Gymnázium, Praha 6, Arabská 14

Obor programování

ROČNÍKOVÝ PROJEKT

Tomáš Černý, Martin Havlíček, David Koňařík,
Michail Spiridonov

Informační systém pro školní jídelny

Duben 2019

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V ..... dne ..... Tomáš Černý .....

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V ..... dne ..... Martin Havlíček .....

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V ..... dne ..... David Koňařík .....

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V ..... dne ..... Michail Spiridonov .....

Název práce: Informační systém pro školní jídelny

Autoři: Tomáš Černý, Martin Havlíček, David Koňařík, Michail Spiridonov

Abstrakt: Cílem práce bylo vytvořit počítačový systém pro správu zařízení hromadného stravování určený pro zaměstnance i strávníky. Systém řeší tvorbu jídelníčku, správu seznamu strávníků a jejich kont a volbu jídel ze strany strávníků. Pro tvorbu systému byly využity unikátní techniky a vlastní knihovny, které lze dále využít pro další projekty.

Title: Information System for School Canteens

Authors: Tomáš Černý, Martin Havlíček, David Koňařík, Michail Spiridonov

Abstract: The goal of our project was to create a computer system for managing common canteens aimed at both staff and diners. It deals with creating a menu, managing a list of diners and their monetary accounts, and allows the diners to select food in advance. To create this system we used unique techniques and developed custom libraries, which can now be used for other projects.

Obsah

Úvod

Tato dokumentace a popsaný systém, Mocasys, jsme vytvořili jako skupinovou ročníkovou práci pro předmět Programování ve 3. ročníku. Volbu tématu ovlivnilo několik faktorů. Jeden z nich byla komplexita práce, která měla být adekvátní týmu čtyř kompetentních programátorů. Dalším faktorem byla využitelnost výsledného programu. Námi vytvořený systém pro jídelny může být, po malých úpravách pro jednotlivé provozovny, použit jako náhrada existujícího komerčního software.

Práci jsme pojali částečně i jako experiment v oblasti nástrojů a knihoven pro tvorbu webových aplikací. Vytvořili jsme databázový framework DASCore, který umožňuje jednoduše vytvářet databáze s ukládáním historie a ověřováním uživatelských oprávnění na úrovni databáze. Dále jsme vytvořili knihovnu liwec pro jazyk Scala, která umožňuje vytvářet grafická uživatelská rozhraní čistě v tomto jazyce bez zbytečné komplexity.

Zadání

Cílem ročníkové práce bude vytvoření informačního systému pro školní jídelny (případně další hromadná stravovací zařízení). Pracovníkům jídelny bude poskytovat možnost správy seznamu strávníků a jídel. Strávníkům bude umožňovat volbu jídla na konkrétní dny. Dle konzultací s potenciálními uživateli a dle analýzy existujících produktů bude zvolena další funkcionalita. Součástí ročníkové práce bude rovněž hardwarové řešení pro výdej jídel pomocí identifikace fyzickými tokeny.

1. Architektura

Mocasys se skládá z několika částí běžících na několika zařízeních.

- *Backend* je samotná databáze systému. Je postavený na PostgreSQL. Narozdíl od konvenčních návrhů posílají další vrstvy systému SQL příkazy přímo do Backendu přes Middleend.
- *Middleend* je vrstvou mezi ostatními částmi systému a Backendem. Řeší přihlašování a je dostupný jako webová REST služba.
- *Frontend* je samotná webová aplikace. Je určena pro zaměstnance jídelny i strávníky a běží přímo v prohlížeči jako JavaScript.
- Výdejová čtečka je hardware určený k poloautomatickému výdeji jídla. Po přiložení identifikátoru ukáže obsluze zvolené menu.

2. Backend

Kód Backendu je rozdělen na dvě části, samotný Backend Mocasysu a „DASCore“. DASCore poskytuje funkce pro řešení oprávnění přímo v databázi a pro vytváření tzv. temporálních tabulek – tabulek obsahujících i historická data o každé změně.

Samotný Backend definuje tabulky systému a oprávnění nutná k jejich používání. Tabulky jsou definovány standardní SQL syntaxí a pak jsou upraveny SQL funkcí.

Narozdíl od většiny systémů je vrstvám Mocasysu umožněn skoro přímý přístup k provádění SQL dotazů. Pro zajištění bezpečnosti bylo tak nutné implementovat různá opatření přímo v databázi a využít tak pokročilé funkce PostgreSQL.

Při každém dotazu z Middleendu je vytvořeno spojení s databází a je spuštěna funkce `session_user_set(user_id, secret)`, kde `user_id` je ID uživatele získané z session tokenu a `secret` je klíč použitý k odhlášení uživatele z daného databázového připojení a jeho navrácení do výchozího stavu. Tato funkce, napsaná v jazyce Perl, uloží ID uživatele do globální proměnné pro použití při ověřování práv.

Jednotlivé tabulky jsou pro potřeby uživatele databáze reprezentovány pomocí „views“, které mají nastavený atribut `security_barrier`, což zakazuje některé optimalizace a umožňuje tak použít `WHERE` klauzi k filtrování tabulky dle oprávnění uživatele.

Omezení úprav je realizováno pomocí `INSTEAD OF` triggerů, které při nedodržení podmínek zahlásí chybu a ukončí spojení.

