FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace pro sledování finančních příspěvků pro sportovní kluby
Student:	Jan Mikolášek
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem této práce je navhrnout a implementovat aplikaci pro sledování finančních příspěvků pro sportovní kluby. Aplikace bude sledovat pohyby na účtu konkrétní organizace a automaticky je pomocí bankovního API párovat ke konkrétním hráčům, turnajům atd. Pro komunikaci s bankovními institucemi zvolte vhodné řešení na základě rešerše. Softwarové řešení bude složeno ze serverové a klientské části.

Pro implementaci serverové části použijte JavaScriptový framework Node.js a pro klientskou část proveďte analýzu možných techologií a vyberte vhodného kandidáta.

1) Analyzujte možnosti komunikace s bankovními institucemi
2) Na základě předchozí analýzy navrhněte vhodné softwarové řešení
3) Implementujte prototyp webové aplikace
4) Aplikaci podrobte vhodným testům
5) Na základě výsledků testování proveďte nebo alespoň navrhněte potřebné úpravy a opravy aplikace

---

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 25. února 2021 v Praze.

Bakalářská práce

APLIKACE PRO SLEDOVÁNÍ FINANČNÍCH PŘÍSPĚVKŮ PRO SPORTOVNÍ KLUBY

Jan Mikolášek

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: Ing. Filip Glazar
13. května 2021

Odkaz na tuto práci: Jan Mikolášek. *Aplikace pro sledování finančních příspěvků pro sportovní kluby.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Tímto bych rád poděkoval svému vedoucímu práce Ing. Filipovi Glazarovi za poskytnutí rad a zkušeností k tvorbě práce. Moje poděkování si také zaslouží přítelkyně Daniela a kamarád Honza, kteří mi pomáhali s kontrolou práce. Nakonec bych největší poděkování věnoval rodině, bez které bych to nedotáhl takhle daleko a která mě za dobu celého studia plně podporovala.

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 10. května 2020 .....

Abstrakt

Tato práce se především zabývá komunikací aplikace s bankou. Popisujeme zde také vytvoření webové aplikace za pomoci moderních technologií. Výstupem je portál, jehož cílem je pomáhat sportovním klubům s organizací příspěvků. Tento portál využívá serverové části, která zajišťuje komunikaci s bankou a databází. V práci najdeme automatizované testy a rozhovor s koncovým uživatelem. Ke konci dokumentu se dozvíme o budoucnosti projektu a jeho použití.

Klíčová slova sportovní klub, finanční příspěvky, banka, Open banking, PSD2, AISP, API, NodeJS, React, Javascript, JS, Bootstrap, MongoDB

Abstract

This thesis mainly deals with communication between an application and a bank. We are describing the creation of web applications using modern technologies. The thesis result is a web portal that helps sports clubs with payment tracking. This portal uses a server that communicates with the bank and the database. We can also seek automatized tests and communication with end-user. At the end of this documentation, we will find out about the future of the project and its uses.

Keywords sport club, payment tracking, bank, Open banking, PSD2, AISP, API, NodeJS, React, Javascript, JS, Bootstrap, MongoDB

Shrnutí

Motivace

Vedení našeho florbalového klubu stojí mnoho času. Nejvíce času zabere kontrolování příspěvků. Každý měsíc musí vedoucí zkontrolovat nové platby a ručně je přiřadit k hráčům. Napadlo mě implementovat aplikaci, která by tento postup provedla automaticky. Tedy pomocí Open banking API stáhla transakce a poté je přiřadila ke hráčovi v databázi. Aplikace by obsahovala i informace o platbě pro jednotlivé hráče.

Cíl práce

Analýza možnosti komunikace s bankou. Navrhnut a implementovat řešení, které se bude skládat ze serverové a klientské části. Aplikace zvládne komunikaci s bankou a s databází, dále dokáže párovat získané transakce ke hráčům. Aplikaci poté podrobíme vhodným testům a na základě těchto testů navrhneme zlepšení.

Postup

Začali jsme analýzou, zda je komunikace s bankou vůbec možná. Popsali jsme si základní informace o Open bankingu, PSD2 a jeho použití v České republice. Ukázali jsme si příklady komunikací s Fio bankou. Dále jsme analyzovali technologie, za pomoci kterých budeme tvořit naši aplikaci. Jedná se hlavně o databázi v MongoDB a JavaScriptové frameworky NodeJS a React. Následovala vlastní implementace skládající se ze serverové a klientské části. Server obstarává komunikací s bankou a databází. Klient následně data stahuje a zobrazuje. Pro testování jsme zvolili programové unit testy a zpětnou vazbu od vedoucího klubu. Následně jsme provedli menší úpravy a navrhnuli budoucí vývoj.

Výsledky práce

Výsledkem naší práce je prototyp aplikace, který zvládne komunikaci s Fio bankou a následné přiřazení transakcí k uživateli. Součástí této práce je databáze vytvořená v MongoDB. K ní se pak připojujeme pomocí serveru, který zajišťuje i komunikaci s bankou. V klientské části najdeme přehled pro hráče, úpravu dat pro adminy a prototyp přihlašovacího systému.

Závěr

Všechny stanovené cíle byly splněny. Věřím, že aplikace bude vedoucím klubu napomáhat a že po navrhnutých úpravách a rozšířeních bude připravena na plný provoz. Tato aplikace se líbila i samotným hráčům, kteří by uvítali přehled o platbách a speciálně vygenerovaný QR kód, který urychlí platby.

Seznam zkratek

PSD2	Payment Services Directive 2
AISP	Account Information Service Provider
PISP	Payment Initiation Service Provider
CISP	Card-based Payment Instrument Issuer
IBAN	International Bank Account Number
API	Application Programming Interface
SSL	Secure Sockets Layer
HTTPS	Hypertext Transfer Protocol Secure
JS	JavaScript
JSON	JavaScript Object Notation
I/O	Input/output
CPU	Central processing unit
NPM	Node Package Manager
UI	User Interface
DOM	Document Object Model
CSS	Cascading Style Sheets
URL	Uniform Resource Locator

Kapitola 1
Úvod

1.1 Motivace

Můj florbalový klub se postupem času velmi rozrostl. Nabrali jsme mnoho nových členů, což vedlo k založení rezervního B týmu a juniorského týmu. Proto vedení klubu nyní stojí daleko více času, než tomu bylo před pár lety. Každý tým potřebuje řešit finanční příspěvky. Komplikace bohužel nastávají při kontrole, kdo již zaplaceno má a komu chybí doplatit. Aktuálně musí vedoucí týmu každý měsíc kontrolovat bankovní účet a podle komentářů u jednotlivých transakcí pak zadat data do tabulky v Excelu. To samozřejmě při vzrůstajícím počtu hráčů zabere mnoho času.

Nedávno jsem proto přišel s návrhem webové aplikace, která by zautomatizovala tento proces. Vedoucí by mohl zakládat příspěvky v portálu a poté kontrolovat, kdo již zaplatil. Aplikace by využívala komunikace open banking API, díky které by mohla komunikovat s bankovním účtem klubu a následně například pomocí variabilního čísla vytvořit statistiku, kdo platil. Tato aplikace by byla přívětivá i pro samostatné hráče. Součástí by totiž bylo přihlášení, po kterém by si hráč mohl zobrazit a zkontrolovat, jaké příspěvky již platil a jaké mu zbývají doplatit.

Po představení konceptu vedoucím klubu jsem se setkal s velkým nadšením. Napadlo mě vytvořit aplikaci jako součást bakalářské práce. Díky tomu získám možnost rad a zkušeností od odborníka, které mi pomohou při řešení problémů.

1.2 Cíle práce

Cílem této práce je navrhnout aplikaci pro sledování finančních příspěvků pro sportovní kluby. Pomocí bankovního API budeme schopni spárovat transakce s konkrétními hráči. Aplikace se bude skládat se serverové a klientské části. Pro obě části využijeme JavaScriptové frameworky. V případě serverové části použijeme Node.js a klientskou část budeme tvořit pomocí Reactu.

Budeme analyzovat komunikace s bankovními institucemi. Povíme si základní informace o Open Bankingu a o směrnici Evropského parlamentu PSD2. Zjistíme, jakým způsobem je tato směrnice implementována u nás a popíšeme si základní komunikaci s bankovní institucí Fio.

V návrhu aplikace si popíšeme nejdůležitější použité technologie. Technologie si rozdělíme do tří sekcí. První bude obsahovat technologie MongoDB starající se o databázi. Ve druhé se podíváme na serverovou část, kde najdeme informace o technologiích jako je Node.js, NPM a JSON. A v poslední sekci zabývající se klientskou částí si popíšeme základy Reactu a Bootstrapu.

V implementaci aplikace se opět budeme zabývat serverovou a klientskou částí aplikace. Tentokrát už budeme popisovat námi navržené řešení. U serverové části si představíme model databáze a všechny dostupné API. V klientské části se budeme věnovat implementaci v Reactu a popisu důležitých komponent.

Po implementaci bude navazovat testování, které si rozdělíme do programového a s uživatelem. Na základě výsledků, pak provedeme a navrhneme změny.

Kapitola 2
Analýza

V této kapitole se budeme věnovat analýze aktuálního řešení. Podíváme se na požadavky, které pomohou jak vedoucím klubu, tak usnadní přehled a platbu hráčů. Dále se budeme snažit najít způsob, jak bude naše aplikace komunikovat s bankovními institucemi.

2.1 Současný stav problému

V současné době musí vedoucí klubu kontrolovat každou nově příchozí platbu na bankovním účtu. Po této kontrole přiřadí transakce k jednotlivým hráčům. Transakce v sobě nese informaci o hráči ve zprávě pro příjemce, pomocí které je pak přiřadí. Tyto zprávy se mnohdy liší. Například dvě platby od stejného hráče mají dvě různé zprávy: *„Mikolášek – platba Slovensko“* a *„Poplatek za sezonu Mikolasek“*.

Transakce a příspěvky jsou vedené v Excelovém souboru, do kterého mají přístup pouze vedoucí. Pokud chce vedoucí dát vědět hráčům kolik dluží, vytvoří podle dat příspěvek ve skupině na sociální síti Facebook.

2.2 Funkční požadavky aplikace

Hlavními požadavky je co největší automatizace procesu přiřazovaní příspěvků. Dále realizovat co nejjednodušší portál a způsob plateb pro hráče.

1. **Kontrola nově příchozích plateb** - Aplikace si sama zvládne stáhnout transakce z bankovního účtu a poté je dokáže přiřadit hráči, například pomocí variabilního čísla.
2. **Úprava dat v aplikaci** - Administrátor bude moci upravit data přímo v portálu. Úprava by měla být přívětivá tak, že ji zvládne člověk s minimálními technickými znalostmi.
3. **Systém přihlášení** - Možnost přihlášení do systému pro každého hráče. K přihlášení bude hráč potřebovat například email a heslo. Vytváření účtu budou mít na starost administrátoři.
4. **Portál hráče** - Po přihlášení se hráči zobrazí příspěvky a platby, které již proběhly. Hráč dále uvidí informace o převodu, který je potřeba provést.

2.3 Komunikace s bankou

V následující části se budeme věnovat analýze komunikace aplikace s bankou. Naším cílem je najít způsob, jak by naše aplikace byla schopná získat informace z klubového finančního účtu. Od banky budeme požadovat list transakcí s jejich informacemi, jako jsou například číslo protiúčtu, částka a variabilní symbol.

2.3.1 Open banking

Open banking nabízí otevřenou, dokumentovanou a podporovanou komunikaci třetím stranám. V případě, že by tato komunikace neexistovala, by developeři museli sáhnout po takzvaném screen scrapingu. To znamená volání neveřejných komunikací, jako je například parsování HTML stránek. Jelikož to není podporovaná komunikace, není zaručeno, že zůstane stejná.

