Flashcards

Obsah

1 Přehled

Aplikace Flashcards má stejnou strukturu jako webová aplikace, hlavní části jsou naprogramovány s použitím HTML, CSS a Javascriptu, ale na rozdíl od ostatních webových aplikací se nevykresluje ve webovém prohlížeči, ale v Electronu. Data se ukládají do oddělené databáze, takže obě části aplikace běží nezávisle na sobě. O databázi se stará Django Framework (Python) a běží to na localhostu. Klient a server komunikují asynchronně skrz JSON requesty.

2 Použité knihovny

Aplikace používá mnoho různých knihoven na usnadnění používání některých funkcí. Na serverové části je použito Django, které ovšem vyžaduje nainstalovaný Python 3 (je doporučeno používat nejnovější verzi). Na straně klienta je použit Electron, který renderuje všechny vizuální aspekty místo prohlížeče, a ten pro své fungování potřebuje Node.js. Pro hezčí vzhled je použit Bootstrap, skládá se ze tří souborů: `.css`, které je upravené kvůli sladění některých barev s celkovým vzhledem, `.js` a `popper.js`, které jsou potřeba pro některé funkce. Pro dynamické měnění obsahu na stránce a komunikaci se serverem je použita JQuery.

- Server
  - Python 3.7 nebo vyšší (*www.python.org/downloads/windows*)
  - Django 2.1.5 (nainstalováno příkazem `pip install Django==2.1.5`)
- Uživatelské rozhraní
  - Node.js (*www.npmjs.com/get-npm*)
  - Electron (*www.electronjs.org*)
  - Electron packager
  - Bootstrap předkompilované soubory (stažené z *www.getbootstrap.com/docs/4.3/getting-started/download/*)
    - CSS, Javascript a Popper
  - JQuery (*www.jquery.com*)

3 Konfigurace serveru

Jak bylo zmíněno dříve, o všechna ukládaná data se stará Django běžící offline na localhostu na portu 8000. Všechny soubory týkající se serveru se nachází ve složce `./server/`. Ve složce `./server/server` je obecné nastavení serverové aplikace, ve složce `./server/cards/` jsou pak soubory této konkrétní aplikace s modely pro databázi a funkcemi na zpracovávání příchozích a odchozích requestů.

3.1 Spuštění serveru

```
python manage.py runserver localhost:8000
```

- Spuštěn ze složky, ve které se nachází soubor `manage.py` (`./server/`)

3.2 Základní nastavení serveru

- Soubory v `./server/server`.
- Zde se nastavují všechny nainstalované aplikace využívající tento server (v tomto případě pouze ‘cards’), typ použité databáze (‘sqlite3’) a které urls jsou použity pro příjímání requestů (`urls.py`).
- Dále se zde nachází secret key pro případ publikování databáze na internet

3.3 Nastavení aplikace Cards

Všechny soubory v `./server/cards/`. Pro tuto aplikaci jsou potřebné pouze `models.py`, `urls.py` a `views.py`. Ostatní jsou buď automaticky vytvořené, nebo prázdné.

Základní adresa k serveru

```
localhost:8000/cards
```

3.3.1 Modely

V tomto souboru se definují databázové objekty s jejich proměnnými a vztahy s ostatními objekty. Každý model slouží jako šablona pro záznamy do databáze a vytvoří si tabulku, kam tyto záznamy budou ukládány. Jednotlivé řádky tabulky jsou propojovány s jinými tabulkami skrze relationship fields.

