FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Webová aplikace pro správu osobních financí

Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie

Autor práce: **Tomáš Novotný**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.
Ústav nových technologií a aplikované informatiky

Liberec 2023

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Webová aplikace pro správu osobních financí

Jméno a příjmení:	**Tomáš Novotný**
Osobní číslo:	M18000088
Studijní program:	B2646 Informační technologie
Studijní obor:	Informační technologie
Zadávající katedra:	Ústav nových technologií a aplikované informatiky
Akademický rok:	2021/2022

Zásady pro vypracování:

1. Seznamte se s problematikou tvorby responzivních webových aplikací, včetně aktuálních možností pro implementaci pravidel výměny dat mezi klientem a serverem.
2. Na základě získaných znalostí navrhněte webovou aplikaci pro správu osobních financí. V rámci návrhu porovnejte možnosti REST API a GraphQL a zvolte, která z technologií bude vhodnější pro implementaci v rámci výsledné aplikace.
3. Vytvořený návrh prakticky implementujte, a vytvořte responzivní webovou aplikaci pro správu osobních financí. Řešení zdokumentujte a doplňte o automatizované testy.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] HOQUE, Shama. Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js. 2nd Edition. Packt Publishing, 2020. ISBN 978-1839215414.

[2] MDN JavaScript [online]. Mozilla foundation [cit. 2021-10-1]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

Vedoucí práce:	Mgr. Jiří Vraný, Ph.D. Ústav nových technologií a aplikované informatiky
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	22. května 2023

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2021

Prohlášení

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

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

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

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

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

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

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

17. května 2023 Tomáš Novotný

Webová aplikace pro správu osobních financí

Abstrakt

Tato bakalářská práce se zabývá tvorbou webové aplikace pro správu osobních financí, která má pomoci mladým párům a rodinám hospodařit se svými finančními prostředky. Cílem práce je zjistit, jaké jsou možnosti výměny dat mezi klientem a serverem, a vybrat nejvhodnější způsob komunikace. Autor popisuje aktuální situaci na trhu a následně navrhuje vlastní aplikaci, kterou uživatelé ocení pro její jednoduchost a pohodlnost při používání jak na počítači, tak i na mobilních zařízeních. Autor definuje požadavky na aplikaci. Klient je napsán pomocí frameworku React, serverová část s využitím Node.js s Express.js a databáze MongoDB. Pro komunikaci mezi frontendem a backendem autor vybral dotazovací jazyk GraphQL. V závěru práce je popsáno testování vybraných částí aplikace unit a integračními testy. Výsledná webová aplikace je veřejně dostupná na internetu.

Klíčová slova: Node.js, GraphQL, React, MongoDB, webová aplikace, správa financí, finanční gramotnost

Web application for personal finance management

Abstract

This bachelor thesis deals with the development of a web-based personal finance management application to help young couples and families manage their finances. The aim of the thesis is to find out what are the possibilities of data exchange between the client and the server, and to choose the most suitable way of communication. The author describes the current situation on the market and then proposes his own application, which users will appreciate for its simplicity and convenience when using it both on computers and mobile devices. The author defines the requirements for the application. The client is written using the React.js framework, the server side using Node.js with Express.js and the MongoDB database. For communication between the frontend and backend the author has chosen GraphQL query language. The thesis concludes with unit and integration tests for selected parts of the application. The resulting web application is publicly available on the Internet.

Keywords: Node.js, GraphQL, React, MongoDB, web application, financial management, financial literacy

Poděkování

Rád bych poděkoval svému vedoucímu práce, panu doktoru Jiřímu Vranému, za veškeré rady a čas, který mi věnoval, a také že umožnil vznik této práce.

Obsah

Seznam zkratek

API	Application Programming Interface
BSON	Binary JavaScript Object Notation
CORS	Cross-Origin Resource Sharing
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSX	JavaScript Syntax Extension
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
NoSQL	Not only Structured Query Language
ODM	Object Document Modeling
ORM	Object Relational Modeling
REST	Representational state transfer
SDL	Schema Definition Language
SQL	Structured Query Language
SSL	Secure Socket Layer
UI	User interface
URL	Uniform Resource Locator
XML	Extensible Markup Language

1 Úvod

Tato bakalářská práce se zabývá vytvořením responzivní webové aplikace pro správu osobních financí v jedné domácnosti, kterou využijí především mladé páry a rodiny řešící problematiku finanční gramotnosti a hospodaření se svými i společnými penězi. Aplikace má za úkol shromažďovat informace zadané uživatelem, konkrétně údaje o příjmech a výdajích, a následně je přehledně zobrazit. Uživatel tak jasně uvidí, jaká je jeho finanční situace. Zároveň mu aplikace nabídne vývoj financí za poslední měsíce a také je zde systém správy plateb za druhého neboli „dlužník“, který by měl lidem v páru usnadnit rozhodování, kdo bude hradit další výdaj.

V první, rešeršní části práce jsou popsány aktuálně nejstahovanější mobilní aplikace na správu financí, které jsou dostupné na trhu. Dále je pozornost soustředěna na představení nové aplikace a jejích funkčních, uživatelských a technologických požadavků. Následně autor navrhuje způsob, jakým bude komunikovat serverová část s klientskou a z nových poznatků vybírá nejvhodnější technologii. Stejně tak řeší, jak bude komunikace probíhat mezi serverem a databází. Konec této části je věnován jednotlivým použitým technologiím.

V další části práce je popsán proces implementace databáze včetně jejího schématu. Poté je zmíněna serverovou část, která vysvětluje, jak je tato část naprogramována, jaké obsahuje backend schéma, resolvery a middleware. Podobně je vysvětlena klientská část, která popisuje, jak fungují jednotlivé komponenty a kontext frontendu. Zároveň je zde popsán princip nasazení aplikace včetně kódu na jednotlivé služby (hosting) včetně uvedení URL adresy. v neposední řadě autor navrhuje další rozšíření, která by aplikaci ještě více přizpůsobila potřebám uživatele a mohou být předmětem dalšího vývoje.

Na závěr je aplikace podrobena unit a integračním testům a celý tento proces je zdokumentován, aby se zamezilo případným problémům s funkčností.

2 Rešerše

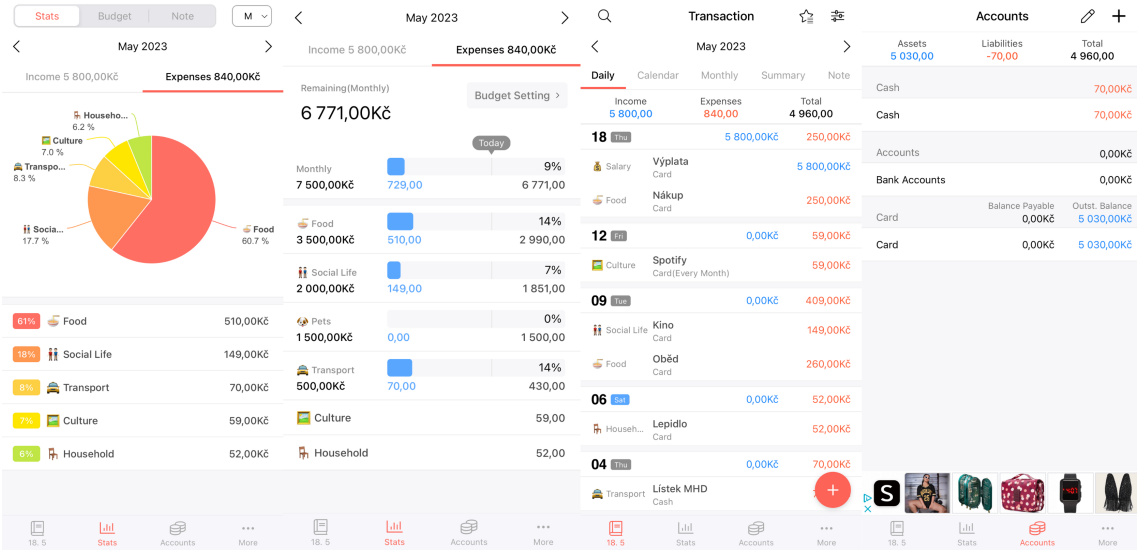
Společnost se dlouhodobě potýká s problémem nedostatečné finanční gramotnosti, jejíž ovládání není samozřejmostí pro každého. Zcela zásadní je u mladých lidí, kteří by měli umět hospodařit se svými penězi co možná nejdříve, a vyhnuli se tak finančním problémům v budoucnosti. Pomoci se jim snaží různé aplikace, které jim umožňují přehledně sledovat své finance. V těchto aplikacích může uživatel zaznamenávat svoje příjmy a výdaje, kategorizovat je a mít přehled, za co utrácí nejvíce peněz a kde je možné ušetřit. Aplikace procházejí neustálým vývojem a přizpůsobují se novým trendům a dnešní moderní době.

2.1 Existující aplikace

Tato část práce se zaměří na popis fungování a specifik již existujících aplikací, které jsou dostupné na trhu. Pro tyto účely byly vybrány tři nejstahovanější a nejpoužívanější aplikace, jež jsou dostupné na Google Play (platforma pro stahování aplikací pro zařízení s operačním systémem Android) a App Store (pro zařízení s operačním systémem iOS).