Na verzované, temporální, tabulky je použito rozšíření PostgreSQL `temporal_tables` (viz Tem, 2019) a pomocné funkce, které k jedné tabulce vytvoří další pro ukládání historie a agregační view, které zobrazí stav tabulky v globálně určeném čase.

Tato funkcionalita je obsažena ve funkcích, jejichž použití lze ukázat na tabulce `food`:

```
-- Vytvoří tabulku, ve které budou uložena data
CREATE TABLE IF NOT EXISTS food_current (
    id serial PRIMARY KEY,
    name text NOT NULL
);

-- Vytvoří tabulku pro ukládání historie a agregační view
SELECT version_table('food');

-- Stanoví nutná oprávnění k čtení a úpravám tabulky
SELECT dascore_setup_table('food',
    select_perm := $$ perm('food.select') $$,
    modify_perm := $$ perm('food.modify') $$);
```

2.1 Databázové schéma

Databázové schéma bylo navrženo obvyklými technikami jako jsou „many-to-one“ a „many-to-many“ vztahy a někdy i vztahem „one-to-one“. Bylo rozhodnuto, že se oddělí uživatel (tabulka `users`), strávník (tab. `diners`) a osoba (tab. `people`), což má několik důvodů. Ne každý uživatel musí být strávník. Zároveň jedna osoba může mít více uživatelů, ale jeden uživatel může mít k sobě přiřazeného pouze jednoho strávníka. Příkladem budiž pracovník jídelny. Strávník může potenciálně být i třeba firma, nebo jiný subjekt, takže je tak nechán prostor pro rozšíření.

Jak již bylo zmíněno, tabulky mají historii. Každá tabulka, kromě tabulky, která ukládá data pro autentifikaci heslem, má suffix *current* a párovou historickou tabulku, která má suffix *history*.

One to One

- users ← user_passwords_data
- people ← diners

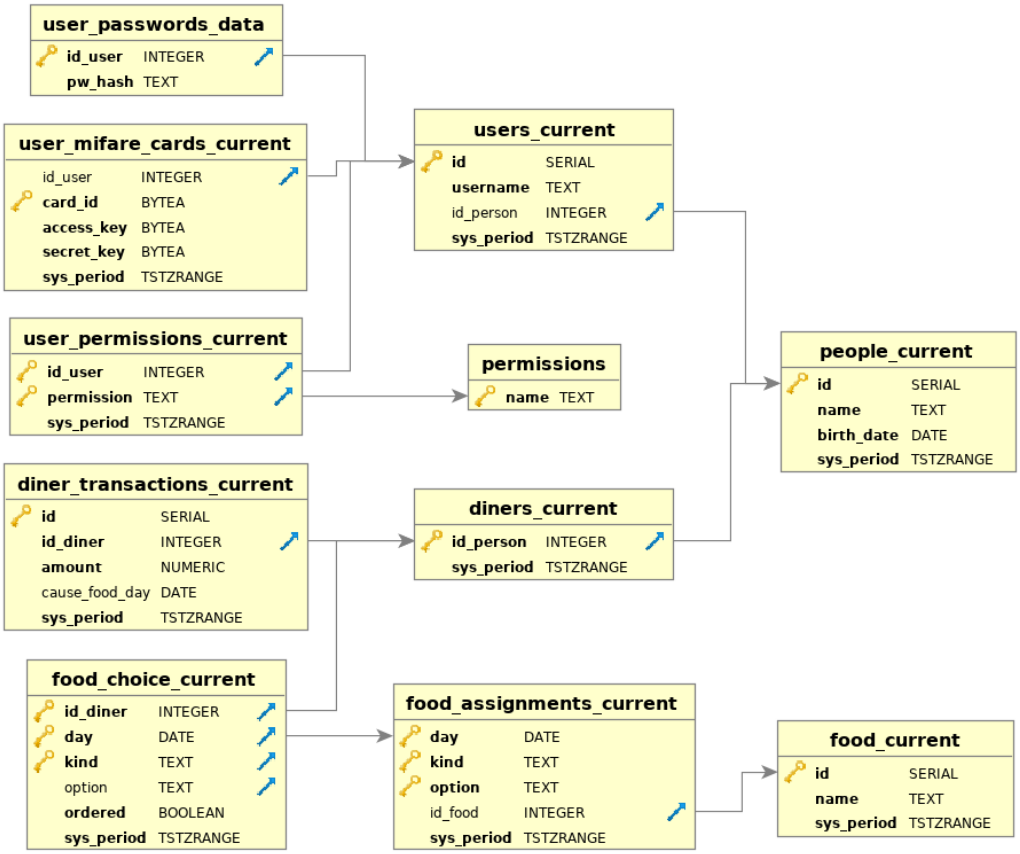
One to Many

- people ← users
- user_mifare_cards ← users
- food ← food_assignments
- diner_transactions ← diners

Many to Many

- diners ← food_choice → food_assignments
- users ← user_permissions → permissions

Na obrázku 2.1 můžeme vidět schéma s jednotlivými poli a jejich datovými typy.



Obrázek 2.1: Databázové schéma. Vytvořeno programem DBVis.

3. Middleend

Middleend je napsán v jazyce Typescript pro Node.js za pomoci RESTful frameworku Restify. Ekosystém Node.js byl zvolen kvůli své škálovatelnosti a rychlosti vývoje. Typescript, který je transpilován do JavaScriptu, pak pomáhá omezit problémy spojené se slabě typovaným JavaScriptem a přináší funkce v JavaScriptu nepřítomné nebo které by se v JavaScriptu musely implementovat ručně či náročně. Restify slouží jako užitečná abstrakce nad Node.js API.