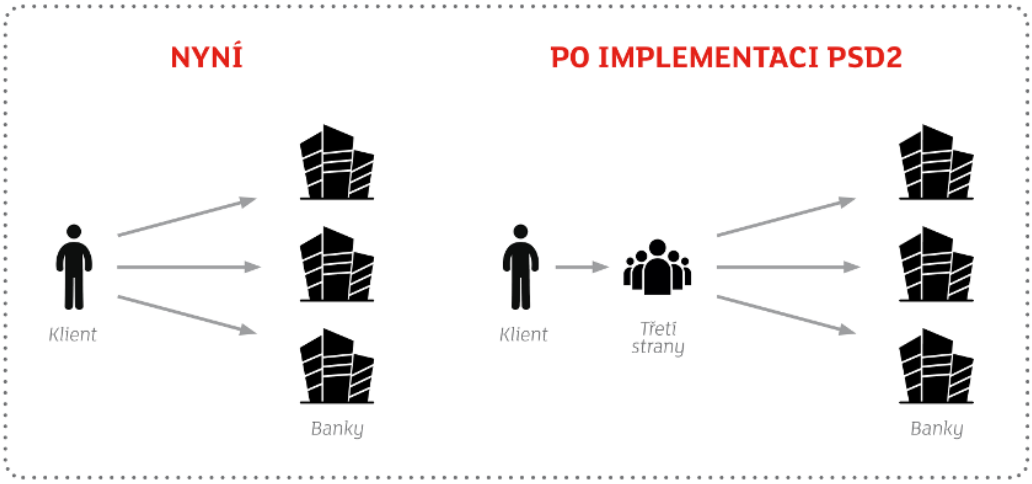
Open banking přináší mnoho výhod pro multibanking. To nám zajišťuje spravování několika účtů z rozdílných bankovních institucí pomocí jedné aplikace. Před příchodem open bankingu byl zákazník nucen využívat ke správě svých financí jen software, který byl povolený příslušnou bankou. S aplikacemi pro multibanking přicházejí už i samotné bankovní instituce, které se snaží napojit na své konkurenty. To umožňuje zákazníkovi například z internetového bankovnictví od Monety Money Bank ovládat i účet z Fio banky. [3][4]

2.3.2 PSD2

PSD2 (Payment Services Directive) je směrnice přijatá Evropským parlamentem 25. listopadu 2015 a 13. ledna nabyla účinnosti v ČR. Tato směrnice, jak už anglický název napovídá, se týká platebních služeb a jedním z hlavních cílů je 4 představení alternativních způsobů, jak lze ovládat bankovní účet. To dovoluje vstoupení nových účastníků na finanční trh. Výsledkem by tedy měly být nové služby a stále rozvíjející se produkty pro zákazníka.

Díky této směrnici jsou banky nuceny poskytovat informace třetím stranám, pokud to zákazník povolí. K těmto informacím pak přistupují pomocí Open banking API, které musí mít banka zdokumentované a plně je podporovat. Přineslo nám to tři nové typy služeb, a těmi jsou **AISP** - Account Information Service Provider, **PISP** - Payment Initiation Service Provider a **CISP** - Card-based Payment Instrument Issuer.

V závěru bych chtěl podotknout, že PSD2 není jen o Open bankingu, ale směrnice například řeší i online podvody, kdy za neautorizovanou platbu bude zákazníkovi účtováno maximálně 50 EUR. [1][2]



Obrázek 2.1 Ilustrace komunikace před a po implementaci PSD2 [1]

2.3.2.1 AISP

AISP bude základním stavebním kamenem pro naši aplikaci. Umožňuje totiž třetí straně nahlédnout do historii transakcí. To se nám bude hodit při implementaci získání posledních transakcí z bankovního účtu. Dále nám umožňuje nahlédnout na informace o platebním účtu. To pro nás ale nebude podstatné, neboť budeme vždy pracovat jen s klubovým účtem.

2.3.2.2 PISP

PISP umožňuje třetí straně předvyplnit platební příkaz, který pak přijde uživateli ke schválení. To vede k rychlejším platbám, což se hodí aplikacím, které fungují na častých platbách, jako jsou například mikrotransakce u her.

2.3.2.3 CISP

CISP je druh komunikace, která odpoví ANO/NE, zda má uživatel dostatek prostředků na kartě pro určitou částku. Zde musím podotknout, že se nejedná o klasické debetní nebo kreditní karty, ale karty vydané obchodníky, kde zákazník zrovna platí. [1][3]

2.4 Český standard pro Open banking

„Hlavními přínosy Českého standardu je jednodušší integrace třetích stran na banky." [5, str. 12] Tento standard definuje a sjednocuje API komunikace mezi rozdílnými bankami. Komunikace probíhá pomocí http dotazů a banka po zpracování dotazu vrátí JSON odpověď. Pro příklad si můžeme ukázat http příkaz pro výpis mých transakcí: GET /my/payments. Návratovou hodnotou pak bude list transakcí, které budou například obsahovat informace o částce, odesílateli, příjemci, variabilním symbolu, poznámce anebo datu založení či poslání.

Standard se snaží zachovat vysoký stupeň univerzálnosti. I přes to se mohou poskytovatelé finančních služeb odchylovat v určitých bodech, neboť každý má odlišný systém. I pro české banky platí, že dle PSD2 musejí poskytovat dokumentaci ke svému řešení. [5]

2.5 Komunikace s bankou Fio

Víme, že klub má založený účet u banky Fio, proto se s ní budeme snažit najít možný způsob komunikace.

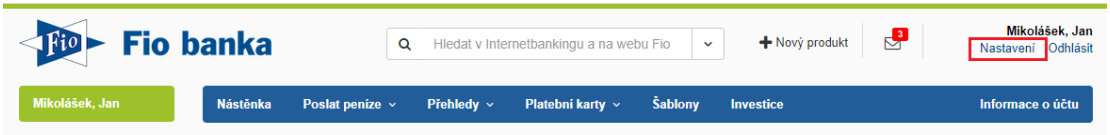
Z přiložené dokumentace pro API Bankovnictví od Fio banky si můžeme přečíst, že umožní přístup jak k získávání dat, tak k podávání příkazů. My se zaměříme jen na získávání dat (AISP), neboť naše aplikace bude potřebovat pouze číst data. Tato komunikace probíhá pomocí SSL protokolů s minimálně 128bitovým šifrováním

Od majitele nebo uživatele s patřičným oprávněním potřebujeme vygenerovat token. Toho může dosáhnout pomocí internetového bankovnictví. Token je unikátní řetězec o 64 znacích a má přiřazená práva: *„Sledování účtu“* nebo *„Sledování účtu a zadávání platebních a inkasních příkazů“*. Tento token je spárován jen s jedním účtem a lze ho kdykoliv deaktivovat. [6]

2.5.1 Návod k získání tokenu

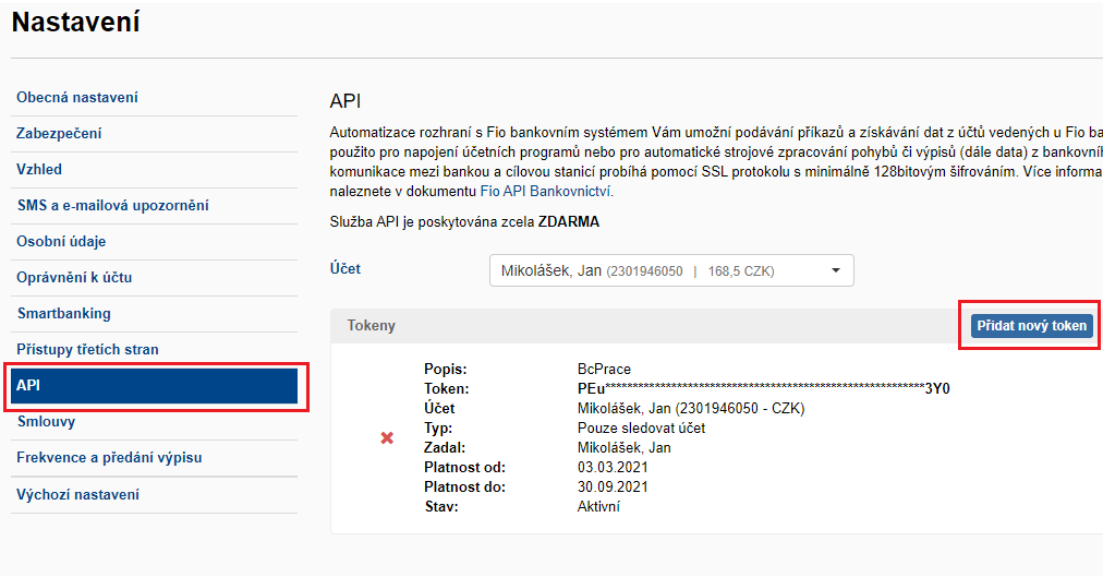
Postup vygenerování tokenu pro majitele bankovního účtu:

1. Prvním krokem je přihlášení oprávněné osoby do internetového bankovnictví.
2. Dále se musí prokliknout do **Nastavení**, které se nachází v pravé horní části portálu, pod jménem uživatele.



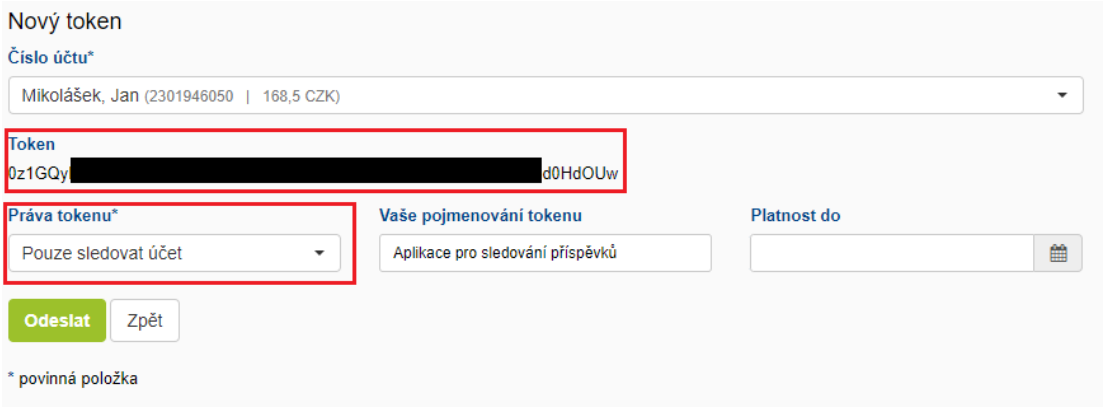
Obrázek 2.2 Tlačítko nastavení [7]

3. V nastavení klikne na **API**. A poté na **Přidat nový token**.



Obrázek 2.3 Nastavení [7]

4. V otevřeném formuláři si může pojmenovat token, vyplnit platnost a především nastavit práva tokenu. Pro náš účel postačí *„Pouze sledovat účet“*. **Uložíme token**. Poté klikne **odeslat a autorizuje**.



Obrázek 2.4 Nastavení tokenu [7]

2.5.2 Komunikace

Komunikace probíhá pomocí https metod GET a POST. Protože my potřebujeme informace pouze číst, vystačíme si s GET žádostmi. Nyní si ukážeme komunikace, které můžeme využít pro naši aplikaci.