2.1.1 Money Manager Expense & Budget

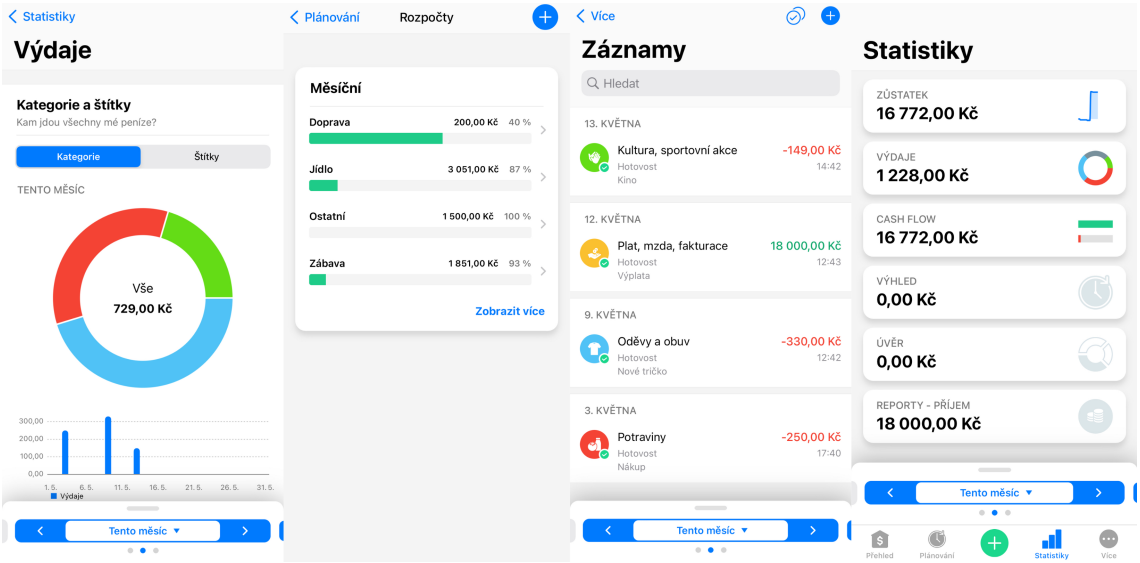
Jedná se o nejpopulárnější aplikaci na Google Play, kde má přes deset milionů stažení a celkové hodnocení 4,6/5. Na App Store má hodnocení 4,7/5, počet stažení Apple neuvádí. Aplikace nabízí standardní možnosti: uživatelé mohou zadávat výdaje a příjmy, kategorizovat je, a dokonce určit, jakým způsobem byla platba provedena (hotovost, platební karta, bankovní převod). Na hlavní stránce se nachází přehled všech záznamů, které lze filtrovat podle měsíce. Dále je zde možnost zobrazit údaje v kalendáři nebo souhrnně po měsících. Nechybí možnost zobrazit poznámky. V další záložce *Stats* si uživateli zobrazí rozpis kategorií spolu s nastaveným měsíčním limitem a aktuální situací. Najdeme zde také měsíční rozdělení kategorií v grafu. V poslední záložce *Accounts* vidíme přehled všech používaných účtů a celkové souhrny financí. Aplikace sice není v češtině, ale nabízí českou měnu. Je zde možnost upravovat předpřipravené kategorie, zálohovat data na iCloud nebo je exportovat do Excelu či poslat na e-mail. V aplikaci se nacházejí reklamy, které lze vypnout po zaplacení premium verze. Pro běžné používání není nutná registrace.



Obrázek 2.1: Prostředí aplikace Money Manager Expense & Budget

2.1.2 Wallet – rozpočty, správa peněz

Mobilní aplikace má na Google Play přes pět milionů stažení. Hodnocení 4,5/5 je stejné v obou obchodech. Aplikace nabízí totožné standardní funkce (viz Money Manager Expense & Budget. Na hlavní stránce uživatel najde celkový zůstatek financí, sečtené výdaje a cash flow za vybrané období. Výdaje si může zobrazit jak v přehledném grafu, tak i podle jednotlivých kategorií s nastaveným měsíčním limitem. Nechybí možnost zobrazení všech záznamů a nastavení plánovaných plateb. Oproti výše uvedené aplikaci se jich na obrazovku vejde téměř o polovinu méně. Aplikace je v češtině, podporuje českou měnu a je zdarma. V placené verzi je možné vlastnit neomezené množství účtů, připojit svůj bankovní účet, mít tak automaticky propojené záznamy, automatickou kategorizaci a možnost sdílet a spravovat finance s rodinou nebo přáteli. Wallet umožňuje využít webový prohlížeč, který nabízí stejné funkce jako na telefonu. Pro používání je nutné se zaregistrovat.



Obrázek 2.2: Prostředí aplikace Wallet - rozpočty, správa peněz

2.1.3 Správa financí – výdajů, peněz

Aplikace je k dispozici pouze na Google Play, má přes pět milionů stažení a hodnocení 4,9/5. Základní funkce jsou stejné jako u předchozích aplikací. U zadávání výdajů a příjmů vyžaduje pouze částku, kategorii, datum a popis. Lze přidat i fotografii. Přehledy jsou jednodušší, ale stále zobrazují nejdůležitější data. Aplikace je v češtině, podporuje české koruny a je zdarma bez reklam.

Obrázek 2.3: Prostředí aplikace Správa financí - výdajů, peněz

2.2 Motivace pro vytvoření nové aplikace

Hlavní motivací k vytvoření nové aplikace je pomoc mladým párům, které žijí v jedné domácnosti, aby přehledně viděly, jak nakládají se svými penězi, a naučily se tak lépe hospodařit s financemi. Již existující aplikace popsané v kapitole 2.1 dokazují, že pokud je aplikace jednodušší, lidé ji využívají v obdobné míře jako jiné, komplikovanější, a dokonce ji lépe hodnotí. Tímto směrem bude proto vedena i tato práce. Zadávání by mělo být co nejjednodušší a mělo by zahrnovat jen nejnutnější informace. Z přehledných grafů uživatelé snadno vyčtou, kolik peněz jim zbývá v daném období a kolik peněz si mohou uložit stranou, a tvořit si tak finanční rezervu. Zároveň jim aplikace usnadní proces řešení společných výdajů a jejich rozdělení, což je její hlavní výhoda.

2.3 Požadavky na aplikaci

2.3.1 Funkční požadavky

Uživatelé budou mít možnost vytvořit si vlastní účet pomocí e-mailu a hesla a přihlásit se do aplikace. Dále budou moci založit a spravovat společnou domácnost (jedna na uživatele), ve které žijí, a přidávat členy. Aplikace jim poskytne přehled jejich finanční situace pomocí přehledných grafů a statistik, které zobrazí zbývající peníze za určité časové období, výdaje, příjmy a úspory. Pro větší přehlednost budou jednotlivé záznamy kategorizovány a tříděny. Kromě toho aplikace umožní uživatelům přidávat výdaje do tzv. „dlužníku“ (správce společných výdajů), ve kterém bude přehledně vidět, který z členů zaplatil za dané období více či méně a kolik si navzájem dluží.

2.3.2 Uživatelské rozhraní

V rámci požadavků na uživatelské rozhraní aplikace bude kladen důraz na jeho jednoduchost, intuitivnost a přehlednost, aby bylo snadno ovladatelné pro uživatele bez vyšších technických dovedností. To znamená, že uživatelé budou mít snadný přístup ke všem funkcím a možnostem aplikace bez složitého učení. Rozhraní bude navrženo tak, aby uživatelé mohli snadno a rychle najít potřebné informace a provádět požadované akce. Budou využívány intuitivní ikony, jednoduché menu a přehledné rozvržení obrazovky. Dalším požadavkem na uživatelské rozhraní je jeho responzivita, což znamená, že aplikace bude optimalizována pro použití na různých zařízeních, a to včetně počítačů, tabletů a mobilních telefonů. Uživatelé budou mít možnost pohodlně a efektivně používat aplikaci na jakémkoli zařízení dle svých preferencí a potřeb.

2.3.3 Technologické požadavky

Serverová část aplikace (backend) bude implementována pomocí robustního a škálovatelného programovacího jazyka Node.js, který umožní efektivní zpracování požadavků a snadnou údržbu. [1] Server bude propojen s databází MongoDB pomocí knihovny *mongoose* (více v kapitole 2.6.5 a 2.5.1). Pro komunikaci mezi klientem a serverem bude použit protokol HTTPS, který zajistí šifrovaný přenos dat. Backend bude hostován na cloudové službě Vercel.com (více v kapitole 2.6.6 na straně 20).

Klientská část (frontend) bude implementována pomocí knihovny *React*, která nabízí rychlou a interaktivní tvorbu uživatelského rozhraní. Stejně jako server bude hostován na Vercel.com. *React* umožňuje vytvářet uživatelská rozhraní z jednotlivých částí nazývaných komponenty, a díky tomu je snadno udržovatelný, což usnadňuje vývoj webových aplikací. [2]

2.4 Komunikace mezi serverem a klientem

V této kapitole budou porovnány technologie REST API (Representational state transfer Application Programming Interface) a GraphQL (Graph Query Language) a ze srovnání se následně vybere vhodnější aplikace pro implementaci.

2.4.1 REST API

REST API je rozhraní využívající REST architekturu, která definuje způsoby komunikace mezi klientem a serverem či dvěma platformami. Poprvé bylo představeno v roce 2000 počítačovým vědcem doktorem Royem Fieldingem, který ve své práci vývojářům popisuje relativně vysokou úroveň flexibility a svobody. Komunikace obvykle probíhá pomocí protokolu HTTP (Hypertext Transfer Protocol), jenž využívá metody POST (vytvoření dat), GET (získání dat), PUT (změna) a DELETE (smazání). Metody reprezentují základní operace CRUD (Create, Read, Update, Delete), které lze provádět nad daty (zdroji), jež jsou na serveru k dispozici. Každý zdroj má definované unikátní URL adresy neboli endpointy, na které se klient dotazuje. Nejčastěji dostaneme odpověď ve formátu JSON nebo XML. [3]

REST API je bezstavové, požadavky na server jsou tedy nezávislé. To znamená, že si neukládá předchozí požadavky a každý z nich je posuzován samostatně. Díky tomu je škálovatelný, což je pozitivní vlastnost, s níž by měl zvládat větší nápor uživatelů, resp. požadavků. REST si ukládá dočasná data (cache) do mezipaměti, aby dokázal rychleji odpovědět při větším objemu dat nebo při opakovaném volání. [4]

2.4.2 GraphQL

GraphQL je dotazovací jazyk pro vytváření API, který není vázán na konkrétní databázi. GraphQL byl vytvořen v roce 2012 v souvislosti s vývojem aplikace Facebook. V roce 2015 byl uvolněn jako open-source (otevřené programové vybavení) s veřejně dostupnými zdrojovými kódy. Tímto krokem se značně rozšířil. Autorem je tedy firma Meta Platforms (dříve Facebook). V roce 2019 právě Facebook a spousta dalších firem (např. Apollo, AWS, Gatsby, IBM a další) založily GraphQL Foundation, aby vytvořily neutrální domov pro ochrannou známku GraphQL a prosadily ho jako průmyslovou specifikaci pro navrhování efektivnějších API. [5]

Komunikace mezi frontendem a backendem probíhá pouze pomocí jednoho endpointu. Klient nejčastěji odesílá HTTP požadavek typu POST. Na straně serveru jsou definována schémata a resolvery. Schéma popisuje všechny dostupné Types (typy), se kterými následně pracují Queries (dotazy) a Mutations (mutace). Každé Query specifikuje vstupní data a také data, která bude API vracet (v REST technologii metoda GET). Mutations definují, jak se data budou ukládat, měnit nebo mazat (POST, PUT, DELETE). Schéma tedy zahrnuje pravidla, podle kterých se klient bude řídit, aby dostal požadovaná data. Resolver obsahuje samotnou logiku, podle které vrátí uživateli požadovaná data ve formátu JSON.

