zároveň je konzistentní se skutečnými informacemi o uživateli. Pouze v případě, kdy v úložišti prohlížeče tato informace není, je potřeba počkat na odpověď ze serveru s tímto objektem a vložit ji do datové vrstvy až později po obdržení odpovědi. Tímto způsobem lze naplnit funkční požadavek F7.

4.5 Souhrn návrhu

Hlavním prvkem výsledného návrhu je vyjmutí personalizovaných a krátkodobých dat z úvodních HTML odpovědí, což umožňuje jejich ukládání ve sdílené mezipaměti. Tento přístup zlepšuje rychlost načítání stránek a celkovou efektivitu serveru tím, že minimalizuje potřebu opakovaných dotazů na server pro obsah, který je vhodný pro kešování.

Front-endová JavaScriptová aplikace hraje zásadní roli v dynamickém načítání personalizovaného a krátkodobého obsahu. V případě, že tato data nejsou dostupná v lokálním úložišti prohlížeče, aplikace vyšle XHR požadavek na server skrze GraphQL API, aby získala aktuální data. Komunikace skrze GraphQL API probíhá i při přidání nebo odebrání produktů z košíku, či seznamu oblíbených.

V případě, že dojde ke změně uživatelského kontextu – například při přihlášení, změně země nebo jazyka – je nutné lokální úložiště přenačíst, aby odráželo nový kontext a zajistilo se, že uživatel získá relevantní a aktualizované informace.

Sdílená mezipaměť, ať už se jedná o reverzní proxy server nebo CDN, je nastavena tak, aby podporovala rozšíření kešovacího klíče specifickou hodnotou cookie. Tím je umožněno efektivní kešování různých obsahů poskytovaných z jednoho zdroje v různých uživatelských kontextech. Důležitou součástí realizace tohoto návrhu je také modifikace logiky výměny cookies tak, aby hlavičky `Set-Cookie` nebyly součástí kešovatelných odpovědí. Tím je zajištěno, že kešované odpovědi neobsahují žádné uživatelsky specifické údaje, což umožňuje jejich bezpečné ukládání ve sdílené mezipaměti.

Vzhledem k tomu, že soukromá HTTP mezipaměť prohlížeče nepodporuje snadné rozšíření kešovacího klíče pomocí cookies, není v tomto návrhu využívána.

Kapitola 5
Implementace

Kapitola popisuje realizaci navrženého řešení a diskutuje výběr využitých technologií. Zahrnuje úpravu webové aplikace, implementaci front-endové části a zavedení kešovacího middlewaru.

5.1 Postup implementace

Implementace je rozdělena na tři části, ty se dělí podle umístění těchto částí v architektuře celého systému – server, middleware a klient. První je úprava back-endu WPJshopu a následuje implementace front-endové části – tyto dvě části již samy o sobě musí tvořit funkční celek. Ve skutečnosti se jejich implementace částečně prolíná, avšak pro přehlednost jsou rozděleny do samostatných kapitol. Poslední krok je zavedení kešovacího middlewaru. Všechny implementované části odpovídají specifikovanému návrhu v předchozí kapitole. Výsledkem je funkční aplikace, která může být nasazena do produkčního prostředí a zároveň umožňuje debugování v lokálním vývojovém prostředí. Přibližný popis jednotlivých částí implementace je následující:

1. **Úprava back-endu**

   Vyjmutí personalizovaných a krátkodobých dat z HTML odpovědí, kterým bude umožněno kešování – primárně se jedná o úvodní stránku, kategorii produktů, detail produktů a blog. Rozšíření veřejného GraphQL API o „koncové body“, které budou volány z JS aplikace. Omezení využití cookies a relací, aby `Set-Cookie` hlavička nebyla vracena v úvodních HTML odpovědích. Generování rozšiřujícího cookie klíče pro sdílenou mezipaměť. Nastavení `Cache-Control` hlavičky a podpora pro revalidaci.

2. **Implementace front-endové JS aplikace**

   Vytvoření JS aplikace, která manipuluje s personalizovanými daty uživatele. Data získává prostřednictvím GraphQL API. Využívá lokální uložiště prohlížeče. Personalizovaná data jsou renderována na straně klienta – košík, oblíbené produkty a další potřebné prvky.

3. **Nastavení sdílené mezipaměti**

   Zavedení sdílené mezipaměti a její nastavení. Konfigurace využití rozšiřující cookie hodnoty k rozšíření kešovacího klíče. Implementace reverzní proxy pro testování funkčnosti při lokálním vývoji.

5.2 Úprava webové aplikace

Všechny úpravy byly provedeny v jádře systému WPJshop. Jelikož je jádro společné pro všechny e-shopy na této platformě, je nezbytné zajistit, aby nedošlo k omezení jejich funkčnosti. Z tohoto důvodu je většina nově implementovaného kódu podmíněna dvěma novými moduly – `JS_SHOP` a `PROXY_CACHE`. Tyto moduly umožňují, aby bylo možné nové funkce aktivovat pouze na vybraných e-shopech. To umožňuje nejprve otestovat implementaci na jednom e-shopu a následně ji rozšířit na další.

Modul `JS_SHOP` podmiňuje logiku vkládání nově vzniklé JS aplikace do vráceného obsahu. Modul `PROXY_CACHE` řídí veškerou serverovou logiku týkající se kešování, což zahrnuje například vkládání kešovacích hlaviček do odpovědí, generování rozšiřujícího cookie klíče atp. Existence dvou modulů zajišťuje možnost postupného nasazení – nejprve může být aktivována JS aplikace a po jejím odladění lze spustit kešování ve sdílené mezipaměti.

5.2.1 Redukce využití cookies a relace

Jak již bylo zmíněno v návrhu 4.4.1, pro umožnění kešování obsahu nějakého zdroje ve sdílené mezipaměti je potřeba zaručit, aby daný zdroj nevracel v odpovědi hlavičku `Set-Cookie`. Cílem této práce je primárně umožnit kešování HTML odpovědí na úvodní požadavky; dále je však možné kešování umožnit i pro ostatní odpovědi s dlouhodobými daty.

Pro tento účel je potřeba odstranit nebo upravit veškerou logiku webového aplikačního serveru, která tuto hlavičku do potencionálně kešovatelných odpovědí nastavuje.

5.2.1.1 Vytváření cartID cookie

Tato cookie je nastavena vždy při prvním úvodním požadavku na zdrojový server. Slouží jako identifikátor uživatelových položek v košíku. Vytvořena je pokaždé, pokud již v cookie neexistuje; její hodnota je při vytvoření vzata z aktuálního uživatelova klíče relace. Pro získání cartID hodnoty je napříč celým kódem volána statická metoda `Cart::getCartID()`. Její volání je potřeba v případě, kdy je potřeba z databáze získat uživatelovy položky z databáze, nebo je potřeba z různých důvodů identifikovat uživatele (měřící prvky GTM).

Kód této metody je vidět na obr. 5.1. Pokud cookie `cartID` neexistuje, je vytvořena nová.[1] Pokud již tato cookie existuje, `cartID` je vzato z ní a není potřeba cookie znovu vytvářet.

Jak je popsáno v kapitole 4.4.1, je možné upravit tuto logiku tak, aby cookie `cartID` byla vytvořena až v případě, kdy uživatel do košíku vloží nějakou položku. Díky tomu může být hlavička `Set-Cookie` vrácena až v odpovědi na nekešovaný XHR požadavek. Tato úprava je vidět na obrázku 5.2, kde metoda `getCartID`, v případě že cookie `cartID` stále neexistuje, vrací pouze konstantu. Tato konstanta značí absenci tohoto unikátního identifikátoru. Jediným případem, kdy je dovoleno cookie vytvořit je v okamžiku, kdy parametr funkce `$allowEmpty` má hodnotu `false`. S touto hodnotou parametru je funkce volána pouze při vkládání produktu do košíku. Tím je zajištěno, že hlavička `Set-Cookie` se vrátí pouze v odpovědi na požadavek pro vložení produktu do košíku.

[1] Pokud je vytvořena cookie na straně serveru, aplikační server při generování HTTP odpovědi do ní zároveň vloží HTTP hlavičku `Set-Cookie`, aby daná cookie byla vrácena klientovi.

```
class Cart
{
  ...
  public static function getCartID()
  {
    $cookie = getVal('cartID', $_COOKIE, '');
    if ($cookie) {
      return $cookie;
    }
    $userKey = session_id();
    SetCookies('cartID', $userKey, 86400 * 365);

    return $userKey;
  }
  ...
}
```

Výpis kódu 5.1 Metoda `Cart::getCartID`

```
class Cart
{
  public static $NO_CART_ID = 'no-cartID';
  ...
  public static function getCartID($allowEmpty = true)
  {
    $cookie = getVal('cartID', $_COOKIE, '');
    if ($cookie) {
      return $cookie;
    }
    if ($allowEmpty) {
      return self::$NO_CART_ID;
    }

    $userKey = session_id();
    SetCookies('cartID', $userKey, 86400 * 365);

    return $userKey;
  }
  ...
}
```

Výpis kódu 5.2 Upravená metoda `Cart::getCartID`

5.2.1.2 Proměnná relace – PurchaseState

Tzv. „PurchaseState“ je datová struktura, která ve WPJshopu slouží k reprezentaci veškerých informací souvisejících se stavem nákupu uživatele. Obsahuje informace o uživatelových produktech v košíku, zvolené dopravě/platbě, celkové ceně nákupu, aktivovaných slevových kupónech atp. Tato struktura je serializována a ukládána ve stavové relaci pro každého uživatele. Primárně

slouží k uchování stavu nákupu uživatele, ale částečně je využita i jako kešovací vrstva, která snižuje počet potřebných dotazů do relační databáze.

Problémem s tím spojeným je opět nutnost vytváření této struktury pro každého uživatele a to i v případě, kdy uživatel na e-shop přistoupil poprvé. Z tohoto důvodu je uživateli vrácena cookie `PHPSESSID`, která slouží k opětovnému přístupu k relaci při následujících požadavcích.

Řešení tohoto problému se částečně podobá principu, kterým je vyřešen problém s `cartID` cookie. Novým uživatelům, kteří stále nemají přiřazené `cartID`, není potřeba generovat pokaždé nový PurchaseState, a to z toho důvodu, že všichni tito uživatelé mají tuto datovou strukturu stejnou. Z tohoto důvodu lze jednou za čas vygenerovat prázdný PurchaseState, uložit ho v in-memory databázi a z ní ho poskytovat všem uživatelům bez `cartID`.

Výpis kódu 5.3 znázorňuje provedenou úpravu. Metoda `getPurchaseState()` je volána napříč celou aplikací k získání tohoto objektu, stačí tedy upravit pouze tuto část kódu. Pokud `cartID` se rovná konstantě `$NO_CART_ID`, objekt je vyžádán z in-memory databáze, v opačném případě je získán z relace uživatele. Pokud vytvořená proměnná `$purchaseState` je prázdná, což značí, že mezipaměť nebo relace požadovaný objekt neobsahuje, je vygenerován nový PurchaseState. Nově vygenerovaný objekt je vždy uložen do in-memory databáze nebo do relace podle toho, zda má uživatel vygenerované `cartId`.

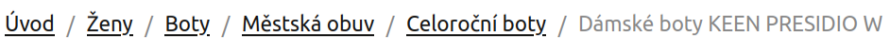
```
public function getPurchaseState(): PurchaseState
{
  if (Cart::getCartID() === Cart::$NO_CART_ID) {
    // Získání PurchaseState objektu z in-memory databáze -- sdílená mezipaměť
    $purchaseState = getCache('cart.empty_purchase_state');
  } else { // Získání PurchaseState objektu z relace uživatele
    $purchaseState = unserialize($this->session->get('cart.purchase_state'));
  }

  if (!$purchaseState) { // Generování nového PurchaseState
    $purchaseState = $this->generatePurchaseState();
    $this->persistPurchaseState($purchaseState);
  }

  return $purchaseState;
}
// Metoda na uložení PurchaseState objektu do příslušného uložiště
private function persistPurchaseState($purchaseState)
{
  if (Cart::getCartID() == Cart::$NO_CART_ID) { // in-memory-sdílená mezipaměť
    setCache('cart.empty_purchase_state', $purchaseState, 2 * 60);
  } else { // relace uživatele
    $this->session->set('cart.purchase_state', serialize($purchaseState));
  }
}
```

Výpis kódu 5.3 Úprava generování PurchaseState

Tímto způsobem je zajištěno, že PurchaseState objekt je vložen do relace uživatele vždy až při vložení produktu do košíku a případná hlavička `set-cookie` s `PHPSESSID` je vrácena v nekešovatelné odpovědi.

5.2.1.3 Proměnná relace – poslední aktivní kategorie

WPjshop využívá drobečkovou navigaci ke znázornění aktuální polohy uživatele v hierarchii webové stránky – e-shopy mnohdy obsahují několik vnořených kategorií, které tvoří stromovou strukturu. Jeden produkt může být však zařazen do více kategorií najednou, z toho důvodu je v některých případech nemožné jednoznačně určit cestu v hierarchii ke konkrétnímu produktu, pokud není známa historie navštívených kategorií uživatele.



Obrázek 5.1 Drobečková navigace – hierarchická cesta k produktu

Pro zobrazení správné drobečkové navigace je v současném řešení využívána relace uživatele, kam je ukládán identifikátor poslední navštívené sekce. Díky tomu lze jednoznačně určit hierarchickou cestu ke každému produktu, která odpovídá uživatelově historii procházení kategorií. Drobečková navigace je vkládána do HTML obsahu na straně serveru a je vždy vrácena v úvodní odpovědi s HTML obsahem.

Navržený princip kešovacího mechanismu se současným principem vylučuje. V případě, kdy je úvodní HTML požadavek uživatele obsloužen ze sdílené mezipaměti, není možné zobrazit personalizovanou navigaci a zároveň není možné do relace na zdrojovém serveru uložit aktuálně navštívenou kategorii. V případě, že by požadavek došel až na server, využití relace může způsobit vrácení hlavičky `Set-Cookie`.

Zvoleným řešením je opuštění od aktuálního principu s ukládáním poslední navštívené sekce. Produktů, které jsou zařazeny ve více kategoriích najednou, není ve většině případů mnoho. Tímto způsobem není relace potřeba k uložení poslední navštívené kategorie, avšak za tu cenu, že drobečková navigace nebude reflektovat uživatelovu historii procházení.

Alternativním řešením, které by tuto funkcionalitu zachovalo, je přenesení pamatování poslední navštívené kategorie na stranu klienta do lokální paměti prohlížeče. Tímto způsobem může být relevantní drobečková navigace získána pomocí XHR požadavku a vyrenderována po vzoru CSR nebo SSR.

5.2.2 Generování cookie pro rozšíření kešovacího klíče

Pro umožnění dereference jednoho zdroje na více možných podob podle kontextu uživatele je potřeba rozšířit kešovací klíč HTTP mezipaměti. Díky tomu bude umožněno na jedné URL zobrazit rozdílný obsah a zároveň všechny tyto obsahy kešovat.

Jak již bylo popsáno v návrhu 4.4.3, toto lze umožnit pomocí proměnné v cookie, kterou reverzní proxy server bude nastaven využívat k rozšíření kešovacího klíče. Pro správné chování kešovacího mechanismu bude tato cookie předávána pouze mimo kešovatelné odpovědi, buď přes XHR nebo např. přesměrování. Rozšiřující klíč je krátký textový řetězec, který reprezentuje uživatelův kontext. Každý e-shop má předem určenou množinu těchto rozšiřujících klíčů a jeden z těchto klíčů je označen jako výchozí. Výchozí klíč jako jediný není nastavován do cookie, aby při první návštěvě e-shopu uživatelem nebyla okamžitě vracena hlavička `Set-Cookie`, která zabraňuje uložení odpovědi v mezipaměti.

Vzhledem k bezpečnosti může být namísto originální hodnoty rozšiřujícího klíče vrácen jeho hash, aby potencionální útočník nemohl předpovědět jiné hodnoty z množiny generovaných klíčů a přistoupit tak k ostatním, jemu neurčeným odpovědím ve sdílené mezipaměti.