- model `Tag(tag_name, previous_success_rate, card_count)` ... šablona pro okruhy
  - `tag_name` ... CharField (max. 100 znaků); reprezentuje jméno daného okruhu
  - `previous_success_rate` ... IntegerField; reprezentuje procentuální úspěšnost posledního testu daného okruhu
  - `card_count` ... IntegerField, volitelný (default=0); reprezentuje počet karet propojených k okruhu
  - každá proměnná má svoje `get` a `set`, které vrací, nebo upravují jejich hodnoty
  - `add_card` a `remove_card` metody slouží k navýšení, nebo snížení počtu karet v `card_count` proměnné
- model `Card(card_front, card_back, tag_count, tags)` ... šablona pro kartičky
  - `card_front` ... CharField (max. 200 znaků); reprezentuje text zobrazený na přední straně kartičky
  - `card_back` ... CharField (max. 200 znaků); reprezentuje text zobrazený na zadní straně kartičky
  - `tag_count` ... IntegerField, volitelný (default=0); reprezentuje počet okruhů, ve kterých je daná kartička
  - `tags` ... ManyToManyField; propojuje kartičky s libovolným počtem okruhů
    - příkazem `tags.add(tag)` je propojena kartička s okruhem, `tags.remove(tag)` zničí vazbu mezi nimi
    - v každém okruhu se automaticky vytvoří proměnná `cards`, kam se naopak ukládají všechny propojené kartičky. To znamená, že jejich spojení je oboustranné a lze k nim přistoupit z obou dvou.
  - každá proměnná má svoje `get` a `set`, které vrací, nebo upravují jejich hodnoty
  - `add_tag` a `remove_tag` metody slouží ke zvýšení, nebo snížení počtu okruhů v `tag_count` proměnné

3.3.2 Views a Urls

Zde jsou definovány funkce, které spravují příchozí a odchozí JSON a Http requesty z/do klienta. Všechna data posílaná requesty jsou v JSON formátu pro lepší komunikaci s Javascriptem.

- request `cards`
  - musí být poslán na url `localhost:8000/cards/cards`
  - vrací seznam JSON objektů, každý reprezentuje jednu kartičku z databáze
  - JSON objekt neobsahuje všechny atributy kartičky - `{id, card_front, card_back}`
- request `tags`
  - musí být poslán na url `localhost:8000/cards/tags`
  - vrací seznam JSON objektů, každý reprezentuje jeden okruh z databáze
  - JSON objekt neobsahuj všechny atributy kartičky - `{id, tag_name}`
- request `tag(tag_id)`
  - musí být poslán na url `localhost:8000/cards/tags/tag_id/`
  - vrací JSON objekt se všemi informacemi o okruhu specifikovaným `tag_id`
  - vrací `{id, tag_name, success_rate, card_count, cards}`
- request `card(card_id)`
  - musí být poslán na url `localhost:8000/cards/cards/card_id/`
  - vrací JSON objekt se všemi informacemi o kartičce specifikované `card_id`
  - vrací `{id, card_front, card_back, tag_count, tags}`

- request `add_tag`
  - musí být poslán na url `localhost:8000/cards/add_tag/`
  - dostane JSON objekt od klienta - `{type, id, tag_name, success_rate, card_count, cards}`
    - podle hodnoty `type` argumentu určí, která akce se provede
    - vytvoří nový okruh (`type: new`),
    - zaktualizuje jméno (`type: update`),
    - zaktualizuje úspěšnost posledního testu (`type: test`),
    - smaže okruh (`type: delete`).
- request `add_card`
  - musí být poslán na url `localhost:8000/cards/add_card/`
  - dostane JSON objekt od klienta - `{type, id, card_front, card_back, tag_count, tags}`
    - podle hodnoty `type` argumentu určí, která akce se provede
    - vytvoří novou kartičku (`type: new`),
    - zaktualizuje jméno a automaticky všechny vazby mezi okruhy (`type: update`),
    - smaže kartičku (`type: delete`).
- request `import_all`
  - musí být poslán na url `localhost:8000/cards/import/`
  - dostane dvojici seznamů JSON objektů, které obsahují okruhy i kartičky

    ```
    [
       [{type: "new", card_front, card_back, tag_count, tags}],
       [{type: "new", tag_name, success_rate: 0, card_count: 0}]
    ]
    ```
  - funkce sama určí, jestli je položka unikátní a přidá ji do databáze - automaticky spojí okruhy s kartičkami na základě indexů druhého listu (v každé `tags` proměnné je seznam indexů položek z druhého listu, které mají být propojeny)
  - když položka již existuje, je přeskočena

V souboru `urls.py` jsou definovány url adresy kam se posílají jednotlivé requesty. Každý request má svojí adresu pro lepší organizaci.