Po spuštění se serverovému objektu Restify předají funkce pro parsování příchozích žádostí, CORS mezivsrstva (z angl. Cross-Origin Resource Sharing). Zmíněné funkce jsou přímo z frameworku Restify a přidružených NPM knihoven. Před nasazením hlavního routeru je ještě přidána vlastní vrstva, která ověřuje klientův „session token“, o kterém si povíme v podsekci o autentifikaci.

Hlavní router skládá dohromady všechny endpointy, které mohou být umístěny přímo na hlavní router, nebo mohou být součástí podřadných routerů. Jeden takový podřadný router má na starosti přihlašování. Je samozřejmě možné dávat endpointy přímo na serverový objekt, ale tím bychom se zbavili jisté flexibility, a nemohli tak snadno určit kořenovou cestu pro všechny endpointy. Zároveň si každý router může určit, jakou relativní cestu budou mít routery podřadné.

Veškerá komunikace s klinty je ve formátu JSON.

3.1 Konfigurace

Paramatry pro Middleend lze nastavit ze dvou míst. Z konfiguračního souboru, jenž se nachází ve složce `/config` a stanovením proměnné prostředí (angl. Environment Variable) s názvem `NODE_CONFIG`, jejíž hodnoty mají prioritu nad konfiguračním souborem. Formát konfigurace je JSON:

```
{
    "db": {
        ...
    },
    "server": {
        "port": 8000,
        ...
        "sessionTokenExpiresInMillis": 86400000,
        "rootPath": ""
    },
    "_desc": "Local development config."
}
```

Middleend používá dva Postgres účty. První (cesta `db.middleend`) je určen například pro ověřování a změnu hesel. Druhý (cesta `db.qdb`) zajišťuje žádosti na endpoint `/qdb`. Druhý účet má práva pouze na pohledy na tabulky v db a jeho permise dále omezují permise konkrétního uživatele. Objekty v polích `db.*` mají položky `host`, `port`, `user`, `password`, `database`, `max`. Všechna kromě `max` jsou na první pohled srozumitelná. `Max` určuje nejvyšší možný počet klientů v poolu.

Cesta	Typ	Popis
server.port	int	Port serveru
server.origins	[string]	Povolené zdroje žádostí
server.logRequests	bool	Logování obsahu žádostí
server.sessionSecret	string	Tvorba klíčů pro session token
server.sessionTokenExpiresInMillis	int	Doba platnosti session tokenu
server.rootPath	string	Cesta hlavního routeru
server.authentication.password	boolean	Povolení přihlášení heslem
server.authentication.reader	boolean	Povolení přihlášení kartou

Tabulka 3.1: Seznam všech používaných hodnot kromě databázového připojení

Chceme-li například změnit port aplikace, nastavíme `NODE_CONFIG` na řetězec „{"server": 9000}“. Při spouštění v produkci je nutné nastavit `NODE_ENV` na „production“.

3.2 Autentifikace

Autentifikace může probíhat různými způsoby. Po úspěšné autentifikaci uživatele nebo například čtečky, obdrží klient „session token“, který použije pro další žádosti. Posílá se v HTTP hlavičce `Authorization` ve formátu „Token <hodnota>“. Klient také obdrží jeho dobu platnosti v milisekundách od času autentifikace. Důvod pro autentifikaci hlavičkou místo běžných cookies je, že tak jednoduše zabráníme XSS (angl. Cross-site scriping).

Session token má následující podobu:
`<iv>.<tělo>.<timestamp vytvoření>.<hmac>`. `Iv` je inicializační vektor (viz TP). Algoritmus pro vytvoření byl převzán z knihovny node-client-sessions společnosti Mozilla (viz Mozilla). Funguje tak, že si vytvoříme klíč pro šifrování dat, která jsou ve formátu JSON, a klíč pro podepsání konečné zprávy. Klíče vytvoříme z náhodného řetězce („session secret“) pomocí funkce HMAC. Session secret musí zůstat tajný, jinak by si útočník mohl sám podepsat session tokeny. Symetricky zašifrujeme data, přičemž přidáme inicializační vektor. Poskládáme jednotlivé části dohromady a spočítáme z nich HMAC (angl. Hash Message Authentication Code), jehož hodnotu přidáme na konec celého tokenu. HMAC slouží jako ověření tokenu, pouze Middleend je schopen ji vytvořit (viz Gilles, 2012). Inicializační vektor, zašifrovaná data, HMAC hodnotu závěrečně zaenkódujeme do Base64 (viz Josefsson, 2006).

Aby byl předložený session token platný, musí splňovat několik podmínek. Nesmí být starý, tj. $t_v + t_e \geq t$, kde t je nynější čas, t_v je čas vytvoření a t_e je čas, za který se token stane neplatným. Po dešifrování si spočítáme znovu HMAC, a ten se musí rovnat HMAC hodnotě z tokenu. Poslední podmínkou je úspěšné parsování dešifrovaných JSON dat.

3.2.1 Autentifikace heslem

Pro solení a hashování hesel využívá Middleend algoritmus scrypt, který je pro Node.js dostupný jako NPM balíček stejného jména (viz barrysteyn, 2016). V porovnání s jinými kryptografickými algoritmy používanými pro zabezpečení

hesel, jako je pbkdf2 nebo bcrypt, má pár výhod. Používá exponenciální čas a paměť, takže hardware optimalizovaný pro jeho výpočet není natolik účinný, jako je hardware například pro pbkdf2 (viz Preziuso, 2015). V praxi je ale pbkdf2 s mnoha iteracemi účinný a používáný.