Tabulka 2.1 Pohyby na účtu za určené období [6, str. 7]

Struktura:	.../periods/{token}/{datum od}/{datum do}/transactions.{format}	
	Token	unikátní vygenerovaný token
	Datum od	začátek stahovaných příkazů ve formátu rrrr-mm-dd
	Datum do	konec stahovaných příkazů ve formátu rrrr-mm-dd
	Formát	formát pohybů
Příklad:	Získání pohybů v období od 25.8.2012 do 31.8.2012 v xml	
`https://www.fio.cz/ib_api/rest/periods/aGEMQB9Idh35fh1g51h3ekkQwyGlQ/2012-08-25/2012-08-31/transactions.xml`		

Tabulka 2.1 nám ukazuje jak získat transakce od-do podle data. Výsledek bychom mohli použít například při kontrole předchozích transakcí na účtu, bez nutnosti kontaktování zodpovědné osoby.

Tabulka 2.2 Pohyby na účtu od posledního stažení [6, str. 7]

Struktura:	.../last/{token}/transactions.{format}	
	Token	unikátní vygenerovaný token
	Formát	formát pohybů
Příklad:	Získání pohybů od posledního stažení v JSON	
`https://www.fio.cz/ib_api/rest/last/aGEMtmwcsWAjPzhg3bPH3j7Iu15g56d66AdEbfIjqIgMR9Idh6u3/transactions.json`		

V tabulce 2.2 vidíme komunikaci, kterou budeme nejvíce využívat. Výsledkem jsou totiž všechny transakce od poslední kontroly. To nám zlehčí práci, neboť si nebudeme muset pamatovat datum a čas poslední kontroly pokaždé, když budeme chtít získat nové údaje z banky.

Tabulka 2.3 Nastavení zarážky [6, str. 7]

Struktura:	.../set-last-date/{token}/{datum}	
	Token	unikátní vygenerovaný token
	Datum	datum posledního neúspěšně staženého výpisu ve formátu rrrr-mm-dd
Příklad:	Nastavení data posledního neúspěšného stažení pohybů na 27. 7. 2012	
`https://www.fio.cz/ib_api/rest/set-last-date/Pu5CMBu5nYBthKaqM0FaUlY7JIjUnY0FaUlY7JIjU1eUl/2012-07-27`		

Komunikace v tabulce 2.3 bude sloužit jako nastavení zarážky. Tato zarážka je pak využívána v tabulce 2.2. Využití je jasné, možnost chyby v přenosu, testování nebo v případě resetování hodnot.

2.5.3 Odpověď

Fio banka nabízí následující formáty pro přehled pohybů na účtě CSV, GPC, HTML, JSON, OFX, FIO XML. Protože budeme psát backendovou část aplikace v NodeJS, budeme používat formát dat JSON. JSON je podmnožina jazyka JavaScript. Výsledkem je objekt, který vždy obsahuje dvojice klíč: hodnota. [12]

Po dotazu o pohybech se nám vrátí objekt s informacemi o účtu a TransactionList. Pro nás bude významný TransactionList, který obsahuje pole objektů (transakcí), které budeme následně ukládat do databáze.

Výpis kódu 2.1 Zkrácený příklad odpovědi o pohybu na účtě

```
{
    "accountStatement": {
        "info": {
            ...          //informace o ucte
        },
        "transactionList": {
            "transaction": [
                {
                    "column1": {
                        "value": 100.00,
                        "name": "Objem",
                        "id": 1
                    },
                    "column2": {
                        "value": "201840372",
                        "name": "Protiúcet",
                        "id": 2
                    },
                    "column3": {
                        "value": "0600",
                        "name": "Kód banky",
                        "id": 3
                    },
                    ...      // dalsi informace o transakci
                },
                ...          // pokracuje list transakci
            ]
        }
    }
}
```

Transakce jsou prezentovány jako objekty v poli a mají specifickou strukturu. Objekt obsahuje několik klíčů s názvem Column{0-27}. Bohužel bez dokumentace tedy předem nedokážeme určit spárování Column s typem hodnoty. Pro přehled je vytažena tabulka 2.4 s mapováním na jednotlivé atributy a jejich popisem. [6, str. 25]

Tabulka 2.4 Namapovní jednotlivých sloupců v odpovědi [6, str. 25]

ID sloupce	Atribut	Popis
Column22	ID pohybu	unikátní číslo pohybu - 10 numerických znaků
Column0	Datum	datum pohybu ve tvaru rrrr-mm-dd+GMT
Column1	Objem	velikost přijaté (odeslané) částky
Column14	Měna	měna přijaté (odeslané) částky dle standardu ISO 4217
Column2	Protiúčet	číslo protiúčtu
Column10	Název protiúčtu	název protiúčtu
Column3	Kód banky	číslo banky protiúčtu
Column12	Název banky	název banky protiúčtu
Column4	KS	konstantní symbol
Column5	VS	variabilní symbol
Column6	SS	specifický symbol
Column7	Uživatelská identifikace	uživatelská identifikace
Column16	Zpráva pro příjemce	zpráva pro příjemce
Column8	Typ pohybu	typ operace
Column9	Provedl	oprávněná osoba, která zadala příkaz
Column18	Upřesnění	upřesňující informace
Column25	Komentář	komentář
Column26	BIC	bankovní identifikační kód banky protiúčtu dle standardu ISO 9362
Column17	ID pokynu	číslo příkazu
Column27	Reference plátce	bližší identifikace platby dle ujednání mezi účastníky platby

Kapitola 3
Návrh aplikace

3.1 Databáze MongoDB

Pro náš portál budeme potřebovat databázi, ve které budeme uchovávat například hráče, jejich příspěvky a transakce. Především z důvodu dobré komunikace s NodeJS a ukládání dat v podobě JSON dokumentů, použijeme **MongoDB**.

MongoDB je NoSQL (nerelační) databáze. Entity jsou reprezentovány v podobě kolekcí JSON dokumentů, které nemají předem daný formát, jako je to v relační databázi v případě tabulek. [8]

3.1.1 Jednoduchost a flexibilita

Nabízí developerům vytvoření databáze během pár minut. Dále díky formátu JSON dokumentů můžeme ukládat téměř cokoliv, aniž bychom potřebovali vytvářet mapování. Velkou výhodou je zde vnořování objektů, které by se v relační databázi řešilo propojením několika tabulek.

3.1.2 Relace mezi dokumenty

Propojení dokumentů zde funguje většinou přes vygenerovaný index (_id). Více méně můžeme propojovat přes cokoliv, pak si ale musíme dát pozor, aby se data dále neměnila. Jak už bylo zmíněno v předešlém bodě, existuje zde i možnost vnořování objektů.

Databáze nám povoluje založení i vlastních indexů. Nad těmi je pak vyhledávání rychlejší. MongoDB nabízí vlastní vyhledávací query nástroj. Na následujicí straně si ukážeme příklad možných relací.

Výpis kódu 3.1 Představení relací mezi dokumenty.

```
1  //Dokument uzivatele
2  {
3    _id:"user001",
4    fullname:"Jan Mikolasek",
5    team:{                    // napojeni 1:1 pomoci vnorenych objektu
6      _id:"team01",
7      name:"Muzi A"
8    },
9    transaction_ids: ["trans001", "trans002", ...] // napojeni 1:m pomoci pole id
10 }
11
12 //Dokumenty transakci
13 {
14   _id:"trans001",
15   name:"Turnaj bratislava",
16   amount: 2200.00
17 }
18
19 {
20   _id:"trans002",
21   name:"Prispevek sezona 2021",
22   amount: 1000.00
23 }
24
25 ...
```

V kódu 3.1 si můžeme všimnout dokumentů uživatele a transakcí. Potřebujeme uživatele přiřadit k týmu *„Muzi A“*. Pro tuto relaci jsme zvolili vnoření objektů, viz řádek 5 s klíčem *„team“*. Dále potřebujeme k uživateli přiřadit několik transakcí. Můžeme opět postupovat pomocí vnořování, ale my si tentokrát vybereme napojení pomocí _id. Nejdříve vytvoříme transakci jako samotný dokument a poté připojíme pomocí pole, viz řádek 9 a klíče *„transaction_ids“*.

3.1.3 Dotazy a vyhledávání

Protože používáme NoSQL databázi, nemáme k dispozici SQL dotazy. MongoDB nabízí svoje řešení, které funguje na základě queries (dotazů). Dotazy mají tvar objektu JSON a používají se například při zakládání, úpravě nebo hledání.

Nyní si ukážeme pár příkladů SQL dotazů, ke kterým pak vytvoříme ekvivalentní query. Pro příklad můžeme využít dat, které jsme nastínily v kódu 3.1. Začneme tím nejjednodušším, čímž je získání všech dat z transakcí.

Výpis kódu 3.2 Vyber všechny dokumenty mezi transakcemi.

```
1 //SQL dotaz
2 SELECT * FROM transaction
3
4 //query
5 {}
6
7 //Priklad v kodu
8 db.transaction.find({})
```

Můžeme si všimnout, že zmizely výrazy **SELECT * FROM**. Příkaz hledá v dokumentové knihovně transakcí pomocí funkce **transaction.find()**. Té pak stačí předat prázdný filtr (objekt), neboť hledáme všechny dokumenty. Pojďme se podívat na trochu složitější příkaz.

Výpis kódu 3.3 Úvod do podmínek

```
1 //SQL dotaz
2 SELECT * FROM transaction WHERE variableSymbol = "181818" OR amount > 1500
3
4 //query
5 let qr = { $or: [ { variableSymbol: "181818" }, { amount: { $gt: 1500 } } ] }
6 db.transaction.find(qr)
```

Výsledkem budou transakce s variabilním symbolem 181818 nebo s částkou větší než 1500. Můžeme zde vidět podmínky $or a $gt. $or je logický operátor, který vyžaduje správně vyhodnocenou alespoň jednu podmínku. $gt znamená greater than, tedy musí být větší než požadované číslo.

Operátorů a způsobů, jak prohledávat data, nabízí MongoDB mnoho. Pro více informací bych doporučil MongoDB manual, kde se můžete dozvědět všechny informace o poskytovaných službách. [9]

3.2 Serverová část aplikace

Pro náš portál budeme potřebovat vytvořit jednotnou aplikaci, která obstará komunikaci s bankou a s databází. Dalším úkolem této aplikace bude přihlašovací systém, jenž ověří, zda hráč zadal správné přihlašovací údaje nebo se shoduje jeho cookie.

3.2.1 Node.js

Node.js je open-source a multiplatformní JavaScript runtime prostředí. Je postaven na Chrome V8 JS enginu běžícím mimo prohlížeč. Velkou výhodou je, že aplikace běží velmi efektivně v jednom procesu, aniž bychom museli vytvářet nové vlákno pro každou žádost. Node.js nabízí soubor asynchronních I/O řešeních, které brání v blokování kódu. Většinou i externí knihovny sdílejí tuto politiku a snaží se psát neblokující systém. Opak je tedy velmi ojedinělá výjimka.

Když Node.js provádí operace jako je zápis a čtení z databáze nebo vytváření http žádostí, místo aby čekáním blokovaly CPU cykly, umožňuje Node.js pokračovat v dalších operacích, než se vrátí předchozí odpověď. To povoluje obsluhu tisíce různých požadavků, bez nutnosti vytvoření více vláknové politiky.