Výhodou je, že se schéma může jednoduše a flexibilně rozšířit o další dotazy a mutace podle potřeby, aniž by to nějak ovlivnilo frontend. Klient zároveň vždy ví, jaké dotazy může pokládat a která data může získat, a to vše pomocí jednoho koncového bodu. Typový systém API propustí pouze správně definované dotazy, v případě chyby tak nezatěžuje server následným zpracováním. Další výhodou je, že si frontend může přesně definovat strukturu o informacích, které potřebuje. Nepřenáší se tedy všechna data, díky čemuž může být odpověď serveru rychlejší. Jeden dotaz se dá použít na více místech aplikace, ale pokaždé může vracet jiná data. GraphQL umí vhodně nahradit odkazy na další záznamy, čímž vytvoří pomocí jednoho dotahu komplexní vnořený seznam, u kterého uživatel přesně definuje, která data požaduje nazpět. [4]

Značnou nevýhodou může být náročná implementace, která vyžaduje popsaní všech možných scénářů pro práci s daty. S tím je spojen fakt, že pro získání veškerých dat je nutné definovat všechny parametry na straně klienta. Náročnější, dlouhé dotazy mohou navyšovat požadavky na server. GraphQL má také problémy s cashováním, kdy musí data pokaždé stáhnout znovu, což může být nevhodné pro větší projekty, na kterých se podílí větší množství uživatelů. GraphQL vrací pouze chybový stav HTTP 400 Bad Request. Ostatní chybové hlášky jsou součástí JSON odpovědi. [6]

2.4.3 Výběr technologie

Obecně nelze říct, která z výše uvedených technologií je lepší. Obě mohou být vhodné při řešení konkrétních případů. Pro účely této práce byla zvolena technologie GraphQL, a to z následujících důvodů. Velkou výhodou je pouze jeden koncový bod, na který se klient dotazuje. Na straně serveru budou jasně definovaná schémata. Zároveň si klient bude moci přesně určit, která data potřebuje, a nebude tak docházet k posílání nepotřebných informací. Vzhledem k tomu, že má aplikace rozsáhlé schéma, bude mnohem jednodušší získat data, která jsou vnořená v jednotlivých strukturách. To vše bude probíhat za pomoci jednoho dotazu.

Názorný příklad: Máme domácnost, která řeší stav financí v několika obdobích a každé období obsahuje desítky záznamů o příjmech a výdajích, zároveň v sobě tato domácnost drží informaci tzv. „dlužníku“ (kdo za koho kolik peněz zaplatil, a tudíž mu druhý peníze dluží). Pokud budeme potřebovat zobrazit všechna tato data, použijeme pouze jeden dotaz. Výpis si klient jednoduše změní, pokud bude potřebovat vypsat pouze názvy jednotlivých období. REST API by muselo provést několik dotazů – domácnost, období, v každém období jednotlivé příjmy a výdaje a nakonec dlužník.

2.5 Komunikace s databází

Server s databází mohou komunikovat na přímo (bez nutnosti použití modulů), ale ve většině případů se využívají tzv. ODM (Object Document Modeling) nebo ORM (Object Relational Modeling) knihovny. Již podle názvu lze říct, že ORM mapuje objekty aplikace na tabulky v relační databázi. Není potřeba používat SQL dotazy, protože se právě provádějí ve stejném jazyce, jako je serverová část. Pro Node.js jsou známé např. Sequelize nebo Bookshelf. Tato aplikace však využívá ODM, protože slouží pro nerelační (NoSQL) databáze, jako je např. MongoDB. Objektový model se mapuje na formát podobný JSON nebo BSON. [7]

2.5.1 Mongoose

Mongoose je knihovna založená na ODM pro Node.js a databázi MongoDB. Umožňuje vývojářům vytvořit schéma na úrovni aplikace. Kromě něj poskytuje validátory, díky kterým lze synchronně nebo asynchronně ověřovat příchozí data (schéma) již na aplikační vrstvě. Výhodou je automatické vytváření *Timestamps* neboli časových značek vytvoření a změny dokumentu. Další výhodou je jednoduché propojení referencí mezi dokumenty v různých kolekcích. Metoda *populate()* nahrazuje referenční ID (*ObjectID*) příslušným záznamem v databázi. Takto lze efektivně získávat data z databáze bez nutnosti ručního propojování kolekcí. Metoda *save()* automaticky vloží záznam do databáze a ve chvíli, kdy jsou již data v databázi uložena, tak provede aktualizaci dat včetně časových značek. Kvůli tomuto zjednodušení zpracování dat byl Mongoose pro tuto práci vybrán místo přímé komunikace bez knihovny. [8]

2.6 Použité technologie

2.6.1 Node.js

Node.js je open-source, multiplatformní, událostmi řízené prostředí, které je navrženo k vytváření škálovatelných aplikací. Node.js je založen na programovacím jazyce JavaScript. Díky Node.js lze použít Javascript kromě klientské části i v části serverové. Hlavními výhodami použití je rychlé zpracování požadavků, jednoduché vytvoření API, široká nabídka knihoven a rozšíření a multiplatformnost, kdy lze kód použít na různých operačních systémech. Node.js byl použit pro tvorbu serverové části této bakalářské práce.[1]

2.6.2 Express.js

Express.js zkráceně Express je open-source framework, který používá Node.js. Výhodou je zjednodušení vytváření API a webových aplikací a lze říct, že díky ní programátor napíše méně kódu, než kdyby framework nepoužíval. Express je *unopinionated* nebo nedogmatický, což znamená, že nenabízí žádné postupy a vylepšení jako nejlepší, ale je na vývojáři, jaké komponenty a postupy zvolí a jak chce kód napsat. Express byl použit pro tvorbu serverové části spolu s Node.js. [9]

2.6.3 React

React je knihovna pro webová uživatelská rozhraní. Tato rozhraní knihovna umožňuje vytvářet pomocí jednotlivých komponent, které se následně využívají k zobrazení stránek. Jejich výhodou je nezávislost, což usnadňuje práci ve větších skupinách vývojářů. React dále nabízí jednoduchou správu stavu aplikace pomocí *props*, které se mohou předávat mezi komponentami. Další výhodou je virtuální DOM. To znamená, že React dokáže do již vygenerované HTML stránky vložit nové komponenty a prvky, aniž by se stránka musela celá vykreslit znovu. React podporuje JSX, díky kterému lze psát HTML a CSS kód přímo v Javascriptu, což zjednodušuje tvorbu UI. React byl použit pro tvorbu klientské části této bakalářské práce. [2]

2.6.4 GraphQL

GraphQL je dotazovací jazyk na API. Klient dostane vždy data, která si přesně definoval, což urychlí komunikaci, protože nebudou přenášena nepotřebná data. GraphQL má pouze jeden endpoint, neboli koncový bod, na který posílají klienti své dotazy. Rozhraní používá typy k přesnému definování toho, co lze pomocí dotazu získat. [10]

2.6.5 MongoDB

MongoDB je open-source dokumentově orientovaná databáze typu NoSQL (nerelační databáze). Na rozdíl od relačních databází využívá místo tabulek dokumen-

ty a pole. Data se ukládají ve formátu JSON, resp. BSON (binární reprezentace JSON objektů). Díky použití dokumentů lze jednoduše pracovat s nestrukturovanými daty a zároveň snadno vytvářet dotazy. MongoDB bylo použito jako databáze této bakalářské práce. Databáze komunikuje s aplikací pomocí knihovny Mongoose (viz kapitola 2.5.1). [11] [12]

2.6.6 Vercel

Vercel.com je cloudová platforma specializující se na rychlé a jednoduché nasazení webových aplikací. Optimalizací výkonu poskytuje uživatelům možnost vytvářet a hostovat webové aplikace efektivně a spolehlivě. Integrace s Git repozitáři (GitHub, GitLab a Bitbucket) umožňuje automatizované nasazení změn. Díky podpoře vlastních domén s SSL (Secure Socket Layer) certifikáty nabízí zabezpečené spojení. Tyto certifikáty se využívají při vytváření HTTPS (Hypertext Transfer Protocol Secure) spojení. Pro účely této práce postačuje bezplatná verze. [13]

3 Implementace aplikace

3.1 Databáze

V rámci implementace je využívána dokumentově orientovaná NoSQL databáze MongoDB (více v kapitole 2.6.5). Pro komunikaci mezi serverem a databází je zvolená knihovna Mongoose, která slouží jako ODM nástroj pro MongoDB v prostředí Node.js (kapitola 2.5.1).


```
_id: ObjectId('61de8798e92eca1b2da106ed')
email: "pepazdepa@gmail.com"
name: "Pepa"
surname: "z depa"
password: "$2a$12$3V6V6P2JsNT.RbPfNM/tT.KPUCjPiqo2lLDhZvunmAFDldKtSgiCu"
color: "blue"
▸ households: Array
createdAt: 2022-01-12T07:47:36.691+00:00
updatedAt: 2022-01-12T07:47:36.691+00:00
__v: 0
```