4 Klient

Klient uživatelského rozhraní je vytvořeno stejným způsobem jako moderní webové aplikace, ale místo renderování v prohlížeči jako Firefox nebo Chrome se vykreslují v Electronu. Výhoda tohoto řešení je lepší fungování offline a také vytváření vlastního okna pro aplikaci pro hezčí vzhled, ale na druhou stranu je nezbytné nainstalovat Node.js, což dělá klienta náročného na úložiště (kolem 100 MB minimálně). Zase ale mohou být využity ostatní Node.js rozšíření, jako například file manager. Díky tomu máme přístup k souborům na lokálním disku, čehož by nešlo dosáhnout s běžnou webovou aplikací, protože Javascript žádnou takovou funkci neposkytuje.

4.1 Složky a soubory

Všechny soubory jsou umístěny ve složce `./user_interface/`.

- `/assets/` obsahuje obrázky a ikony.
- `/lib/` obsahuje předkompilované Bootstrap soubory, Jquery a speciální CSS soubor, který otáčí kartičkou.

- `/node_modules/` obsahuje všechny Node.js package všetně Electronu; z těchto souborů se postaví konečný spustitelný soubor
- `/src/` obsahuje Javascript a HTML soubory, které zajišťují funkcionalitu aplikace
- Dále složka obsahuje soubory `package-lock.json` a `package.json`
  - První z těchto dvou popisuje přesný strom souborů stažených do node-modules složky
  - Druhý slouží ke spouštění aplikace z terminálu a ke konečnému zabalení aplikace do spustitelné formu

4.2 Struktura

Konkrétní obsahy jednotlivých souborů jsou popsány v následujících kapitolách, tady je popsán pouze proces. Vykreslovací proces začíná v `render.js` souboru, který je nastaven ke spuštění v `package.json`.

Tento soubor vytvoří okno a načte do něj `mainWindow.html`, který určuje rozložení a vzhled stránky. Aplikace je postavena jako jednostránková aplikace, to znamená, že všechny části jsou na jedné stránce, ale většina je skrytá a zobrazuje se jen ta momentálně potřebná část. O viditelnost se stará Javascriptový soubor.

Vzhled HTML stránky je vylepšen Bootstrapem. Do `mainWindow.html` jsou nalinkovány `.css` soubory, které jsou upravené oproti těm oficiálním o některé barvy, `.js` soubory a `popper.js`, které jsou potřebné pro některé funkce Bootstrapu.

Do `mainWindow.htlm` je také nalinkován `main.js` soubor, který tvoří jádro aplikace a obsahuje všechny hlavní funkce aplikace. Například se stará o viditelnost obsahu, získává input od uživatele a posílá a přijímá requesty z databáze. Všechny jeho funkce jsou popsány v některé z dalších kapitol.

4.2.1 Instalace Electronu

```
npm install electron --save-dev
```

- spuštěno v hlavni složce projektu

4.2.2 Instalace Electron-packageru

```
npm install electron-packager --save-dev
```

- spuštěno ve složce, ve které jsou node-modules

4.2.3 Spuštění klienta z terminálu

```
npm start
```

- spuštěno ve složce `user_interface`

4.2.4 Zabalení aplikace do package

Pro zabalení aplikace do složky obsahující spustitelný soubor a vše potřebné pro spuštění souboru bez instalace spustit

`npm run` + jedna z možností `package-win/package-linux/pakcage-mac`

Tento příkaz je definován v `package.json` v atributu `"scripts":{}` a zabalí aplikaci do složky `/user_interface/rele`
Použije k tomu `electron-packager` a výsledný package funguje bez instalování čehokoliv. **To platí jen**

pro vizuální stránku aplikace, server potřebuje mít nainstalované Django a Python a musí být spuštěn z příkazové řádky (viz Spuštění serveru)

4.3 render.js

Jednoduchý soubor, popisují se jenom vlastnosti okna a který HTML soubor se má použít.

