Ponzi Bank

Technická zpráva k projektu do předmětu ITU
FIT VUT v Brně, 2020

Název týmu
Tým printf

Autoři
David Černý, xcerny74
Lukáš Baštýř, xbasty00
Sotirios Pupakis, xpupak01

Poznámky/pokyny:

- Může se stát, že otázky či doporučení v jednotlivých kapitolách nebudou sedět pro některá zadání. Upravte si tak, aby kapitola obsahovala, co má (třeba i vysvětlení, proč v dané kapitole není co psát).
- Sdělení ve společných částech TZ se chápe jako vyjádření všech členů týmu. Proto je nutno se na něm v týmu shodnout.
- Dodržujte připravené formátování a strukturování textu (pokud logika věci nevyžaduje jinak).
- Obrázky musí obsahovat popisek a musíte se na ně odkázat z textu.
- Pokud není explicitně uvedeno, můžete psát text společně a text je kolektivním dílem.
- Ačkoli jsou reference až na konci dokumentu, odkazy na ně doplňte zejména v první fázi řešení projektu, kdy provádíte průzkum a studium.

Obsah

1. Zadání a organizace týmu

1.1 Cíl

Cílem projektu je vytvořit systém pro transakce virtuálních tokenů které mohou sloužit jako forma měny. Výsledný systém bude obsahovat dvě klientské bankovní aplikace (aplikace pro správu zůstatků na účtech), API se kterou dané aplikace komunikují a rozhraní pro správu celého systému. Systém bude umožňovat transakce bez použití skutečné měny - např. platby mezi hráči nějaké hry nebo pro správu kapesného.

1.2 Tým

Tým se skládá ze tří studentů FITu.

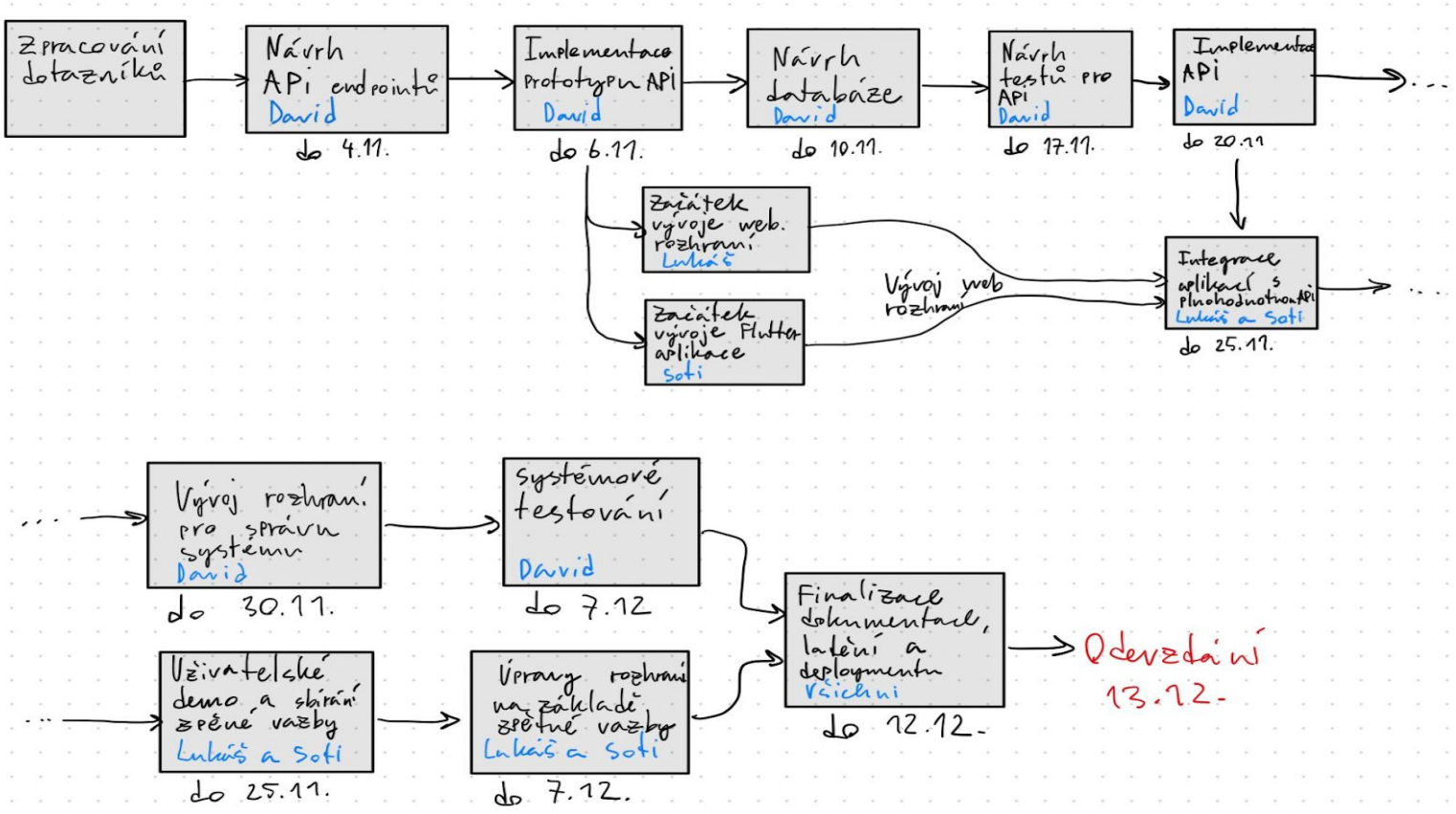
Složení:	David Černý	- David
	Lukáš Baštýř	- Lukáš
	Sotirios Pupakis	- Soti

David má hlavně na starosti API, návrh databáze a webového rozhraní pro správu systému.