KDF	6 letters	8 letters	8 chars	10 chars	40-char text	80-char text
DES CRYPT	< $1	< $1	< $1	< $1	< $1	< $1
MD5	< $1	< $1	< $1	$1.1k	$1	$1.5T
MD5 CRYPT	< $1	< $1	$130	$1.1M	$1.4k	1.5×10^{15}
PBKDF2 (100 ms)	< $1	< $1	$18k	$160M	$200k	2.2×10^{17}
bcrypt (95 ms)	< $1	$4	$130k	$1.2B	$1.5M	$48B
scrypt (64 ms)	< $1	$150	$4.8M	$43B	$52M	6×10^{19}
PBKDF2 (5.0 s)	< $1	$29	$920k	$8.3B	$10M	11×10^{18}
bcrypt (3.0 s)	< $1	$130	$4.3M	$39B	$47M	$1.5T
scrypt (3.8 s)	$900	$610k	$19B	$175T	$210B	2.3×10^{23}

Časová hodnota v závorkách algoritmů určuje jejich náročnost.

Tabulka 3.2: Odhadovaná cena hardwaru k prolomení hesla za 1 rok. (viz Percival, 2009)

Ceny hardwaru v tabulce 3.2 nebudou odpovídat dnešnímu stavu, protože se výkon i cena hwardwaru od roku 2009, kdy byla tabulka vytvořena, změnily. Každopádně je rozumné se domnívat, že crackování funkce scrypt je stále nejdražší i dnes.

3.3 Tunelování Postgresu pro Frontend

Připojení k databázi je zprostředkováno NPM modulem `pg` (viz NodePg). Ten umožňuje asynchronní přístup, předpřipravené žádosti a „poolování“ (tj. máme „pool“ klientů, které si půjčujeme a vracíme je, jakmile je nepotřebujeme).

Jakmile je uživatel autentifikován platným session tokenem, je půjčen klient k databázi. Dále se pošle databázi žádost, která zavolá backendovou funkci `session_user_set(user_id, secret)`, kde `user_id` je identifikátor uživatele a `secret` je náhodný řetězec, který potřebujeme pro odhlášení po spuštění uživatelského SQL. Tím se určí uživatel a jeho permise. Poté se spustí uživatelské SQL a odpověď se pošle uživateli. Před posláním je ještě zpracována a některé části odpovědi z databáze jsou vynechány, nebo přetransformovány. V posledním kroku je uživatel odhlášen backendovou funkcí `session_logout(secret)`, kde `secret` je náhodný řetězec, který jsme si vygenerovali při přihlášení.

3.4 Transformace Postgres odpovědi

Transformaci zajišťuje třída `MiddleResponse`, která přijme původní odpověď databáze. Odpověď databáze obsahuje proměnnou `field` obsahující pole definic sloupců vrácených z databáze.

```
"fields": [
    {
```

```
        "name": "id",
        "tableID": 17249,
        "columnID": 1,
        "dataTypeID": 23,
        "dataTypeSize": 4,
        "dataTypeModifier": -1,
        "format": "text"
    },
    ...
]
```

V definici každého typu se chceme zbavit `tableID`, `columnID`, `dataTypeID` a nahradit je jmény, aby mohl Frontend data snadněji parsovat, čehož docílíme dotázáním se databáze. Můžeme si všimnout, že v původní odpovědi už dostaneme název sloupce, jehož název může být i agregační sloupec nebo sloupec přejmenovaný apod. Dotazování z výkonostních důvodů provedeme při spuštění Middleendu. SQL dotazy vypadají následovně:

```
SELECT oid, typname FROM pg_type; -- id datových typů -> jména dt.
SELECT oid, relname FROM pg_class; -- id tabulek -> jména tabulek
```

Druhý SQL dotaz vrátí více než tabulky. Počet řádků je však dostatečně malý a nepotřebuje moc paměťi navíc (viz PgType), (viz PgClass).

Odpovědi si uložíme do JavaScript objektu, který mapuje číselné identifikátory na řetězce. JavaScript ale neumožňuje používat čísla jako klíče, tudíž si je převedeme při ukládání a čtení na řetězec.

Konečná odpověď z Middleendu může vypadat následovně:

```
{
    "rows": [
        [1, "Adam", "Warlock", "2014-04-08T22:00:00.000Z"],
        [2, "Harry", "Potter", "1990-07-31T22:00:00.000Z"]
    ],
    "rowCount": 2,
    "fields": [
        {
            "tableName": "wizards",
            "columnName": "id",
            "dataTypeName": "int4",
            "dataTypeSize": 4,
            "dataTypeModifier": -1,
            "format": "text"
        },
        ....
    ]
}
```

4. Frontend

Webové rozhraní Mocasysu je napsané jako SPA (z angl. Single Page Application) v jazyce Scala s využitím kompilátoru Scala.js. Byl vybudován na vlastním frameworku „liwec“, který je sám postaven na JavaScript knihovně domvm (viz Dom, 2019). Využívá přístup „virtual DOM“ známý například z knihoven React a Vue.