1. Na začátku souboru se musí zahrnout všechny Node.js moduly, které budou použity, včetně Electronu a je potřeba příradit mu proměnné `app` a `BrowserWindow`.
2. Vše ostatní by mělo být v bloku `app.on("ready", function() { code });`, který před spuštěním počká až se dokončí načtení modulů a inicializaci okna.
3. Vytvořit instanci classy `BrowserWindow`, kde se nadefinují vlastnosti okna.
   - Výška, šířka okna (i s minimálními hodnotami, aby byl vidět nějaký obsah), kde na obrazovce se má okno zobrazit a nadpis. Je možné přidat ještě ikonku, ale to udělá Electron-packager.
4. Metodou `.setMenu` nastavíme lištu menu (v této aplikaci je defaultně vypnutá).
5. Načtení HTML kódu ze souboru (`mainWindow.html` v tomto případě).
6. Na konci je pojistka, aby se vypnula aplikace po zavření všech oken.

Více informací o Electron API a dalších možnostech nastavení je na *www.electronjs.org/docs*.

4.4 mainWindow.html

V hlavičce tohoto souboru je mnoho `link` a `script` tagů, které načítají javascriptová a css rozšíření. V zakomentovaných řádcích jsou online CDN odkazy pro případ, že by ty stažené soubory přestaly fungovat (např. kvůli novější verzi). Dále jsou zde dva speciální `script` tagy kolem ostatních, aby je Electron správně načetl při renderování.

Skoro všechny tagy inheritují vlastnosti z `bootstrap.css` a obsahují spoustu `class` atributů. Ty které jsou nějakým způsobem ovládány Javascriptem jsou také pojmenovány pomocí unikátního `id` atributu.

Většina obsahu se generuje dynamicky, takže HTML samo o sobě obsahuje jen přípravená místa pro naplnění daty z databáze.

Více detailních informací o `class` atributech je v dokumentaci Bootstrapu (*www.getbootstrap.com/docs/4.2/getting-started/introduction/*).

4.5 main.js

Tento soubor obsahuje kombinaci JQuery API a čistého Javascriptu. Všechny příkazy z JQuery jsou označeny `$` a zajišťují lepší import dat do HTML souboru a také usnadňují komunikaci se serverem.

Po načtení `mainWindow.html` do Electronu se spustí blok `$(document).ready(function() { code });`. Nejprve zavolá funkci `reset()`, která skryje vše kromě titulní strany. Poté čeká na **click events** na tlačítkách v liště menu a spustí příslušnou funkci, když **event** nastane.

4.5.1 Home button

Spustí pouze funkci `reset()` a zobrazí titulní stranu.

4.5.2 Vytváření karet

Funkce, která se spustí při kliknutí na tlačítko s `id=create_card_button` a které se nachází v navigační liště. Po kliknutí se spustí funkce `create_card()`.

- `create_card()` - nemá žádné parametry; je pouze organizační
  - Pomocí JQuery vyprázdní všechna pole a zobrazí container, který obsahuje HTML definici pro vytváření karet.
  - Pomocí `load_information("tags")` pošle request, načte všechny okruhy z databáze a pro každý vytvoří prázdný checkbox.
  - Čeká na kliknutí na Cancel (`id=cancel`) nebo na Save (`id=save_new_card`).
    - Cancel zavolá funkci `reset()`.
  - Po kliknutí na Save se uloží hodnoty z polí do proměnných.
  - Zkontroluje se, jestli karta se stejnými hodnotami již neexistuje v databázi.
    - Pomocí `load_information("cards")` se postupně načtou všechny karty v databázi a porovnají se skrz funkci `card_is_unique(new_front, new_back, all_cards)`.
  - Když je unikátní, uloží se, které okruhy byly vybrány a vytvoří se JSON objekt, jenž je následně poslán do databáze `post_information(suffix, create_card_object(viz Ostatní funkce))`.
  - Poté se funkce zavolá znovu a uživatel může vytvořit další kartu.
  - Když není unikátní, vyprázdní se všechna pole a vyskočí upozornění.
- `card_is_unique(new_front, new_back, all_cards)` - jako argumenty má nové hodnoty pro předek a zadek karty a seznam karet k porovnání
  - Pokud se rovnají přední i zadní strany u alespoň jedné karty ze seznamu, vrátí funkce `false`, jinak vrátí `true`.