Lukáš se podívá na webovou aplikaci pro uživatele, kde budou mít možnost kontroly účtu, posílání peněz a dalších bankovních operací.

Sotiho práce se skládá hlavně z tvorby mobilní aplikaci, která bude mít podobnou funkcionalitu jako webová aplikace pro uživatele.

1.3 Roadmapa



David

1. Zpracování dotazníků
2. Návrh API endpointů (do 4.11.)
3. Implementace prototypu API (do 6.11.)
4. Návrh databáze (10.11.)
5. Návrh testů pro API (10.11.)
6. Implementace API (20.11.)
7. Vývoj rozhraní pro správu systému (30.11.)
8. Systémové testování (7.12.)

Lukáš a Soti

1. Začátek vývoje webového rozhraní a Flutter aplikace (po implementaci prototypu API)
2. Integrace aplikací s plnohodnotnou API (do 25.11.)
3. Uživatelské demo a sbírání zpětné vazby (do 25.11.)
4. Úpravy rozhraní na základě zpětné vazby (do 7.12.)

Společné

1. Finalizace dokumentace, deploymentu a ladění

1.4 Rizika a opatření

1. Problém se setupem API a databáze na zařízení vývojáře (pro účely vývoje rozhraní)
    - **Řešení 1:** Dockerizace API a databáze, jednotný způsob spuštění skrze docker-compose
    - **Řešení 2:** Spuštění API na veřejné IP adrese pro účely vývoje
2. Nejasnosti se zadáním, neplnění závazků, atd.
    - **Řešení 1:** Sync se všemi členy týmu 1x týdně
        - Každý člen shrne jakou práci vykonal a jaké má plány na další týden
3. Nedokončený/nefunkční backend blokující vývoj
    - **Řešení 1:** vývojáři klientských aplikací dostanou přístup k maketě API která bude poskytovat dummy data aby mohli vyvíjet frontend nezávisle na stavu backendu

2. Průzkum a zkušenosti

2.1 Existující řešení

Inspiraci, jak vizuální tak funkční, jsme přebrali z několika bankovních aplikací pro banky (RB, KB, ČSOB).
Přínosem je lehký přístup k informacím o účtu. U mobilní aplikace je často problém “horšího” zabezpečení. Z aplikace, která je často zabezpečena pinem, je možné přímo posílat peníze. Na druhou stranu ale člověk musí mít přístup k telefonu.

Lukáš

RB webová aplikace

Webová aplikace pro RB[1] dostala v posledních letech vzhledový upgrade, který vedl k lepší přehlednosti základních informací o účtech. Na druhou stranu je nyní značně těžší pracovat s nastavením. Co se mobilní aplikace týče, ta je dělaná hlavně na rychlé QR platby a zjištění stavu účtu (ne/proběhlá platba, zůstatek, poslední transakce). Zabezpečení na webu je pomocí nové aplikace (něco jako google authenticator), která je vázána přímo na účet. Do mobilní je poté potřeba zadat pin nebo pomocí biometrie.

Soti

KB aplikace[2]

Prozkoumal jsem funkčnost mobilní banky od Komerční Banky. Zabezpečení je řešeno buď heslem, KB klíčem, nebo otiskem prstu. Tímto dojde k zjednodušení přihlášení, což je dobrý krok pro uživatele. Dále jsou v aplikaci jednoduše viditelné všechny důležité údaje o účtu a to hned na první obrazovce.

---

[1] "Raiffeisenbank: Banka inspirovaná klienty." https://www.rb.cz/. Datum přístupu: 1 lis. 2020.
[2] "Mobilní banka | Komerční banka." 28 čvc. 2016, https://www.kb.cz/cs/mobilni-banka. Datum přístupu: 1 lis. 2020.

David

ČSOB bankovní aplikace[3]

ČSOB mobilní bankovní aplikace se mi líbí svým čistým designem, má ale i své nedostatky - ne vždy je jasné kde najít klíčová nastavení (např. limity pro platby online), také je obtížné vytváření šablon pro platby.

Ledger Live[4]

Ledger Live není vyloženě bankovní aplikace ale aplikace pro správu kryptoměny, je velmi přehledná ale narozdíl od běžných bankovních aplikací jí zase chybí některé užitečné funkce - vzory pro platby, možnost poslat zprávu příjemci.

2.2 Uživatelské potřeby

Uživatelé budou hlavně lidé, kteří potřebují mezi sebou spravovat nějakou virtuální měnu.

2.3 Shrnutí

Z průzkumu vyšlo, že většině dotázaných stačí praktické informace o účtu (zůstatek a poslední platby) a možnost posílání peněz/tokenů. Co se mobilní aplikace týče, tam by uvítali QR scanner na rychlé platby.

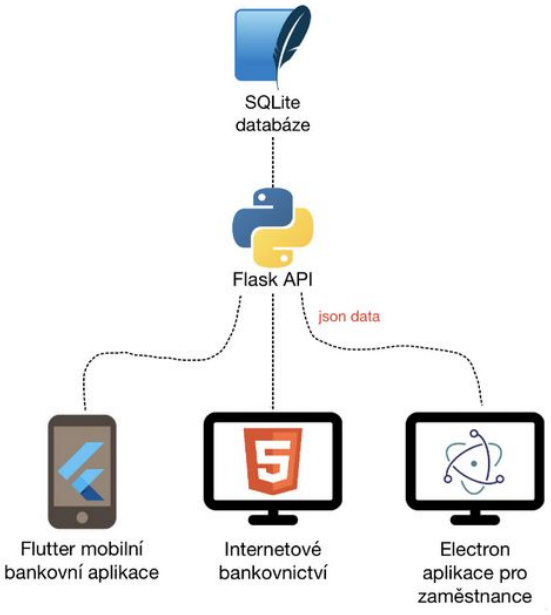
3. Architektura řešení

3.1 Architektura systému

Systém se skládá ze tří částí:

Backend

- **Flask API (Controller)**
    - Zpracovává požadavky zaslané z frontendu a poskytuje data zobrazovaná v aplikacích, pro tyto akce využívá databázi



[3] "Smartbanking, mobilní aplikace na online bankovnictví | ČSOB." https://www.csob.cz/portal/lide/ucty/internetove-a-mobilni-bankovnictvi/smartbanking. Datum přístupu: 1 lis. 2020.
[4] "Ledger Live : Most trusted & secure crypto ...." https://www.ledger.com/ledger-live. Datum přístupu: 1 lis. 2020.