Dalším přínosem je výhoda pro frontend vývojáře, kteří píšou weby v JavaScriptu. Ti nyní zvládnou napsat backend bez nutnosti se naučit zcela nový programovací jazyk. [10]

3.2.1.1 NPM

Node Package Manager je správce JS balíčků. Umožňuje jejich instalaci a údržbu. NPM je součástí standardní instalace Node.js. NPM zvládne také distribuci řešení a správu relací mezi balíčky. Všechny balíčky můžeme najít v NPM repozitáři. [11]

3.2.2 JSON

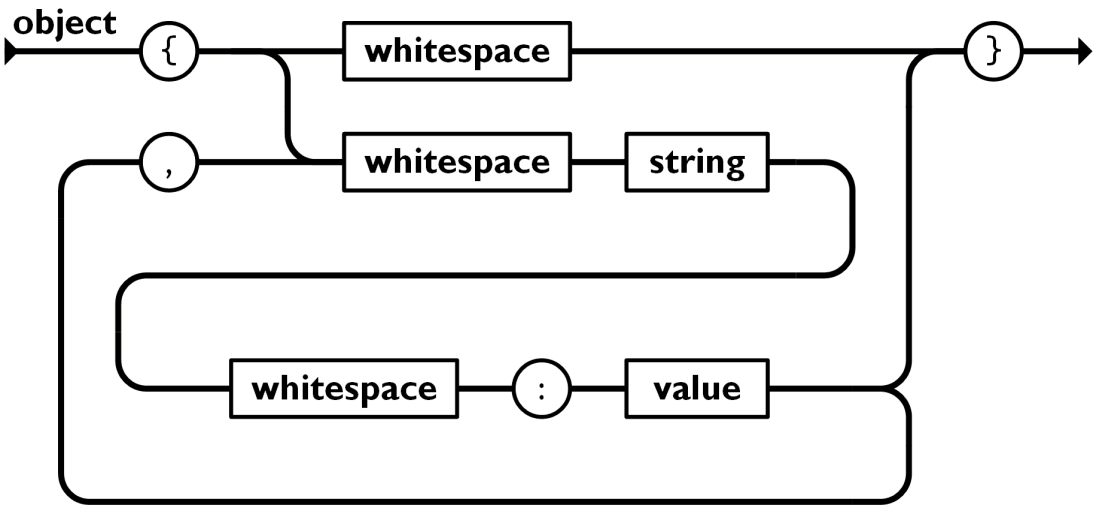
JSON (JavaScript Object Notation) je podmnožina programovacího jazyka JavaScript. Je snadno čitelný člověkem a jednoduše zpracovatelný pro stroj. Jedná se o textový nezávislý formát, který však využívá konvencí jazyků rodiny C (C, C++, C#, Java, JavaScript…). Díky tomu je JSON ideální kandidát pro výměnu dat mezi různými jazyky.

JSON může být strukturován dvěma možnostmi:

1. Kolekce párů skládající se ze jména a hodnoty. V mnoha jazycích to můžeme vidět jako objekt, strukturu, klíčové pole nebo hash tabulku.

2. Seřazený seznam hodnot. V jazycích reprezentovaný například jako pole, vektor, list nebo sekvence.

Toto jsou univerzální struktury dat, které obecně všechny moderní programovací jazyky v nějaké formě podporují. Nyní si ukážeme diagramy těchto struktur.

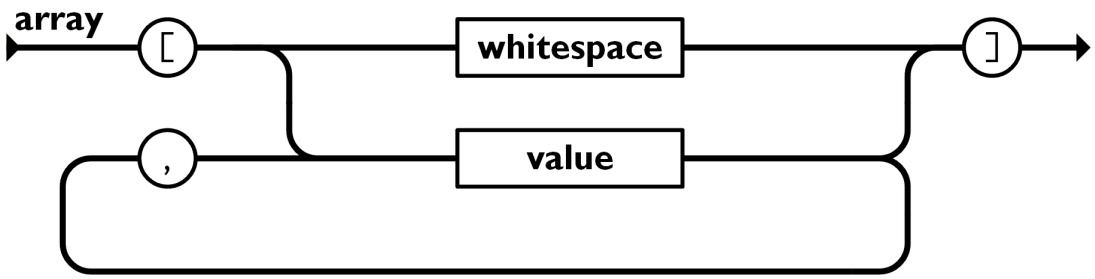


Obrázek 3.1 Diagram validace JSON objektu [12]

Můžeme vidět, že objekt je vždy ohraničen složenými závorkami. Další zajímavostí je možnost předání prázdného objektu, tedy jak je naznačeno **{whitespaces}**. Whitespaces jsou takzvané bílé znaky. Patří sem například mezerník a nová řádka.

Pokud nepotřebujeme získat prázdný objekt, vytvoříme vždy dvojici **string: value**. Zde se setkáváme s dalšími pojmy. **String** (řetězec) je jednoduše definován jako řetězec znaků Unicode, nad ním běží úniková pravidla. Tomu se také často říká název nebo klíč hodnoty. **Value** (hodnota) je pak vyjádřena typy: řetězcem, číslem, objektem, polem, booleanem nebo prázdnou hodnotou (null).

V případě, že chceme přidat další hodnotu, přidáváme čárku a opakujeme proces, který zakončujeme pravou složenou závorkou.



Obrázek 3.2 Diagram validace JSON pole [12]

Jak můžeme vidět, tak struktura pole je mnohem jednodušší. Opět zde máme hraniční znaky, tentokrát v podobě hranatých závorek, a znovu lze vytvořit pole o žádném prvku. Popis pole je také přímý. Jedná se o řadu hodnot, které jsou odděleny opět čárkou, a na rozdíl od objektu, nemají svůj název/klíč. [12]

3.3 Klientská část aplikace

V této sekci si ukážeme technologie, které nám pomůžou postavit portál, který se bude ukazovat hráčům. Co se týče dynamické a programovací části, zvolil jsem **ReactJS** a ke stylistické části použijeme **Bootstrap**.

3.3.1 React

React je knihovna napsaná v JavaScriptu pro vytváření uživatelského rozhraní. Byla vytvořena společností Facebook Inc. a nyní je podle průzkumu Google Trends nejvíce populárním způsobem pro tvoření UI. Základní kámen každé React aplikace je komponenta. Komponenta je většinou menší část rozhraní. Pokud chceme vytvořit co nejvíce univerzální aplikaci, začneme s postavením několika menších nezávislých komponent, které pak skládáme dohromady. Každá aplikace má takzvanou root-aplication, většinou referovanou jako App.js, na kterou se připínají další komponenty.

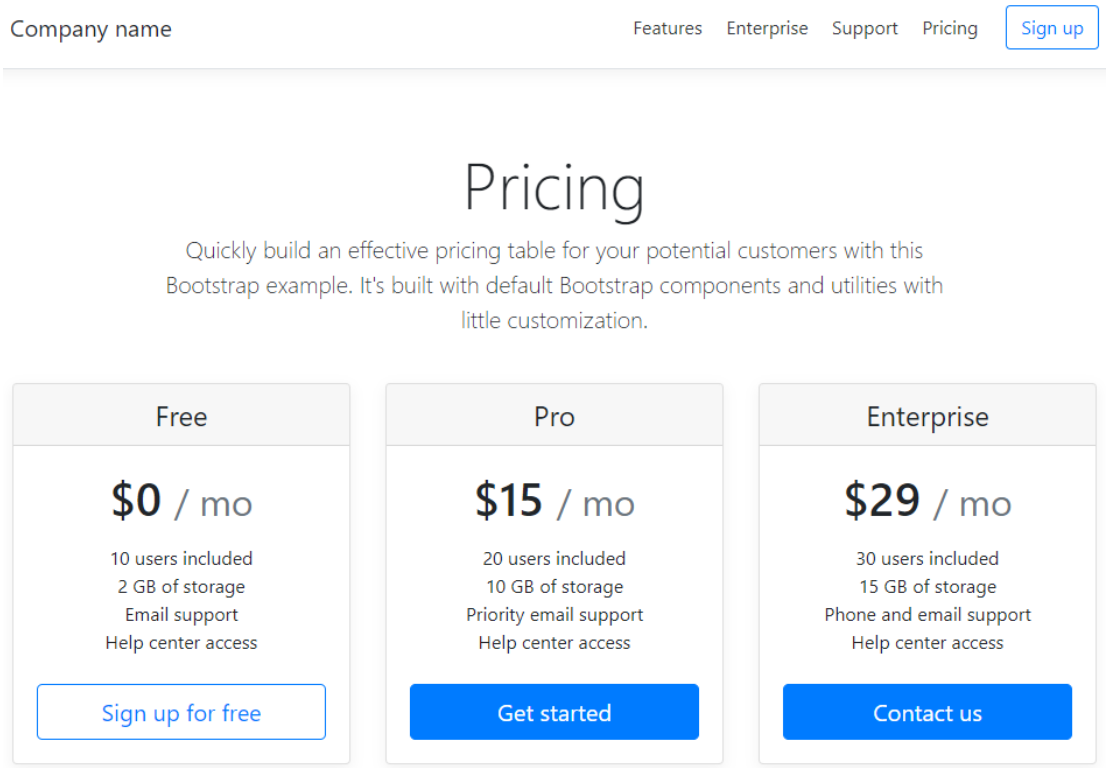
Výhodou komponent je dynamika. Nemusíme zasahovat do DOM například přes jquery kdykoliv, když budeme potřebovat udělat změnu. Místo toho React ve svých třídách používá render metodu, která se vždy při změně stavu znova načte, což při dobrém návrhu a nezávislosti tříd neovlivní žádné jiné třídy. [13]

V neposlední řadě bych se rád zmínil o states (stavech) a props (vlastnostech). Props jsou využívány pro statické vlastnosti komponent. Po celou dobu se totiž nemění. Opakem je pak State, který slouží pro dynamické vlastnosti a může se po dobu běhu měnit pomocí setState. Při změně se zavolá opět funkce render, která překreslí komponentu. [14]

3.3.2 Bootstrap

Bootstrap je jednoduchý nástroj pro vytváření designu webových stránek. Nejen svými předpřipravenými komponenty, ale i například responzivním mřížkovým systémem, ulehčuje práci a šetří čas developerům. Velkou výhodou je responzivní design, který se dokáže přizpůsobit jak počítačovým obrazovkám, tak tabletu nebo mobilním zařízením. To vše bez nutnosti, aby designér ovládal CSS kód.

Stránka je obalena kontejnerem, který se většinou skládá z několika řádků a sloupců. To vytvoří jakousi mřížku, která se v případě zmenšení obrazovky přeskládá pod sebe. Bootsrap nám dále nabízí celou řadu různých předdefinovaných komponent. Jako příklad bych uvedl navigační panely, rozbalovací nabídky, modální okna, systém formulářů nebo celou řadu pěkně designovaných tlačítek.][15]



Obrázek 3.3 Příklad stránky napsané čistě v Bootstrapu [16]

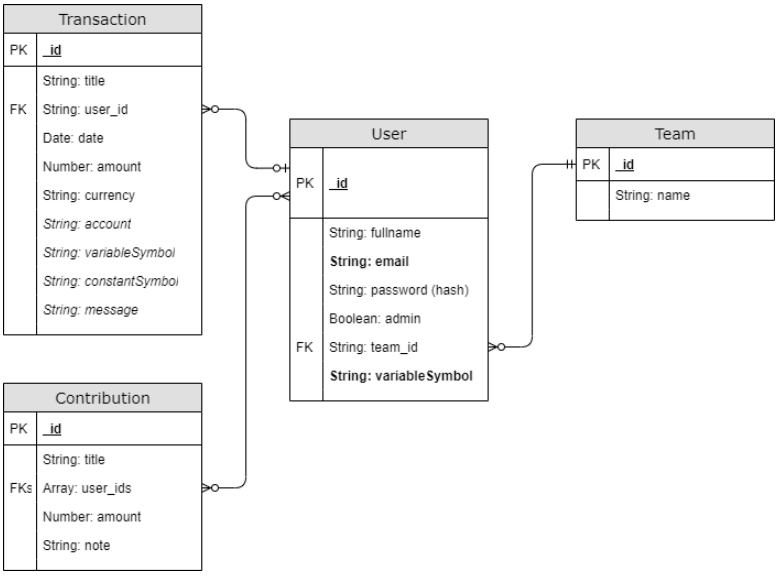
Na obrázku 3.3 můžeme vidět stránku využívající Bootstrapu. Na ní vidíme hned několik komponent. Například nahoře se nachází navigační panel, také si můžeme všimnout karet, které obsahují informace o ceně s tlačítky. V neposlední řadě zde nalezneme všechen text, který je podpořen typologií z Bootstrapu.