4.5.3 Vytváření okruhů

Funkce, která se spustí při kliknutí na tlačítko s `id=create_card_button` a které se nachází v navigační liště. Po kliknutí se spustí funkce `create_tag()`.

- `create_tag()` - nemá parametry; je pouze organizační
  - Pomocí JQuery vyprázdní pole na vstup a zobrazí container s HTML obsahem pro vytváření okruhů.
  - Čeká na kliknutí na Cancel (`id=cancel`) nebo na Save (`id=save_new_card`).
    - Cancel zavolá funkci `reset()`.
  - Po kliknutí na Save se uloží hodnoty z pole do proměnné.
  - Zkontroluje se, jestli okruh již neexistuje v databázi.
    - Pomocí `load_information("tags")` se načtou jména všech okruhů a porovnají se
  - Když je unikátní, vytvoří se, funkce se zavolá znovu a uživatel může přidat další okruh.
  - Když není unikátní, vyprázdní se pole a vyskočí upozornění.

4.5.4 Editace karet

Funkce, která se spustí při kliknutí na tlačítko s `id=edit_card_button` a které se nachází v navigační liště. Vypíše všechny karty do tabulky a umožní jejich úpravu. Jako první se volá funkce `list_cards_to_edit()`.

- `list_cards_to_edit()` - nemá parametry; je pouze organizační
  - Vyprázdní se předchozí tabulka, takže zůstanou jen nadpisy sloupečků, a zobrazí se container s tabulkou.

  - Načtou se názvy a id všech karet v databázi do seznamu, na který je pak zavolána funkce `sort_card_list(card_list)`, která list seřadí.
  - Pro každé id se vytvoří řádek v tabulce, zbytek ale zůstane prázdný.
  - Pomocí `load_information()` se načtou všechny informace o jednotlivých kartách a přidají se do odpovídajících řádků podle id.
  - V každém řádku se také vytvoří tlačítko Edit a tlačítko Delete.
  - Po stisknutí Delete se zeptá na potvrzení a pomocí `post_information("add_card/", create_card_object ...))` se pošle request do databáze, kde se kartička smaže. Znovu se zavolá `list_cards_to_edit()`, aby se obnovil seznam karet.
  - Po stisknutí Edit se zavolá `edit_card(card_info)`
- `edit_card(card)` - jako parametr bere objekt JSON, který obsahuje informace o kartě
  - Funkce funguje dost podobně jako `create_card()`, ale pole a checkboxy se vyplní podle aktuálních informací o kartě.
  - Po kliknutí na Save se opět pošle request do databáze pomocí `post_information("add_card/", create_card_object("update", ...))`.
  - Po uložení nebo kliknutím na Cancel se zavolá funkce `list_cards_to_edit()`.
- `sort_card_list(card_list)` - využívá předdefinovanou funkci `sort()` a porovnává přední strany karet.

4.5.5 Editace okruhů

Funkce, která se spustí při kliknutí na tlačítko s `id=edit_card_button` a které se nachází v navigační liště. Vypíše všechny karty do tabulky a umožní jejich úpravu. Jako první se volá funkce `list_cards_to_edit()`.

- `list_tags_to_edit()` - nemá parametry; je pouze organizační
  - Vyprázdní se předchozí tabulka, takže zůstanou jen nadpisy sloupečků, a zobrazí se container s tabulkou.
  - Načtou se názvy a id všech okruhů v databázi do seznamu, na který je pak zavolána funkce `sort_tag_list(tag_list)`, která list seřadí.
  - Pro každé id se vytvoří řádek v tabulce, zbytek ale zůstane prázdný.
  - Pomocí `load_information()` se načtou všechny informace o jednotlivých okruzích a přidají se do odpovídajících řádků podle id.
  - V každém řádku se také vytvoří tlačítko Edit a tlačítko Delete.
  - Po stisknutí Delete se zeptá na potvrzení a pomocí `post_information("add_tag/", create_tag_object( ...))` se pošle request do databáze, kde se okruh smaže. Znovu se zavolá `list_tags_to_edit()`, aby se obnovil seznam okruhů.
  - Po stisknutí Edit se zavolá `edit_tag(tag_info)`
- `edit_tag(tag)` - jako parametr bere objekt JSON, který obsahuje informace o okruhu
  - Funkce funguje dost podobně jako `create_tag()`, ale pole se vyplní podle aktuálních informací o okruhu.
  - Po kliknutí na Save se opět pošle request do databáze pomocí `post_information("add_tag/", create_tag_object("update", ...))`.
  - Po uložení nebo kliknutím na Cancel se zavolá funkce `list_tags_to_edit()`.
- `sort_tag_list(tag_list)` - využívá předdefinovanou funkci `sort()` a porovnává názvy okruhů.