    - Na úspěšné požadavky o přihlášení odpovídá vrácením JWT tokenu[5], který budou aplikace následně používat pro ověření API requestů
    - Veškerá komunikace mezi frontendem a API probíhá ve formátu *json*
- **SQLite databáze (Model)**
    - Ukládá všechna data v systému (informace o účtech, transakcích a zůstatcích)

Uživatelský frontend (View)

- Zobrazuje informace k jednotlivým účtům (zůstatky, historii transakcí, atd.) a umožňuje uživateli efektivně interagovat s API

Administrátorský frontend (View)

- Zobrazuje informace o provozu systému (seznam uživatelů, seznamy transakcí, informace o databázi a API, logy, atd.)

3.2 Architektura aplikace/í

Lukáš

Webová aplikace (React)

Pro webovou aplikace hodlám použít React (JS knihovna pro vývoj UI), díky které by měla aplikace fungovat na moderních/běžných prohlížečích. Vzhledem k tomu, že se jedná o webovou aplikaci, bude potřeba také použít HTML.
Samotná aplikace bude frontend (view) pro naše společné API (backend - controller). Jako model bude použita databáze sqlite.

Soti

Flutter mobilní aplikace

Mobilní aplikace bude psána ve frameworku Flutter, který zajistí stejné zobrazení napříč všemi bolními zařízeními. Flutter jako takový je stavěn na MVC architektuře, kterou budu tedy používat.

David

Rozhraní pro správu systému

Frontend aplikace slouží jako *View* zobrazující data získaná z API. Backend aplikace bude mít na starosti komunikaci s API - zpracování příchozích dat a vygenerování odchozích requestů. API slouží jako *Controller* (validuje správnost příchozích dat a komunikuje s modelem) SQLite databáze slouží jako model obsahující všechna data

3.3 Datový model

- **Data o klientovi**

[5] "JWT.io." https://jwt.io/. Datum přístupu: 1 lis. 2020.

    - Uživatelské jméno
    - Číslo/identifikátor účtu
    - Zůstatky pro každou měnu
    - Hash hesla
    - Typ účtu (administrátor/běžný uživatel)
    - Nastavení účtu (heslo, atd.)
    - Je účet zablokovaný?
- **Transakce**
    - Čas transakce
    - Měna transakce
    - Objem transakce
    - Zpráva příjemci
    - Odesílatel
    - Příjemce
    - Je transakce “anonymní”? (pokud to daná měna povoluje)
- **Nastavení systému**
    - Jak dlouho je platný JWT token?
    - Měny a jejich nastavení

API endpointy

- **/api/v1/** - verze api (v budoucnu může dojít k úpravám)

Veřejné endpointy

- **[POST] /api/v1/auth** - zaslání přihlašovacích údajů
    - V případě platných údajů vrátí endpoint JWT token
    - Odhlášení není třeba oznamovat serveru, stačí zahodit JWT token a pro opětovné přihlášení požádat server o nový

Zabezpečené endpointy

Při volání API není třeba určovat identitu uživatele, ta je získána z JWT tokenu.

Klientské

- **[GET] /api/v1/user_data** - získá základní data o uživateli (pouze pro aktuálně přihlášeného uživatele)
- **[GET] /api/v1/settings** - získá aktuální nastavení uživatele
- **[POST] /api/v1/settings** - změní aktuální nastavení uživatele (atributy které chceme změnit jsou v tělerequestu, ostatní zůstanou nezměněné)
- **[GET] /api/v1/balances** - získá zůstatky pro každou měnu
- **[GET] /api/v1/currencies** - získá seznam všech měn a jejich nastavení
- **[GET] /api/v1/<currency>/transactions** - získá historii transakcí pro danou měnu
- **[POST] /api/v1/send** - pošle transakci (informace o transakci jsou v těle requestu)

Administrátorské

- **[GET] /api/v1/users** - získá základní informace o všech uživatelích
- **[GET] /api/v1/users/<user_id>** - získá detailnější informace o uživateli

- **[POST] /api/v1/users/<user_id>** - změní nastavení uživatelského účtu (blokace, atd.)
- **[DELETE] /api/v1/users/<user_id>** - smaže uživatelský účet
- **[GET] /api/v1/transactions** - získá log transakcí (posledních n transakcí)
- **[GET] /api/v1/export/<target>** - exportuje požadovanou komponentu (logy, databázi)
- **[GET] /api/v1/import** - importuje databázi (nahradí stávající databázi)
- **[POST] /api/v1/<currency>/settings** - edituje nastavení měny (atributy které chceme změnit jsou v těle requestu, ostatní zůstanou nezměněné)
- **[POST] /api/v1/restart** - restartuje API a databázi

3.4 Vybrané technologie a implementace

Pro backend jsme se rozhodli použít *Flask* knihovnu a databázi *SQLite*. Pro autentizaci uživatelů budou použity *JWT (JSON Web Tokens)*. Pro rozhraní správy systému bude použit framework *Electron*. Pro webové rozhraní pro uživatele bude použit framework *React*.
Pro mobilní aplikaci bude použit framework *Flutter*.