Kapitola 4

Implementace aplikace

4.1 Serverová část aplikace

4.1.1 Popis databáze



Obrázek 4.1 Naznačení dokumentových knihoven a relací

Na obrázku 4.1 je nastíněna databáze, která bude ukládat hodnoty o týmech, hráčích a jejich transakcích s příspěvky. Každý dokument obsahuje svůj unikátní identifikátor (_id).

Srdcem celé databáze je uživatel, který nese informace o jméně, týmu, uživatelském přihlášení a variabilním čísle, podle kterého se systém snaží spárovat transakce. Tučně vyznačené vlastnosti jsou povinné a indexované.

Další si popíšeme transakce nesoucí v sobě informace o platbě, názvu a datu provedení. Napojení je uskutečněno vlastností *„user_id“*. Teoreticky bychom mohli použít i variabilní číslo. Tohle řešení by nutilo propsání v případě změn u uživatele a transakce by pak ztratila další informaci shodující se s bankovní transakcí.

Poslední zajímavé dokumenty nesou příspěvky uživatele. Opět zde máme titul, částku a komentář. Napojení na uživatele je realizováno pomocí pole uživatelských identifikátorů, tedy *„user_ids“*. Jak můžeme vidět, neexistuje přímá relace mezi transakcí a příspěvkem. Důvodem je možné hromadění příspěvků, ke kterému dochází například koncem roku v případě turnajů, nedoplatků či pokut. Pokud bychom chtěli takovou relaci, museli bychom po hráči požadovat vytvoření právě jedné platby pro každý příspěvek.

4.1.2 Struktura

```
server
├── banks ..... obsahuje řešení pro komunikaci s bankou
│   └── fiobank.js ..... knihovna přizpůsobená pro komunikaci s Fio bankou
├── database
│   ├── models ..... implementace databázových modelů
│   └── database.js ..... komunikace s MongoDB
├── node_modules
├── .env ..... nastavení prostředí
├── .gitignore
├── package.json ..... použité balíčky
├── package-lock.json
└── server.js ..... hlavní kód, API, inicializace
```

4.1.3 Nastavení .env

Soubor .env v sobě ukrývá informace o prostředí, měl by být tajný, neboť v sobě ukrývá citlivá data, jako je například token pro přístup k bance nebo přihlašovací údaje k databázi. Popíšeme si všechna nastavení:

Tabulka 4.1 Popis .env vlastností

BANK_TOKEN	Token vygenerovaný zodpovědnou osobou. Viz kapitola 2.5.1.
DB_USER	Příhlášení uživatele do databáze.
DB_PASS	Heslo k příslušné databázi.
SALT	Číslo nastavení hashování. Čím větší číslo, tím víckrát hashování proběhne.

4.1.4 Dostupné API

API se skládá ze tří hlavních částí. Těmi jsou databáze, banka a přihlašovací systém. API vždy začíná /_api a dále už následuje cesta k potřebné komunikaci. Odpovědí je pak JSON objekt, který v sobě nese příslušná data a boolean **success** indikující, zda přenos proběhl správně. Dostupné cesty jsou zobrazeny v obsahu pod odstavcem.

```
/_api
├── /bank
│   ├── /lasttransactions
│   └── /transactions
├── /database
│   ├── /contributions
│   ├── /add-contribution
│   ├── /update-contribution
│   ├── /teams
│   ├── /add-team
│   ├── /update-team
│   ├── /transactions
│   ├── /add-transaction
│   ├── /update-transaction
│   ├── /delete-transactions
│   ├── /users
│   ├── /add-user
│   └── /update-user
├── /login
├── /logout
└── /isLoggedIn
```

4.1.4.1 Banka

API, které začíná **/_api/bank**, se týka čtení transakcí z bankovního účtu. Komunikace tohoto typu jsou dvě. V případě kladné odpovědi vrací vnořený objekt **data** s transakcemi.

GET /_api/bank/transactions nám vrátí všechny transakce. Můžeme filtrovat parametry **from** a **to** s datem ve formátu **yyyy-mm-dd**.

GET /_api/bank/lasttransactions nám vrátí poslední transakce. Pokud chceme získat transakce od nějakého data, přidáme parametr **date** s hodnotou data ve formátu **yyyy-mm-dd**. Pokud pošleme žádost bez parametru, získáme transakce od posledního stažení (zarážky). V obou případech se nastaví nová zarážka.

Výpis kódu 4.1 Příklad bankovních API

```
1 //transakce od 25. 3. 2021 do 25. 4. 2021
2 GET /_api/bank/transactions?from=2021-03-25&to=2021-04-25
3
4 //posledni transakce od 10. 4. 2021.
5 GET /_api/bank/lasttransactions?date=2021-04-10
```

4.1.4.2 Databáze

Naše další APIs se váže ke komunikaci s databází. Cesta k němu začíná **/_api/database**. Dovoluje nám číst, přidávat a upravovat dokumenty v naší databázi. Jedná se o týmy, uživatele, příspěvky a transakce. Protože se zde funkce opakují, jen se vždy jedná o jinou entitu, zavedeme

si pojem **document**, který může být jakýkoliv jeden prvek z množiny **{contribution, team, transaction, user}**.

GET /_api/database/(document)s nám vrátí všechny dokumenty. Můžeme filtrovat pomocí parametru **id**. V případě transakcí a příspěvků můžeme filtrovat i pomocí **userid**, výsledkem jsou pak dokumenty, které mají relace se chtěným uživatelem.

POST /_api/database/add-(document) vytvoří dokument v požadované knihovně. Objekt je předáván v těle žádosti. Jediná unikátnost nastává v případě uživatele, kdy je heslo před uložením zašifrováno pomocí knihovny node.bcrypt.js. Odpovědí je pak opět výsledek a vytvořená data nebo zpráva o chybě.

POST /_api/database/update-(document) upraví dokument v požadované knihovně. V parametru musí být **id**, podle kterého najdeme chtěný dokument. Žádané vlastnosti k úpravě předáme v těle žádosti. Odpověď je stejná jako při vytváření.

DELETE /_api/database/delete-transactions slouží jen pro testování funkcionality pro zapsání dat z banky do databáze. Pomůže nám vymazat celou knihovnu transakcí. Při implementaci se toto API nejspíš vypne nebo upraví, aby reagovalo na přiložené id a zamezilo se tak ztrátám.

Výpis kódu 4.2 Příklad databázových API

```
// transakce ktere patri uzivateli s id 607
GET /_api/database/transactions?userid=607

// pridame tym se jmenem Juniorka
POST /_api/database/add-team
body: {
  name: "Juniorka"
}

// upravime cenu a uzivatele prispevku s id 789
POST /_api/database/update-contribution?id=789
body: {
  user_ids: ["user001", "user004"],
  amount: 500
}

// smaze vsechny transakce
DELETE /_api/database/delete-transactions
```

4.1.4.3 Systém přihlašování/odhlašování

Tato komunikace bude zajišťovat přihlášení, odhlášení a kontrolu cookies.

POST /_api/login slouží jako přihlášení hráče. Tělo musí obsahovat email a heslo. Po přijetí najdeme uživatele a podle hesla porovnáme pomocí knihovny node.bcrypt.js. Výsledkem je pak id, email a informace, zda je uživatel admin. V případě chyby, jako je špatný email nebo heslo, je odpověď uložena v msg. V neposlední řadě je vytvořen cookie **ClubAppSessionID**.

POST /_api/logout odhlásí uživatele. Uvolní cookies a vrátí pozitivní odpověď.

POST /_api/isLoggedIn zkontroluje validitu cookie. Pokud cookie je v pořádku, vrátí kladnou odpověď a informace o přihlášeném uživateli.

Výpis kódu 4.3 Příklad přihlašovacích API

```
// prihlaseni
POST /_api/login
body: {
  email: "admin@gmail.com",
  password: "tajneheslo123456"
}

// odhlaseni, vycisti cookie
POST /_api/logout

// kontrola zda uzivateli nevyprsel login
POST /_api/isLoggedIn
```

4.2 Klientová část aplikace

Klient bude moci pomocí této aplikace ovládat data. Implementovány jsou operace pro vytváření, úpravu a zobrazení položek. Smazání implementováno není, může být dosaženo přímo na portálu MongoDB. Dále se pokusí spárovat transakce k uživateli pomocí variabilního symbolu a poté mu ukáže přehled o platbách s bankovním převodem.

4.2.1 Struktura souborů

```
client
├── node_modules
├── public ..... nastavení vlastností stránky a ikony
├── src
│   ├── components ..... všechny použité komponenty
│   ├── helpers ..... pomocné funkce
│   ├── images ..... obrázky
│   ├── pages ..... stránky s komponentami
│   ├── stores ..... funkce pamatující si informace o přihlášeném
│   ├── App.js ..... root aplikace
│   └── App.test.js
├── .env ..... nastavení prostředí
├── .gitignore
├── package.json ..... použité balíčky
└── package-lock.json
```

4.2.2 Nastavení .env

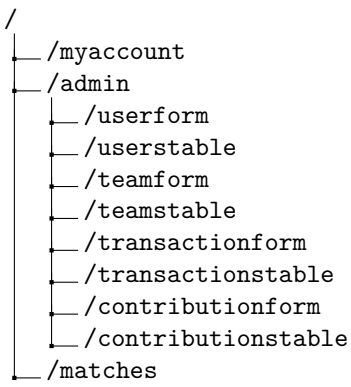
Stejně jako u serverové části aplikace, i zde najdeme nastavení prostředí. Výhodou Reactu je, že toto prostředí je už předdefinováno. U serverové aplikace jsme museli nainstalovat pomocnou knihovnu. Každé námi vytvořené nastavení musí začínat **REACT_APP_** a poté už si můžeme zvolit vlastní název. Volání v kódu pak vypadá následovně: process.env.REACT_APP_URL.

Tabulka 4.2 Popis .env vlastností

SERVER_URL	Url na domácí stránku serveru
IBAN	IBAN bankovního klubového účtu
BANK_ACCOUNT	Číslo bankovního účtu v českém formátu
Ostatní...	Zbytek se pak referuje na jednotlivá API

4.2.3 React Routes

Po přihlášení uživateli React Router ukáže navigační panel se stránkou odpovídající jeho lokaci. Na stránkách pod adminem se nachází správa databáze. Myaccount je viditelný hráčem a ukáže mu informace k jeho platbám. Matches by měly v budoucnu ukazovat nadcházející zápasy týmů.