4.5.6 Testy

Funkce, která spravuje výběr okruhu k prozkoušení a typ testu, poté zavolá odpovídající funkci. Spustí se při kliknutí na tlačítko s `id=test_button`, které se nachází v navigační liště.

- `test_main()` - nemá parametry; slouží jen k organizaci
  - Vyprázdní se předchozí obsah a zobrazí se container pro výběr okruhu.
  - Načtou se všechny okruhy z databáze a pro každý se vytvoří tlačítko.
  - Po kliknutí na vybrané tlačítko se zavolá `test_type(event.data[0], event.data[1])`
- `test_type(tag_id, tag_name)` - jako parametry bere id a název okruhu
  - Zobrazí se container pro výběr typu testu se třemi tlačítky, jedním pro každý z typů testu.
  - Po kliknutí na dané tlačítko se zavolá funkce `load_cards("browse"/"choices"/"write", tag_id, is_reversed())`.
- `is_reversed()` - nebere žádné parametry;
  - Zjistí, jestli je přepínač "směru" kartiček sepnutý.
  - Směr znamená, jestli se bude jako otázka brát přední, nebo zadní strana kartičky a odpověď bude ta druhá.
  - Vrací `true` nebo `false`.
- `load_cards(typem, tag_id, is_reversed)` - jako parametry bere typ testu, id okruhu a směr kladení otázek
  - Načte z databáze všechny kartičky, které jsou v okruhu určeným parametrem `tag_id`, do seznamu.
    - Pokud je parametr `is_reversed true`, přední a zadní strany se automaticky prohodí.
  - Po načtení všech karet se zavolá funkce `group_similar_cards(all_cards)`, která vyfiltruje seznam.
  - Podle hodnoty parametru `type` se zavolá odpovídající funkce `browse`/`choices`/`write`, která provede vlastní test.
- `group_similar_cards(all_cards)` - jako parametr bere seznam kartiček, který má vyfiltrovat
  - Projde seznam kartiček a na každou zavolá funkci `contains_similar_front(return_card_list, card.card_front)`. (V `return_card_list` jsou kartičky, které již byly zkontrolovány, nebo spojeny)
  - Pokud vrátí `true`, kartičky se spojí do jedné, aby nebyl problém při hledání správné odpovědi při kartičkách, které mají stejnou přední stranu, ale jiné zadní strany (synonyma).
- `contains_similar_front(card_list, card_front)` - jako parametry bere seznam kartiček a přední stranu porovnávané kartičky
  - Projde všechny kartičky v seznamu a porovná jejich přední strany s přední stranou zadanou v parametrech.
  - Vrátí `[true, index]` nebo `[false, null]`.

Test typu "Browse"

- `browse(all_cards, tag_id)` - jako parametry bere seznam kartiček a id okruhu
  - Funkce, která se stará o typ testu, kde se prochází všechny kartičky a otáčí se.
  - Na začátku se zavolá funkce `update_browse_progress_bar(current_index, count)` a `change_flipcard(all_cards[current_index].card_back)`, které nastaví kartičku na první v seznamu a vynuluje ukazatel nahoře.
  - Podle počtu kartiček se rozhodne, jestli budou vidět tlačítka NEXT a PREVIOUS.
  - Po kliknutí na tlačítka s `id=browse_next` resp. `id=browse_previous`, znovu se zavolají funkce na aktualizování horního ukazatele a změnu hodnot na kartičce. Dále se znovu rozhodne o viditelnosti NEXT a PREVIOUS pomocí funkce `show_next_previous(current_index, count)`.