4. Návrh GUI

4.1 Webová aplikace [xbasty00]

4.1.1 Požadavky na GUI

Možnost zobrazení informací o účtu, různá vyplňovací okna pro zadávání hodnot. Možnost upravit některá nastavení (například změna jazyka) - dropdown menu.

4.1.2 Makety

Interaktivní wireframe rozhraní, PDF verze wireframu

4.1.3 Pilotní test

Testy proběhly na běžných uživatelích pomocí zaslání makety. Následně jsem je požádal, jestli jsou schopni poslat na cizí účet prostředky ze svého účtu a následně si zobrazit historii pohybů na účtu.

4.1.4 Vyhodnocení testu a revize návrhu

Nikdo údajně neměl problém se splněním těchto zásadních, ale také nejdůležitějších úkolů. Jediné, co se uživatelům nelíbilo bylo nezarovnání horních ikon s hlavním menu barem. Jinak se uživatelům UI líbilo a neměli s ním problém.

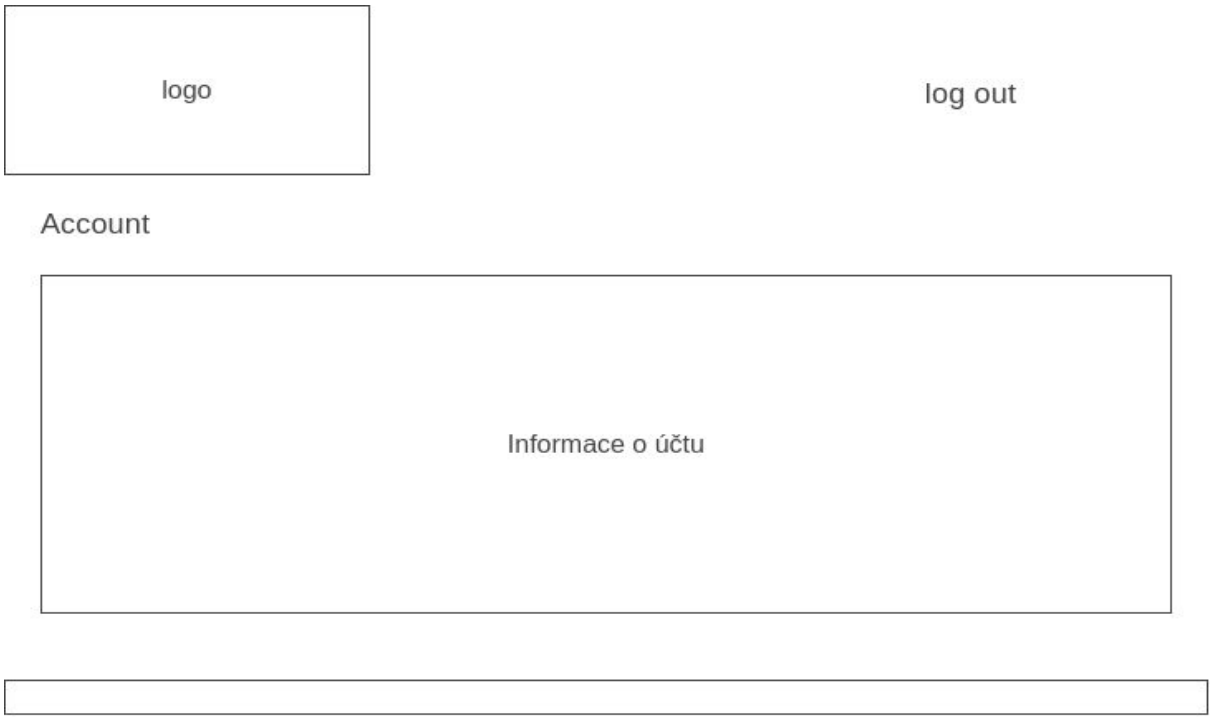
4.2 Mobilní aplikace [xpupak01]

4.2.1 Požadavky na GUI

GUI bude hlavně přehledně zobrazovat uživateli všechny informace o účtu, jako zůstatek, provedené transakce, číslo a další. Také bude umožňovat uživateli pohodlně zadat odchozí platbu.

4.2.2 Makety


lišta
informace o účtu + rychlé akce
Historie
Transakce
Tansakce
Transakce
Transakce
Transakce
Transakce



4.2.3 Pilotní test

Pomocí wireframů jsem na své rodině otestoval, jak se jim líbí rozložení a intuitivnost aplikace. Hlavně jestli dosáhnout na nejdůležitější prvky jednou rukou a nemusí si mobil přidržovat.

4.2.4 Vyhodnocení testu a revize návrhu

Všichni byli s návrhem spojeni a neměli žádné závažné připomínky. Dostal jsem pouze doporučení na barvy.