4.1 Liwec

Liwec dělí aplikaci na tzv. komponenty, někdy nazývané widgety či zobrazení, které reprezentují jednotlivé soběstačné části grafického rozhraní. Komponenty jsou třídy dědící abstraktní třídu `Component`. Při změně jakéhokoliv atributu komponenty je spuštěna funcke `render()`, její výsledek porovnán s aktuálním stavem a změny vykresleny. Hlídání změn využívá `Proxy` (viz ?)MdnJsProxy], funkcionalitu JavaScriptu, která umožňuje obalit objekt a při jakémkoliv přísupu k objektu zavolat handler funkci.

Liwec umožňuje programovat aplikace výhradně ve Scale. Obsahuje DSL (z angl. Domain Specific Language) pro tvorbu HTML a CSS, které využívají flexibilitu Scaly, což dovoluje mít pro jednu komponentu pouze jeden soubor bez nutnosti vkládat další jazyky jako text.

HTML tagy jsou reprezentovány objekty, což dovoluje například vytvářet „pseudo-komponenty“ – funkce, které dle daných parametrů vrátí HTML tag. Tyto funkce zabírají méně paměti než plné komponenty, nemohou ale udržovat vnitřní stav pomocí hlídaných atributů, plní tak jinou roli než plné komponenty.

U CSS bývá problémem „přelévání“ stylů z jedné komponenty do druhé kvůli příliš širokým selektorům či sdíleným jménům tříd. Tento problém je někdy řešen komplikovanými vývojovými metodologiemi, liwec však implementuje „scoped CSS“, koncept převzatý z Vue. Každému DOM elementu je přidán atribut, který unikatně identifikuje komponentu, které je součástí. Ke každé části každého CSS selektoru je pak přidána ekvivalentní podmínka.

CSS styly komponent jsou při kompilaci převedeny makrem na textovou reprezentaci a uloženy do souborů, poté sloučeny externím skriptem. Výsledná aplikace tak nemusí obsahovat kód na generování CSS.

```
class Counter(var i: Int = 1) extends Component {
    def render() = scoped(div(
            button("Increment",
                   onClick := { _ => i += 1 }),
            div(s"Counted to \$i")
    ))
    cssScoped { import liwec.cssDsl._
        e.div (
            color := "crimson",
            fontSize := "20pt",
        )
    }
}
```

4.1.1 Interní struktura

Liwec je kompilován nástrojem SBT. Jeho tzv. projekt je rozdělen na několik dílčích projektů:

- *macros* obsahuje makra využívaná v DSL liwecu, konkrétně `scoped` pro úpravu HTML pro „scoped CSS“. Dále pak `css` a `cssScoped`, makra pro zpracování CSS DSL.
- *htmlCodegen* a *cssCodegen* jsou programy, které ze stránek MDN získají data ve strukturované formě a vytvoří pak podle nich třídy a funkce, které pak jsou součástí DSL.
- *liwec* obsahuje samotný kód obsažený v zkompilovaném výstupu, mimo jiné typy pro DSL, reprezentaci knihovny `domvm` ve Scale (tzv. „binding“) a kód pro routování stránek dle URL.

4.1.2 HTML DSL

Vytváření HTML komponent pomocí DSL, sady prvků jazyka tvořících vnořený „jazyk“, je jednou z definujících vlastností liwecu. Tento přístup je inspirován mimo jiné JavaScript knihovnou Mithril (viz Mit, 2019) a Scala knihovnou ScalaTags (viz Sca).

DSL je implementováno pomocí různých funkcí Scaly, několik z nich exklusivních tomuto jazyku. Tagy jsou reprezentovány statickými objekty (konkrétně `lazy val`), které lze volat jako funcke akceptující libovolné množství argumentů a vracející objekt `ElementVNode` z knihovny `domvm`. Tyto objekty lze chápat jako „typesafe“ wrappery funkce `el` z `domvm`.

Argumenty tagů jsou instance „vlastnosti“ (angl. trait) `VNodeApplicable[T]`, kde `T` je typ elementů, na které lze tuto instanci použít. Každá instance `VNodeApplicable` má metodu `applyTo(vn: T): Unit`, která zmutuje předaný objekt. Instancí `VNodeApplicable` jsou mimo jiné objekty reprezentující atributy tagů a jejich hodnoty a komponenty. Pomocí tzv. implicitních tříd lze pak jako `VNodeApplicable` brát i například řetězce, další elementy a vybrané monády (`Seq` a `Option`), pokud obsahují další `VNodeApplicable`.

Atributy tagů jsou objekty, které po použití operátoru `:=` vrátí objekt reprezentující název atributu i jeho hodnotu. Tento objekt je instancí `VNodeApplicable`, takže je možné jej použít jako argument tagů.

4.1.3 CSS DSL

Popis CSS pravidel v liwecu využívá objektů a vlastních operátorů k vytvoření AST (z angl. Abstract Syntax Tree, syntaktický strom). CSS selektory jsou reprezentovány instancemi `Selector`, které jsou vytvářeny primárně pomocí globálních „dynamických objektů“. Díky funkci Scaly je `c.className` přetransformováno na `c.selectDynamic("className")`, což je v liwecu funkce vracející selektor pro CSS třídu nazvanou `className`. Komplexní selektory lze pak vytvářet pomocí vlastních operátorů vlastnosti `Selector`.