Nastavování rozšiřujícího klíče může být realizováno pomocí Symfony EventSubscriberu. Ten umožňuje navěsit spuštění metod na události, které jsou v průběhu zpracování HTTP požadavku Symfony aplikací vyvolávány. Jednou ze Symfony událostí je `KernelEvents::RESPONSE`. Ta je vyvolána vždy po tom, co Symfony controller vrátí `Response` objekt; tedy těsně před tím než server odešle odpověď. Tím je umožněno modifikovat libovolnou odpověď včetně hlaviček. [44]

Na ukázce kódu 5.4 je vidět implementace Symfony služby, která využívá `EventSubscriber` k nastavování zmiňované cookie. Proměnná `$cacheContext` obsahuje referenci na Symfony službu sloužící ke generování řetězce reprezentující uživatelův kontext. Proměnná `$desiredKey` obsahuje rozšiřující klíč, který by měl být nastaven do cookie. `$activeKey` je současná hodnota rozšiřujícího klíče v cookie. `$defaultKey` je výchozí rozšiřující klíč (výchozí kontext). Pokud se požadovaný klíč shoduje s již aktuální hodnotou cookie, není vyžadována žádná akce. Pokud uživatel stále žádnou cookie nemá, je mu nastavena cookie s požadovanou hodnotou. Pokud se požadovaný klíč shoduje s výchozím klíčem, žádná cookie se nenastavuje, pouze je smazána případná cookie se starou hodnotou.

Celá tato logika slouží k nastavení správného rozšiřujícího klíče do cookie v případě jeho změny. Hlavička `Set-Cookie` je tedy vrácena pouze když se změní uživatelův kontext.

```php
class CacheKeyChangeListener implements EventSubscriberInterface
{
    private static $cacheKeyword = 'cachekey'; // Název cookie

    public static function getSubscribedEvents()
    {
        // Registrace event subscriberu pro každou odpověď ze serveru
        return [KernelEvents::RESPONSE => [['changeCacheKeyCookie', 200]]];
    }

    public function changeCacheKeyCookie(ResponseEvent $event): void
    {
        $response = $event->getResponse();
        $cacheContext = Contexts::get(CacheContext::class);

        // Získání požadaovaného správného klíče pro aktuální kontext uživatele
        $desiredKey = $this->hashKey($cacheContext->getKey());

        // Aktuálně nastavený klíč v cookie
        $activeKey = $event->getRequest()->cookies->get(self::$cacheKeyword);

        // Pokud cookie už obsahuje požadovaný klíč, nedělat nic
        if ($desiredKey == $activeKey) {
            return;
        }

        // Získání výchozího klíče
        $defaultKey = $this->hashKey($cacheContext->getDefaultKey());

        // Pokud se výchozí klíč rovná požadovanému klíči, nedělat nic
        // Případně je pouze smazána cookie s původní hodnotou
        if ($desiredKey == $defaultKey) {
            if ($activeKey) {
                $response->headers->clearCookie(self::$cacheKeyword);
            }
            return;
        }
        // Nastavení požadovaného klíče do cookie
        $response->headers->setCookie(
        new Cookie(self::$cacheKeyword, $desiredKey));
    }

}
```

Výpis kódu 5.4 Implementovaný EventSubscriber pro nastavování rozšiřujícího klíče mezipaměti

5.2.3 Oprava rozdílné dereference pro XHR požadavky

I přesto, že ve zvoleném návrhu je povolená dereference jednoho zdroje na více různých obsahů podle kontextu uživatele (díky rozšíření kešovacího klíče pomocí cookie hlavičky), je potřeba, aby z pohledu uživatele měl jeden zdroj pouze jednu podobu. Jinými slovy, aby pro každého uživatele

byl z konkrétní URL načten vždy ten stejný obsah.

Příkladem nesprávného chování je současná kategorie produktů WPJshopu. Pokud je kategorie `/muzi?page=2` (druhá stránky kategorie „Muži“) vyžádána úvodním požadavkem (např. kliknutím na hypertextový odkaz), je vrácen kompletní obsah včetně navigace, záhlaví a zápatí stránky. Pokud je ale stejný požadavek vyžádán přes JS (XHR), je vrácena pouze konkrétní část stránky s výpisem produktů, kterou JavaScript dynamicky vloží do DOM. Tímto způsobem nelze efektivně kešovat tuto stránku, protože tyto dva požadavky mají vždy stejný kešovací klíč, ale rozdílný obsah. Důvodem, proč se toto v současném řešení děje, je to, že zdrojový server rozhoduje v jakém formátu je vrácen podle hlavičky požadavku `X-Requested-With: XMLHttpRequest`, která je obsažena pouze v XHR požadavcích.

Tento problém je vyřešen pomocí přidání query parametru `xhr=1` do všech požadovaných URL na získání kategorie z JavaScriptu. Požadovaná URL z příkladu tedy nyní pro XHR požadavky vypadá `/muzi?xhr=1&page=2`. Díky tomu je kešovací klíč XHR požadavku v mezipaměti rozdílný od úvodních požadavků.

Alternativním řešením by bylo přidání hlavičky `X-Requested-With` do hlavičky `Vary` v odpovědích.

5.2.4 Nastavení Cache-Control hlavičky

Kešovatelným odpovědím ze serveru je potřeba nastavit příslušnou hlavičku `Cache-Control`, která umožňuje kešovat HTTP odpovědi v mezipaměti.

Uvažované zdroje WPJshopu, jejichž HTML odpovědi lze kešovat, jsou: **úvodní stránka, kategorie produktu, detail produktu, blog a ostatní obsahové stránky**. Všechny tyto zdroje implementují vlastní business logiku v jejich třídách `HomeView`, `CategoryView`, `ProductView` atp. Společnou mají svojí abstraktní nadtřídu `View`, která definuje jejich jednotné rozhraní a implementuje metodu `getResponse`; návratová hodnota této metody je `Response` objekt, který obsahuje kompletní HTML obsah.

Přímočarou úpravou (viz. kód 5.5) je vložení kódu na přidání kešovací hlavičky přímo do metody `getResponse` ve třídě `View.php`. Kešovací hlavička s direktivy se do odpovědi vloží v případě, kdy atribut třídy `$proxyCacheEnabled` obsahuje „truthy“ hodnotu. Tento atribut je implicitně nastaven na `false`; všechny podtřídy mají možnost hodnotu tohoto atributu předefinovat na `true`, tím se povolí kešování pro zdroje, které konkrétní podtřída View obsluhuje.

Direktivy kešovací hlavičky jsou `public, max-age=0, s-maxage=1800`. Uložené odpovědi jsou tak „čerstvé“ půl hodiny ve sdílené mezipaměti. V mezipaměti prohlížeče odpovědi nejsou čerstvé nikdy, nemůže se tedy stát že by požadavek byl obsloužen přímo prohlížečem. Každý požadavek je buď obsloužen sdílenou mezipamětí, nebo je přeposlán na zdrojový server.

```php
// View.php
abstract class View
{
   protected $proxyCacheEnabled = false; // Implicitně vypnuto
   ...
   public function getResponse()
   {
      ...
      $response = new Response($this->render());
      ...
      if ($this->proxyCacheEnabled && !getAdminUser()) {
         // pokud je povolený modul a zároveň není přihlášený administrátor
         $this->setProxyCacheHeaders($response, $etag ?? null);
      } else {
         // původní cache-control direktivy
         $response->headers->addCacheControlDirective('must-revalidate', true);
         $response->headers->addCacheControlDirective('no-store', true);
      }

      return $response;
   }

   private function setProxyCacheHeaders($response)
   {
      // Nutný příznak, aby Symfony automaticky nemodifikovalo hlavičky
      $response->headers->set(
        AbstractSessionListener::NO_AUTO_CACHE_CONTROL_HEADER, 'true');

      // cache-control: public, max-age=0, s-maxage=1800
      $response->setPublic();
      $response->setMaxAge(0);
      $response->setSharedMaxAge($this->proxyCacheEnabled['ttl'] ?? 30 * 60);
   }
    ...
}
```

Výpis kódu 5.5 Úprava View – kešovací hlavička

5.2.5 Revalidace

Podporu pro revalidaci lze jednoduše přidat rozšířením předchozí úpravy. Pokud je atribut `$proxyCacheEnable` asociativní pole a obsahuje hodnotu `revalidate`, je volána metoda `getEtag`. Tato metoda implicitně vrací prázdný řetězec, jejím předefinováním v jednotlivých podtřídách je umožněno implementovat vlastní logiku na generování etag řetězce pro každou podtřídu zvlášť. Pokud se vygenerovaný etag shoduje s hodnotou `If-None-Match` v hlavičce požadavku, je vrácen HTTP status 304 (Not Modified), jinak je pokračováno v generování odpovědi. Do hlaviček odpovědí je včetně `Cache-Control` vkládána i hlavička `etag` obsahující vygenerovaný řetězec. Tímto způsobem je umožněno jednoduše vypínat a zapínat možnost revalidace pro konkrétní podtřídy třídy View.

```php
abstract class View
{
   ...
   public function getResponse()
   {
      // Pokud je revalidace povolena pro konkrétní View
      if ($this->proxyCacheEnabled['revalidate'] ?? false) {
         $etag = $this->getEtag();

         // Kontrola, zda etag v požadavku se shoduje s aktuálním etagem
         if (in_array('"'.$etag.'"', $this->request->getETags())) {
            // Pokud se etag shoduje, je vrácen HTTP status 304
            $response = new Response(status: 304);
            $this->setProxyCacheHeaders($response, $etag);
            return $response;
         }
      }
      ...
      if ($this->proxyCacheEnabled && !getAdminUser()) {
         // Nastavení kešovacích hlaviček a etagu
         $this->setProxyCacheHeaders($response, $etag ?? null);
      }
      ...
    }

   private function setProxyCacheHeaders($response, $etag)
   {
      ...
      // Pokud je etag nastaven, vloží jeho hlavičku do odpovědi
      if ($etag) {
         $response->setEtag($etag);
      }
   }

   protected function getEtag(): string
   {
      // Implicitně prázdné, předefinováno v poděděných třídách
      return '';
   }
   ...
}
```

Výpis kódu 5.6 Úprava View – revalidace

5.2.6 GraphQL API

Pro umožnění výměny informací mezi front-end aplikací a webovým serverem je využito GraphQL API. Je potřeba implementovat všechny koncové body veřejného rozhraní, které umožní získat všechny personalizovaná a krátkodobá data, která bude klientská aplikace potřebovat podle vzniklého návrhu. Ke specifikaci celého současného rozhraní využívá WPJshop PHP knihovnu

GraphQLite[2]. Ta pomocí programátorem specifikovaných anotací tříd a metod dokáže vytvořit kompletní GraphQL schéma. Pro implementaci potřebných částí rozhraní stačí pouze rozšířit současné veřejné API o potřebné dotazy a objektové typy.

GraphQL dotazy se dělí na dvě skupiny – „queries“ a „mutations“. Zatímco queries slouží pouze ke čtení dat, mutations slouží k jejich modifikaci (analogicky jako je tomu u REST metod GET a POST). Kromě tohoto klíčového slova se také liší v jejich principu vyhodnocování na serveru – queries se mohou zpracovávat paralelně; mutations se musí zpracovávat sériově. [45]

```
class CartController
{
  #[Query]
  public function cart(ResolveInfo $metadata, bool $invalidateCache=false):Cart
  {
    return $this->cartUtil->getCart(
      $this->getFetchOptions($metadata), $invalidateCache);
  }
  ...
}
```

Výpis kódu 5.7 GraphQLite – query dotaz cart

```
#[Type]
class Cart
{
  #[Field]
  public function getItems(): array
  {
    $items = [];
    foreach ($this->purchaseState->getProducts() as $item) {
      $items[] = new CartItem($item);
    }
    return $items;
  }
  ...
}
```

Výpis kódu 5.8 GraphQLite – objektový typ Cart

Ukázka implementace query dotazu `cart` a jeho návratového typu `Cart` je vidět ve výpisech kódu 5.7 a 5.8. Metoda ve třídě `CartController` je anotována PHP atributem `#[Query]`, tím je metoda zaregistrována GraphQLite knihovnou a bude zpřístupněna v API. Do parametrů metody je vkládán objekt s metadaty o dotazu a parametr dotazu `$invalidateCart`. Metadata obsahují informaci o všech specifikovaných polích dotazu atp. Parametry jsou přeposlány do metody pomocné třídy `CartUtil`, která obsahuje veškerou business logiku k vytvoření instance třídy `Cart` se všemi potřebnými daty. Třída `Cart` je anotována PHP atributem `#[Type]` a její metoda `getItems()` atributem `#[Field]`. Tím je zaregistrován objektový typ s položkou `items`, který obsahuje jednotlivé produkty v košíku. Metoda `getItems()` získává jednotlivé položky

[2] `https://graphqlite.thecodingmachine.io`

košíku z `PurchaseState` objektu (viz. 5.2.1.2) a vytváří z nich instance třídy `CartItem`, která je také zaregistrována jako GraphQL typ.

Analogicky, jako je tomu v předchozí ukázce, jsou implementovány všechny dotazy a typy, které jsou potřeba pro front-end aplikaci.

Specifikace rozhraní

Tabulky podrobně popisují implementované rozhraní. Tabulka 5.1 znázorňuje všechny dotazy typu query a tabulka 5.2 znázorňuje všechny dotazy typu mutation. Následující tabulky 5.3 až 5.13 popisují jednotlivé objektové typy, které se v dotazech vyskytují. Objektové typy jsou ve všech tabulkách znázorněny tučným písmem.

Query dotaz **`cart`** slouží k získání košíku aktivního uživatele, jako parametr má boolean `invalidateCache`, který slouží jako příznak, zda návratový objekt `Cart` může být vygenerován z `PurchaseState` objektu, nebo má být `PurchaseState` invalidován a aktualizován z databáze. Query **`redirectLocation`** slouží pro získání informace (podle IP adresy uživatele), zda má být přesměrován na jiný kontext (např. měna). Query **`gtmData`** slouží pro získání personalizovaných GTM objektů. Query **`me`** vrací informace o přihlášeném uživateli. Query **`productsFavorites`** vrací pole uživatelových oblíbených produktů. Query **`alsoBought`** vrací textový řetězec obsahující požadovaný počet (parametr `limit`) doporučených produktů a jako jediný slouží pro SSR renderování HTML obsahu, který je vkládán do postranní lišty košíku.

Mutation dotaz **`cartUpdate`** slouží k aktualizaci produktů v košíku uživatele, vstupní parametr `items` je pole produktů s jejich množstvím. V odpovědi na tento dotaz je objekt `Cart`, který obsahuje kompletní informace o košíku včetně celkové ceny. Mutation **`productFavoritesUpdate`** slouží k přidání (nebo odebrání) produktu do oblíbených.