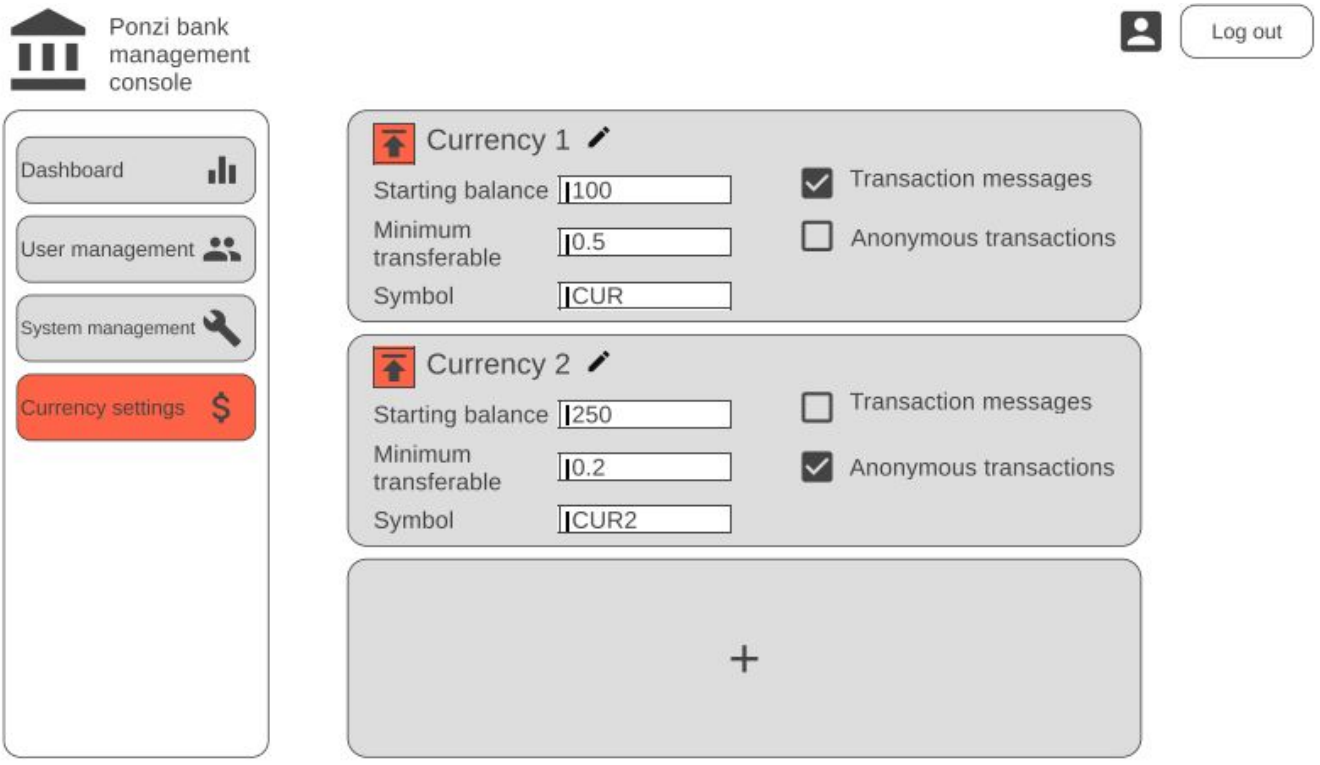
4.3 Rozhraní pro správu systému [xcerny74]

4.3.1 Požadavky na GUI

Administrátor by pomocí rozhraní pro správu systému měl mít možnost získat rychlý přehled o stavu systému: aktuálně vypisované logy, aktuální vytížení systému, atd. Dále by měl pomocí rozhraní být schopen rychle spravovat uživatele (přidávat, odstraňovat, upravovat uživatelské účty). Důležitá je i správa jednotlivých měn - v rozhraní bude možné přidávat různé virtuální měny/tokeny a nastavovat jejich chování - výchozí zůstatek dané měny při vytvoření účtu, nejmenší přenositelná částka, zda lze posílat transakce v rámci systému anonymně, zda je možné posílat zprávu příjemci, atd. Také bude v rozhraní možné exportovat a importovat databázi, celý systém restartovat a exportoval logy o provozu systému.

4.3.2 Makety

[Interaktivní wireframe rozhraní](), [PDF verze wireframu]()



4.3.3 Pilotní test

Pilotní test byl realizován na uživateli středního věku s mírně nadprůměrnými uživatelskými schopnostmi (používá matlab, je schopen nainstalovat OS), není to ale IT profesionál. Uživateli bez předchozích znalostí s rozhraním (viz. interaktivní wireframe) byly podány následující úkoly:

- **Přihlásit se do systému** - proběhlo bez problému
- **Přidat novou měnu** - taktéž proběhlo bez problému
- **Přidat nového uživatele** - uživatel rychle rozpoznal příslušné tlačítko, vše bez problému
- **Restartovat systém** - uživatel se místo restartování systému odhlásil (to bylo možná ale způsobeno špatnou formulací požadavku), na druhý pokus už úkol splnil bez problému

Uživateli se GUI celkově líbilo, nic by na něm neměnil.

4.3.4 Vyhodnocení testu a revize návrhu

Výsledky testu ukázaly, že rozhraní je dostatečně intuitivní na to, aby i uživatel bez zkušeností byl schopen vykonat zadané úkony - napomáhají tomu výrazná tlačítka doprovázená ikonami. V návrhu není potřeba v podstatě nic měnit, bude ale ještě zhodnoceno zda tlačítko pro restart systému neumístit přímo do hlavního okna aplikace.

Druhá část

---

5. Implementace GUI - aplikace XY (nebo část aplikace XY) [login autora]

Tato kapitola včetně podkapitol musí být vypracovaná každým členem týmu zvlášť, na jeho vlastní část GUI aplikace, kterou bude autorsky řešit.
Kapitola bude mít číslo pravděpodobně vyšší, podle počtu kapitol s Návrhem GUI (tj. podle počtu členů týmu).