Obrázek 3.1: Ukázka uloženého objektu - uživatele v databázi

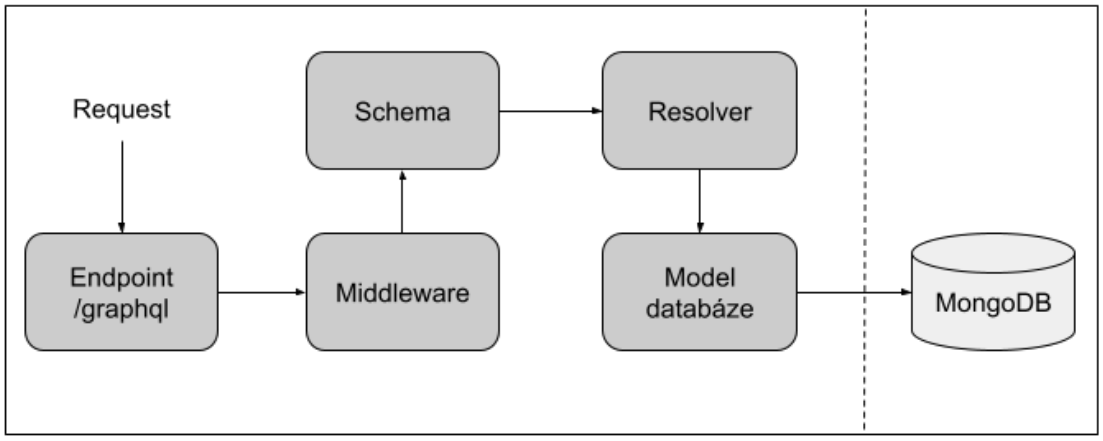
3.2 Server

V této kapitole bude zaměřena pozornost na serverovou část aplikace. Pro její vývoj byl použit Node.js spolu s frameworkem Express.js (viz kapitola 2.6.2). Hlavní spouštěcí soubor je index.js. Na začátku jsou importovány používané moduly: *express*, *bodyParser*, *mongoose*, *routerHandler* a *isAuthenticated*. Jako první je vytvořena instance aplikace pomocí modulu express, na který navazuje middleware, který ověřuje, zda hlavička dotazu obsahuje ověřovací token, a tedy jestli je uživatel přihlášen (více v kapitole 3.2.4). Na to navazuje nastavení hlaviček pro CORS (Cross-Origin Resource Sharing), což je technika, která využívá hlavičky protokolu HTTP a umožňuje bezpečnou komunikaci mezi webovými aplikacemi na různých doménách. Zde je nastavena adresa klienta (podrobnosti v kapitole 3.3.4), metody, které lze využít (POST, GET, OPTIONS), a obsah hlavičky (content-type a Authorization). Poté je nastavena cesta "/", kterou komponenta *routerHandler* obsluhuje

všechny koncové body API, v tomto případě to je pouze */graphql*, pomocí kterého bude probíhat veškerá komunikace. V závěru už je pouze nastavení připojení databáze MongoDB pomocí *connection string*, což je řetězec, který obsahuje adresu serveru, port, jméno, databáze, uživatelské jméno a heslo. Tento string spolu s adresou klienta je uložen v konfiguračním souboru *.env*, který se nachází v kořenovém adresáři. Obsahuje vždy klíč a hodnotu. Tím jsou citlivé údaje odděleny od zbytku kódu, a je tak zajištěna jejich bezpečnost. Zároveň mohou být použity kdekoli na serveru, proto je vhodný i pro údaje, které se často se opakují (např. adresa databáze). Použití je vidět na obrázku 3.2, kde se nastavuje URL adresa klienta.

```
app.use((req, res, next) => {
    res.setHeader('Access-Control-Allow-Origin', process.env.ORIGIN_URL);
    res.setHeader('Access-Control-Allow-Methods', 'POST,GET,OPTIONS');
    res.setHeader('Access-Control-Allow-Headers', 'content-type,Authorization');
    if (req.method === "OPTIONS") {
        return res.sendStatus(200);
    }
    next();
});
```

Obrázek 3.2: Ukázka nastavení CORS hlaviček

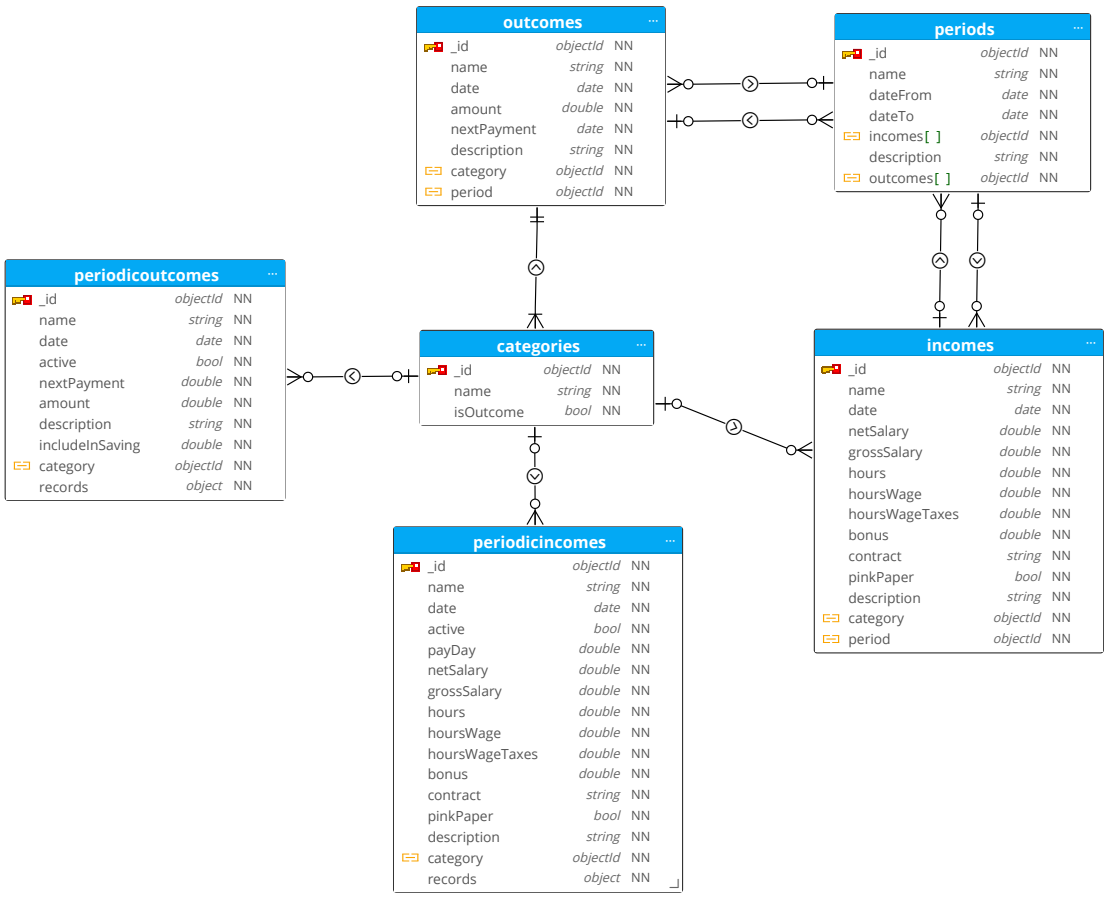


Obrázek 3.3: Schéma serveru

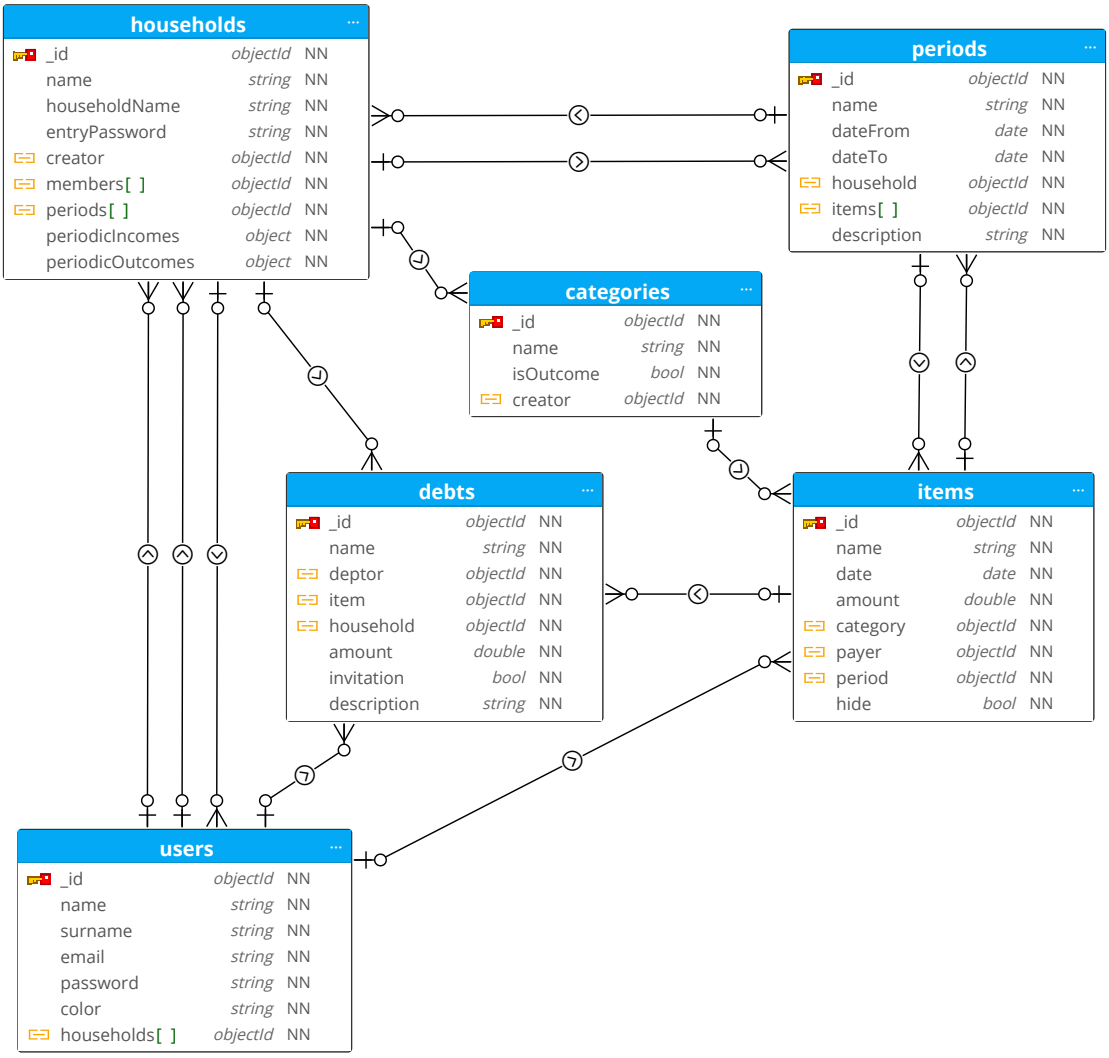
3.2.1 Model

Každý model aplikace je napsán pro větší přehlednost v jednom souboru ve složce models. Na začátku bude vždy načten modul Mongoose. Poté se vytvoří nové schéma, ve kterém se definují názvy proměnných, jejich datový typ, volitelnost, výchozí hodnota, reference a na závěr časové značky vytvoření a změny záznamu,