Tabulka 5.1 GraphQL public API query dotazy

GraphQL query dotazy

Název	Parametry	Návratový Typ	Popis
cart	invalidateCache :Boolean!	**Cart!**	košík (produkty, sum ceny)
redirectLocation		**RedirectLocation**	přesměrování na doporučený kontext
gtmData		**GTMData**	GTM objekty
me		**User**	přihlášený uživatel
productsFavorites		[**ProductFavorite**]!	oblíbené produkty uživatele
alsoBought	limit: Int	String!	doporučené produkty (HTML)
contexts		**Contexts!**	informace o aktivním kontextu (měna, jazyk)

Tabulka 5.2 GraphQL public API mutation dotazy

GraphQL mutation dotazy

Název	Parametry	Návratový Typ
cartUpdate	items: [**CartItemInput**!]!	**CartUpdateResult!**
productFavoritesUpdate	productId: Int! action: **ProductFavoriteEnum**!	**Result!**

Tabulka 5.3 GraphQL typ Cart

Cart

Název	Typ	Popis
url	String!	odkaz k přejití na detail košíku
totalPrice	**TotalPrice!**	objekt s celkovou cenu (s DPH, bez DPH)
items	[**CartItem!**]!	položky v košíku
freeDeliveryFrom	Float	hranice ceny, od které je doprava zdarma

Tabulka 5.4 GraphQL typ CartItem

CartItem

Název	Typ	Popis
id	Int!	identifikátor položky v košíku
productId	Int!	identifikátor produktu
variationID	Int	identifikátor varianty
name	String	název produktu
pieces	Float!	počet kusů v košíku
piecePrice	Price!	cena jednoho kusu
totalPrice	Price!	součet ceny všech kusů
priceOriginal	Price!	původní cena bez slevy
discount	Float!	sleva v procentech
product	**ProductPublic!**	objekt produktu
note	String	poznámka produktu
gtmActions	**GtmActions**	GTM objekty položky

Tabulka 5.5 GraphQL typ User

User

Název pole	Typ	Popis
id	Int!	ID uživatele
email	String!	e-mail
name	String!	jméno
surname	String!	příjmení
userName	String!	celé jméno

Tabulka 5.6 GraphQL typ RedirectLocation

RedirectLocation

Název pole	Typ	Popis
location	String	URL přesměrování
name	String	titulek URL
flag	String	vlajka k URL
actualName	String	titulek aktivní
actualFlag	String	vlajka aktivní

Tabulka 5.7 GraphQL typ ProductFavorite

ProductFavorite

Název pole	Typ	Popis
productID	Int!	identif. produktu
productTitle	String!	titulek produktu

Tabulka 5.8 GraphQL typ CartItemInput

CartItemInput

Název pole	Typ	Popis
cartId	Int	identif. košíku
productId	String!	identif. produktu
variationId	Int	identif. varianty
pieces	Float!	počet kusů
note	String	dodatečné data

Tabulka 5.9 GraphQL typ Result

Result

Název pole	Typ	Popis
code	Int!	kód odpovědi
isSuccess	Bool!	úspěšnost operace
message	String	zpráva o výsledku operace

Tabulka 5.10 GraphQL typ CartUpdateResult

CartUpdateResult

Název pole	Typ	Popis
result	**Result!**	status akce
cart	**Cart!**	košík

Tabulka 5.11 GraphQL typ Contexts

Contexts

Název pole	Typ	Popis
currency	**Currency!**	code name roundType rate ...
language	**Language!**	code name
country	**Country!**	code name

Tabulka 5.12 GraphQL typ GTMActions

GTMActions

Název pole	Typ	Popis
cartAdd	String!	gtm objekt – přidání prod.
cartRemove	String!	gtm objekt – odebrání

Tabulka 5.13 GraphQL typ GTMData

GTMData

Název pole	Typ	Popis
user	String!	GTM objekt – uživatelské info.

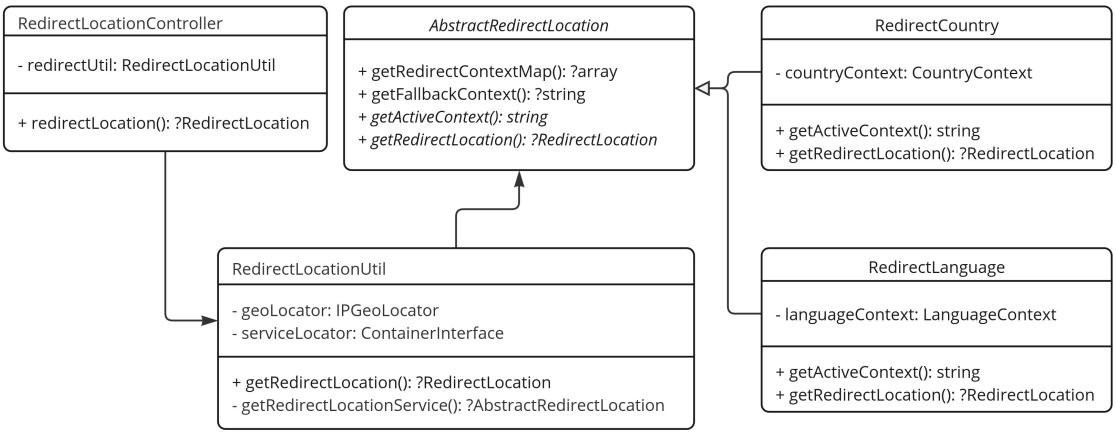
5.2.7 Nastavení kontextu dle geolokace

Řešení musí umožňovat, aby uživateli, který zobrazí konkrétní e-shop v jiném jazyce než mu je určeno, bylo nabídnuto vyskakovací okno s odkazem na změnění jazyku na správný. Správný jazyk je určen z IP adresy uživatele. Tento princip je kromě jazyku aplikovatelný i na libovolný jiný kontext, například aktivní země e-shopu nebo měna.

Dle návrhu 4.4.4 je potřeba informaci o tom, zda má být uživatel přesměrován, získat prostřednictvím XHR požadavku. K tomu účelu byl vytvořen GraphQL endpoint `redirectLocation`, který vrací stejnojmenný objekt (viz. tabulka 5.6), obsahující informaci o tom, zda má front-end aplikace uživateli zobrazit vyskakovací okno s odkazem na přesměrování.

Struktura tříd webového serveru, který implementuje tento GraphQL endpoint je vidět na diagramu 5.2. Třída `RedirectLocationController` obsahuje metodu `redirectLocaiton()`, která je zaregistrována jako GraphQL query dotaz. Uvnitř této metody je volána Symfony služba `RedirectLocationUtil`, která je společně s příslušnou implementací abstraktní třídy `AbstractRedirectLocation` zodpovědná za celou logiku generování odpovědi. Jednotlivé implementace abstraktní třídy (`RedirectCountry, RedirectLanguage`) slouží k poskytnutí potřebné logiky pro různé skupiny e-shopů, které mnohdy vyžadují přesměrování na základě jiného kontextu. Pro každý e-shop je v konfiguračním souboru specifikováno, jaká implementace abstraktní třídy má být využita a jaké je mapování mezi zeměmi získaných z IP a kontextem.

Na ukázce kódu 5.9 je vidět metoda `getRedirectLocation()`, ta na základě získané země z IP adresy, mapování těchto zemí na kontexty a aktivního kontextu determinuje, zda má být vygenerována odpověď s objektem na přesměrování. Pokud se aktivní kontext shoduje s doporučeným kontextem podle IP adresy, je vrácena prázdná odpověď, jinak je vrácena odpověd s `RedirectLocation` objektem.



Obrázek 5.2 Diagram tříd implementující logiku přesměrování podle geolokace

```php
class RedirectLocationUtil implements ServiceSubscriberInterface
{
...
  public function getRedirectLocation(): ?RedirectLocation
  {
    // Získání instance služby s logikou přesměrování
    $redirectService = $this->getRedirectLocationService();
    // Kód země podle IP adresy
    $geoIPCountry = $this->getGeoIpCountry();

    // Získání doporučeného kontextu podle země z IP adresy
    $recommendedContext = $redirectService
        ->getRedirectContextMap()[$geoIPCountry] ?? null;
    // V případě že pro konkrétní zemi není určen doporučený kontext,
    // použije se kontext pro výchozí zemi
    $recommendedContext ??= $redirectService->getFallbackContext();

    // Aktuálně aktivovaný kontext uživatele
    $actualContext = $redirectService->getActiveContext();
    // Pokud se doporučený kontext shoduje s aktuálním, nic se neprovede
    if ($recommendedContext === $actualContext) {
      return null;
    }

    // Vygenerování a vrácení přesměrování na doporučený kontext
    return $redirectService
        ->getRedirectLocation($actualContext, $recommendedContext);
  }
```

Výpis kódu 5.9 RedirectLocationUtil – logika metody getRedirectLocation

Příklad konfigurace logiky přesměrování v konfiguračním souboru na nějakém konkrétním e-shopu může vypadat jako ve výpisu kódu 5.10. Využita je implementace na přesměrování podle doporučeného jazyka. Mapování zemí je z CZ na češtinu, PL na polštinu a zbytek na angličtinu.

```
$cfg['Modules']['js_shop'] = [
    'redirectLocation' => [
        'type' => 'language',
        'map' => [
            'CZ' => 'cs',
            'PL' => 'pl',
        ],
        'fallback' => 'en',
    ],
];
```

Výpis kódu 5.10 Konfigurace logiky přesměrování v konfiguračním souboru e-shopu

5.3 Implementace front-endové části

Následující část, po úpravě logiky webového serveru (back-endu), je implementace front-endové JS aplikace, která slouží k renderování a získávání personalizovaných či krátkodobých dat až po tom, co je načten prvotní HTML obsah. Díky tomu je umožněno kešovat všechny odpovědi na úvodní požadavky a dosáhnout tím téměř okamžitého načítání stránek e-shopu. Zároveň je využito lokálního úložiště prohlížeče, aby požadovaná data nemusela být stahována ze serveru při každém požadavku.

5.3.1 Využité technologie

Aplikace je vyvinuta v **Reactu** s využitím **TypeScriptu**. React je populární a již využívaná technologie ve WPJshopu. To dělá z Reactu ideálního kandidáta, neboť vývojáři této společnosti jsou s ním již obeznámeni. TypeScript nabízí typovou kontrolu, ta zvyšuje udržitelnost tím, že odhaluje potencionální chyby již ve fázi vývoje a kompilace. Kromě toho je využit **Webpack** jako nástroj pro modularizaci a optimalizaci těchto zdrojů. Transformací zdrojových souborů tímto nástrojem je dosaženo menšího rozsahu výsledné aplikace a lepší kompatibility napříč všemi prohlížeči.

Využita je i JS knihovna `graphql-codegen`[3]. Ta umožňuje automatické generování TypeScript typů, které odpovídají typům v GraphQL API WPJshopu. Díky tomu je odlehčeno od nutnosti manuální definice těchto typů při vývoji, zároveň lze v budoucnu jednodušeji odhalit chyby při případné upravě tohoto API.

Dále je využita knihovna `i18next-scanner`[4] pro automatické generování souboru s přeložitelným textem. Aplikace obsahuje texty, které se mohou lišit v závislosti na zobrazovaném jazyku stránky. Všechny tyto texty jsou v aplikaci obaleny do funkce `_t`. Touto knihovnou jsou extrahovány všechny texty do samostatného souboru, který je načten vždy při inicializaci JS aplikace. Tímto způsobem lze jednoduše měnit jazyk textu aplikace pomocí jednoho poskytnutého souboru.

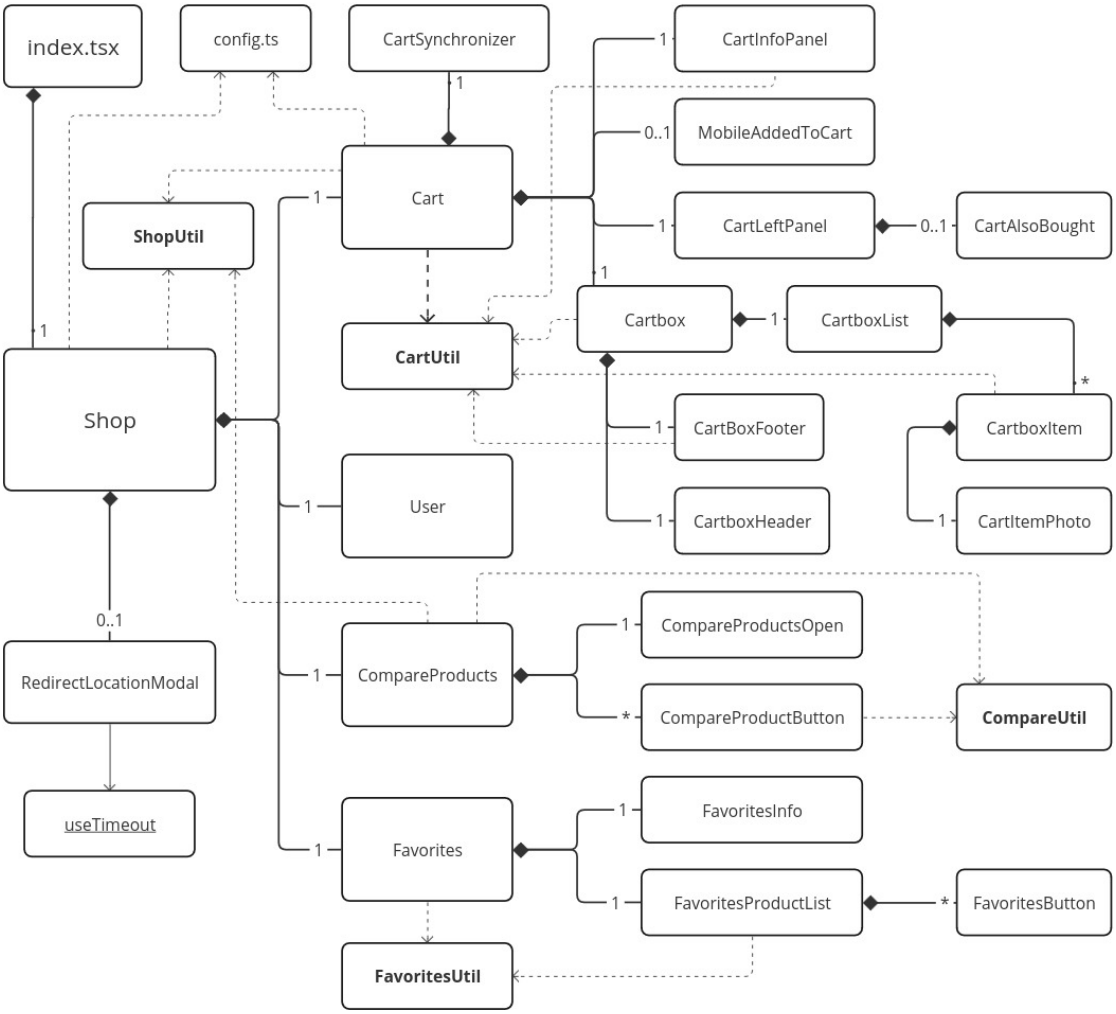
5.3.2 Struktura aplikace

React komponenty jsou základní stavební bloky aplikací vytvořených v knihovně React. Slouží k definování a správě uživatelského rozhraní. Každá komponenta v Reactu může být definována buď jako funkce nebo třída. Komponenty přijímají vstupy, známé jako „props“ (vlastnosti), které umožňují komponentám přijímat data z nadřazených komponent a použít je ve svém výstupu.

[3] `https://the-guild.dev/graphql/codegen`
[4] `https://github.com/i18next/i18next-scanner`

Návratový typ komponent je JSX, což je syntaxe podobná HTML, ale umožňuje vkládat JS kód přímo mezi tagy HTML. Komponenty jsou do sebe vnořené a tvoří stromovou strukturu. [46]

Struktura implementované React aplikace je vidět na obrázku 5.3. Znázorněny jsou stěžejní komponenty, jejichž název odpovídá názvu komponenty v programu a na diagramu jsou vyobrazeny normálním písmem. Pomocné „util“ soubory jsou zvýrazněny tučně a tzv. React hooks jsou podtrženy.



Obrázek 5.3 React aplikace – diagram kompozice komponent

index.tsx Tento soubor slouží jako vstupní bod celé aplikace. Metoda `ReactDOM.render` inicializuje HTML element s id `js-shop` jako kořenový element pro celou React aplikaci. Zároveň je inicialována Knihovna Sentry pro odchytávání chyb a knihovna Apollo pro dostupnost GraphQL klienta napříč celou aplikací.

```
const root = document.getElementById('js-shop');
if (root) {
   render(
      <Sentry.ErrorBoundary>
         <ApolloProvider client={apolloClient}>
            <Shop client={apolloClient}/>
         </ApolloProvider>
      </Sentry.ErrorBoundary>, root);
}
```

Výpis kódu 5.11 index.tsx

config.ts Konfigurační soubor aplikace. Specifikuje v konstantě `config` jednotlivé moduly, které jsou pro konkrétní e-shop povoleny. Slouží k tomu, aby se pro jednotlivé e-shopy mohla některá funkcionalita aplikace vypnout nebo zapnout. Například pokud některý e-shop nevyužívá porovnávání produktů, nebo nepoužívá uživatelské účty.