Selektor je možné buď zavolat jako funkci přímo, nebo zavolat jeho metodu `->`. Obě funkce berou libovolné množství parametrů: samotné CSS vlastnosti a další pravidla. Takto vzniklé pravidlo je pak zpracováno makry `css` a `cssScoped`.

4.2 Tabulky a formuláře

Velkou část Frontendu tvoří rutinní tabulky a formuláře, které skoro přesně odpovídají SQL tabulkám. Proto byly vytvořeny obecné třídy a funkce pro vytváření těchto konstruktů.

Framework tabulek se skládá ze dvou vrstev. Nižší umožňuje z jakéhokoliv seznamu a specifických objektů sloupců vytvořit tabulku, druhá se pak zaměřuje na vytváření tabulek z libovolného SQL dotazu.

Framework formulářů je navržen vesměs dynamicky. Jádrem každého formuláře je `Map[String, Any]`, slovník z řetězce na libovolný datový typ. Tento přístup sice znatelně ztěžuje ověřování správnosti kódu při kompilaci, výsledný kód je ale velmi krátký a framework nevyžaduje k funkčnosti žádná komplexní makra.

4.3 Hlavní menu

Menu můžeme reprezentovat rekurzivně dvojící tříd `MenuItem` a `SubMenu`. Obě tyto třídy dědí z traitu (interface v Javě) `MenuNode`, což zjednodušuje psaní algoritmů, jenž využívají instance těchto tříd. Limitujeme tak počet případů. `MenuItem` uchovává text odkazu a funkci, která se zavolá při kliknutí na odkaz. `SubMenu` si uchovává `MenuItem` seznam `MenuNode`, abychom mohli menu reprezentovat rekurzivně.

```
trait MenuNode
class MenuItem(value: String, action: Event => Unit) extends MenuNode
class SubMenu(item: MenuItem, children: Seq[MenuNode]) extends MenuNode
```

Celé menu můžeme předávat jako instanci `SubMenu`. Menu pak vyrenderujeme procházením do hloubky přibližně takto:

```
def renderMenu(node: SubMenu): liwec.htmlDsl.VNodeFrag =
    for (child <- node.children) yield child match {
        case menu: SubMenu =>
            li(h4(menu.item.value, onClick := menu.item.action),
                ul(renderMenu(menu)))
        case item: MenuItem =>
            li(item.value, onClick := item.action)
    }
```

V implementaci je kořen menu renderován jinak kvůli stylování, princip ale zůstává stejný. Některé html atributy byly v ukázce kódu vynechány.

4.4 Globální stav

Stav aplikace udržuje instance třídy `AppStateCls`, ke které se přistupuje skrz proměnnou `AppState`. Zde jsou uloženy údaje jako uživatel, session token z Middleendu a instance `ApiClient`. Dále poskytuje abstrakční vrstvu nad tímto api klientem, abychom mohli zachytávat chyby a zobrazovat je globálně jako zprávy uživateli.

4.5 Zprávy uživateli

Zprávy uživateli (např. chybové) jsou renderované komponentou `Messenger`, ke které lze přistupovat z `AppState`. Zprávy jsou předávány jako instance traitu `Message`, jenž po konkrétních třídách požaduje implementovat `render` a `duration`.

Volitelná metoda traitu `Message setCancel` umožňuje obdržet funkci k ručnímu odstranění. Toho je například využíváno u zprávy `OfflineMessage`, která se zobrazí, pokud uživatelovo zařízení ztratí připojení. Tj., když zavolá se `offline` event. `OfflineMessage` pak implementuje `online` event, v jehož handleru zavolá předanou odstraňovací funkci.

Skrz tento třídový návrh můžeme docílit různého stylování s např ikonami a dodatečnými akcemi. `Messenger` si zprávy ukládá do slovníku `Map[Int, Message]`, kde klíč je náhodně generovaný. Společně s přidáním do slovníku je vytvořen časovač pomocí funkce `setTimeout`, která po uplynutém intervalu spustí předanou funkci k odstranění ze slovníku zpráv. Návratová hodnota funkce `setTimeout` je handler, který když předáme funkci `clearTimeout`, tak zrušíme odpočet. Datový typ `duration` je `Option[Int]` a pokud je hodnota této proměnné `None`, zpráva nebude automaticky odstraněna.

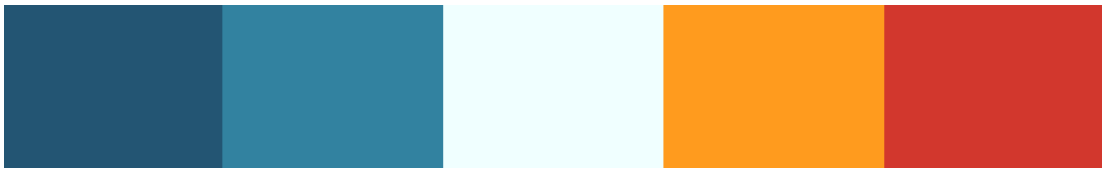
4.6 Drag'n'drop operace

Některé stránky (přiřazování jídel, úprava permisí) využívají operaci drag'n'-drop (česky táhni a pusť) k usnadnění a zpříjemnění uživatelského rozhraní. Táhnutý prvek musí implementovat událost `ondragstart` a musí mít atribut `draggable`. Místa, kam může být táhnutý prvek umístěn, pak musí implementovat události `ondrop` a `ondragover` (viz W3DnD). Dále se hodí implementovat na `ondragleave` na místě puštení, abychom mohli například vrátit bravu okraje na původní barvu, který jsme obarvili při `ondragover`.