které Mongoose spravuje automaticky. Aplikace používá následující podporované datové typy: String (řetězec), Number (číslo), Date (datum a čas), Boolean (logický typ true nebo false), Array (pole hodnot), ObjectId (ID dokumentu v databázi) a Decimal128 (desetinná čísla).[14] Následně se ze schématu vytvoří model pomocí mongoose.model(), kde se specifikuje název modelu a výše definované schéma. Na závěr se modul exportuje.



Obrázek 3.4: Model databáze 1. část



Obrázek 3.5: Model databáze 2. část

3.2.2 Schéma

Nachází se v adresáři `/graphql/schema` a je důležitou částí serveru, protože definuje strukturu API a typy dat, které bude moci klient získat a modifikovat. Vytváří se pomocí *buildSchema()* z knihovny *graphql*, která je definována na začátku souboru. Tato funkce přebírá jako parametr textový řetězec, který vytvoří GraphQL schéma pomocí speciálního jazyka SDL (Schema Definition Language). Tento jazyk popisuje typy, dotazy a mutace, které GraphQL využívá. Jako první je potřeba definovat typy, které jsou součástí API (*Household, User, Period, Item, Debt, Category, Income, Outcome, PeriodicIncome, PeriodicOutcome*). Model a schéma mají různé datové typy. Porovnání je v tabulce 3.1. V modelu nebylo potřeba definovat identifikátor, jelikož se vytvoří automaticky. Ve schématu je nutné, aby skrze něj bylo možné přistupovat k jednotlivým záznamům.

typ	model	schéma
řetězec	String, Date	String
číslo	Number/Decimal128	Int/Float
identifikátor	ObjectId	ID
logická hodnota	Boolean	Boolean
list/pole	[Type]	[Type]
povinné	required: true	Type!

Tabulka 3.1: Porovnání typů pro práci s modelem a schématem.

Pro každou kolekci v databázi je definován *input* typ, který slouží k seskupení a jednoduššímu strukturování vstupních dat. Ta se předávají při provádění mutace. Dále se již definují *queries* (dotazy), které určují, jaká data bude server vracet. Konkrétně půjde o *login* pro přihlášení uživatele, *Categories, Users, Items, Jobs, Debts, PeriodicIncomes a PeriodicOutcomes*. Aby se naplno využil potenciál GraphQL, tak query *periodsSelect* bude vracet údaje o všech obdobích a klient si vybere, která data bude po API požadovat. Na závěr je nutné definovat *mutations* (mutace), které určují, jaká data může uživatel měnit. Pro každý model server definuje tři mutace (create, edit, delete). Výjimku tvoří přidání/odebrání člena do/z domácnosti, kde chybí mutace pro změnu.

3.2.3 GraphQL Resolver

Ve složce `/graphql/resolvers` se pomocí *queries* v resolveru implementují jednotlivé dotazy a mutace a vrací se odpovídající struktury (viz kapitola 3.2.2). Po zavolání resolveru se data načtou z databáze a následně se pošlou klientovi. Na prvky, které obsahují *ObjectId*, se naváže konkrétní objekt v databázi pomocí funkce *bind()*. Tím bude dosaženo výhody GraphQL, kdy jeden dotaz bude obsahovat všechny informace a bude na volbě klienta, která data chce získat. V případě chyby na straně serveru je uživateli zaslána chybová hláška. Výše uvedené metody popisují, jak lze získat data ze serveru, resp. z databáze. Následující funkce definují, jak se dají data vytvářet, měnit nebo odstranit.

createUser

Na základě e-mailu a hesla funkce vytvoří uživatele a zapíše ho do databáze. Heslo je zašifrováno knihovnou *bcrypt* metodou *bcrypt.hash*. Klient obdrží vlastnosti uloženého uživatele (heslo je nahrazeno prázdným stringem). V případě, že je již e-mail zaregistrován, je vyhozena chybová hláška.

login

Vstupní parametry funkce tvoří e-mail a heslo, které je pomocí metody *bcrypt.compare* porovnáno s heslem uloženým v databázi. V případě shody je na základě uživatelského ID a e-mailu vytvořen token knihovnou *jsonwebtoken* (*jwt.sign*),

která slouží pro práci s JWT (JSON Web Tokens). Nastavení expirace tokenu jsou dvě hodiny. Klient obdrží uživatelské ID, e-mail, token, dobu platnosti tokenu a informaci o domácnosti, ve které se nachází. V případě zadání špatných údajů obdrží uživatel chybovou hlášku.

createHousehold

Tento resolver vytváří novou domácnost po zadání jejího názvu, který ještě nebyl použit, a hesla. To je následně zašifrováno (*bcrypt.hash*). Nová domácnost je vložena do databáze a zároveň je přiřazena uživateli. Výsledek uložené domácnosti je vrácen klientovi (bez hesla). Při použití stejného názvu je vyhozena chybová hláška.

addToHousehold

Funkce přidává uživatele do domácnosti. Vstupními argumenty jsou název domácnosti, ID uživatele a vstupní heslo. Po kontrole pomocí middlewaru (viz kapitola 3.2.4), zda je uživatel autentizován, je v databázi nalezena domácnost, do které se chce uživatel připojit. Pokud jsou hesla porovnaná metodou *bcrypt.compare* stejná, tak se uživatel zapíše do domácnosti a domácnost je k němu přiřazena. Klientovi je vrácena informace o domácnosti. V případě, že uživatel zadá nesprávné heslo, nebo je již zapsán ve stejné či jiné domácnosti, je vyvolán nový error.

removeFromHousehold a removeHousehold

Tento resolver není do aplikace implementován, protože pro smazání člena z domácnosti či odstranění samotné domácnosti je důležité, aby v ten moment neobsahovala žádná data, popř. aby byla data zálohována a připravena na přesun do nové domácnosti. Tato funkce je navržena v rámci dalšího rozšíření aplikace (viz kapitola 3.4).

create/edit/deletePeriod

Následující resolvery jsou určeny pro práci s obdobími v domácnosti. Během volání je vždy ověřeno, zda je uživatel přihlášen. Pro vytváření a editaci období jsou předepsány povinné vstupní parametry (viz schéma v kapitole 3.2.1.) Při úspěšném zapsaní do databáze je období následně vráceno jako potvrzení. Vstupním parametrem pro smazání je identifikátor období. Klientovi je vrácen logický typ *true*, pokud dojde k odstranění, nebo *false*, když odstranění neproběhne. Pokud období obsahuje jakékoli prvky (příjmy, výdaje), tak současně dojde ke smazání těchto prvků. V případě, že nastane chyba, kdy uživatel není přihlášen, období se stejným názvem již existuje či období nelze najít, je ze strany serveru vrácen error s příslušnou hláškou.

create/edit/deleteItem

Tyto funkce slouží pro práci s jednotlivými výdaji. Při volání je zkontrolováno, jestli je uživatel přihlášen. Po získání všech potřebných proměnných (viz kapitola

3.2.1) je výdaj zapsán či změněn. Do databáze se uloží samotný výdaj a také odkaz v podobě ID výdaje do pole *items* v daném období. Při úspěšném zapsání obdrží uživatel požadovaný výdaj. Pro odstranění je potřeba ID výdaje. Pokud se výdaj odstraní z databáze a jeho odkaz z pole v daném období, tak se klientovi vrátí *true*, jinak *false* nebo error s chybovou hláškou.

create/edit/deleteCategory

Resolvery jsou určeny pro práci s kategoriemi, které mohou sloužit pro příjmy nebo výdaje. Pro vytvoření a změnu je potřeba název, ID uživatele a informace, zda se jedná o kategorii pro výdaj. Klientovi je vrácena kategorie, pokud byla zdárně zapsána do databáze. Ke smazání může dojít pouze v případě, kdy není kategorie používána. Jestli je to pravda, tak server odpoví *true*. Tak jako u předchozích resolverů, i u kategorií se kontroluje, jestli je uživatel přihlášen a také jsou zde chybové hlášky, když není kategorie nalezena nebo kdy je její jméno již použito.

create/edit/deleteIncome

Resolvery obsluhují jednorázové příjmy uživatele. Pro vytvoření a změnu jsou potřeba určité parametry (viz kapitola 3.2.1). Opět se zde kontroluje, zda je uživatel autentifikován. Příjem je uložen v databázi v kolekci *incomes* a odkaz v podobě ID je uložen v kolekci *periods* v poli *incomes*. Uživateli se stejně jako v předchozích případech vrátí samotný příjem. Pokud nastane výjimka, je vyhozena příslušná chybová hláška.

create/edit/deleteOutcome

Resolvery, které slouží pro práci s výdaji, se chovají stejně jako příjmy (viz create/edit/deleteIncome).

create/edit/deletePeriodicIncome

Tyto funkce obsluhují pravidelné výdaje. Principiálně fungují stejně jako normální jednorázové výdaje s tím rozdílem, že se v databázi ukládají pouze do kolekce *periodicincomes*. Jednotlivé pravidelné příjmy se zařazují jako klasické příjmy a jejich historie je uložena v poli *records*. Kontrola přihlášení uživatele a chybové hlášky jsou stejné jako v předchozích resolverech. Jelikož se jedná o šablonu pro jednoduché vytvoření pravidelných příjmů, tak při odstranění zůstávají již vytvořené příjmy v databázi.

create/edit/deletePeriodicOutcome

Metody pro práci s pravidelnými výdaji fungují stejně jako výše uvedené create/edit/deletePeriodicIncome.

create/edit/deleteDebt

Tyto metody pracují s vytvářením, aktualizací a mazáním dluhů mezi uživateli. Pro zpracování jsou nutné vstupní parametry (viz kapitola 3.2.1). Při úspěšném zapsání do databáze server posílá dluh zpět klientovi. I zde se kontroluje, zda je uživatel přihlášen, a také zde nechybí chybové hlášky, které jsou v případě potřeby odeslány klientovi.