5.1 Webová aplikace [xbasty00]

5.1.1 Implementace

Implementace proběhla prakticky v čistém JavaScriptu. S databází se komunikuje pomocí AJAXu (async await).
Volání funkcí je ve většině případů podmíněno nějakým vstupem od uživatele (psaní, kliknutí na tlačítko) -> volání funkce na změnu stavu -> znovu vykreslení komponenty.
V celém projektu je jedna velká datová struktura na uložení dat (username, token, atd…), kterou si jednotlivé komponenty předávají pomocí ‘headeru’. ‘Header’ je vlastní komponenta, která slouží jako navigační lišta na horní straně obrazovky. Klikáním na jednotlivé odkazy se vždy tato struktura pošle do nové stránky, která s ní může dále pracovat.
Jako jednu z posledních věcí jsem implementoval hodiny a odpočítávadlo do vypršení platnosti tokenu. Na to je použita použita React Hook ‘useEffect’, která se spustí, když se komponenta poprvé vytvoří. V ní je následně timer, který každou vteřinu aktualizuje čas. V ‘Login.jsx’ je tento ‘useEffect’ také použit a to na pingnutí serveru, když uživatel načte přihlašovací stránku.
Toto jsou jediné dvě místa, kde aktualizace komponenty nezáleží na uživateli.
Pro propojení všech jednotlivých stránek jsem pouzil React Router. Díky němu se dá mezi stránka jednoduše přepínat pomocí volání ‘History.push({pathname: “/stranka”, extradata:})’
Oproti mému původnímu návrhu zde chybí stránka ‘Settings’. Od té jsem nakonec upustil, jelkož mě nenapadlo, co by tam vlastně mohlo být. Všechna data jsou uložena pouze po dobu užívání aplikace a poté se zahazují. Tudíž by se žádná nastavení v budoucnu neaplikovala.

5.1.2 Použité nástroje a knihovny

Použil jsem knihovny/komponenty od Reactu na urychlení práce. Na sjednocení vzhledu jsem použil Bootsrap (React) theme.
Celý projekt je prakticky poskládán z “Container” a “Cards”.

“Container” se na obrazovce sám centruje tudíž již není třeba centrovat jednotlivé prvky pomocí ‘.css’ stylů. Následně se v něm dají dělat jednotlivé řádky a odstavce s různou velikostí a zarovnáním.
‘Card’ byly použity na vnitřní položky. Dá se u nich nastavit samostatné pozadí a obrys, který je například u historie transakci zbarven do zelena, pokud se jedná o příchozí platbu a nebo do červena, pokud se jedná o odchozí.
‘Modal’ byl použit na různá vyskakovací okna. Opět je v něm použit ‘Container’, aby se všesamo centrovlo.
Pro komunikaci s api jsem použil knihovnu ‘Axios’, což je HTTP klient založený na “slibech” (podobně jako druhé civčení z ITU). Mám zde ‘async’ funkci, ve které pomocí ‘try/catch’ a následně ‘await get/post’ čekám na odpověď od serveru. Pokud přijde chybový kód, vyskočí vyskakovací okno s chybovým hlášením. Pokud vše proběhne v pořádku, jsou data uložena a dále se s nimi pracuje
Na graf historie účtu jsem použil knihovna ‘react-chartsjs-2’ což je wrapper (balič?) pro ‘chartjs’, díky kterému se tyto grafy dali v reactu použít. pro výpočet dat je zde funkce, která vezme historii a projde ji pro danou měnu od začátku do konce a postupně přičítá/odečítá příchozí/odchozí hodnoty. Následně je podle toho graf vykreslení. Opět jsem zde použil barvy pro lepší vizualizaci.. Zelenou pro příchozí platby, červenou na odchozí platby.

Všechny použití knihovny i komponenty práci velice ulehčili. Pro mě byla hlavně největší výhoda, že nebylo téměř třeba upravovat ‘.css’ styly, protože to ‘Container’ za mě vycentroval, nebo Bootstrap theme za mě jednoduše upravil barvu.

5.1.3 Finální testování

Finální testování proběhlo na rodině a přátelích (přes internet). První verze byla celé bílá a velice statická. To vedlo k tomu, že se uživatelům aplikace zdála příliš statická.
Horní tlačítka na stránky také měla zakulacené rohy, na což většina uživatelů upozorňovala. Kdyby měli normální ostré rohy (tudíž na sebe přesně doléhali), bylo by to lepší.
Dále se nelíbila přihlašovací obrazovka, kde chyběl jakýkoliv text. Byl zde pouze formulář na přihlášení.
Nicméně i s těmito námitkami uživatelé neměli problém s používáním aplikace, tudíž se nemusel nijak upravovat funkcionalita elementů (pouze design).

5.1.4 Vyhodnocení testu

Finální testy byli velice užitečné. I přes “ošklivé” rozhraní uživatelé neměli problém s používáním aplikace, tudíž se nemusela nijak upravovat funkcionalita elementů (pouze design).
Díky zpětné vazbě jsem tedy rozhraní upravil. Nyní je v černé/tmavé barvě a je mnohem plnější. Přidal jsem vodorovné čáry jako “stínování” u některých elementů. Přihlašovací obrazovka nyní obsahuje “logo” a řádný text. Horní tlačítka na změnu stránek jsem předělal, aby měli ostré hrany.
Po všech těchto změnách se GUI již velice líbilo a nikdo již nepřišel s dalším návrhem na vylepšení/změnu.

5.1 Mobilní aplikace [xpupak01]

5.1.1 Implementace

GUI aplikace je implementovaná pomocí frameworku Flutter. GUI pracuje s kontrolery, kterým se říká provideři. Ti se starají o správu dat a upozorňují GUI, že je potřeba překreslení, pokud došlo k jejich změně. Dodržení MVC návrhu bylo kritické, jelikož celé GUI závisi na asynchroní práci s daty, která když se dokončí, tak upozorní GUI, že je potřeba zobrazit změny.