Shop.tsx Hlavní komponenta obsahuje klíčovou logiku a globální stav celé aplikace. Sdílení globálního stavu je zprostředkováno funkcí React Context, která umožňuje jednoduše přistoupit k tomuto stavu ze všech vnořených komponent bez nutnosti manuálně předávat vlastnosti (props) na každé úrovni. Globální stav zahrnuje položky v košíku, oblíbené produkty, informace o uživateli a další data potřebná napříč aplikací. Při aktualizaci stavu dochází k jeho serializaci a ukládání do SessionStorage prohlížeče. Při inicializaci této komponenty je ověřeno, zda jsou data dostupná v paměti prohlížeče; pokud ano, stav se načte odtud, v opačném případě jsou data vyžádána ze serveru prostřednictvím jednoho XHR požadavku na GraphQL API.

Na ukázce kódu 5.12 je zobrazena metoda `render`. Proměnná `config` určuje, jaké komponenty jsou vráceny. Tyto komponenty jsou poté obaleny v komponentě `DataContext.Provider`, což umožňuje zmíněné sdílení stavu do těchto a všech dalších vnořených komponent. Kromě globálního stavu je prostřednictvím komponenty `DataContext.Provider` sdílena i metoda `updateData`, která slouží k aktualizaci globálního stavu.

```
render() {
   return (
    <DataContext.Provider value={{
       ...this.state.data,
       client: this.props.client,
       updateShopData: this.updateData
    }}>
       {config.moduleCart && <Cart ref={this.cartComponent}/>}
       {config.moduleFavorites && this.state.data.me && <Favorites/>}
       {config.moduleUser && <User/>}
       {config.domainRedirectModal && this.state.data?.redirectLocation &&
        <RedirectLocationModal redirect={this.state.data.redirectLocation}/>}
       {config.moduleCompare && <CompareProducts/>}
    </DataContext.Provider>
   );
}
```

Výpis kódu 5.12 Shop.tsx – metoda render

Cart.tsx Rodičovská komponenta celého košíku. Větví se na uživatelský košík pro desktopové zobrazení (`Cartbox`), uživatelskou lištu s ikonou košíku (`CartInfoPanel`), mobilní vyjíždějící lištu po přidání do košíku (`MobileAddedToCart`) a komponentu sloužící pro synchronizaci napříč více okny prohlížeče (`CartSynchronizer`). Všechny tyto části jsou vyrenderovány příkazem `ReactDOM.createPortal` do elementu s id js-shop-cart.

`Cart` komponenta je zopovědná za komunikaci se serverem v případě, že je přidán/odebrán produkt košíku. Při změně obsahu košíku jsou poslány všechny položky košíku požadavkem na server (GraphQL mutation **`cartUpdate`**), ten aktualizuje stav databáze tak, aby odpovídal těmto přijatým počtům. V odpovědi na požadavek je celý stav košíku včetně přepočítané ceny. Tímto způsobem je synchronizována lokální paměť prohlížeče s databází.

Po přijetí odpovědi je vyslán dodatečný požadavek na získání doporučených produktů a personalizovaného GTM `user` objektu (dotaz **`alsoBought`** a **`gtmData`**). Tento požadavek záměrně není součástí prvního požadavku na aktualizaci košíku, aby synchronizace košíku proběhla co nejrychleji a nezdržovala uživatele. Doporučené produkty jsou vloženy do postranní lišty košíku. K tomu slouží komponenta `CartAlsoBought`. GTM `user` objekt je uložen do lokální paměti prohlížeče, aby při dalším načtení stránky mohl být vložen do GTM datalayer.

```
// Pole produktů v košíku (parametry dotazu)
const data = this.getDataForUpdate();

// Požadavek na GraphQL API serveru.
// Konstanta UPDATE_CART obsahuje dotaz (string)
this.context.client.mutate({ mutation: UPDATE_CART, variables: {items: data}})
 .then((result) => {
    // Výsledek obsahuje kompletní stav košíku
    //Aktualizace globálního stavu aplikace
    this.context.updateShopData(
     { cart: ShopUtils.prepareCartData(result.data.cartUpdate.cart) }
    );
    // Získání doporučených produktů a GTM dat
    this.updateAlsoBought(false);
    this.updateSessionRelatedGTM();
 })
 .catch(handleNetworkAbort);
```

Výpis kódu 5.13 Cart.tsx – GraphQL mutation dotaz na aktualizaci košíku

CartInfoPanel.tsx Ikona košíku v uživatelské liště stránky. Zobrazuje číslo celkového počtu produktů v košíku a složí k otevření postranního košíku (desktop), nebo k přejití na detail košíku (mobil). Vyrenderováno do elementu s id `js-shop-cart-info`.

MobileAddedToCart.tsx Komponenta je viditelná pouze při mobilním zobrazení. Když uživatel přidá produkt do košíku, namísto zobrazení postranního košíku s celým seznamem produktů se objeví spodní vyjíždějící lišta. Tato lišta nabízí možnost přejít přímo do košíku, nebo pokračovat v nákupu.

CartSynchronizer.tsx Komponenta zajišťující synchronizaci všech položek v košíku napříč více otevřenými okny stejného prohlížeče – pokud uživatel změní produkty košíku v jednom okně, stejná změna se projeví i ve všech ostatních.

Tohoto chování je docíleno JS události `window.onstorage`, která je vyvolána pokaždé, když je v kontextu jiného dokumentu (jiného okna) modifikováno lokální úložiště prohlížeče [47]. Při změně je uložen obsah košíku do LocalStorage pod klíčem `js-cart-change`. Ostatní okna tuto změnu odchytí a upraví svůj stav košíku.

UserInfoPanel.tsx Ikona uživatele v uživatelské liště stránky renderovaná do elementu s id `js-shop-user`. Pokud je uživatel přihlášen, je změněn css styl ikony a při kliknutí je přesměrován na stránku s uživatelským účtem. Pokud uživatel přihlášen není, po kliknutí vyskočí přihlašovací modal.

CompareProducts.tsx Komponenta, která zahrnuje logiku pro srovnávač produktů. V DOM se vyhledají všechny tagy s atributem `data-js-shop-compare-product-button` a do nich se pomocí `ReactDOM.createPortal` vloží tlačítko (`CompareProductButton`) pro přidání produktu do srovnávače. Každý takový prvek má také atribut `data-product-id` obsahující identifikátor produktu. Po kliknutí na tlačítko je produkt přidán, nebo odebrán z úložiště prohlížeče (LocalStorage). Tlačítka u aktivních produktů na stránce jsou zvýrazněna, což umožňuje snadno rozpoznat, které produkty jsou momentálně vybrány.

Kromě tlačítek na přidání/odebrání je také vloženo další tlačítko (`CompareProductsOpen`) do tagu s atributem `data-js-shop-compare-product-open`. Kliknutím na toto tlačítko je uživatel přesměrován na URL adresu detailu srovnávače, která ve své query části obsahuje identifikátory všech vybraných produktů. Detail srovnavače je zdroj, který vrací stránku s tabulkou, kde jsou znázorněny rozdíly produktů (jejich parametry atp.).

```
return <>
   // Pro každý produkt je vyrenderováno tlačítko na přidání do srovnavače
   {productsElements.map((value) => {
      const buttonType = value.element.dataset.jsShopCompareProductButton;
      return ReactDOM.createPortal(
       <CompareProductButton
        active={compareItems.has(value.productId)}
        buttonType={buttonType}
        changeActive={(active) =>
        buttonClick(value.productId, active, buttonType)}
       />, value.element);
   })}

    // Vyrenderování tlačítek na otevření detailu srovnavače
   {Array.from(openCompareElements.current || []).map((element) => {
      return ReactDOM.createPortal(
       <CompareProductOpen
        openCompare={() => openCompareProducts(compareItems)}
        productsIds={Array.from(compareItems) || []}
       />, element);
   })}

</>;
```

Výpis kódu 5.14 CompareProducts.tsx – renderování tlačítek

Favorties.tsx Komponenta pro logiku oblíbených produktů. Slouží podobně jako komponenta pro srovnávání produktů, pouze s několika následujícími rozdíly. Oblíbené produkty jsou uchovávány

v globálním stavu aplikace, který se při inicializaci aplikace stahuje ze serveru (pokud není již v prohlížeči). Po přidání nebo odebrání produktu z oblíbených je vyslán požadavek na GraphQL mutation endpoint `productFavoritesUpdate`, který aktualizuje relaci uživatele.

RedirectContextModal.tsx Vyskakující okno, které je zobrazeno pouze pokud je v odpovědi na GraphQL query dotaz `redirectLocation` obsažen objekt s informacemi o přesměrování. Tento dotaz je vyslán jako součást požadavku na získání globálního stavu aplikace pouze v případě, že toto okno nebylo již uživateli zobrazeno. Pokud již zobrazeno bylo, uloží se do LocalStorage příznak, který zajistí, že se již znovu nezobrazí. Tímto způsobem je zajištěno, že se vyskakující okno zobrazí pouze v případě, kdy uživatel přistoupil na e-shop poprvé.

5.3.3 Integrace aplikace

Odkaz na React aplikaci je vložen na konec `body` elementu každé prvotní HTML stránky do `<script>` tagu. Včetně toho je do stránky vložen i soubor s vygenerovanými překlady pro aktivní jazyk. Vše je podmíněno WPJ modulem `JS_SHOP`. Viz. ukázka kódu 5.15.

```
<!DOCTYPE html>
...
<body>
...
{ifmodule "JS_SHOP"}
   <div id="js-shop"></div>
   {* Soubor s překlady *}
   {include "jsShop.translations.tpl"}
{/ifmodule}

...
{block "js-entry"}
    {ifmodule "JS_SHOP"}
    {* Smarty tag pro vygenerování <script src="..."> *}
          {encore_entry_script_tags entry='js-shop'}
    {/ifmodule}
{/block}
</body>
```

Výpis kódu 5.15 Integrace React aplikace do WPJshopu

5.4 Nastavení sdílené mezipaměti

Poslední část implementace je nastavení sdílené mezipaměti. Zvolenou technologií je Bunny.net. WPJ již tuto CDN používá pro kešování statických zdrojů a její rozšíření na potřebnou doménu je nejjednodušší možností, jak zavést sdílenou mezipaměť. Nyní přes tento middleware prochází pouze statický obsah, jakým jsou obrázky, videa a JS/CSS soubory; pro potřebný účel je stejně tak potřeba nasměrovat i zbytek komunikace, aby šlo kešovat i potřebné HTML odpovědi.

Tuto technologii však nelze jednoduše použít pro lokální vývoj, kde je také potřeba mít možnost vyzkoušet kešování ve sdílené mezipaměti. Bez možnosti spuštění aplikace ve stejných podmínkách jako v produkci není možné mnohdy správně replikovat hlášené chyby, nebo odladit aplikaci před jejím nasazením. Z toho důvodu je implementován i Nginx reverzní proxy server. Ten kešuje stejně jako Bunny.net, ale lze ho jednoduše zprovoznit na lokálním stroji.

5.4.1 Konfigurace CDN Bunny.net

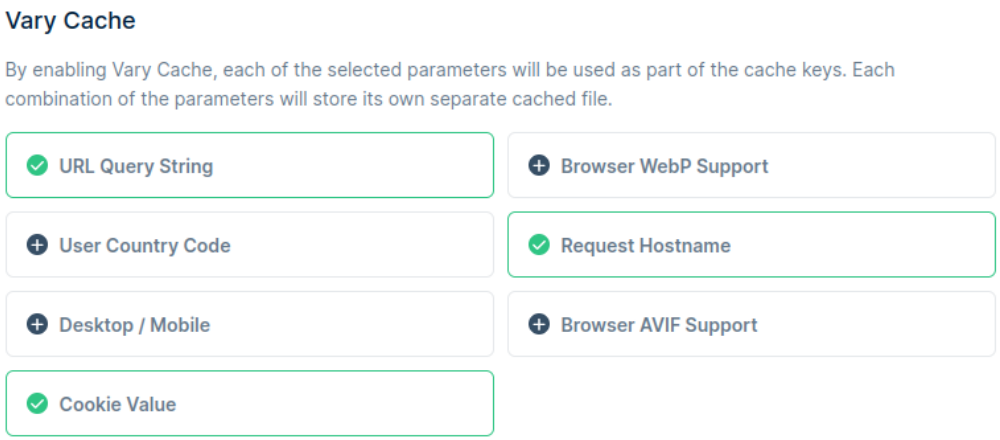
Nyní je v Bunny.net nastavena pro každý e-shop samostatná pull zóna; to je „zóna“, přes kterou uživatelé vyžadují obsah. Každá pull zóna má samostatné nastavení a lze ji například nastavit jiný zdrojový server a jinou logiku kešování. Mimo to ji lze i samostatně monitorovat. [48]

Vzhledem k tomu, že statické soubory, jako jsou obrázky a videa, nepotřebují speciální nastavení kešovacího klíče jako dynamický obsah, bude každý e-shop nyní mít dvě oddělené pull zóny. Původní pull zóna zůstane beze změny a bude přidána nová, která bude sloužit pro dynamický obsah a bude přiřazena k hlavní doméně e-shopu (doméně druhého řádu). Například pro e-shop s doménou `example.org` budou nastaveny DNS záznamy následovně:

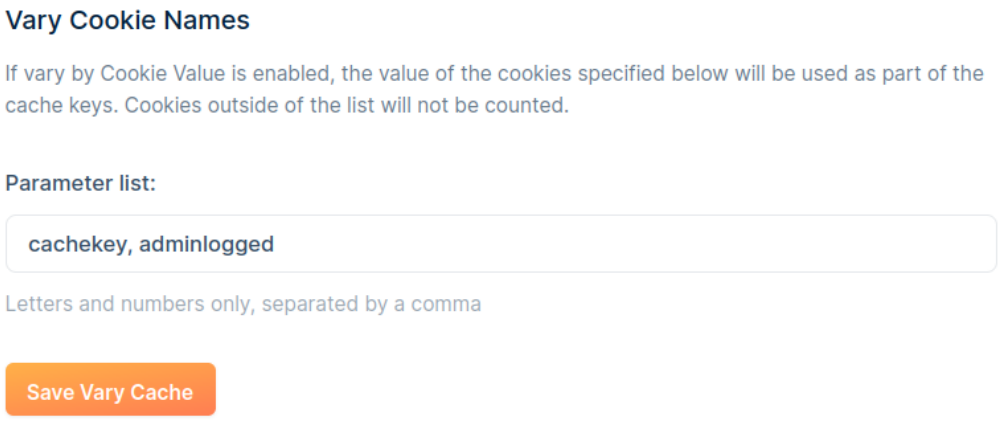
- Doména třetího řádu `data.example.org` bude nadále směrována do původní pull zóny určené pro statické soubory.
- Hlavní doména `example.org` bude směrována do nové pull zóny určené pro dynamický obsah, který dosud neprocházel přes Bunny.net. Tato pull zóna bude mít nastavené rozšíření kešovacího klíče pomocí cookie.

Rozdělení dat do dvou skupin zlepšuje organizaci a efektivnější využívání sdíleného úložiště v mezipaměti. Tím, že se stejný statický obsah neukládá duplicitně kvůli specifickým nastavením kešovacích klíčů, zůstává úložiště efektivnější. Toto uspořádání také umožňuje lépe konfigurovat dodatečné optimalizace pro obrázky a videa, které CDN poskytuje. Navíc lze jednotlivé pull zóny monitorovat samostatně, což také přispívá k lepší přehlednosti.