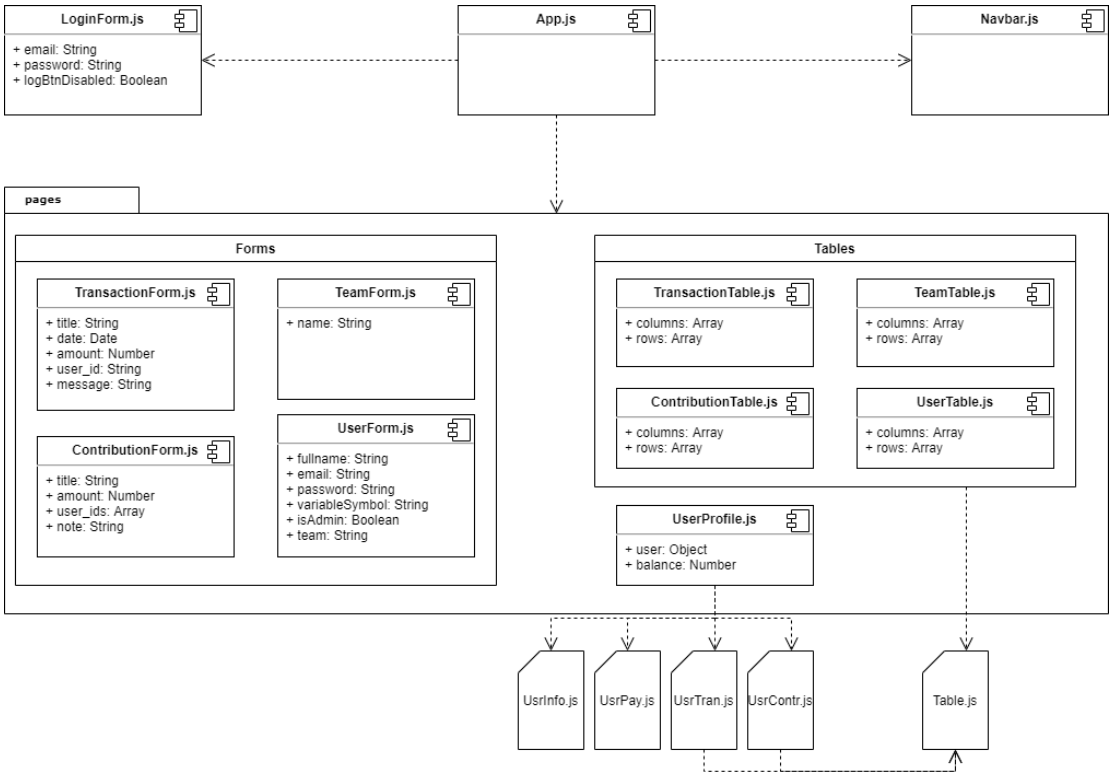


4.2.4 Popis získání posledních transakcí

Pro získání transakcí, které přibyly po poslední kontrole, zavoláme na server požadavek **GET /_api/bank/lasttransactions**. Tento požadavek nejenže vrátí transakce, ale také nastaví novou zarážku pro příští dotaz. V případě kladné odpovědi pak projedeme každou transakci a podle jejího variabilního symbolu se snažíme napojit na uživatele. Výslednou transakci uložíme do databáze. V případě špatného namapování přes variabilní symbol lze uživatele přidat v portálu pomocí **Admin > Transakce**. Funkce se nachází ve třídě Helper.

4.2.5 Popis důležitých komponent

V této sekci si popíšeme strukturu a vlastnosti podstatných komponent. Začneme od kořenové komponenty App, pak budeme postupně pokračovat níže.

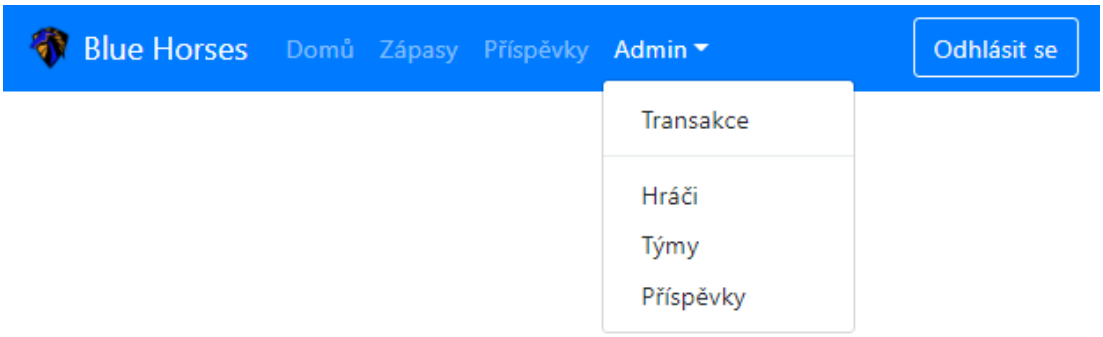


Obrázek 4.2 Struktura vytvořených komponent

4.2.5.1 Kořenová komponenta App

App je první komponenta, která se spustí. Zeptá se serveru, zda je uživatel přihlášen. Pokud ne, ukáže mu přihlašovací formulář, a pokud server vrátí pozitivní zprávu, pak mu naservíruje navigační panel se stránkou odpovídající React routeru. Kontrola přihlášení probíhá pomocí **POST /_api /isLoggedIn**.

4.2.5.2 Navigační panel



Obrázek 4.3 Navigační panel

Na panelu se může uživatel navigovat stránkami. Najde zde i tlačítko pro odhlášení, které pošle žádost na server **POST /_api /logout**. Navigační panel je použit z knihovny Bootstrap s menšími úpravami. Odkazy jsou pak nastaveny pomocí Link z knihovny react-router-dom. Poslední zajímavostí je znak modrého koně, který i s názvem patří stochovskému florbalovému klubu, jemuž by měl portál sloužit.

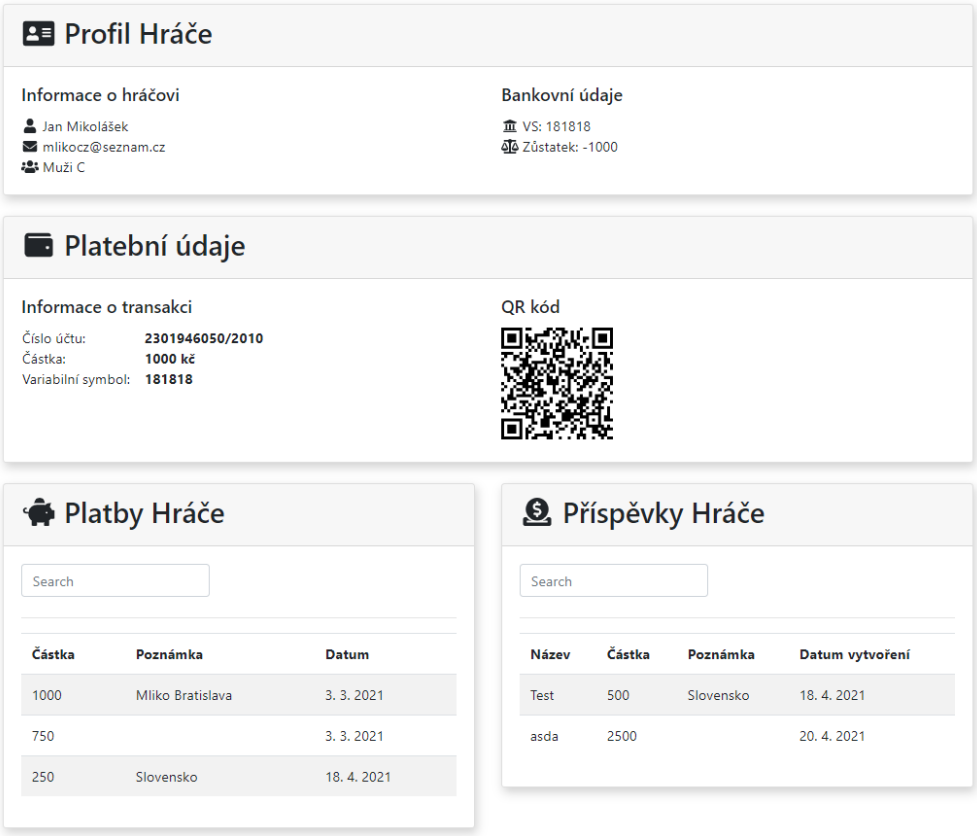
4.2.5.3 Stránky

```
1  import React, {Component} from 'react';
2
3  import UserProfile from '../../components/userprofile/UserProfile'
4
5  class UserProfilePage extends Component {
6    render(){
7      return (
8        <>
9          <UserProfile/>
10       </>
11     );
12   }
13 }
14
15 export default UserProfilePage;
```

Obrázek 4.4 Příklad kódu stránky

Stránky se nacházejí v **/src/pages** a slouží jako jakýsi obal pro komponenty. Neměly by se v nich provádět žádné funkcionality. Tyto stránky jsou volány z React routeru z App.js. Všechny jsou v podstatě stejné, nejdříve si importujeme důležité komponenty a poté je vykreslíme.

4.2.5.4 Profil Hráče



Obrázek 4.5 Ukázka profilu hráče

Příslušné komponenty se nacházejí v **/src/components/userprofile**. Z pohledu počtu komponent je profil hráče nejnáročnější. Ukrývají se v něm hned čtyři a jsou od sebe dobře rozeznatelné díky kartám z knihovny Bootstrap. Pojdmě si je popsat.

Profil Hráče obsahuje informace o hráči a v něm jméno, email a tým hráče. Dále v něm nalezneme jeho bankovní údaje, ve kterých se uživatel dozví o variabilním čísle a zůstatku.

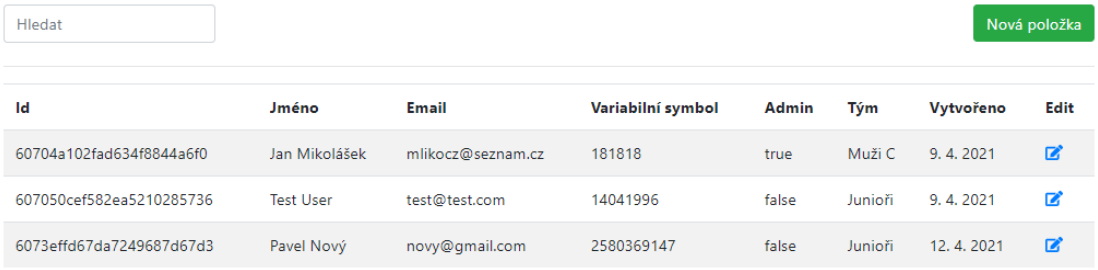
Platební údaje zobrazují informace pro bankovní převod. V případě potřeby lze využít QR kód, který je vytvořen podle následujícího řetězce:
SPD*1.0*ACC:{IBAN}*AM:{Amount}*CC:CZK*MSG:{Email}*X-VS:{VariableSymbol}.

Platby a Příspěvky Hráče obsahují položky, které jsou k nim přiřazeny.

Tabulka 4.3 Popis QR řetězce

SPD, 1.0	Zahajovací fixní hlavička
ACC	IBAN účtu uložený v **./.env**
AM	Obsahuje částku, kterou musí hráč doplatit
CC	Měna
MSG	Zpráva pro příjemce, do které pro informaci uložíme email
X-VS	Variabilní symbol

4.2.5.5 Tabulky

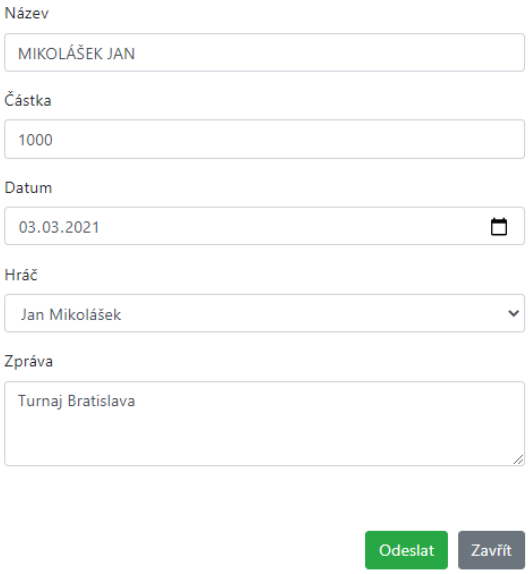


Obrázek 4.6 Tabulka obsahující hráče

Tabulky jsou vykreslovány pomocí knihovny react-bootstrap-table2. Tato knihovna nám umožňuje vytvářet tabulky bez složitého nastavování stylů a pomocí předdefinovaných tříd si je můžeme přizpůsobit. V případě vytvoření nového dokumentu můžeme využít tlačítka **Nová položka** vpravo nad tabulkou. Pokud pak chceme editovat, u příslušné položky klikneme na tlačítko ve sloupci **edit**. Obě tato tlačítka nás zavedou do příslušného formuláře.