5.1.2 Použité nástroje a knihovny

Framework flutter byl použit jako základní stavební kámen, který se stará a celý chod aplikace. Dále jsem použil fluttertoast, který simuloval toasts z androidu, které jsou používal jako rychlý výpis při debuggingu. Provider package byl použit na správu dat a jejich ukládání. Package http byl spolu s asynchronním pouštěním použit pro komunikaci se serverem a získávání a posílání dat.
Pro emulování android zařízení jsem použil android emulátor, který je přibalen u Android Studia. Také jsem použil Android SDK, které jsem pomocí Android Studia stáhl. Jako vývojové prostředí jsem si vybral Visual Studio Code, které je jednou z možností, jak Flutter psát příjemně, jelikož jsou do něj dostupné pluginy přímo pro Flutter a Dart.

5.1.3 Finální testování

Testoval jsem po celou dobu vývoje, abych věděl, na co se zaměřit, co pozměnit a co nechat již ve fázi vývoje. Testování probíhalo hlavně na členech rodiny, jelikož momentální situace mi nedovolila osobní testování s dalšími lidmi. Testování probíhalo vysvětlením nových funkcionalit a zadání úkolů, které měl tester provést. Dělal jsem si poznámky, jak se tester choval, na co klikal a jaká chování od aplikace očekával.

5.1.4 Vyhodnocení testu

Po testu jsem se vždy s testrem vedl konverzaci na téma, co fungovalo jak očekával a kde očekával jiné chování aplikace. Poté jsem navrh možné řešení, které jsem s testrem probral. Pokud takové řešení vyhovovalo více testrům, tak došlo k jeho implementaci.

5.1 Rozhraní pro správu systému [xcerny74]

5.1.1 Implementace

GUI projektu je implementováno pomocí reaktivního frameworku *Vue.js*[6], jako kontejner pro webovou aplikaci slouží okno prohlížeče Electron[7], díky tomu je možné použít webové technologie při tvorbě desktopové aplikace. Aplikace je strukturována jako sbírka Vue komponent - komponenta je soubor obsahující šablonu pro renderování obsahu komponenty a kód který poskytuje funkcionalitu, o překreslení rozhraní po změně dat se stará samotný

[6] "Vue.js." https://vuejs.org/. Datum přístupu: 13 pro. 2020.
[7] "Electron." https://www.electronjs.org/. Datum přístupu: 13 pro. 2020.

framework Vue. Některé komponenty jsou použity jako views (tzn. jednotlivé obrazovky ve kterých se zobrazuje obsah), znovupoužitelné komponenty jsou zakomponovány do views. Každá komponenta obsahuje svá vlastní data nutná pro vykreslení, také může tato data poskytovat jiným podkomponentám. Aktualizace dat probíhá pomocí časovačů, které se při vytvoření komponenty aktivují a průběžně data asynchronně aktualizují z backendu pomocí knihovny Axios[8] pomocí Javascript mechanismu *promises*, po aktualizaci dat se view automaticky překreslí. Časovače se automaticky vypnou pokud se otevře jiné podokno aplikace kde daná data není potřeba aktualizovat. Globální data - např. instance třídy pro přístup k API a token přihlášeného uživatele se ukládají ve Vuex datové komponentě a jsou dostupné všem komponentám v aplikaci. O přepínání views se stará komponenta routeru.

5.1.2 Použité nástroje a knihovny

Při implementaci bylo použito poměrně hodně knihoven, asi hlavní knihovnou byl samotný *Vue.js*, tato knihovna je poměrně uživatelsky přívětivá a po prvotním seznámení už bylo jednoduché ji použít. Pro styly byl použit css framework *Bootstrap*[9], částečně pomocí samotných stylů a částečně pomocí pomocné knihovny *Bootstrap Vue*[10], která zjednodušuje používání některých Bootstrap prvků spolu s Vue frameworkem, např. integrace s komponentou pro paginaci (tzn. přepínání aktuálně zobrazené stránky v seznamech záznamů) nebo zobrazování upozornění v rozhraní. Pro zobrazení grafů byla použita knihovna *Chart.js*[11] spolu s pomocnou knihovnou *Chartkick* která zjednodušuje použití knihovny Chart.js. Pro generování avatarů (ikonek s iniciály uživatele) byla použita knihovna *Avatar*. Dříve zmíněná knihovna *Electron* sloužila pouze jako kontejner obsahující okno prohlížeče ale bez běžných ovládacích prvků které prohlížeč obsahuje, lze například nastavit minimální velikost okna prohlížeče.
I přesto, že je tento projekt prvním, který jsem realizoval pomocí webových technologií jsem nenarazil při práci na žádné zásadní problémy při implementaci. Automatické reaktivní aktualizace dat hodnotím velmi kladně, pouze rozdělení jednotlivých komponent na samostatné soubory má i nedostatky - ne vždy je jednoduché dostat příslušná data do cílové komponenty, obzvlášť když jsou data uložena o několik úrovní výše. Asi největší problém jsem měl s css, nakonec se ale podařilo aby bylo rozhraní správně rozložené a reagovalo i na velikost obrazovky.

5.1.3 Finální testování

Finální testování proběhlo se stejným testovacím subjektem. Vzhledem k tomu, že jsem se při implementaci poměrně striktně držel původního wireframu/makety, odpovědi byly také obdobné - všechny úkoly prováděné na původní maketě se podařilo zreprodukovat, pouze tlačítko pro restart systému nebylo podle původních připomínek přemístěno protože pro něj nebylo nalezeno lepší umístění.

---

[8] "axios/axios: Promise based HTTP client for the ... - GitHub." https://github.com/axios/axios. Datum přístupu: 13 pro. 2020.
[9] "Bootstrap." https://getbootstrap.com/. Datum přístupu: 13 pro. 2020.
[10] "BootstrapVue." https://bootstrap-vue.org/. Datum přístupu: 13 pro. 2020.
[11] "Chart.js." https://www.chartjs.org/. Datum přístupu: 13 pro. 2020.