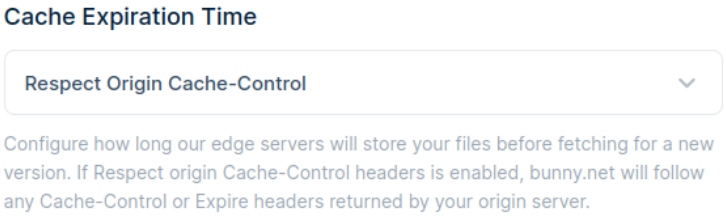
Konfiguraci kešování nové pull zóny v administraci Bunny.net lze vidět na následujících obrázcích. Kešovací klíč se skládá z celé URL (obr. 5.4) a je rozšířen pomocí cookie hodnot `cachekey` a `adminlogged` (obr. 5.5). Cookie `adminlogged` slouží k tomu, aby se přihlášeným administrátorům vždy zobrazoval aktuální obsah. Dále je nastaveno respektování doby expirace v `Cache-Control` direktivách (obr. 5.6). Touto konfigurací je zajištěno kešování veškerého obsahu podle direktiv v kešovacích hlavičkách včetně rozšiřujícího klíče v cookie.



Obrázek 5.4 Bunny.net konfigurace – Vary Cache



Obrázek 5.5 Bunny.net konfigurace – Vary Cookie Names



Obrázek 5.6 Bunny.net konfigurace – Cache Expiration Time

Každé datové centrum Bunny.net má svoji vlastní nezávislou mezipaměť. Toto rozložení na více sdílených mezipamětí může vést ke snížení efektivnosti vytvořeného kešovacího mechanismu. I když je uložena odpověď v mezipaměti jednoho centra, nemusí být uložena v mezipaměti jiného centra. Pokud tedy přijde požadavek na datové centrum bez uložené odpovědi, musí být přeposlán na zdrojový server i přes to, že jiné datové centrum tuto odpověď v mezipaměti obsahuje.

5.4.2 Nginx konfigurace pro lokální vývoj

Pro lokální vývoj je vytvořen Docker obraz s nakonfigurovaným Nginx serverem. Díky tomu lze jednoduše pomocí jednoho příkazu `docker-compose up` spustit Nginx reverzní proxy server na počítači vývojáře. Server je nastaven na port `8090`, namísto samotného `localhost` může vývojář použít `localhost:8090` a veškerá komunikace bude procházet přes lokální reverzní kešovací proxy server.

Namísto využití oficiálního Nginx Docker obrazu je využit vlastní Docker obraz, který vychází z čisté `debian:bullseye` distribuce a Nginx je na něm zkompilován a nainstalován ze zdrojových souborů. Vlastní image byl vytvořen z toho důvodu, aby do něho mohl být přidán Nginx modul `nginx-module-vts`[5], který poskytuje uživatelsky přívětivé zobrazení statistik mezipaměti. Vývojář tedy může při lokálním vývoji jednoduše monitorovat počet požadavků obsloužených z mezipaměti atp.

Konfiguraci Nginx serveru lze vidět na následující ukázce kódu 5.16.

[5] `https://github.com/vozlt/nginx-module-vts`

```
vhost_traffic_status_zone; # Inicializace monitorovacího modulu
proxy_cache_path /tmp/cache levels=1:2 keys_zone=default_cache:60m
                 inactive=120m use_temp_path=off;
server {
  listen 8090;
  server_name _;
  set $origin localhost; #nastavení localhostu jako zdrojový "upstream" server
  proxy_cache_revalidate on; # Povolení revalidace

  # Endpoint pro zobrazení statistik monitoringu
  location /_nginx/status {
    vhost_traffic_status_bypass_limit on;
    vhost_traffic_status_bypass_stats on;
    vhost_traffic_status_display;
    vhost_traffic_status_display_format html;
  }

  #Samostatný endpoint pro GraphQL, veškerá komunikace je jednoduše přeposlána
  location /graphql {
    # komunikace přes GraphQL není zobrazena ve statistikách
    vhost_traffic_status_bypass_stats on;
    proxy_pass http://$origin/graphql;
  }

  # Zóna pro zdroje poskytující statické soubory
  location ~ ^/(data|common|templates|web)/ {
    proxy_cache default_cache; # Povolení kešování
    add_header X-Proxy-Cache $upstream_cache_status;
    proxy_pass http://$origin;
  }

  # Zína pro veškerý zbylý obsah
  location / {
    proxy_cache default_cache; # Povolení kešování
    # Rozšíření kešovacího klíč o hodnotu "cachekey" cookie
    proxy_cache_key "$scheme$request_method$host$request_uri$cookie_cachekey";
    # pokud je nastavena cookie adminlogged, keš je přeskočena
    proxy_cache_bypass $cookie_adminlogged;
    proxy_no_cache $cookie_adminlogged;
    # Přidání hlaviček o statusu keše a s podobou kešovacího klíče do odpovědí
    add_header X-Proxy-Cache $upstream_cache_status;
    add_header X-Proxy-Cache-Key
      $scheme$request_method$host$request_uri$cookie_cachekey;

    proxy_pass http://$origin;
  }
}
```

Výpis kódu 5.16 Nginx – konfigurace reverzního kešovacího proxy serveru pro lokální vývoj

Kapitola 6
Výsledné řešení

Kapitola obsahuje ukázku kešovacího mechanismu v provozu, výsledné zhodnocení, naměřené statistiky a možné úpravy do budoucna.

6.1 Ukázka řešení v provozu

První e-shop, na kterém byl výsledný kešovací mechanismus vyzkoušen, je `www.rockpoint.cz`. Spuštění tohoto e-shopu bylo plánováno ve stejné době, co byla dokončena implementace kešovacího mechanismu. Stal se tak ideální možností na otestování a odladění zbývajících detailů kešovacího mechanismu. V následujících ukázkách jsou obrázky právě z tohoto e-shopu.

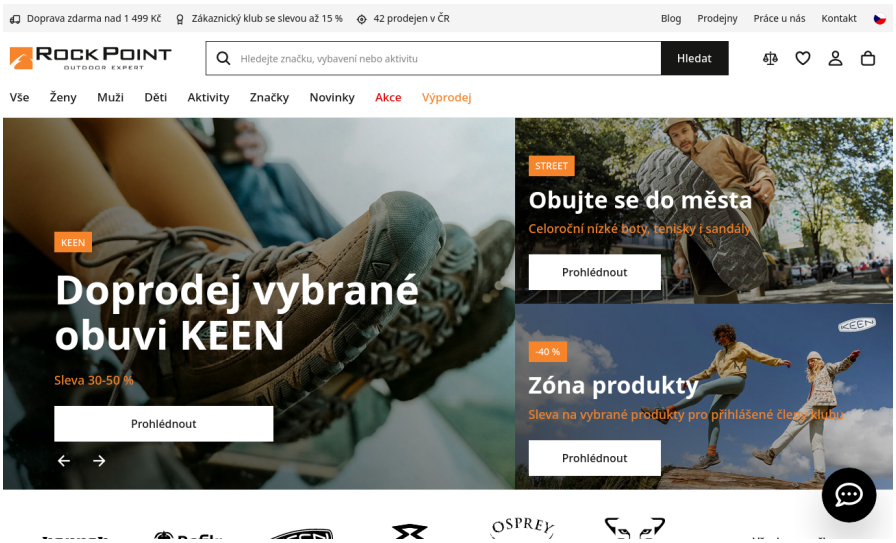
6.1.1 Princip komunikace

V této ukázce je vidět komunikace se serverem v případě, kdy uživatel nemá žádná uložená data v lokálním úložišti prohlížeče (SessionStorage). Na prvním obrázku 6.1 je vidět vzhled domovské stránky e-shopu. Součástí odpovědi na úvodní požadavek je téměř veškerý tento HTML obsah. Na obrázku 6.2 je zároveň vidět tento požadavek v síťovém monitoru prohlížeče, včetně následujícího XHR požadavku. Komprimovaná velikost úvodní odpovědi je 30 kB a doba čekání byla 4 ms (bez započítání doby TCP a TLS handshake). Tato krátká doba odpovědi je zásluha CDN serveru, který celý úvodní požadavek obsloužil z mezipaměti bez navázání spojení se zdrojovým serverem.

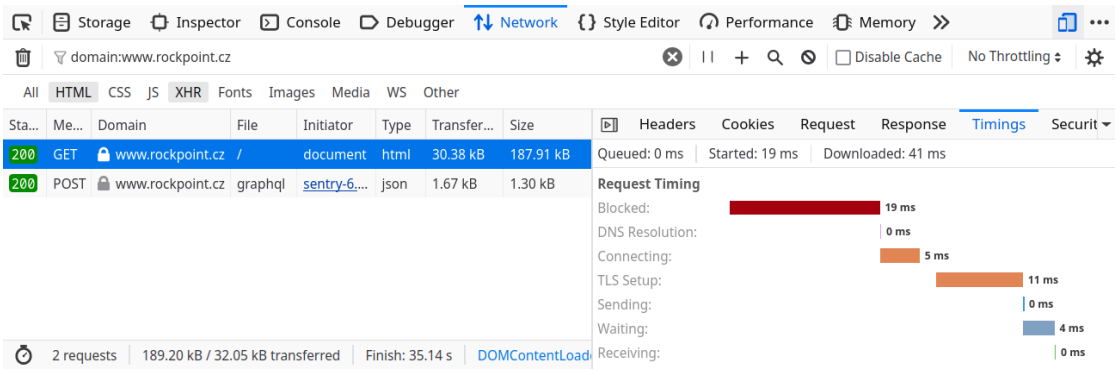
Po zpracování HTML odpovědi na úvodní požadavek je prohlížečem inicializována React aplikace. Pokud požadovaná data nejsou v lokální mezipaměti, nebo je potřeba získat data aktuální, je vznesen XHR požadavek na GraphQL API zdrojového serveru. V tomto požadavku je dotaz na šest GraphQL „queries“: `cart`, `redirectLocation`, `products`, `productsFavorites`, `me`, `contexts` a `gtmData`. Celou GraphQL odpověď ve formátu JSON lze vidět na obrázku 6.3; získaná data obsahují uživatelovy produkty v košíku, oblíbené produkty, informace o uživatelském účtu atp. Tato data jsou po obdržení vyrenderována do příslušných částí stránky.

Tímto způsobem jsou oddělena kešovatelná sdílená data a personalizovaná uživatelská data do dvou samostatných požadavků. Sdílená data jsou poskytnuta ze sdílené mezipaměti CDN serveru (pokud je mezipaměť obsahuje) a personalizovaná data jsou vždy poskytnuta ze zdrojového serveru.

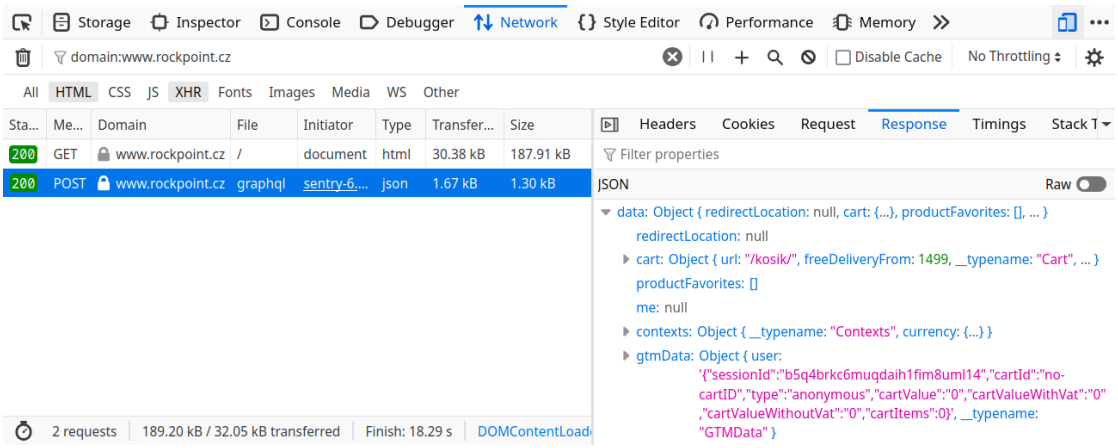
Pokud uživatel přejde na další stránku e-shopu, XHR požadavek na GraphQL API již vznesen není. Veškerá potřebná data jsou už uložena v SessionStorage prohlížeče. Pokud je tedy úvodní požadavek na tuto novou stránku obsloužen také z mezipaměti, server nemusí generovat žádný obsah a je odlehčen o celé jedno poskytnutí stránky uživateli.



Obrázek 6.1 Domovská stránka – www.rockpoint.cz



Obrázek 6.2 Domovská stránky – doba odpovědi



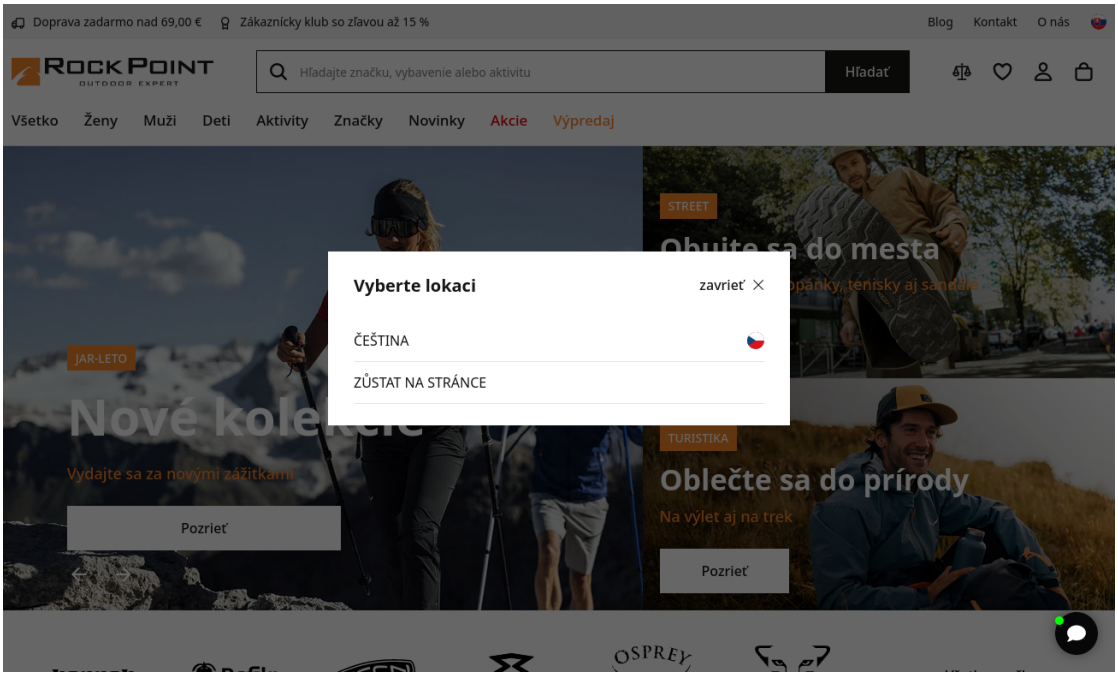
Obrázek 6.3 Domovská stránky – GraphQL požadavek po načtění stránky (první návštěva)

6.1.2 První návštěva – přesměrování podle geolokace

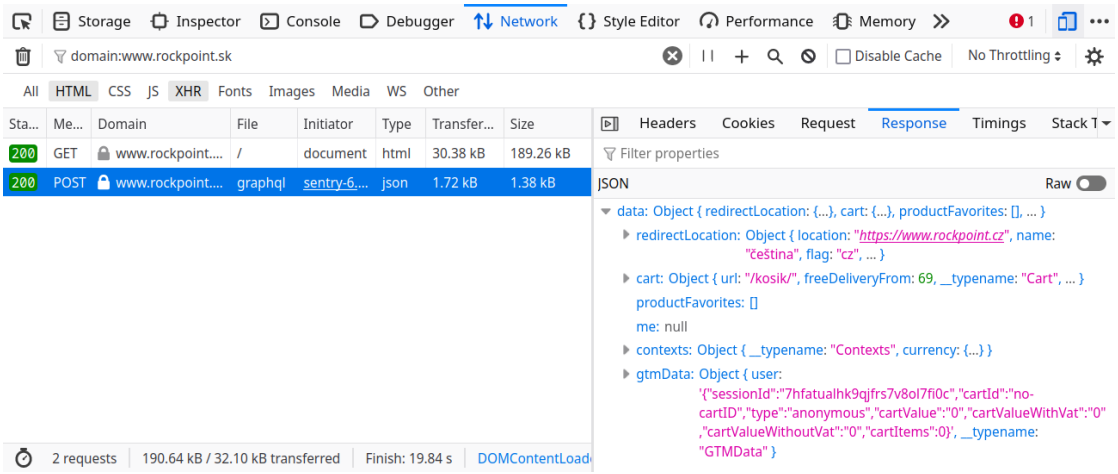
Rockpoint provozuje e-shop pro několik různých zemí a je podle toho rozdělen na jednotlivé domény (www.rockpoint.cz, www.rockpoint.sk, www.rockpoint.pl atp.), které se liší v jazyce a měně obsahu. Může se stát, že uživatel omylem přistoupí na jinou doménu, než by správně potřeboval. K tomu slouží implementovaná funkcionalita „přesměrování podle kontextu uživatele“, která zobrazí uživateli vyskakovací okno s možností přepnutí na správný kontext (v tomto případě doménu).