3.2.4 Middleware

Middleware je funkce, která se provádí mezi příchozími a odchozími požadavky aplikace, nachází se v `/middleware`. Slouží ke zpracování a modifikaci požadavků a objektů žádostí a odpovědí. Server používá jeden middleware *isAuthenticated*, a to pro ověření přihlášení uživatele. Pomocí knihovny *jsonwebtoken*, která slouží pro práci s JWT (JSON Web Tokens), se ověří správnost tokenu, který je vytvořen při přihlášení uživatele do aplikace. Ten se nachází v hlavičce požadavku a má podobu *Authorization: Bearer [token]*. Klíč se pomocí funkce *jwt.verify* a klíčového slova ověří a v případě pravosti nastaví vlastnost requestu *req.isAuthenticated* na *true*. Také se uloží uživatelské ID a e-mail *(req.userId, req.userEmail)*. Tento middleware se následně může využít v jakémkoli resolveru.

3.3 Klient

Tato kapitola se věnuje klientské části aplikace. Pro její vývoj byl použit React.js (viz kapitola 2.6.3). Hlavním spouštěcím souborem je index.js, zde se nachází jednoduchý kód, který vykresluje React elementy do DOM, který se následně vloží do hlavního divu v index.html. Hlavní komponentou je App.js, v níž se nacházejí metody, které se volají při přihlášení (nastaví se do globální proměnné token a uživatelské ID) a odhlášení (všechny uložené proměnné se smažou). Následně se vykreslí komponenta *Navigation* a k ní předdefinované *Routes* (cesty). Např. kam se má uživatel přesměrovat po přihlášení a které prvky navigace uvidí, pokud je, nebo není přihlášen. Nachází se zde také *ToastContainer*, který obsluhuje vyskakovací upozornění (pokud např. uživatel zadá špatné heslo).

3.3.1 Components

Components (komponenty) jsou v React aplikaci velmi důležité, jelikož pomáhají jednoduše strukturovat a organizovat kód do jednotlivých částí, které jsou nezávislé a znovupoužitelné. Obsahují logickou část a většinou vrací HTML (Hypertext Markup Language) segment. V této aplikaci jsou využívány dva typy: *class* a *function* komponenty.

Class komponenty jsou rozšířeny třídou *React.Component*. Obsahují metodu *render()*, ve které se definuje JSX (JavaScript XML) struktura komponenty. Dále mají vlastní stav (*state*) a mohou mít uvnitř sebe vlastní metody.

Function komponenty jsou funkce, které přijímají jako argument *props* a vrací metodu *return* JSX strukturu stejně jako *class* komponenty.

Queries

Dotazy na server jsou implementovány jako *function* komponenty, cesta `/src/components/Queries`. V každém souboru jsou definovány dotazy, které jsou potřebné pro komunikaci se serverem. Z pravidla se jedná o query pro získání dat (zde se definuje, jaká data jsou potřeba od serveru, a nemusí to být všechna, která server nabízí) a mutations pro vytvoření, změnu a smazání záznamů v databázi. Vstupním argumentem je objekt props, který obsahuje potřebné vlastnosti pro vytvoření dotazu. Výstupní hodnota *function* komponenty je již dotaz ve formátu, který odpovídá tělu dotazu v jazyce GraphQL, který se posílá na server.

Podobně jsou implementovány i *class* komponenty, které se také dotazují na server, ale navíc slouží k vytvoření seznamu kategorií, itemů nebo uživatelů. Tyto komponenty vrací JSX strukturu, která představuje část formuláře, jako například seznam tvořený pomocí HTML značky *<option>*. Aby mohl být kód vrácen, musí být zaobalen v jedné HTML značce a tou je *<React.Fragment>* pro snadné vložení do formuláře. Ve výsledku jsou tyto komponenty schopny zobrazit výchozí hodnotu, pokud je poskytnuta v objektu *props*.

Handlers

V rámci aplikace je využívána asynchronní *function* komponenta nazvaná *FetchHandler* (`/src/components/Handlers`), která slouží k odesílání dotazů na server a získávání dat. Tato komponenta přijímá jako argument *props*, které obsahují *token* a *requestBody* (které je získáno z komponent Queries v kapitole 3.3.1). V závislosti na existenci proměnné *token* je nastavována hlavička dotazu. Pokud je *token* k dispozici, do hlavičky je vložen *content-type* (typ obsahu) s hodnotou *application/json* a *Authorization* (autorizační token) ve formátu *Bearer [token]*. Pokud *token* není k dispozici, je nastavena pouze hodnota *content-type*. Dotaz je odeslán pomocí asynchronní funkce *fetch*. Informace přijaté ze serveru jsou uloženy do objektu *returnPackage*, který má dvě vlastnosti. Úspěšně získaná data jsou vložena do vlastnosti *data* a chybové hlášky do prázdného pole *errors*.

Modals

Tato část (`/src/components/Modals`) je zaměřena na práci s vyskakovacími dialogovými okny (*modals*), která slouží k interaktivnímu zadávání dat v rámci aplikace. Jedná se o *function* komponentu, jež jako argument přijímá *props*, díky kterým se dají nastavit následující vlastnosti: zobrazení a skrytí okna, titulek a samotné tělo, které obvykle představuje formulář pro zadávání dat. Díky konfigurovatelným tlačítkům lze definovat chování dialogového okna. Obvykle se používají dvě tlačítka – jedno slouží k potvrzení a odeslání dat a druhé k zavření okna. Můžeme nastavit,

zda se tlačítka zobrazí, nebo ne, jaký budou mít styl, text a jakou akci mají vykonat. Tímto způsobem lze přizpůsobit vzhled a chování vyskakovacího okna specifickým požadavkům aplikace a uživatelského rozhraní.

Forms

V aplikaci jsou implementovány *function* komponenty pro každý formulář určený k odesílání dat na server, cesta `/src/components/Forms`. Tyto komponenty přijímají objekt (argument) *props*, který obsahuje vlastnosti pro správu daného formuláře. Pro tvorbu a responzivní zobrazení formulářů je využívána komponenta *Form*, která je součástí knihovny *react-bootstrap*. Tímto je zajištěno správné zobrazení formuláře nejen na mobilních zařízeních a také je zde zaručen základní styl. Každý prvek formuláře je zabalen do skupiny *Form.Group*, která slouží k seskupení souvisejících prvků a zajistí správné zarovnání a stylizaci. Ve skupině je dále popisek (*Form.Label*) a příslušné pole *Form.Control*, které umožní zadávat hodnoty. Pokud jsou vlastnosti pro výchozí data definovány (v objektu *props*), jsou tato data automaticky použita pro předvyplnění odpovídajících polí formuláře. V opačném případě jsou pole prázdná a umožňují vytvoření nového prvku.

Functions

Klient využívá dvě *function* komponenty pro formátování dat podle potřeb uživatele. Nachází se v `/src/components/Functions` a konkrétně jde o *FormatDate*, který přijímá dva argumenty *date* a *format* a vrací formátované datum podle zadaných parametrů. Podle potřeby lze zvolit formát *dd.mm.yyyy* nebo *yyyy-mm-dd*. Při nezadání žádného z formátů je zvolen formát výchozí (*yyyy-mm-dd*). V případě, kdy je vstupní datum ve špatném formátu, je vráceno výchozí datum *1991-01-01*.

Druhá komponenta slouží k formátování čísel. Komponenta přijímá dva argumenty: *amount*, který představuje číslo, a *decimals*, ten určuje počet desetinných míst. Formát čísla je nastaven na český standard, kde se desetinná část odděluje čárkou.

3.3.2 Context

Context, česky kontext, ve složce `/src/context` umožňuje sdílet stav (vlastnosti) a funkce mezi komponentami bez nutnosti předávat tyto informace skrz *props*. Slouží tedy k usnadnění komunikace a ke sdílení dat mezi různými částmi aplikace. V aplikaci je definován *AuthContext*, který zahrnuje vlastnosti: *token* (pro autentizaci uživatele), *userId* a *householdId*. Také obsahuje funkce: *login* (pro aktualizaci tokenu, jeho expiraci, uživatelské ID a e-mail a ID domácnosti), *updateHouseholdId* (pro aktualizaci ID domácnosti v případě přihlášení) a na závěr *logout* (pro odhlášení a vynulování příslušných vlastností kontextu).

3.3.3 Pages

Každá stránka webové aplikace má vlastní třídu, kterou rozšiřuje třída *Component* z knihovny *React* a nachází se v `/src/pages`. Úvodní stránka je *Welcome*, ze které se lze dostat na stránku registrace (*Register*) nebo přihlášení (*Login*). Na těchto stránkách se nachází pouze formulář, který odešle data na server a zpětně klientovi vrací odpověď s informací o uživateli.