4.7 Umělecký styl

Paleta barev byla inspirována výrobcem zbraní Maliwan v počítačové hře Borderlands (viz Mal, 2019). Konečná paleta byla získána nástrojem s adresou colormind.io, což je generátor palet využívající strojové učení. Finální paleta je vidět na obrázku 4.2.

Obrázek 4.1: Logo Mocasys



Obrázek 4.2: Paleta Frontendu

5. Hardware jídelní čtečky

Hardware je založen na mikropočítači, který je osazen procesorem s architekturou ARM. Z mnoha dnes dostupných ARM počítačů jsem zúžili výběr na následující čtyři: Omega 2, NanoPi Neo, Orange Pi Zero a VoCore 2. Tato zařízení byla uvážena, protože jsou malá a mají nainstalovaný ethernet port, který je nezbytný pro připojení do k ostatním komponentám, protože jsme nechtěli zvolit bezdrátové připojení do sítě, které bývá nestabilní a nespolehlivé.

VoCore 2 vyhovoval ve všech bodech, ale byl zamítnut kvůli vysoké ceně. Omega 2 bylo zamítnuto, protože nebyla žádná výhoda oproti konkurenci. NanoPi Neo a Orange Pi Zero jsou velmi podobné. Zprvu bylo zvoleno Orange Pi Zero díky nepatrně lepší výbavě.

5.1 Displej a NFC čtečka

Pro zobrazení pro koncové uživatele byl vybrán standardní rezistivně dotykový displej o rozlišení 480 na 320 pixelů a o průměru 4 place. Ten komunikuje s počítačem pomocí protokolu SPI, aby byla dosažena nízká odezva a plynulost displeje. Tento displej je použit pro obsluhu čtečky. Naopak pro strávníky je zde displej s technologií OLED, rozlišením 128x64 pixelů, průměrem 1,3 palce, modré barvy.

Roli samotného čtení čipů nebo mobilů, zajišťuje NFC čtečka založena na čipu PN532 a desce 3. verze. Modul čte na frekvenci 13,56 MHz.

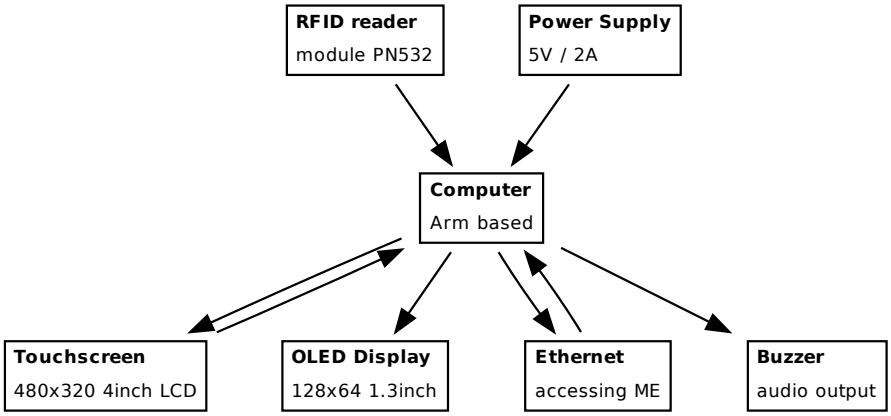
5.2 Napájení

Napájení Orange Pi Zero (dále už jen OPZ) bylo navrženo s technologií PoE (Power over Ethernet), kde OPZ má na své desce rospojené spojení pro vypnutí této funkce. Stačilo tedy jen spojit vývody na desce pomocí cínu. Tím se krajní piny ethernet portu propojí s interním regulátorem OPZ. Tím se zjednoduší zapojení, ale pokud spojení přesáhne 10 metrů, začne se snižovat napětí díky vysokému odporu ethernetového drátu, což má za následek nedostatečné napájení pro čtečku.

5.3 Potíže s SPI a volba Raspberry Pi A+

Při zapojení jsme zjistili, že OPZ obsahuje jen dva přepínací porty SPI, což je dostačující pro dotykový displej (obraz a dotyk), ale poté nezbývají další SPI piny pro OLED displej ani čtečku, které měly být také připojeny pomocí SPI pro dosažení nízké latence. Proto byla NFC čtečka přepnuta do protokolu I2C, a tím byla zajištěna komunikace. U displeje byl problém obtížnější, protože přepínání protokolů je zajištěno odporem mezi spojeními. Pro obavu z poškození displeje byl displej vyměněn za jeho menší verzi (0,96 palce), který nativně podporuje protokol I2C, a tím se vyřešily problémy s počtem pinů.

Počítač byl nakonec nahrazen za Raspberry Pi A+ první generace kvůli softwarové nekompatibilitě grafiky dotykového displeje. Ten nativně neobsahuje ether-



Obrázek 5.1: Hardware schéma

netový port, a proto musel být použit adaptér z USB na ethernet port, pomocí kterého je zajištěna komunikace se serverem a opravňování softwaru přes protokol SSH. Upravování kódu je díky tomu možné pouze přes SSH, protože počítač má jen jeden USB port a bylo by velmi časově náročné provádět úpravy pomocí dotykového displeje. Všechny komponenty hardwaru jsou vidět na obrázku 5.1.