Pokud prohlížeč uživatele v LocalStorage nemá uložený příznak toho, že uživatel na stránku již přistoupil, součástí požadavku na získání personalizovaných dat skrze GraphQL API je i query dotaz `redirectLocation`. Na základě výsledku tohoto query je uživateli zobrazeno okno s odkazem na přesměrování. Na obrázku 6.4 lze vidět vyskakovací okno na doméně e-shopu www.rockpoint.sk. Server podle IP adresy uživatele determinoval, že se jedná o uživatele přistupujícího z České republiky a v odpovědi na query dotaz `redirectLocation` vrátil odkaz na českou doménu (viz. obrázek 6.5).

Kromě změny domény lze tuto funkcionalitu využít i pro jiné účely. Může se například jednat pouze o změnu měny či jazyka na jedné doméně.



Obrázek 6.4 Slovenská doména – nabídka přesměrování na českou doménu



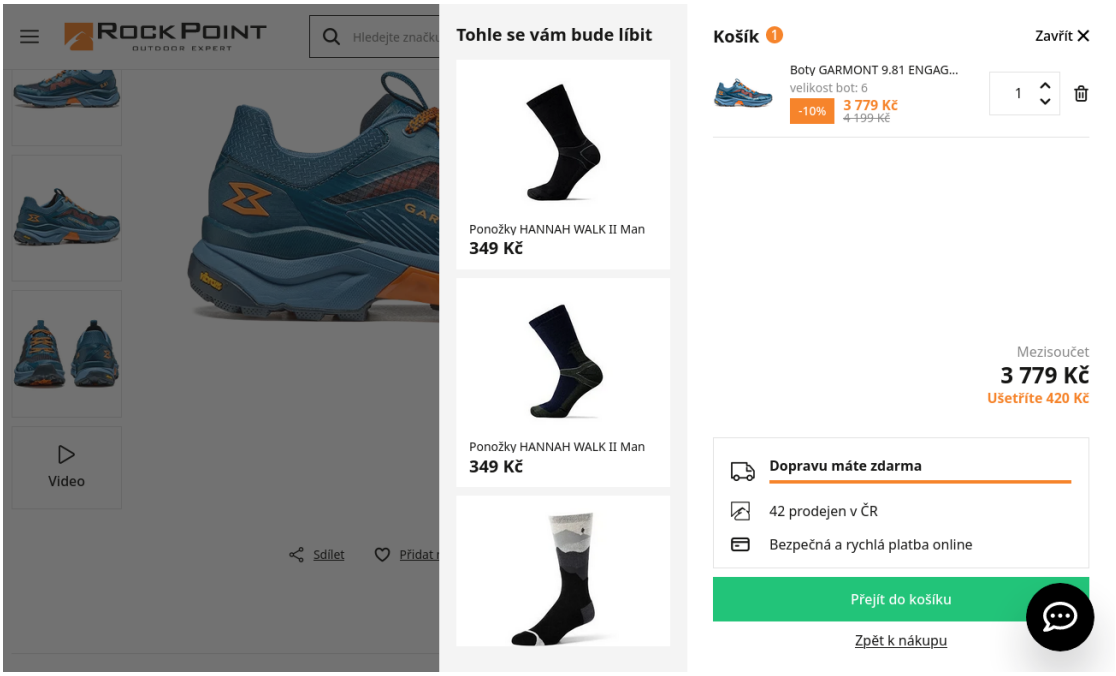
Obrázek 6.5 Slovenská doména – GraphQL odpověď s informacemi o přesměrování

6.1.3 Vložení produktu do košíku

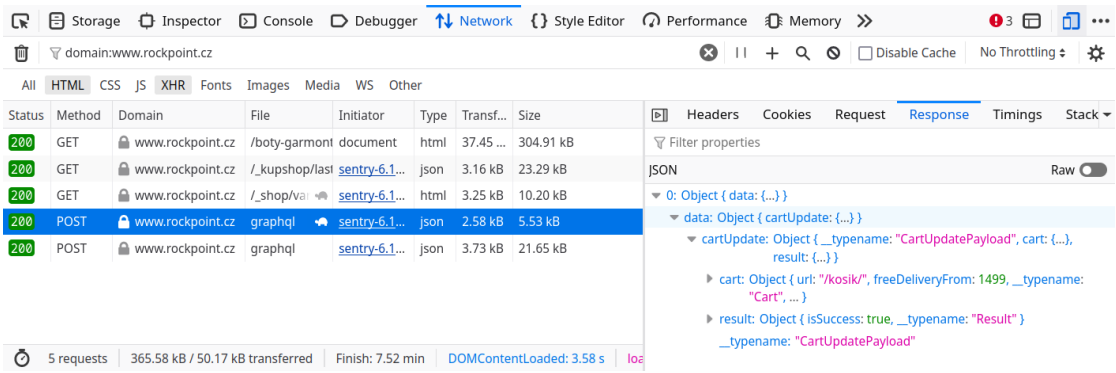
Veškerý obsah košíku je v novém řešení získáván přes GraphQL endpoint ve formátu JSON. Celý košík je renderován na straně klienta React aplikací a veškerá logika s ním spojená je také součástí této aplikace. Vzhled košíku je vidět na obrázku 6.6 a nijak se vizuálně neliší od jeho předchozí verze. Na rozdíl od předchozí verze však jeho obsah není součástí HTML odpovědi na úvodní požadavek, díky čemuž lze veškerý tento HTML obsah kešovat ve sdílené mezipaměti (CDN).

Po kliknutí na tlačítko „Přidat do košíku“, je skrze React aplikaci vyslán GraphQL mutation dotaz `cartUpdate`, který v těle požadavku obsahuje všechny produkty v košíku včetně nově přidaných. Server požadavek zpracuje, aktualizuje stav databáze a vrátí odpověď se stavem košíku včetně přepočítané ceny. Pokud se jedná o první produkt uživatele v košíku, součástí odpovědi je i hlavička `Set-Cookie` s hodnotou `cartID`, která slouží jako klíč k ke košíku uživatele.

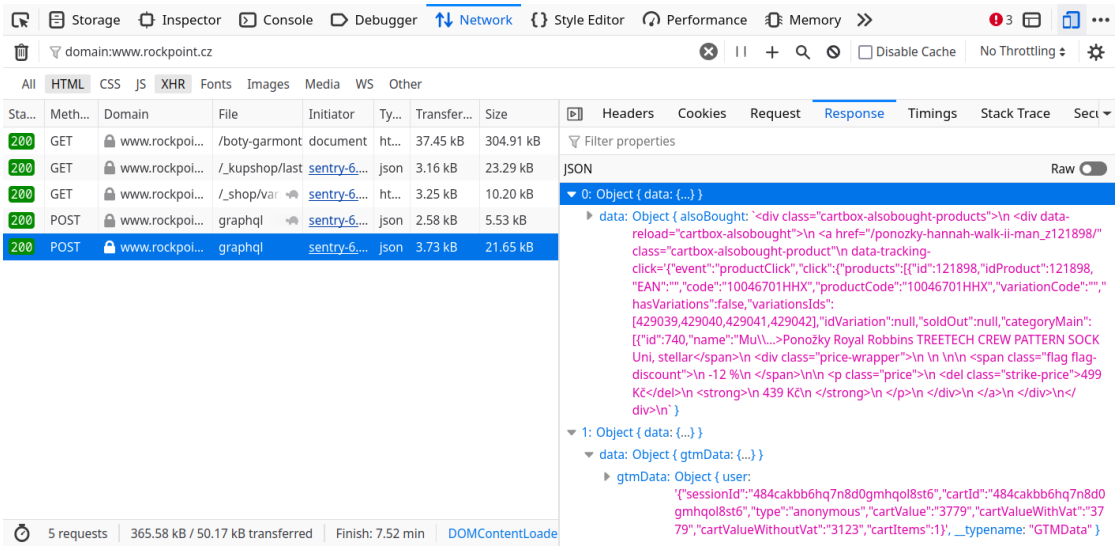
Poté, co klientská React aplikace přijme odpověď na přidání do košíku, je vyslán další požadavek pro získání doporučených produktů, které se v košíku zobrazují (sloupec v košíku „Tohle se vám bude líbit“) a pro získání personalizovaného GTM objektu. Doporučené produkty jsou získány již ve formátu HTML a jsou jednoduše vloženy do DOM stránky (server side rendering). Personalizovaný GTM objekt je uložen do LocalStorage prohlížeče a je vložen do datové vrstvy při následujícím načtení stránky.



Obrázek 6.6 Vložení do košíku – košík renderovaný Reactem



Obrázek 6.7 Vložení do košíku – odpověď na GraphQL požadavek pro přidání produktu



Obrázek 6.8 Vložení do košíku – odpověď na druhý GraphQL požadavek pro získání doporučených produktů a GTM objektu

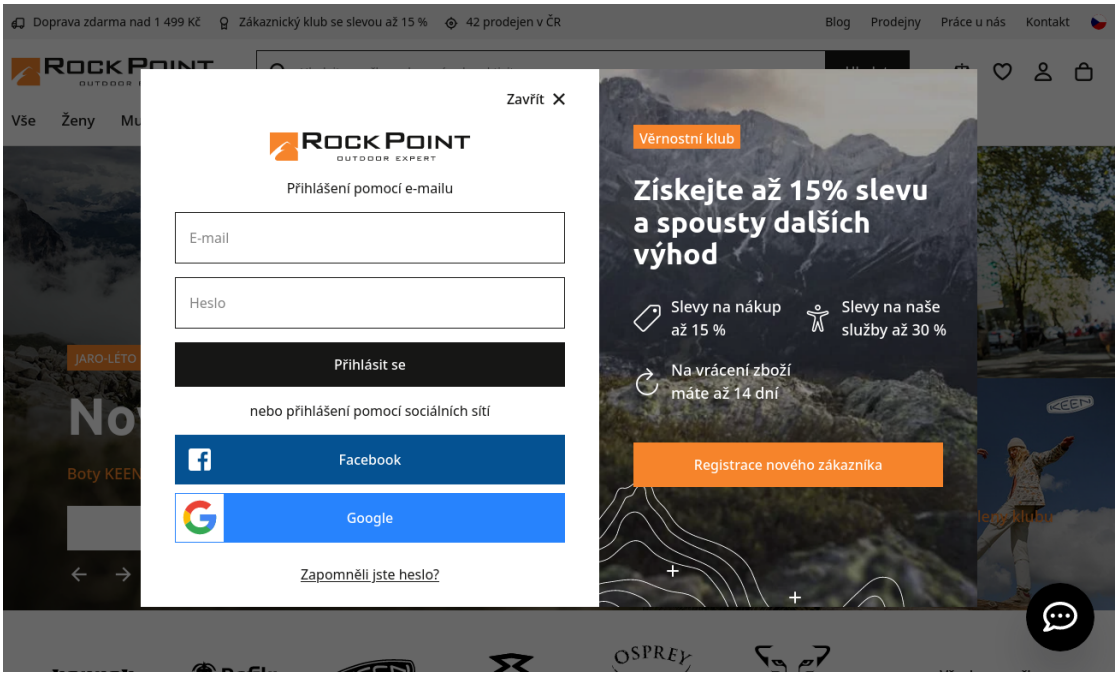
6.1.4 Přihlášení uživatele

Princip přihlášení uživatele se v novém řešení nezměnil, zachovaný přihlašovací formulář lze vidět na obrázku 6.9. Zajímavá je však odpověď na POST požadavek po odeslání přihlašovacího formuláře. Ten reprezentuje celkový princip vrácení cookies ze serveru. Na obrázku 6.10 lze vidět, že odpověď je HTTP 302 přesměrování a obsahuje pět `set-cookie` hlaviček. Účel jednotlivých cookies je vypsán zde:

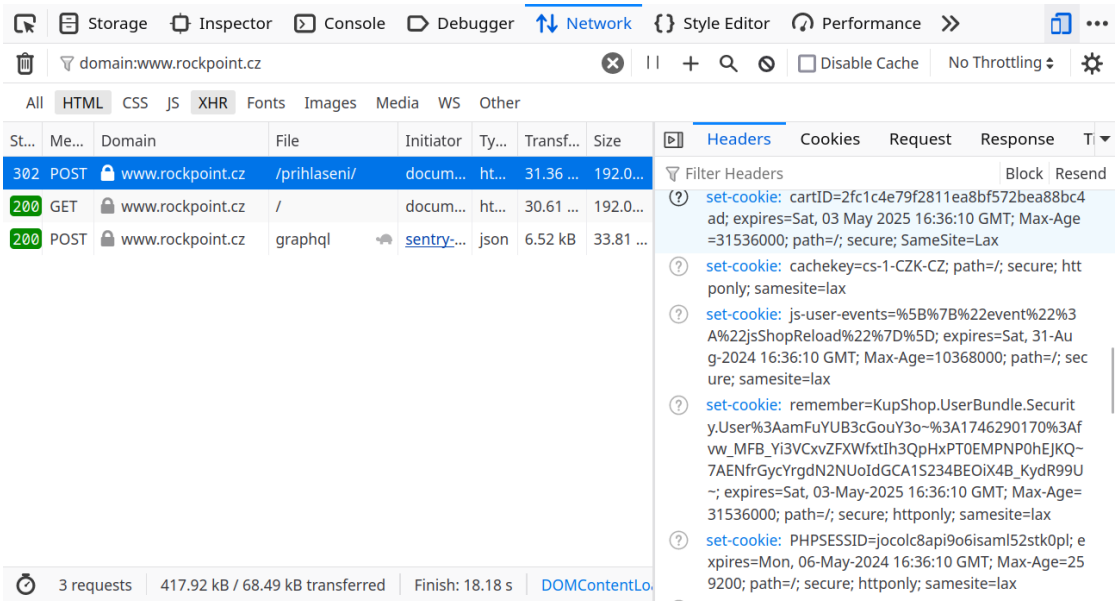
- **cartID** je klíč na položky košíku přihlášeného uživatele, které jsou uchovány v relační databázi.
- **cachekey** slouží k rozšíření kešovacího klíče v HTTP sdílené mezipaměti CDN. Uživatel má nastavený jiný kontext než nepřihlášení uživatelé – v tomto případě se jedná o jinou cenovou hladinu, díky které uživatel vidí slevu na všechny produkty. Takto je zajištěno, že uživateli nebude ze sdílené mezipaměti poskytnut obsah zobrazující produkty bez slevy, cena produktů je totiž součásti kešovatelného HTML obsahu. Stejný rozšiřující klíč je poskytnut i všem ostatním uživatelům se stejnou cenovou hladinou na české doméně.
- **js-user-events** obsahuje příznak k přenačtení lokální paměti prohlížeče, kterou využívá React aplikace k uchování stavu napříč všemi stránky e-shopu. Po přihlášení uživatele je nutné tuto lokální mezipaměť přenačíst.
- **remember** je tzv. „remember me“ cookie a slouží k dlouhodobému zapamatování přihlášeného účtu uživatele. Nastavena je Symfony frameworkem, který tuto funkcionalitu poskytuje.
- **PHPSESSID** slouží jako klíč k relaci uživatele, ve které jsou uchovány různé programově nastavené proměnné (např. PurchaseState).