Po úspěšném přihlášení je uživatel přesměrován na stránku *Periods*, kde může přehledně sledovat, vytvářet a upravovat svá období. Ta se řadí od nejnovějších po nejstarší, na každé lze kliknout, a dostat se tak do samotného období na stránce *Period*. Zde může uživatel přidávat pravidelné výdaje a příjmy a jednorázové výdaje (položky), které se vkládají přes příslušné tlačítko v horní části obrazovky. Také zde vidí celkové součty a v případě, že je uživatel součástí rodiny, tak vidí i údaje ostatních členů. Pravidelné výdaje a příjmy si lze nastavit na stránce *Household*, a tím si vytvořit šablonu pro opakované zadávání v období. Správa rodiny (vytvoření a přidání či odebírání členů) probíhá na stránce *Members*. Přehled o vytvořených kategoriích je na stránce *Categories*. Vytvořené kategorie se sdílí mezi všemi členy domácnosti.

Poslední důležitá věc, dlužník, se nachází na stránce *Debts*. Zde mohou uživatelé zadávat své dluhy, které mají u jiných členů rodiny. Mají dvě možnosti, jak to provést. První je zadání dluhu z listu všech vytvořených výdajů (lze využít našeptávač), kde po kliknutí vyskočí formulář s již vyplněnými údaji. Druhou možností je přes tlačítko, zde je potřeba vyplnit všechny údaje (využije se v případě, že výdaj není zadán v systému).

3.3.4 Hosting

Pro nasazení serveru a klienta byla použita službaVercel.com, která je plně dostačující v Hobby verzi (zdarma) pro základní testování funkčnosti (několik uživatelů) a běžné používání. Zároveň také nabízí přehled, kolik peněz jsme již vyčerpali z měsíčního limitu (výčet v tabulce 3.2). [15] Vercel je propojen s GitHub účtem autora práce, na kterém je uložen samotný kód aplikace ve dvou neveřejných projektech: bp-client-2022 a bp-server-2022. Vercel automaticky monitoruje hlavní větev *main*, a pokud dojde k její změně, tak kód z této větve otestuje a následně nasadí na produkci. Aplikace je veřejně dostupná na URL adrese duo-wallet-two.vercel.app (klient) a bp-server-2022.vercel.app (server). V případě, kdy by došlo ke změně adresy ze strany autora nebo poskytovatele hostingu, je aktuální URL adresa vždy připnuta k GitHub projektu.

Omezení	Hodnota
Vytvořená nasazení	100 / den
Maximální čas pro nasazení	45 minut
Maximální přenos dat	100 GB / měsíc
Maximální velikost souboru	100 MB
Maximální velikost request body	4,5 MB
Připojené projekty přes Git	3
Velikost disku	13 GB

Tabulka 3.2: Výčet hlavních omezení služby Vercel ve verzi zdarma

Pro nasazení databáze se použila služba MongoDB, která vyhovuje požadavkům práce ve verzi zdarma. Nejpodstatnější omezení jsou vypsána v tabulce 3.3. Verze zdarma nabízí sdílenou operační paměť RAM a procesor, tudíž se může stát, že v čase nejvyššího vytížení se dotaz bude zpracovávat déle než obvykle, maximálně však půjde o jednotky sekund. Databáze je hostována na fyzickém serveru, který se nachází v německém Frankfurtu. [16]

Omezení	Hodnota
Maximální počet spojení	500
Maximální velikost databáze	512 MB
Maximální velikost dokumentu	16 MB
Počet databází (atlasů)	1

Tabulka 3.3: Výčet hlavních omezení služby MongoDB ve verzi zdarma

3.4 Možné další rozšíření

Vzhledem k zaměření aplikace na oblast financí a jejich správy se nabízí nespočet možných rozšíření, ať už v oblasti nových funkcí, či technologických vylepšení, které by uživateli ještě více ulehčily management vlastního rozpočtu. Během testování aplikace vyvstalo mnoho návrhů na konkrétní vylepšení, které by již byly nad rámec této bakalářské práce, a proto mohou být v budoucnu předmětem dalšího vývoje.

Jedním z návrhů může být zaměření na podporu šetření peněz uživatelů, kteří si díky tomu mohou snadněji vytvářet finanční rezervu. Uživatelům by aplikace pomáhala sledovat a optimalizovat jejich výdaje, a tak dosahovat předem stanovených finančních cílů.

Další funkcí je import výpisů z banky, který umožní automatické nahrávání záznamů do aplikace, což by mělo ušetřit uživatelům čas strávený ručním nahráváním příjmů a výdajů do aplikace. Na to navazuje možnost nahrávat faktury a účtenky, které by aplikace archivovala pro možné budoucí zpětné dohledání. Odesílání upozornění je dalším možným rozšířením aplikace. Tato funkce by

umožňovala uživatelům nastavit si upozornění na různé události, jako je blížící se splatnost faktur, dosažení limitu výdajů nebo výstraha na moc časté výdaje.

Dalším rozšířením je možnost ukládání fotek, účtenek a faktur. Uživatelé by si mohli jednoduše nahrát fotografie nebo dokumenty ke konkrétním transakcím v aplikaci.

Pokud bude mít aplikace všechna potřebná data převážně o utržených příjmech, bude moci na základě těchto dat připravit potřebné informace k podání daňového přiznání, nebo jej dokonce za uživatele vyplnit.

Samotná aplikace již obsahuje části kódu, které zatím nejsou využity, ale již se s nimi počítá v dalším rozšíření popsaném výše. Autor práce tak při vývoji naznačil, které funkce se budou realizovat mezi prvními, např. šetření či různé resolvery (viz kapitola 3.2.3). Než budou všechna výše popsaná rozšíření implementována, je třeba aktuální verzi aplikace řádně a dlouhodobě testovat a naslouchat názorům uživatelů, kteří mohou přijít s dalšími inovacemi a nastavením priorit.

4 Testování

Kapitola se zaměřuje na testování samotné aplikace, které slouží k ověření správného fungování kódu. Jeho cílem je odhalit nedostatky a zajistit, aby aplikace splňovala očekávané výsledky. Testování funguje na principu porovnávání předpokládaného a skutečného výstupu nebo chování systému. Běžně se testy provádějí automaticky před nasazením na produkci, aby se zachytily chyby, které mohly vzniknout v rámci vývoje.

4.1 Unit testy

Cílem unit testování je ověřit funkčnost každé komponenty, malé části kódu, nezávisle na sobě a zjistit, zda splňují očekávané výstupy. Tímto budou odhaleny chyby, čímž se zajistí bezproblémové fungování aplikace. K nalezení jsou ve složce `/src/components/__tests__` a `/src/__tests__`. Pro toto testování byl použit nástroj Jest ve spojení s React knihovnou. Jest je Javascriptový framework, díky kterému lze testy psát a spouštět a společně s React knihovnou je možné také testovat jednotlivé komponenty. V rámci práce bylo vytvořeno několik scénářů, ve kterých je simulováno správné zobrazení komponent a prvků v nich. Testuje se také předvyplňování formulářů (s výchozími údaji i bez). Během testování je nutné definovat všechny možné situace, které mohou nastat, díky čemuž se docílí lepších výsledků. Na závěr testování bude pomocí metriky coverage zjištěno, kolik procent kódu je pokryto.

V klientské části je testována každá komponenta, formulář a dotaz zvlášť v samostatném souboru. Vzhledem k tomu, že některé funkce jsou velmi obsáhlé (přes 500 řádků kódu), autor se rozhodl vybrat jen ty nejnutnější, aby na nich ukázal, jak princip testování funguje. Kompletní testování celé aplikace je možné např. V budoucím pokračování práce. Na serverové části práce se nepodařilo unit testy realizovat. Nejčastější problém nastal při samotném testování, kdy se nepřipojily knihovny/modely, bez kterých testování nebylo možné. I přesto je ukázka nefunkčního kódu součástí práce pro případné další zkoumání. Vybrané testy, které byly realizované na straně klienta, jsou popsány níže.

4.1.1 Dotazy

Některé komponenty obsahují šablony pro dotazování na server. Úkolem těchto testů je zkontrolovat, zda jsou *queries* a *mutations* definované správně a jestli vracejí validní data. Na začátku každého scénáře se připravují testovací data (samotný dotaz a jednotlivé proměnné), která budou použita. Tyto objekty simulují reálnou databázi, do které bude aplikace připojena. Následně je zavolána komponenta a výsledek porovnán s ručně zadaným očekávaným objektem. Pro úspěšné projití testu je potřeba, aby tyto údaje byly stejné. Tímto se docílí toho, že výsledek bude obsahovat správné specifické hodnoty, strukturu a formát. V rámci jedné komponenty je vždy potřeba definovat testy pro dotazy na vytváření, editaci, smazání a získání dat. Jedná se o kategorie, pravidelné příjmy a výdaje.

4.1.2 Formuláře

V těchto testech se nejdříve vytvoří instance potřebných komponent, které daný formulář využívá. Tyto instance simulují data, která zadává uživatel nebo se dostávají z databáze. Vytvořené objekty jsou následně vloženy jako vstupní parametr (props) do testované komponenty (formulář) při jejich renderování. Poté pomocí testovací knihovny získáme reference na jednotlivé prvky formuláře, nejčastěji je dostaneme pomocí názvů atributů či popisků. Dále ověřujeme získané požadované vlastnosti a porovnáváme je spolu s očekávanými. V případě, že se všechny hledané požadavky shodují, test je vyhodnocen jako úspěšný. Testy zahrnují různé scénáře, např. zobrazení prázdného formuláře, zobrazení formuláře se zadanými výchozími hodnotami nebo interakce s prvky formuláře, jako je vyplnění políček, odeslání formuláře apod. Cílem je tedy ověřit, zda se formulář správně zobrazuje, reaguje na interakci a poskytuje očekávané výsledky. Jelikož aplikace obsahuje velký počet formulářů a většina zahrnuje podobné prvky, vybral autor pro testování tři, které jsou nejvíce rozličné. Konkrétně formulář pro přihlášení, kategorie a dluhy.