5.1.4 Vyhodnocení testu

Celkově z finálního testu nevzniklo žádné nové zásadní zjištění, přístě by možná bylo lepší přehodnotit postranní lištu a umístit do ní více prvků, které musí být vždy viditelné - například dříve zmíněné tlačítko pro restart systému nebo informace o stavu systému.

6. Závěr

6.1. Závěr [xbasty00]

Stručně popište dosažený výsledek a hlavní zjištění z testování. Důležité uživatelské zkušenosti nepopisujte obecně (“uživatelům se to líbilo”), ale co nejvíce konkrétně (co a jak uživatelé použili, zvládli, pochopili, ocenili).
Zakončete krátkým shrnutím Vaší zkušenosti s prací celého týmu a Vaší roli. Co pro Vás bylo při práci v týmu přínosné (2-3 věci) a co byste příště udělali jinak (pouze 1 věc).
Kapitola bude mít číslo pravděpodobně vyšší, podle počtu kapitol s Návrhem GUI a s Implementací (tj. podle počtu členů týmu).

Jsem rád, že se mi povedlo dosáhnout tíženého cíle. Tudíž se hlavně naučit dělat webové rozhraní pomocí ‘ReactJS’. V druhé řadě samozřejmě nesmím zapomenout na vlastní webové rozhraní. K maketě nemá moc daleko, tudíž jsem zde také spokojen.
Uživatelé s jeho ovládáním většinou neměli žádný problém a pochopili moje důvodu pro odstranění některých věcí (‘Settings’). V posledním testování ocenili změnu tématu z bílého na tmavé. To přispělo k celkovému vyplnění rozhraní.
Práce v týmu byla probíhala dobře a nemám si zde na co stěžovat.
Moje role byla jednoduchá. Měl jsem udělat rozhraní pro bankovní aplikaci (jako každý člen týmu) -> Největší podíl na celé práci tudíž připadá Davidovi Černému, který sám udělal celé API, se kterým jsme poté pracovali.
Velkým přínosem pro mě bylo právě to, že se o API postaral David, který ho dělal v pythonu (flasku), jelikož již má bohaté zkušenosti s pythonem. Tudíž to pro něho bylo značně jednodušší, než kdybych ho dělal já.
Jedinou věc, co bych udělal jinak, je začít na projektu pracovat trochu dříve.

6.2. Závěr [xpupak01]

Výsledek tohoto projektu je aplikace, která komunikuje s API a zajišťuje hezké zobrazení dat získána právě z toho API. Jedná se o aplikaci, která simuluje jednoduchou bankovní aplikaci, kde hlavním účelem je posílat peníze, zobrazit si zbytek na svém účtu a podívat se na historii transakcí. Uživatelům vyhovoval způsob hlášení chyb vzniklých při komunikaci se serverem. Dále se jim líbila jednoduchost GUI a hlavně přizpůsobivost GUI na různé zařízení.
V týmu se mi pracovalo velice dobře, jelikož jsme si hned ze startu ujasnili, co budeme dělat, kdo co bude dělat a také podle toho navrhli strukturu celého repozitáře na verzovacím systému, abychom mohli pracovat na svých částech a nedocházelo ke konfliktům ve verzích. Osobně jsem se staral o mobilní klientskou aplikaci. Největší díky patří Davidovi,

který na sebe vzal vybudování API. Toto udělal rychle a pokud byly nějaké další požadavky na endpointy, či nalezení chyby, tak David zajistil rychlou nápravu, či doplnění.
Pro projekt jsem si vybral framework, se kterým jsem měl zkušenosti pouze ze stránky GUI, jelikož jsem v něm dělal pouze demo aplikaci bez reálných dat. Díky tohoto projektu jsem si vyzkoušel, jak Flutter pracuje s daty, jak se řeší komunikace se serverem a hlavně v něm udělal reálnou aplikaci, kterou můžu paté dále rozvíjet a základní codebase použít i dále, pokud dojde třeba k přepsání pro reálné využití.

6.2. Závěr [xcerny74]

Celkově dosažený výsledek mírně předčil moje očekávání, podařilo se implementovat všechny prvky z původně navržené makety a některé vlastnosti rozhraní byly i vylepšeny - např. paginace, vyhledávání uživatelů pomocí regexu, responzivní design a filtrovatelné grafy. Uživatelé ocenili zejména design rozhraní a poutavé grafy, celkově je ale systém zamýšlen spíše pro administrátory, není tedy nutné všechny prvky ovládání co nejvíce zjednodušovat. Trochu matoucí bylo umístění paginace pro tabulky nad samotnou tabulku, to má ale své opodstatnění - velikost tabulky se může měnit a ovládací prvek je lepší umístit nad samotnou tabulku aby bylo dobře viditelné.
S prací v týmu nebyl problém, naše grafická rozhraní byla nezávislá a mou rolí bylo implementovat backend, členové týmu backend hodnotili kladně, také mi pomáhali hledat v něm chyby. Líbily se mi občasné videohovory kde jsme si navzájem prezentovali naše projekty a sbírali feedback. V příštím projektu bych možná vybral téma které nevyžaduje tak rozsáhlý backend.

Reference

- https://flutter.dev/
- https://www.udemy.com/course/learn-flutter-dart-to-build-ios-android-apps/
- https://pub.dev/
- **Vue Js reaktivní framework** - https://vuejs.org/
- **CSS framework Bootstrap** - https://getbootstrap.com/
- React – A JavaScript library for building user interfaces (reactjs.org)
- https://react-bootstrap.github.io (react-bootstrap.github.io)
- Bootstrap · The most popular HTML, CSS, and JS library in the world. (getbootstrap.com)
- reactchartjs/react-chartjs-2: React wrapper for Chart.js (github.com)