Po přijetí odpovědi jsou všechny tyto cookies uchovány v prohlížeči a uživatel je přesměrován na domovskou stránku. Požadavek na domovskou stránku lze také vidět na obrázku 6.10. Ten je možné obsloužit ze sdílené mezipaměti, pokud mezipaměť obsahuje odpověď s nyní již rozšířeným kešovacím klíčem. Při načítání HTML obsahu domovské stránky je inicializována React aplikace, která, vzhledem k přítomnosti hodnoty `jsShopReload` v cookie `js-user-events`, je nucena

přenačíst stav aplikace v lokální paměti prohlížeče. Vyslán je XHR GraphQL požadavek shodný s požadavkem z obrázku 6.3. Ten získá produkty v košíku, oblíbené produkty a další data spojená s uživatelským účtem.



Obrázek 6.9 Formulář na přihlášení k účtu



Obrázek 6.10 Přihlášení k účtu – odpověď na požadavek

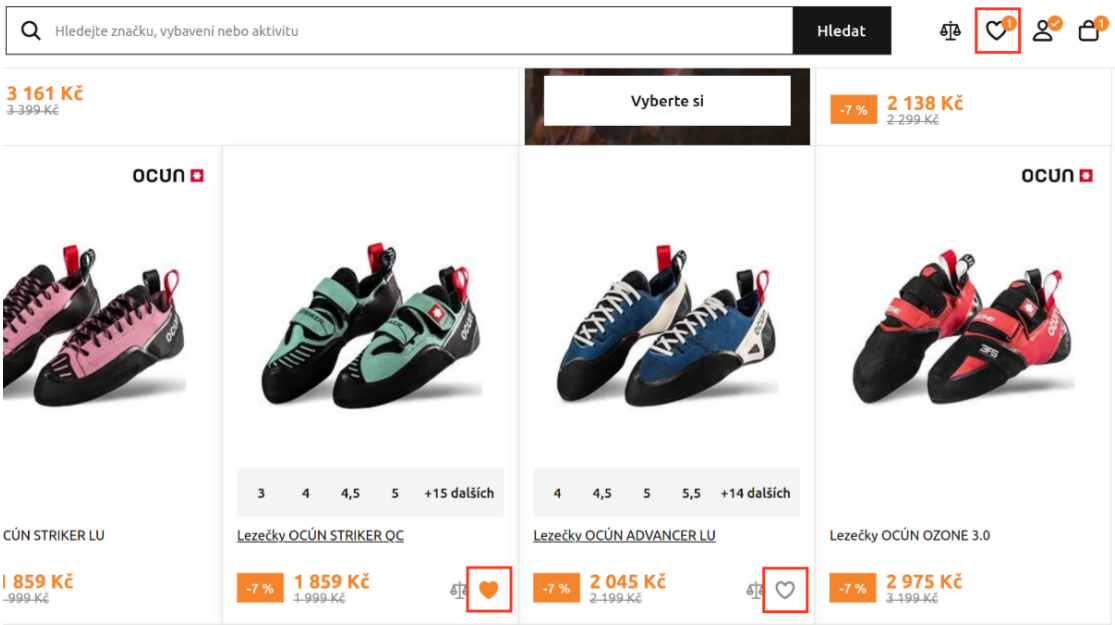
6.1.5 Oblíbené produkty

Přihlášený uživatel má možnost zařadit jednotlivé produkty e-shopu do seznamu oblíbených. Seznamy všech uživatelů jsou uchovány v relační databázi. Jak je vidět na obrázku 6.11, oblíbené produkty jsou zvýrazněny v kategorii produktů vybarvenou ikonou srdce. V původním řešení byla informace o uživatelově oblíbených produktech vždy součástí HTML odpovědi ze serveru. Nové řešení, které umožní ukládat HTML odpovědi ve sdílené mezipaměti, přenáší informaci o oblíbených produktech uživatele v GraphQL XHR odpovědi (viz. 6.3). Oblíbené produkty jsou uchovány i v lokální paměti prohlížeče (SessionStorage) a jsou součástí globálního stavu React aplikace, která je zároveň i renderuje do DOM stránky. Nutnost jejich získání ze zdrojového serveru je potřeba pouze tehdy, pokud je SessionStorage prázdná, stejně jako je tomu u produktů v košíku a dalších.

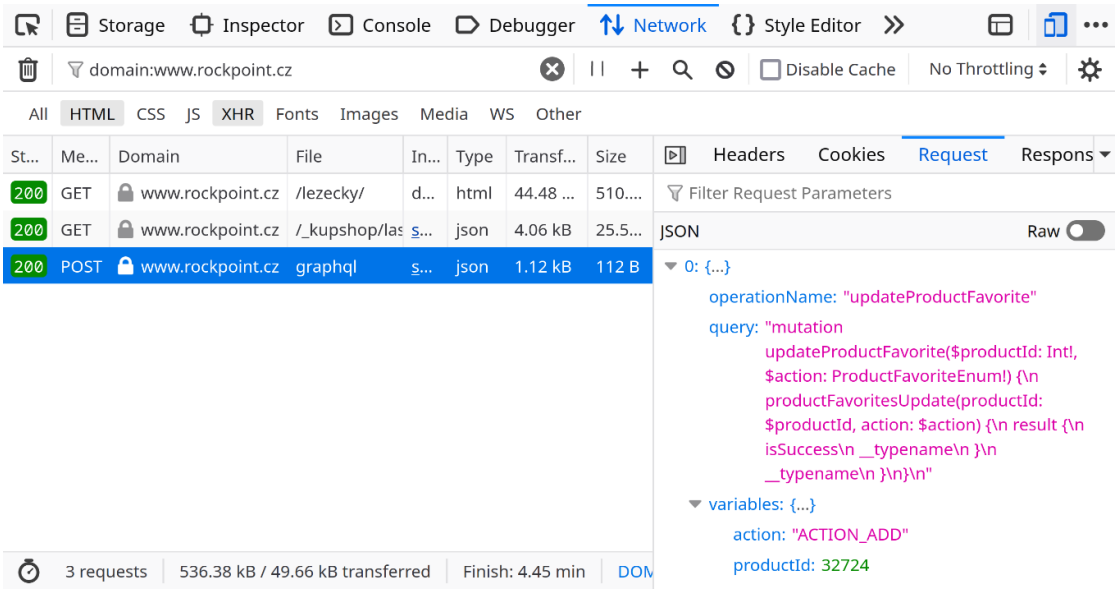
Po kliknutí na přidání produktu do oblíbených je produkt zvýrazněn (vybarví se ikona srdce). Produkt se přidá do SessionStorage a odešle se GraphQL požadavek na server. Tento požadavek lze vidět na obrázku 6.12. Odebrání produktu z oblíbených probíhá stejným způsobem.

Samotná stránka se všemi oblíbenými produkty kešovatelná není. Při jejím zobrazení jsou všechny oblíbené produkty uživatele součástí HTML odpovědi. React aplikace v tomto případě tedy nemusí nic renderovat, protože požadavek na zobrazení stránky s oblíbenými produkty vždy doputuje na zdrojový server a součástí odpovědi mohou být i personalizovaná data.

Podobně jako oblíbené produkty fungují nově i produkty ve srovnavači. Pro ně slouží ikona váhy na stránce (viz. obrázek 6.11). Rozdíl od oblíbených produktů je však takový, že produkty srovnavače jsou uchovány pouze v paměti prohlížeče (LocalStorage) a neukládají se na serveru. Ikona na zobrazení stránky se všemi porovnávanými produkty odkazuje na URL obsahující všechny identifikátory produktů ve srovnávači.



Obrázek 6.11 Oblíbené produkty – zvýraznění na stránce



Obrázek 6.12 Přidání do oblíbených – GraphQL požadavek

6.2 Zhodnocení a naměřené statistiky

Vytvořený kešovací mechanismus se po nasazení do ostrého provozu skutečně ukázal jako efektivní řešení. Výrazně je zkrácena průměrná doba čekání na HTML odpovědi úvodních požadavků na zdroje e-shopu. Zlepšení je obzvláště znát u typů zdrojů, které vyžadují spuštění rozsáhlých SQL dotazů, mnohdy trvajících stovky milisekund. Mezi takové zdroje, v případě e-shopu `www.rockpoint.cz`, patří především kategorie produktů s mnoha položkami. Například vygenerování HTML odpovědi zdrojovým serverem pro kategorii „Muži“, která je položena vysoko ve stromu kategorií a zobrazuje i položky všech podkategorií, trvá 1–2 sekundy. Díky tomu, že lze nyní v novém řešení celou tuto HTML odpověď kešovat ve sdílené mezipaměti CDN, je potřeba generovat tuto stránku pouze párkrát za hodinu, namísto generování při každém požadavku. Kromě kategorií produktů jsou kešovány i zdroje jako úvodní stránka, kategorie produktů, detail produktu a blog. Obsluha požadavků na tyto zdroje, pokud je odpověď přítomna ve sdílené mezipaměti, je většinou do 10 ms. Jedná se tedy o výrazné zlepšení oproti původnímu stavu WPJshopu, který musel generovat odpověď pro každý požadavek.

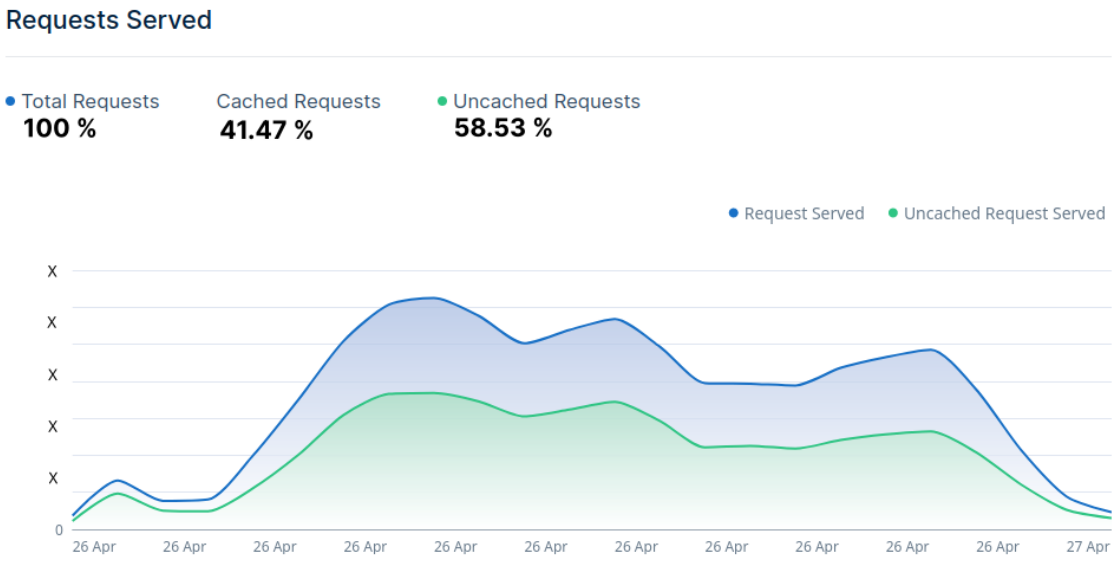
CDN Bunny.net navíc poskytuje i statistiky o celkovém počtu obsloužených požadavků a celkové velikosti poskytnutých dat. V rámci těchto statistik lze vidět i poměr, kolik z toho bylo poskytnuto z mezipaměti a kolik toho muselo být získáno ze zdrojového serveru. Vzhledem k tomu, že každý e-shop využívá dvě pull zóny (jednu pro statický obsah a druhou pro dynamický obsah), je možné získat i poměrně nezkreslenou představu o efektivitě vytvořeného řešení.

Na obrázcích 6.13 a 6.14 lze vidět Bunny.net grafy naměřených statistik z e-shopu rockpoint.cz za den 26.4.2024. Originální grafy poskytují přesný počet požadavků a šířku pásma v MB/GB. Pro zachování diskrétnosti jsou tyto hodnoty převedeny na procenta. Grafy vychází z pull zóny pro dynamická data, nejsou tedy ovlivněny komunikací pro získání statických dat, kterými jsou obrázky, videa, JS soubory a CSS soubory.

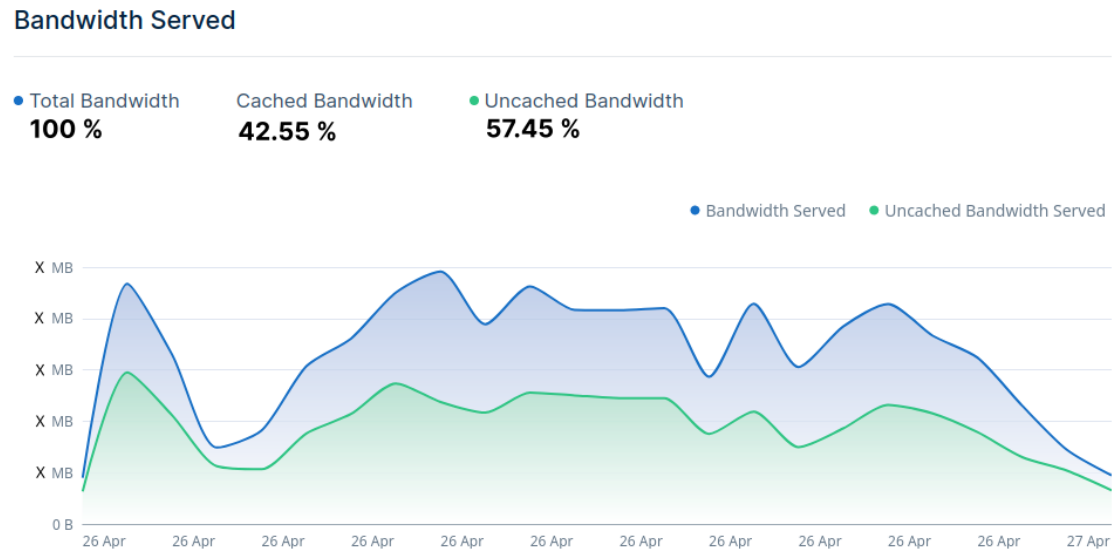
Na prvním grafu je vidět, že daného dne bylo z mezipaměti obslouženo 41.47 % požadavků. Nejvíce požadavků bylo na e-shop vzneseno kolem 10:00, 14:00 a 21:00 hodin. Modrá oblast znázorňuje celkový počet požadavků v danou denní hodinu a zelená oblast znázorňuje počet požadavků přeposlaných na zdrojový server v danou hodinu.

Na druhém grafu je vidět, že daného dne bylo poskytnuto celkově 42.55 % obsahu / šířky

pásma z mezipaměti. Nejvíce šířky bylo využito kolem 3:00 hodin a mezi 10:00 a 20:00 hodiny. Modrá oblast značí celkovou velikost obsahu poskytnutého v danou denní hodinu a zelená oblast značí velikost obsahu poskytnutého ze zdrojového serveru v danou hodinu.



Obrázek 6.13 Bunny.net – vizualizace poměru všech požadavků a požadavků přeposlaných na zdrojový server (za den)



Obrázek 6.14 Bunny.net statistika – poměr všech požadavků a požadavků přeposlaných na zdrojový server (za den)

Oba grafy jsou si podobné, až na oblast 3:00 hodin ráno. V tuto dobu probíhají rozsáhlé synchronizace s externími systémy. To vysvětluje velký objem přenesených dat v nízkém počtu požadavků.

Lze také namítnout, že procenta požadavků a šířky pásma obsloužených z mezipaměti jsou malá. Je však potřeba vzít v potaz, že se jedná o rozsáhlý e-shop, který má v nabídce přes 17000

produktů a obsahuje přes 1400 produktových kategorií. Zároveň je poskytován obsah v různých jazycích a v různých cenových hladinách.