V případě, že chceme například zobrazit název týmu hráče, nastává nám problém, protože dokument hráče obsahuje pouze identifikátor týmu. Proto před vykreslení dat vytvoříme mapu, která má jako klíče _id a jako hodnoty názvy dokumentů, jenž pak zobrazíme.

4.2.5.6 Formuláře



Obrázek 4.7 Formulář pro vytvoření a upravení transakce

Všechny styly formulářů jsou vytvořeny pomocí knihovny Bootstrap. Každá knihovna používá stejný formulář jak pro vytvoření, tak pro editaci. V případě vytváření nové položky je postup jednoduchý. Zobrazí se nám prázdný formulář, který po vyplnění můžeme odeslat serveru na zpracování.

Pokud chceme položku editovat, musíme to formuláři nějak sdělit. V našem případě použijeme parametr v URL se jménem id. Příklad cesty k formuláři na obrázku 4.7: /admin/transaction-form?id=607ac767fc7c6c1ac4e5a6ca. V tomto případě se při inicializaci formuláře přečte hodnota parametru a pomocí ní pak vyhledáme dokument, který uložíme do stavu komponenty. Při uložení pak na server zavoláme update funkci s parametrem id dokumentu.

Poslední zajímavostí je vytváření relací, které probíhá přes identifikátory. Ve formuláři samozřejmě nemůžeme ukázat několik _id řetězců, protože by administrátor při každém uložení musel koukat k jakému dokumentu se identifikátor páruje. Použijeme tedy opět mapování. Například do výběrového pole vložíme možnosti, které budou ukazovat název, ale ponesou si v sobě informaci o identifikátoru dokumentu.

Kapitola 5
Testování

Pro testování jsem zvolil programové unit testy, které pomůžou při aktualizaci a implementaci na jiné prostředí. K získání zpětné vazby mi pomohl rozhovor s vedoucím klubu, pro který je aplikace vyvíjena.

5.1 Programové testy serveru

Na serveru k testování používáme knihovnu Jest. Otestujeme zde připojení k databázi a k bance. Hlavním důvodem je ověření správného nastavení prostředí. Testy se nacházejí v souborech /banks/fiobank.test.js a /database/database.test.js.

Tabulka 5.1 Popis server testů

Database	
Connection	Otestování připojení na databázi
Transactions	Získání transakcí
Contributions	Získání příspěvků
Users	Získání uživatelů
Teams	Získání týmů
Bank	
Get transactions between dates	Funkce getTransactionsBetween pro zjištění transakcí mezi daty
Get last transactions	Funkce lastTransactions pro zjištění posledních transakcí
Get last transactions by date	Funkce lastTransactionsByDate pro zjištění transakcí od data
Without token	Zkoušení bankovní komunikace se špatným tokenem

5.2 Programové testy klienta

K testování klienta použijeme opět Jest, který díky použití Reactu nemusíme instalovat, protože je součástí základní instalace. Abychom plně otestovali klienta musí běžet server, neboť většina testů kontroluje právě komunikaci s ním. Opět se zde budeme snažit zjistit validitu nastavení prostředí. Protože se při implementaci bude nastavovat bankovní účet, na který budou posílány příspěvky, je připraveno mnoho testů pro validaci bankovních informací za pomoci knihovny iban.js.

Tabulka 5.2 Popis klient testů

Komunikace	
Pro testy komunikací je zde připraveno 18 testů, které testují všechna dostupná API. V případě POST žádostí je doplněn parametr test, abychom nevytvářeli nové dokumenty v databázi.	
Bankovní informace	
Correct IBAN format	Kontrola validity formátu IBAN
IBAN equals with bank account number	Souhlasí IBAN s bankovním číslem, které nám vrací API
IBAN equals with .env account number	Souhlasí IBAN s bankovním číslem, které je v nastavení prostředí
.env account number equals with the bank	Souhlasí naše bankovní číslo s tím, co nám vrací API

5.3 Prezentování vedoucímu klubu

Měl jsem možnost prezentovat řešení vedoucímu sportovního klubu, kdy jsem mu ukázal vytváření účtů, přiřazování transakcí a příspěvků. Postup i přehled v tabulkách přišel účastníkovi jednoduchý, jediná výtka přišla v podobě filtrování podle jednotlivých polí. Co se týče úpravy příspěvků, by uvítal vybírání hráčů nejdříve podle týmů a pak upravení jednotlivců. Debatu jsme také vedli ohledně transakcí, ve kterých nelze upravovat data nahraná z banky, jako je variabilní symbol nebo číslo protiúčtu. Mojí odpovědí bylo ponechání stávajícího řešení, pro jednodušší spárování druhou stranou (aplikace -> banka). S čímž vedoucí souhlasil.

Také jsme se bavili o profilu hráče, který se vedoucímu moc líbil, hlavně pak část s platebními údaji, kde největší radost zanechal QR kód. Administrátorovi by se také líbil lepší přehled dlužníků, nyní musí filtrovat obě tabulky. V poslední řadě jsme probírali budoucnost portálu. Bavili jsme o veřejné části, kde by se například nacházeli události a nábor nových hráčů.

Kapitola 6
Úprava aplikace

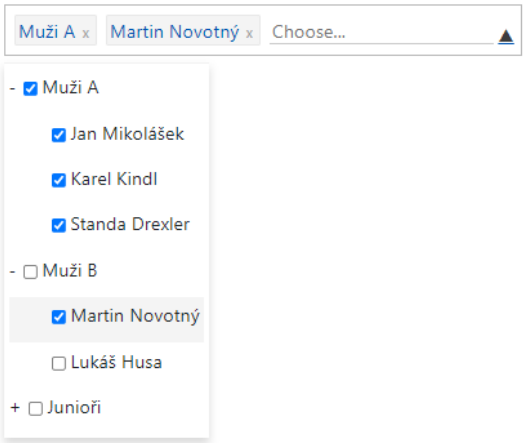
Na základě vlastní iniciativy a po komunikaci s vedoucím klubu jsem navrhnul pár změn a přídavků do budoucna. Protože se jedná o prototyp soustředící se hlavně na komunikaci s bankou, byly některé části zjednodušeny. Například bezpečnost okolo volání API, nebo zabezpečení cookies. Některé změny byly jednoduché a naimplementoval jsem je vzápětí po testování. Pojďme se podívat na seznam změn.

6.1 Moje návrhy:

1. **Bezpečnost API** - Nyní je API přístupné pro kohokoliv, kdo zná adresu. Můžeme například naimplementovat token v těle žádosti.
2. **Zlepšení loginu** - Součástí tohoto bodu bych rád designově zlepšil formulář. Dále je potřeba předávat více bezpečný cookies.
3. **Veřejná část portálu** - Naimplementovat veřejnou část portálu, kde by se nacházely události a zápasy.
4. **TimeJob pro nahrání nových transakcí** – Systém bude například každý den nahrávat nové transakce. Toto se lépe testuje na reálném serveru.

6.2 Návrhy vedoucího klubu:

1. **Filtrování hráčů podle týmů** - Jedná se o přiřazení hráčů ve formulářích. Zde jsem již navrhl změnu, která využije knihovnu react-dropdown-tree-select.



Obrázek 6.1 Filtrování hráčů podle týmů

2. **Filtrování tabulek podle polí** - Změna byla také zaimplementována. Viz obrázek 6.2.

Id	2018	Název protiúčtu	Částka	Uživatel	Variabilní	Datum	Edit
607ac767fc7c6c1ac4e5a6ca	201840372/0600	MIKOLÁŠEK JAN	1000	Jan Mikolášek	181818	3. 3. 2021	
607ac767fc7c6c1ac4e5a6cb	201840372/0600	MIKOLÁŠEK JAN	100			3. 3. 2021	
607ac767fc7c6c1ac4e5a6cc	201840372/0600	MIKOLÁŠEK JAN	750	Jan Mikolášek	181818	3. 3. 2021	

Obrázek 6.2 Filtrování tabulky podle sloupců

3. **Portál admina** - Zlepšit sledování pro admina, aby nemusel filtrovat obě tabulky, kdo již platil. Toto zlepšení bude určitě nutností před publikací.

Kapitola 7
Závěr

Cílem této práce byl návrh a implementace aplikace pro sledování finančních příspěvků pro sportovní kluby. Aplikace měla pomocí open banking API párovat transakce ke hráčům. Proto jsme potřebovali udělat rešerši o možnostech komunikace s bankovními institucemi. Poté navrhnout softwarové řešení, kde jsme předem věděli, že se aplikace bude skládat z klientské a serverové části. Dále jsme si stanovili, že server bude napsán pomocí JavaScriptového frameworku Node.js. Po rešerši jsme měli implementovat prototyp aplikace, kterou následně podrobíme vhodným testům. Na základě testování pak navrhnout změny a opravy aplikace.

Všechny tyto části jsme úspěšně splnili. V rešerši jsme se dozvěděli základní informace o Open bankingu a evropské směrnici PSD2. Podívali jsme se také na zavedení Open bankingu v České republice. Abych mohl správně implementovat komunikaci s Fio bankou, musel jsem si u ní musel založit účet. Prostředí a dokumentace od Fio banky byla velmi přívětivá.

Dále jsme si popsali použité technologie v naší aplikaci. Mezi ně patří MongoDB, NodeJS a React. Všechny tyto technologie podporují komunikaci pomocí JSON. React jsme zvolili, protože je v tuto chvíli nejpoužívanějšího prostředkem pro budování webových aplikací.

V implementaci jsme se zabývali tvorbou a vlastnostmi aplikace jako celku. Popsali jsme si například databázi v MongoDB, API na našem serveru či získání dat z banky a následné uložení do databáze. Snažili jsme se vytvořit aplikaci s co nejvíce nezávislými komponenty, tak abychom měli jednodušší vývoj. K designu jsme nejvíce využívali sady nástrojů od Bootstrapu.

Při testování jsme zkontrolovali správné nastavení prostředí. Díky reálnému využití jsme vedli rozhovor s vedoucím sportovního klubu a dostali tak první zpětnou vazbu. Na základě této zpětné vazby a vlastních poznatků, jsme portál upravili a navrhnuli pár změn.

Z reakce spoluhráčů jsem byl potěšen, neboť každý by měl rád přehled o platbách, který jim aplikace může dodat. Věřím, že s navrženými úpravami bude aplikace připravena pro testování mezi uzavřenou veřejnost a poté k uvolnění na produkci.

Příloha A

Programátorská příručka

A.1 Prerekvizity

Stažení Node.js. Pro plnou funkčnost doporučuji mít staženou minimální verzi v14.16.0. Vytvoření databáze v MongoDB.