6. Instalace

Mocasys je komplexní projekt složený z několika částí, tudíž je instalace komplikovaný proces zahrnující vícero programovacích jazyků a prostředí. Momentálně je jediným podporovaným operačním systémem pro vývoj Mocasysu a spuštění serverových částí GNU/Linux.

Tyto instrukce byly ověřeny na Arch Linuxu a balíčcích z května 2019. Návod předpokládá, že máte stažené všechny repozitáře projektu ve struktuře složek určené submoduly.

6.1 Instalace Backendu

- Nejprve nainstalujte PostgreSQL (viz Pos) verze 11 a vytvořte prázdné databázové schéma.
- Stáhněte, zkompilujte a nainstalujte rozšíření PostgreSQL temporal_tables (viz Tem, 2019).
- Ve složce `mocasys-backend` spusťte skript `setup_db.sh`. Skriptu můžete dát další parametry, které budou předány použitému příkazu `psql`.

6.2 Instalance Middleendu

- Nainstalujte Node.js (viz Nod) a NPM (viz Npm).
- Upravte soubor `mocasys-middleend/config/default.json` tak, aby obsahoval správné nastavení pro připojení k databázi. Middleend využívá dvě databázová spojení, „middleend“ a „qdb“. „Middleend“ by měl být připojen pomocí uživatele, který má přístup k tabulkám `user_passwords_data` a `user_mifare_cards_current`. „Qdb“ používá uživatele `uptest`, který byl vytvořen skriptem `setup_db.sh`.
- Ve složce `mocasys-middleend` spusťte příkaz `npm install`, což nainstaluje NPM balíčky, a `npm run start`, což spustí server middleendu.

6.3 Instalace Frontendu

- Nainstalujte SBT (viz Sbt).
- Ve složce `mocasys-frontend` spusťte příkaz `sbt devel`, poté `devserver.py`, což spustí webserver hostující aplikaci.
- Navštivte ve webovém prohlížeči stránku `http://127.0.0.1:8080`.

Závěr

Výsledný soubor programů splňuje zádání kromě hardwarové části, která se ukázala být problematická. Přestože jsme se rozhodli pro některé části zvolit vlastní frameworky a knihovny, volba se vyplatila. Jakmile jsme je totiž měli hotové, vývoj byl velice rychlý.

V budoucnu plánujeme projekt plně dokončit ve všech aspektech a rozšiřovat ho pro různé typy produkčních provozů

Seznam použité literatury

Node.js. URL https://nodejs.org/en/. Naposledy navštíveno 2019-05-10.

npm. URL https://www.npmjs.com/get-npm. Naposledy navštíveno 2019-05-10.

Postgresql. URL https://www.postgresql.org/. Naposledy navštíveno 2019-05-10.

sbt - the interactive build tool. URL https://www.scala-sbt.org/. Naposledy navštíveno 2019-05-10.

Scalatags. URL https://www.lihaoyi.com/scalatags/. Naposledy navštíveno 2019-05-10.

(2019). Dom viewmodel - a thin, fast, dependency-free vdom view layer. URL https://github.com/domvm/domvm. Naposledy navštíveno 2019-05-09.

(2019). Maliwan. URL https://borderlands.fandom.com/wiki/Maliwan. Naposledy navštíveno 2019-05-10.

(2019). Mithril.js. URL https://mithril.js.org/. Naposledy navštíveno 2019-05-09.

(2019). Temporal tables postgresql extension. URL https://github.com/mlt/temporal_tables/tree/pg11. Naposledy navštíveno 2019-05-09. Fork kompatibilní s PG11.

BARRYSTEYN (2016). Scrypt for node. URL https://github.com/barrysteyn/node-scrypt. Naposledy navštíveno 2019-05-07.

GILLES (2012). How and when do i use hmac? URL https://security.stackexchange.com/questions/20129/how-and-when-do-i-use-hmac. Naposledy navštíveno 2019-01-28.

JOSEFSSON, S. (2006). The base16, base32, and base64 data encodings. URL https://tools.ietf.org/html/rfc4648. Naposledy navštíveno 2019-02-02.

Mozilla (2012). Node client sessions. URL https://github.com/mozilla/node-client-sessions. Naposledy navštíveno 2019-02-01.

NodePg (2019). Postgres for node. URL https://node-postgres.com/. Naposledy navštíveno 2019-05-09.

PERCIVAL, C. (2009). Stronger key derication via sequentialmemory-hard functions. URL https://www.tarsnap.com/scrypt/scrypt.pdf. Naposledy navštíveno 2019-02-04.

PgClass (2019). Postgresql 10 documentation - pg_class. URL https://www.postgresql.org/docs/10/catalog-pg-class.html. Naposledy navštíveno 2019-05-09.

PgType (2019). Postgresql 10 documentation - pg_type. URL https://www.postgresql.org/docs/10/catalog-pg-type.html. Naposledy navštíveno 2019-05-09.

Preziuso, M. (2015). Password hashing: Pbkdf2, scrypt, bcrypt. URL https://medium.com/@mpreziuso/password-hashing-pbkdf2-scrypt-bcrypt-1ef4bb9c19b3. Naposledy navštíveno 2019-02-04.

TP. What is an initialization vector? - definition from techopedia. URL https://www.techopedia.com/definition/26858/initialization-vector. Naposledy navštíveno 2019-02-02.

W3DnD. Html5 drag and drop. URL https://www.w3schools.com/html/html5_draganddrop.asp. Naposledy navštíveno 2019-05-10.

Seznam obrázků

Seznam tabulek