4.1.3 Funkce

Každá testovací funkce má několik scénářů, které pokrývají různé situace podle jejich větvení uvnitř metody. Pro funkci *FormatDate*() jsou definovány čtyři testy, z toho tři podle toho, jaký formát datumu funkce zvládá, a nakonec se přidá test pro kontrolu, kdy nezadáme žádné datum. Druhou testovanou funkcí je *FormatAmount*, u které je definována na vstupu číselná hodnota, a kontroluje se, zda vrací očekávaný počet desetinných míst.

4.1.4 Test coverage

Testovací pokrytí je technika, která určuje, zda definované testy skutečně pokrývají kód aplikace. Říká, kolik procent kódu, funkcí a větví je otestováno. Díky zjištění velikosti pokrytí lze snadno vidět, jak důkladně je kód testován, na kterou část se zapomnělo a naopak pro kterou část bylo napsáno více testů, než bylo potřeba. [17]

Ideální je mít 100% pokrytí kódu, aby se minimalizovaly chyby na minimum. Tato práce má pokrytí 67,5 %, protože šest formulářů nebylo testováno.

```
 src/components/Functions    |    87.5 |    85.71 |     100 |    87.5 |
  FormatAmount.js            |     100 |      100 |     100 |     100 |
  FormatDate.js              |   85.71 |    85.71 |     100 |   85.71 | 20
 src/components/Handlers     |   61.11 |       50 |      75 |   61.11 |
  FetchHandler.js            |   61.11 |       50 |      75 |   61.11 | 16,34,40-41,
 src/components/Modals       |     100 |      100 |     100 |     100 |
  PopUpModal.js              |     100 |      100 |     100 |     100 |
 src/components/Queries      |   72.72 |     42.1 |   68.75 |   73.33 |
  GetCategories.js           |   86.66 |       80 |   85.71 |   86.66 | 50,58
  GetItems.js                |   93.75 |       50 |     100 |   93.33 | 57
  GetUsers.js                |      50 |    16.66 |    37.5 |   52.63 | 48-58,72-76
  QueriesCategory.js         |     100 |      100 |     100 |     100 |
  QueriesPeriodicIncome.js   |      50 |      100 |      50 |      50 | 85-134
  QueriesPeriodicOutcome.js  |      60 |      100 |      50 |      60 | 75-117
----------------------------|---------|----------|---------|---------|-------------

Test Suites: 13 passed, 13 total
Tests:       30 passed, 30 total
Snapshots:   0 total
Time:        22.491 s
```

Obrázek 4.1: Ukázka pokrytí kódu klienta

4.2 Integrační testy

Integrační testy se zaměřují na ověření funkcionality aplikace jako celku. Ověřuje se, zda spolu jednotlivé komponenty a moduly komunikují správě a interagují tak, jak je očekáváno. Pro integrační testování je použita knihovna Puppeteer, která umožňuje vytvářet scénáře, které simulují chování uživatele. Cílem je definovat běžné situace, na které uživatel může v aplikaci narazit. Oproti unit testům toto testování probíhá primárně na větších částech kódu a mezi klientem a serverem. Testování je zaměřeno na klientskou část, ke které má uživatel přístup (složka `/src/_pupperteerTests`). Testy se soustředí na to, jestli správně proběhlo přihlášení uživatele, zda byl přesměrován na odpovídající stránku a jestli se mu zobrazují očekávaná data. Dále také na skutečnost, zda se při vyplnění formuláře zobrazí výsledek, či nikoli.

V této práci je definováno několik testů, které zjistí základní funkčnost aplikace a prověří, zda autor dokáže otestovat kód pomocí integračních testů. Prvním testem je kontrola, jestli se uživatel po vyplnění údajů a kliknutí na tlačítko *přihlásit se* úspěšně přesměruje na uvítací stránku, kde se mu přímo ukáží již vytvořená období a tlačítko na odhlášení (ukázka výsledku v obrázku 4.2). Dále se testují formuláře, tedy když uživatel vyplní např. údaje pro nové období nebo jednorázový výdaj či příjem. Výsledkem by měla být kontrola, zda se daný prvek po odeslaní

opravdu zobrazil na stránce. Tímto se ověří fungování zapisování do databáze.

```
Správná adresa po přihlášení OK
Nadpis:  Tvoje období
Tlačítko pro odhlášení OK
Tlačítko pro vytvoření období OK
Vypsáno alespoň jedno období - konkrétně: 7
```

Obrázek 4.2: Ukázka výsledku integračního testu přihlášení uživatele

5 Závěr

Hlavním cílem této bakalářské práce bylo vytvořit responzivní webovou aplikaci pro správu osobních financí, která uživateli umožní po registraci do systému připojení do domácnosti, kde může jednoduše a efektivně vytvářet a spravovat svoje výdaje a příjmy. Pro ještě větší jednoduchost umožňuje vytváření pravidelných záznamů (šablon). Data jsou uchována v jednotlivých obdobích, u kterých si uživatel může navolit jejich délku podle vlastní potřeby. V každém období se nacházejí grafické přehledy, které uživatele informují o volných finančních prostředcích. Důležitou funkcí celé aplikace je sekce dlužník, díky kterému budou mít především mladé páry přehled o tom, který z dvojice zaplatil více, a tedy budou vědět, kdo by měl platit příště. Dále lze v aplikaci spravovat členy domácnosti a vytvářet si vlastní kategorie pro větší přehlednost.

Oproti běžně užívanému typu komunikace mezi serverem a klientem pomocí REST API byl u této aplikace použit technologicky mladší dotazovací jazyk GraphQL. Výhodu autor vidí v jednoduchosti pouze jednoho koncového bodu a možnosti na straně klienta vyžádat si jenom data, která potřebuje, čímž se zefektivní přenos a rychlost informací.

Aplikace byla podrobena jak unit testům, tak integračním testům. Všechny napsané testy proběhly úspěšně, čímž se splnila poslední část zadání této práce. Vývoj aplikace bude i nadále pokračovat, a to ve spolupráci s vybranými uživateli, kteří budou přicházet se zpětnou vazbou, podle níž se může webová aplikace dále zlepšovat. K tomu mohou pomoci také autorem rozpracované návrhy na vylepšení, jako je např. podpora a motivace v šetření financí, možnost vkládání výpisu z banky, faktur a účtenek nebo možnost připravit pro uživatele potřebné podklady pro formuláře k daňovému přiznání.

Seznam obrázků

Seznam tabulek

Použitá literatura

[1] *Node.js* [online]. OpenJS Foundation, 2023 [cit. 2023-04-01]. Dostupné z: https://nodejs.org/.

[2] *React* [online]. Meta Open Source, 2023 [cit. 2023-04-01]. Dostupné z: https://react.dev/.

[3] *What is a REST API?* [online]. IBM, 2023 [cit. 2023-04-01]. Dostupné z: https://www.ibm.com/topics/rest-apis.

[4] KOĎOUSKOVÁ, Barbora. *GraphQL vs. REST API, kterou architekturu zvolit?* [online]. 2021. [cit. 2023-04-01]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-graphql-versus-rest.

[5] *The GraphQL Foundation Announces Collaboration with the Joint Development Foundation to Drive Open Source and Open Standards | GraphQL* [online]. The GraphQL Foundation, 2022 [cit. 2023-05-05]. Dostupné z: https://graphql.org/blog/2019-03-12-graphql-foundation-announces-collaboration-with-jdf/.

[6] X.LISTO. *Lekce 1 - GraphQL - GraphQL vs. REST* [online]. 2022. [cit. 2023-05-01]. Dostupné z: https://www.itnetwork.cz/javascript/graphql/graphql-graphql-vs-rest.

[7] DHRUW, Deepanshu. *ORM and ODM — A Brief Introduction* [online]. Spider, 2020 [cit. 2023-05-10]. Dostupné z: https://medium.com/spidernitt/orm-and-odm-a-brief-introduction-369046ec57eb.

[8] OBRIZAN, Andriy. *Mongoose vs. MongoDB: What Should You Choose for Your Node.js Project?* [online]. [cit. 2023-05-06]. Dostupné z: https://leanylabs.com/blog/mongoose-vs-mongodb/.

[9] SHARMA, Anubhav. *What Is Express JS In Node JS?* [online]. Simplilearn Solutions, 2023 [cit. 2023-05-22]. Dostupné z: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js.

[10] *GraphQL | A query language for your API* [online]. GraphQL Foundation, 2023 [cit. 2023-04-01]. Dostupné z: https://graphql.org/.

[11] *MongoDB: The Developer Data Platform* [online]. MongoDB, Inc., 2023 [cit. 2023-04-01]. Dostupné z: https://www.mongodb.com/.

[12] SEDLÁČEK, Petr. *Úvod do MongoDB* [online]. 2019. [cit. 2023-04-01]. Dostupné z: https://www.itnetwork.cz/javascript/nodejs/uvod-do-mongodb.

[13] *Vercel: Develop. Preview. Ship. For the best frontend teams* [online]. Vercel Inc., 2023 [cit. 2023-05-09]. Dostupné z: https://vercel.com/.

[14] KARPOV, Valeri. *Mongoose v7.1.1: Getting Started* [online]. 2023. [cit. 2023-04-05]. Dostupné z: https://mongoosejs.com/docs/.

[15] *Limits | Vercel Docs* [online]. Vercel Inc., 2023 [cit. 2023-05-10]. Dostupné z: https://vercel.com/docs/concepts/limits/overview.

[16] *Pricing | MongoDB* [online]. MongoDB, Inc., 2023 [cit. 2023-05-10]. Dostupné z: https://www.mongodb.com/pricing.

[17] SHAH, Hardik. *Why Test Coverage is an Important part of Software Testing?* [online]. 2021. [cit. 2023-05-02]. Dostupné z: https://www.simform.com/blog/test-coverage/.

[18] *Visual Studio Code* [online]. Microsoft, 2023 [cit. 2023-04-01]. Dostupné z: https://code.visualstudio.com/.

[19] HOQUE, Shama. *Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js, 2nd Edition*. 2. vyd. Packt Publishing, 2020. ISBN 978-1839215414.

[20] *MDN JavaScript* [online]. Mozilla Foundation, 2023 [cit. 2023-04-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.