6.3 Možné úpravy do budoucna

Vytvořený kešovací mechanismus byl již od svého návrhu směrován tak, aby dovolil různé úpravy a rozšíření v budoucnu. Jednou z jeho výhod je nezávislost na implementaci sdílené mezipaměti i přes to, že je to jedna ze stěžejních částí kešovacího mechanismu. Jediným požadavkem na sdílenou mezipaměť je možnost rozšíření kešovacího klíče pomocí konkrétní cookie hodnoty. Proto není v budoucnu problém uvažovat i o nahrazení CDN Bunny.net za jinou, flexibilnější implementaci sdílené mezipaměti (jinou CDN nebo reverzní proxy server).

Některé možné úpravy do budoucna již byly zmíněny v průběhu návrhu a implementace. Nicméně, zde jsou vypsány souhrnně, společně i s dalšími:

- **Úprava URL pro kešovací klíč**

  Při kliknutí na reklamu ve vyhledávačích nebo na sociálních sítích je uživatel přesměrován na webovou stránku. Do přesměrované URL jsou však přidávány dodatečné query parametry pro analytické účely. Například parametry `gclid` a `gad_source` jsou přidávány v případě Googlu. V případě Seznamu jsou přidávány `utm_campaign`, `utm_content`, `utm_term` a `utm_source`. Přítomnost těchto parametrů ovlivňuje podobu kešovacího klíče a vzhledem k tomu, že jejich obsah je vždy unikátní, není tyto požadavky možné obsloužit ze sdílené mezipaměti. Tyto query parametry nejsou využity na zdrojovém serveru a tak není potřeba, aby každý takovýto požadavek byl zdrojovým serverem obsloužen.

  Některé kešovací middlewary, jako např. Varnish, umožňují upravit podobu URL pro kešovací klíč. Díky tomu lze i takovéto požadavky obsloužit ze sdílené mezipaměti tím, že specifikované query parametry nebudou součástí kešovacího klíče.

- **Kešovací middleware pro GraphQL API**

  GraphQL queries jsou v současném řešení dotazovány pomocí HTTP metody POST. Ta obecně není jednoduše kešovatelná ve sdílené mezipaměti [27]. GraphQL však umožňuje dotazovat queries i pomocí metody GET, u které jsou dotazy součástí URL požadavku a odpovědi tak lze ukládat v libovolné HTTP mezipaměti. GraphQL queries vytvořené v této práci nejsou vhodné pro kešování, zejména kvůli tomu, že jejich odpovědi obsahují krátkodobá a personalizovaná data. Některé ostatní queries však tento princip využít mohou – například dotaz na získání kategorie, který je využíván u některých e-shopů k dynamickému načítání mobilního menu.

- **Využití ESI (Edge Side Includes)**

  Pokud by byla využita implementace sdílené mezipaměti podporující Edge Side Includes, je možné různé části kešovatelného HTML obsahu uchovávat v mezipaměti na rozdílnou dobu. To může být využito ke zvýšení efektivity kešovacího systému. Například některé části stránky, jako jsou zápatí a navigace, mohou mít nastavenou životnost v mezipaměti vyšší než ostatní části stránky.

- **Využití revalidace**

  Revalidace odpovědí v mezipaměti je jeden ze způsobů, jak lze také zvýšit efektivitu stávajícího řešení. Součástí implementovaného řešení je již podpora pro revalidaci pomocí `Etag` hlavičky, ale pro e-shopy zatím není využita. Pro využití revalidace stačí pouze implementovat jednu metodu na vrácení `Etagu` pro každý typ stránky (viz. 5.2.5). Mimo již implementované logiky pro `Etag` lze přidat analogicky i logiku pro revalidaci pomocí `Last-Modified`.

Kapitola 7
Závěr

Výsledkem diplomové práce je kešovací mechanismus pro e-commerce platformu WPJshop, který je již úspěšně nasazen na prvním e-shopu a je připraven k použití i na dalších e-shopech.

Teoretická část práce se věnuje celkové problematice, předkládá základní principy, technologie a pojmy obecně spojené s kešováním webových aplikací. Zmíněny jsou problémy, se kterými se lze při tvorbě kešovacího mechanismu setkat a jakými způsoby mohou být vyřešeny. Vyčteny jsou i uvažované technologie, které ve výsledném návrhu použity nebyly. Teoretická část práce tak může sloužit i jako základ pro návrh jiných kešovacích mechanismů, které budou využity pro jiné platformy či aplikace. V praktické části je implementován výsledný návrh. Zmíněny jsou všechny důležité části platformy, které bylo nutné upravit nebo implementovat.

Upraven byl způsob poskytování personalizovaných dat tak, aby veškerým stěžejním HTML stránkám bylo umožněno ukládání ve sdílené mezipaměti. Mezi tyto stránky patří úvodní stránka, kategorie produktů, detail produktu, blog a ostatní obsahové stránky. HTML obsah těchto stránek nyní obsahuje pouze data, která lze poskytnou více uživatelům. Zbylá personalizovaná data jsou získávána prostřednictvím JavaScriptové aplikace, která komunikuje s aplikačním rozhraním zdrojového serveru. Tímto způsobem je umožněno poskytnout uživatelům většinu obsahu z mezipaměti, bez nutnosti jeho opakovaného generování při každém požadavku.

Zachovány jsou všechny důležité funkcionality WPJshopu, které byly uživatelům nabízeny v původním řešení. Změněn byl pouze princip fungování některých z nich tak, aby odpovídal výslednému návrhu. Kromě celkového zrychlení systému tak z uživatelského hlediska neproběhla žádná pozorovatelná změna.

Bibliografie

1. *Tvoříme e-shopy - wpj.cz* [online]. WPJ, ©2024 [cit. 2024-02-17]. Dostupné z: `https://www.wpj.cz/`.
2. *Výsledky hledání modul — WPJ Shop nápověda* [online]. WPJ, ©2024 [cit. 2024-05-07]. Dostupné z: `https://napoveda.wpjshop.cz/search?collectionId=%5C&query=modul`.
3. HAUSCHWITZ, Marek. *Modul pro správu B2B předobjednávek v rámci e-shopového řešení wpjshop*. Nám. W. Churchilla 1938/4 130 67 Praha 3 - Žižkov, 2023. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky.
4. *PHP: Preface - Manual* [online]. The PHP Group, ©2001-2024 [cit. 2024-05-07]. Dostupné z: `https://www.php.net/manual/en/preface.php`.
5. *Symfony - Wikipedia* [online]. San Francisco (CA): Wikimedia Foundation, 2024 [cit. 2024-05-07]. Dostupné z: `https://en.wikipedia.org/wiki/Symfony`.
6. *Symfony explained to a Developer* [online]. Symfony SAS, ©2004- [cit. 2024-05-08]. Dostupné z: `https://symfony.com/explained-to-a-developer`.
7. *Symfony - Bundles* [online]. Tutorials Point, ©2024 [cit. 2024-05-08]. Dostupné z: `https://www.tutorialspoint.com/symfony/symfony_bundles.htm`.
8. *The Bundle System* [online]. Symfony SAS, ©2004- [cit. 2024-03-06]. Dostupné z: `https://symfony.com/doc/6.4/bundles.html`.
9. *MariaDB in brief* [online]. MariaDB Foundation, ©2009-2024 [cit. 2024-05-08]. Dostupné z: `https://mariadb.org/en/`.
10. *Introduction - Doctrine Database Abstraction Layer (DBAL)* [online]. Doctrine, [2024] [cit. 2024-05-08]. Dostupné z: `https://www.doctrine-project.org/projects/doctrine-dbal/en/4.0/reference/introduction.html`.
11. *Introduction to Redis* [online]. Redis, [2023] [cit. 2024-05-08]. Dostupné z: `https://redis.io/docs/about/`.
12. *React* [online]. Meta Open Source, ©2024 [cit. 2024-05-08]. Dostupné z: `https://react.dev/`.
13. IMMUKUL. *ReactJS Virtual DOM* [online]. GeeksForGeeks, 2023 [cit. 2024-03-18]. Dostupné z: `https://www.geeksforgeeks.org/reactjs-virtual-dom/`.
14. *GraphQL — A query language for your API* [online]. The GraphQL Foundation, ©2024 [cit. 2024-03-18]. Dostupné z: `https://graphql.org/`.
15. HVÍZDAL, Filip. *Google Tag Manager – základní nastavení* [online]. MarketingPPC, 2024 [cit. 2024-04-24]. Dostupné z: `https://www.marketingppc.cz/marketing/google-tag-manager-nastaveni/`.

16. *The data layer — Tag Manager* [online]. Google, 2023 [cit. 2024-04-24]. Dostupné z: https://developers.google.com/tag-platform/tag-manager/datalayer.
17. *HTTP - Wikipedia* [online]. San Francisco (CA): Wikimedia Foundation, 2024 [cit. 2024-05-07]. Dostupné z: https://en.wikipedia.org/wiki/HTTP.
18. *An overview of HTTP - HTTP — MDN* [online]. Mozilla Corporation, 2024 [cit. 2024-05-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.
19. *What are cookies? — Cookies definition* [online]. Cloudflare, ©2024 [cit. 2024-05-07]. Dostupné z: https://www.cloudflare.com/learning/privacy/what-are-cookies/.
20. *HTML: HyperText Markup Language — MDN* [online]. Mozilla Corporation, 2024 [cit. 2024-05-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.
21. JACKSON-BARNES, Shannon. *SPA Vs. MPA Explained: Differences Between Single-Page Applications and Multi-Page Applications* [online]. Orient Software Development, 2023 [cit. 2024-05-08]. Dostupné z: https://www.orientsoftware.com/blog/spa-vs-mpa/.
22. *SPA vs MPA Applications: What Are the Differences?* [online]. Medium, 2022 [cit. 2024-05-08]. Dostupné z: https://medium.com/@theadkgroup/spa-vs-mpa-applications-what-are-the-differences-7dc004e62397.
23. UGWU, Raphael. *Pre-rendering your React app with react-snap* [online]. LogRocket, 2023 [cit. 2024-05-08]. Dostupné z: https://blog.logrocket.com/pre-rendering-react-app-react-snap/.
24. SHAD, Sherry. *SSR vs CSR: Server Side Rendering vs Client Side Rendering* [online]. Medium, 2023 [cit. 2024-05-08]. Dostupné z: https://medium.com/@sharareshaddev/ssr-vs-csr-server-side-rendering-vs-client-side-rendering-deb1d3855b77.
25. *HTTP Cache (Symfony Docs)* [online]. Symfony SAS, ©2004- [cit. 2024-03-27]. Dostupné z: https://symfony.com/doc/6.4/http_cache.html.
26. FIELDING, R.; NOTTINGHAM, M.; AND, J. Reschke. *HTTP Caching* [Internet Requests for Comments]. RFC Editor, 2022. STD, 98. RFC Editor. ISSN 2070-1721. Dostupné také z: https://www.rfc-editor.org/info/rfc9111.
27. *HTTP caching — MDN* [online]. Mozilla Corporation, 2023 [cit. 2024-05-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching.
28. *What Is Edge Side Includes (ESI)?* [online]. KeyCDN, 2018 [cit. 2024-04-24]. Dostupné z: https://www.keycdn.com/support/edge-side-includes.
29. TSIMELZON, Mark; WEIHL, Bill; CHUNG, Joseph; FRANTZ, Dan; BASSO, John; NEWTON, Chris; HALE, Mark; JACOBS, Larry; O'CONNELL, Conleth. *ESI Language Specification 1.0*. 2001-08. Tech. zpr. W3C. Dostupné také z: http://www.w3.org/TR/2001/NOTE-esi-lang-20010804.
30. *Reverse proxy* [online]. San Francisco (CA): Wikimedia Foundation, 2024 [cit. 2024-05-08]. Dostupné z: https://en.wikipedia.org/wiki/Reverse_proxy%5C#cite_note-apache-forward-reverse-1.
31. *Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX* [online]. F5, ©2024 [cit. 2024-03-15]. Dostupné z: https://nginx.org/en/.
32. *F5 NGINX Plus: Compare NGINX Models* [online]. F5, ©2024 [cit. 2024-03-15]. Dostupné z: https://www.nginx.com/products/nginx/compare-models/.
33. *Beginner's Guide* [online]. F5, ©2024 [cit. 2024-03-15]. Dostupné z: http://nginx.org/en/docs/beginners_guide.html.
34. FERYN, Thijs. *Varnish 6 by example* [online]. Varnish Software AB, 2021 [cit. 2024-05-08]. ISBN 978-9189179974. Dostupné z: https://info.varnish-software.com/resources/varnish-6-by-example-book.

35. *What is a CDN? - Content Delivery Network Explained* [online]. Amazon Web Services, ©2024 [cit. 2024-04-25]. Dostupné z: https://aws.amazon.com/what-is/cdn/.

36. *Reverse proxy* [online]. San Francisco (CA): Wikimedia Foundation, 2024 [cit. 2024-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Content_delivery_network.

37. *Bunny.net - The Content Delivery platform that truly Hops!* [online]. BunnyWay, ©2024 [cit. 2024-03-16]. Dostupné z: https://bunny.net/.

38. SVISNUS000. *How to clear cache memory using JavaScript?* [online]. GeeksForGeeks, 2023 [cit. 2024-04-26]. Dostupné z: https://www.geeksforgeeks.org/how-to-clear-cache-memory-using-javascript/.

39. MERTZ, Jhonny; NUNES, Ingrid. Understanding Application-Level Caching in Web Applications: A Comprehensive Introduction and Survey of State-of-the-Art Approaches. *ACM Computing Surveys* [online]. 2017, roč. 50, č. 6, s. 34 [cit. 2024-05-08]. ISSN 0360-0300. Dostupné z DOI: 10.1145/3145813.

40. *Using HTTP cookies - HTTP — MDN* [online]. Mozilla Corporation, 2024 [cit. 2024-03-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

41. *Web Storage API - Web APIs — MDN* [online]. Mozilla Corporation, 2024 [cit. 2024-03-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API.

42. *IndexedDB API - Web APIs — MDN* [online]. Mozilla Corporation, 2024 [cit. 2024-03-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.

43. *Accept-Language used for locale setting* [online]. W3C, 2003 [cit. 2024-04-11]. Dostupné z: https://www.w3.org/International/questions/qa-accept-lang-locales.

44. *Built-in Symfony Events* [online]. Symfony SAS, ©2004- [cit. 2024-04-18]. Dostupné z: https://symfony.com/doc/6.4/reference/events.html.

45. *Queries and Mutations — GraphQL* [online]. The GraphQL Foundation, ©2024 [cit. 2024-04-27]. Dostupné z: https://graphql.org/learn/queries/.

46. *Your First Component – React* [online]. Meta Open Source, ©2024 [cit. 2024-04-21]. Dostupné z: https://react.dev/learn/your-first-component.

47. *Window: storage event - Web APIs — MDN* [online]. Mozilla Corporation, 2024 [cit. 2024-05-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window/storage_event.

48. *How to create your first Pull Zone* [online]. Bunny.net Support Hub, ©2024 [cit. 2024-05-08]. Dostupné z: https://support.bunny.net/hc/en-us/articles/207790269-How-to-create-your-first-Pull-Zone.

Obsah přiloženého média

- `README.md` ..... popis obsahu média
- `src`
  - `symfony` ..... upravené částí jádra WPJshopu
    - `README.txt` ..... popis souborů v adresáři
    - ⋮
  - `nginx` ..... implementace reverzního proxy serveru pro lokální vývoj
    - `docker-compose.yml`
    - `nginxTools.sh` ..... skript pro spuštění
    - `README.txt` ..... popis aplikace a spuštění pro lokální vývoj
    - ⋮
  - `front-end` ..... implementace front-end části v Reactu
    - `webpack.config.js`
    - `README.txt` ..... popis aplikace
    - ⋮
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - ⋮
- `thesis.pdf` ..... text práce ve formátu PDF