A.2 První spuštění serveru

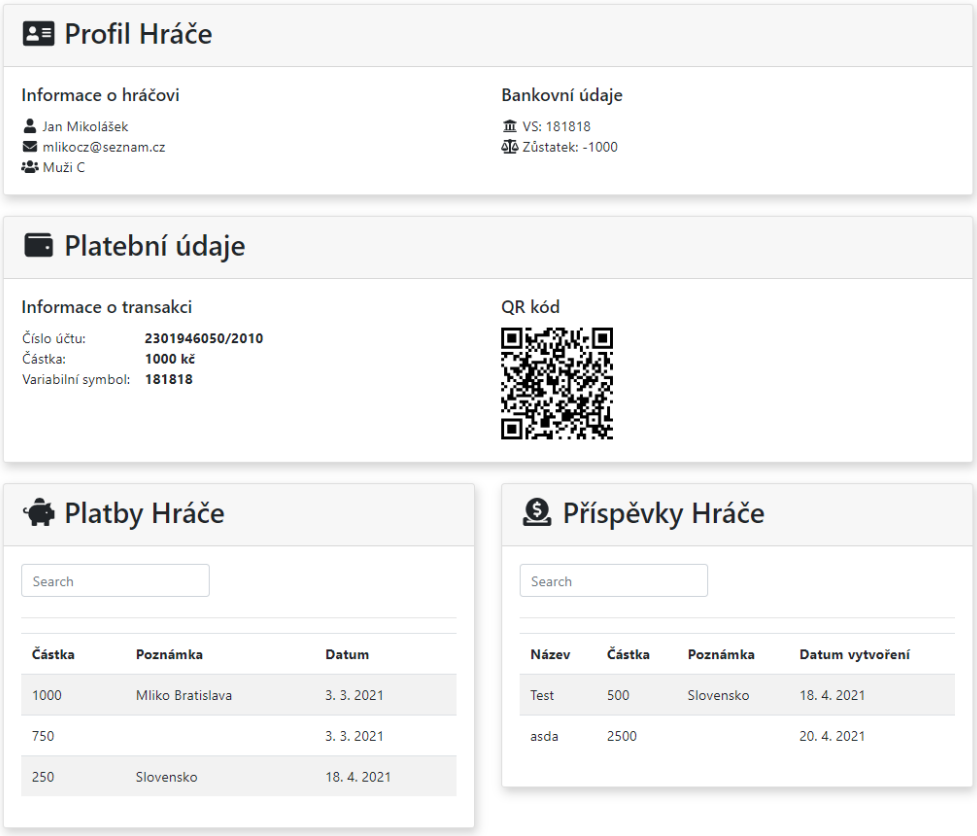
1. Otevřeme si příkazovou řádku a přejdeme do složky **./server**.
2. Před spuštěním nastavíme soubor **.env** pro naše prostředí.
3. Spustíme příkaz **npm install**.
4. Dále spustíme **npm start**. V případě správného nastavení se vypíše jen *„Conected to MongoDB“*.
5. Server necháváme běžet a pokračujeme spuštěním klientské části.

A.3 První spuštění klienta

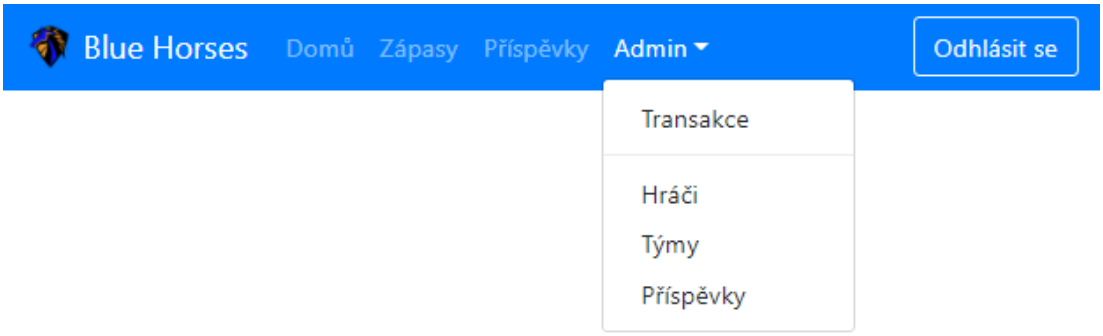
1. Spustíme nové okno s příkazovou řádkou a přejdeme do složky **./client**.
2. Opět nastavíme soubor **.env** a spustíme **npm install**.
3. Provedeme příkaz **npm start**, který se zeptá, zda chceme klienta spustit na jiném portu. Zodpovíme kladně (**y**).
4. Měla by se nám objevit hláška *„Compiled successfully!“* a otevřít portál.

Příloha B

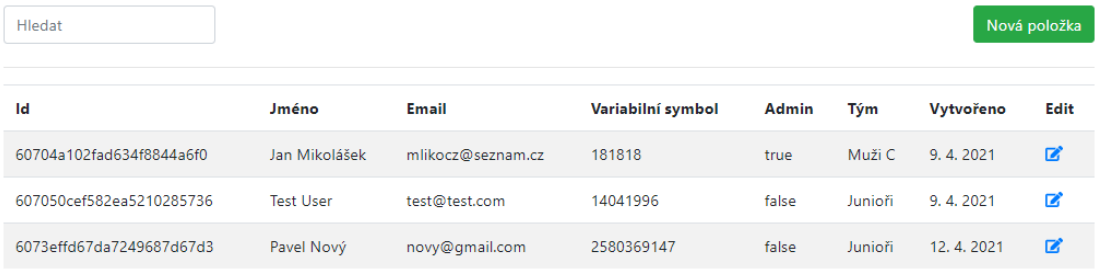
Ukázka porálu



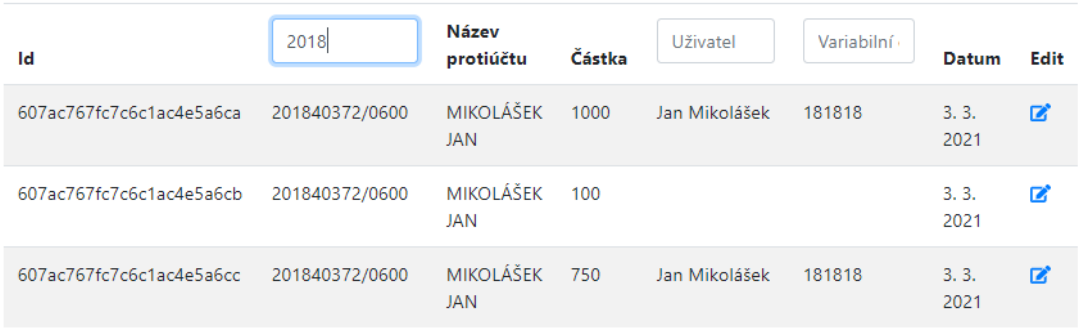
Obrázek B.1 Profil hráče



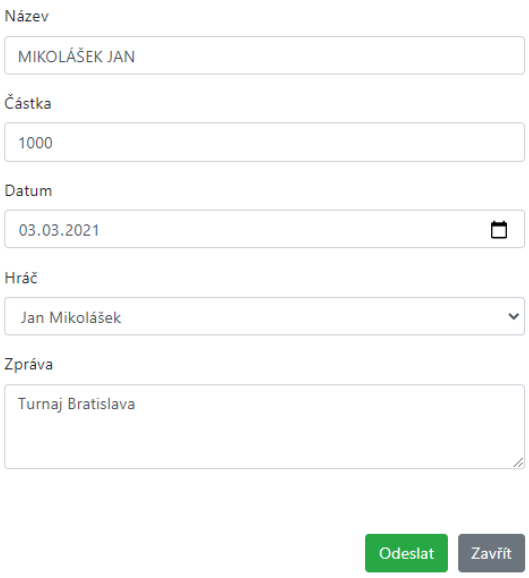
Obrázek B.2 Navigační panel



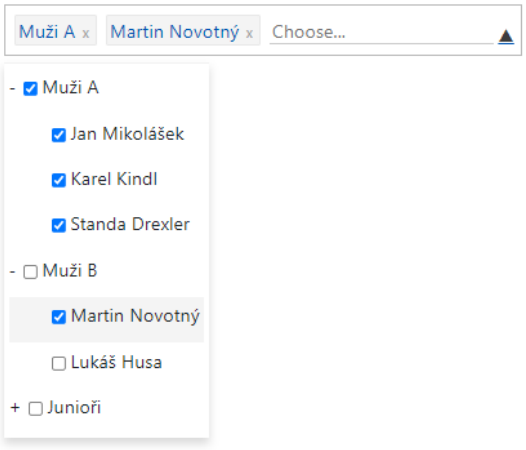
Obrázek B.3 Tabulka obsahující hráče



Obrázek B.4 Filtrování tabulky transakcí podle sloupců



Obrázek B.5 Formulář pro vytvoření a upravení transakce



Obrázek B.6 Filtrování hráčů podle týmů

Literatura

[1] mBank. *PSD2 - Payment Service Directive 2*. [online], [cit 25-04-2021]. Dostupné z: `https://www.mbank.cz/informace-k-produktum/info/jine/psd2.html`

[2] UniCredit Bank. *PSD2, revoluce v elektronických platbách*. [online], [cit 25-04-2021]. Dostupné z:
`https://www.unicreditbank.cz/cs/o-bance/PSD2.html`

[3] Václav Bartáček. *Představení PSD2 nejen pro vývojáře. Blíží se otevřené bankovnictví? Udělejte si v tom jasno.*. [online], [cit 25-04-2021]. Vydáno: 29. 1. 2019 Dostupné z:
`https://zdrojak.cz/clanky/psd2-nejen-pro-vyvojare/`

[4] Veronika Doskočilová. *PSD2 rok poté: multibanking umí 5 bank, API nezpřístupnila polovina*. [online], [cit 25-04-2021]. Vydáno: 24. 1. 2019 Dostupné z:
`https://www.mesec.cz/clanky/psd2-rok-pote-multibanking-umi-5-bank-api-nezpristupnila-polovina/`

[5] Česká bankovní asociace. *Czech Standard for Open Banking*. Verze 4.1. Vydáno: 25. 5. 2020 Platí od: 01. 11. 2020 Dostupné z:
`https://cbaonline.cz/cesky-standard-pro-open-banking`

[6] FIO API BANKOVNICTVÍ. *Fio banka*. [online]. Verze 1.7 Vydáno: 9. 4. 2021 Dostupné z:
`https://www.fio.cz/docs/cz/API_Bankovnictvi.pdf`

[7] Fio Internetbanking. *Fio banka*. Dostupné z: `https://ib.fio.cz/`

[8] Petr Sedláček. *Lekce 5 - Úvod do MongoDB*. [online], [cit 25-04-2021]. Vydáno: 29. 1. 2019 Dostupné z:
`https://www.itnetwork.cz/javascript/nodejs/uvod-do-mongodb`

[9] MongoDB. *The MongoDB 4.4 Manual*. [online], [cit 25-04-2021]. Verze 4.4 Vydáno: 30. 7. 2020 Dostupné z:
`https://docs.mongodb.com/manual/`

[10] Node.js. *Introduction to Node.js*. [online], [cit 3-05-2021]. Dostupné z:
`https://nodejs.dev/learn/introduction-to-nodejs`

[11] Jindřich Máca. *Lekce 1 - Úvod do Node.js*. [online], [cit 3-05-2021]. Dostupné z:
`https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs`

[12] json.org. *Introducing JSON*. [online], [cit 4-05-2021]. Dostupné z: `https://www.json.org/`

[13] Programming with Mosh. *What Is React (React js) & Why Is It So Popular?*. [YouTube video], [cit 4-05-2021]. Vydáno: 27. 6. 2018 Dostupné z: `https://youtu.be/N3AkSS5hXMA`

[14] React. *Tutorial: Intro to React*. [online], [cit 4-05-2021]. verze: 17.0.2 Dostupné z: `https://reactjs.org/tutorial`

[15] Alexandre Ouellette. *What is Bootstrap: A Beginner's Guide.*[online], [cit 4-05-2021]. Vydáno: 26. 3. 2021 Dostupné z: `https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/`

[16] Bootstrap Examples. *Pricing*. Dostupné z: `https://getbootstrap.com/docs/4.0/examples/`

Obsah přiloženého média

- `latexSourceCode` ..... zdrojová forma práce ve formátu LaTeX
- `server` ..... adresář s serverovou částí aplikace
- `client` ..... adresář s klientskou částí aplikace
- `thesis.pdf` ..... práce ve formátu PDF