  - Kliknutím na kartičku se otočí a zobrazí druhou hodnotu.
  - Tlačítko `id=browse_back` znovu zavolá funkci `test_type()`.
- `update_browse_progress_bar(current, max)` - jako parametr bere číslo aktuální kartičky a celkový počet
  - Upravuje Bootstrapový objekt (ukazuje kolikátá kartička je právě prohlížena), aby odpovídal aktuálním hodnotám
- `change_flipcard(front, back)` - jako parametry bere nové hodnoty na přední a zadní stranu
  - Pokud byla kartička ponechána otočená, nejprve se otočí zpátky a až potom se změní hodnoty.
  - Poté dojde ke změně hodnot v HTML stránce.
- `show_next_previous(current_index, max)` - jako parametry bere číslo kartičky a celkový počet
  - Zobrazuje a schovává tlačítka NEXT a PREVIOUS, aby nebyly vidět při první resp. poslední kartičce.

Test typu "Choices"

- `choices(count, correct, wrong, answers, current_word_index, all_cards, tag_info)`
  - Jako parametry bere, celkový počet karet, kolik jich už bylo správně, kolik špatně, číslo momentální kartičky, seznam karet a informace o okruhu - pro další kartičku se zavolá znovu tato funkce, akorát s jinými hodnotami
  - Aktualizují se ukazatele správných/špatných odpovědí a zbývajících odpovědí pomocí funkce `update_write_choices_progress_bar("choices", correct, wrong, count)`.
  - Zobrazí se container se tímto typem testu, nahoře je přední strana kartičky, pod tím jsou čtyři možnosti odpovědí.
  - Funkce `get_random_choices(count, current_word_index, all_cards)` vybere indexy náhodných kartiček, které budou sloužit jako špatné odpovědi.
  - Funkce `get_random_index()` náhodně seřadí čísla od 0 do 3, což způsobí náhodné zamíchání možností mezi čtyři políčka.
  - Pomocí hodnot z těchto dvou funkcí se vyplní možnosti náhodně zvolenými zadními stranami kartiček.
  - Funkce `activate_one()` označí první možnost jako vybranou.
  - Funkce `choose()` sdružuje označování možností jako vybrané po kliknutí.
  - Kliknutím na tlačítko s `id="choices_check_answer` se porovná vybraná možnost se správnou.
    - Zvýší se počet `correct` případně `wrong`, číslo kartičky a uloží se odpověď.
    - Zobrazí se container, kde se ukáže správnost odpovědi a správná odpověď.
  - Pokud je kartička poslední, zavolá se `summary(tag_info, correct, count, answers)`, jinak se zavolá funkce znovu se změněnými hodnotami
- `get_random_choices(count, current_word_index, all_cards)`
  - Jako parametry bere počet kartiček, číslo aktuální kartičky a seznam všech kartiček.
  - Zavolá funkci `count_same_backs(all_cards)` - zredukuje počet karet o ty, které mají stejnou zadní stranu - aby nebylo více stejných možností.
  - Vrátí seznam max. čtyř indexů ze seznamu karet (když je seznam karet kratší než 4, je jich méně).
- `count_same_backs(all_cards)` - jako parametr bere seznam karet
  - Vrátí číslo, které reprezentuje počet unikátních zadních stran v seznamu karet.
- `get_random_index()` - nemá parametry
  - Zamíchá čísla od 0 do 3 v náhodném pořadí - podle toho se přiřazují možnosti.
- `activate_one()` - nemá parametry
  - Odoznačí minulý výběr a označí vždy první možnost jako vybranou.
- `choose()` - nemá parametry

  - Sdružuje **click eventy**, které označují vybranou možnost pomocí `one_choice(x)`.
- `one_choice(index)` - nemá parametry
  - Odoznačí minulou možnost a označí tu která má v id `index`.
- `update_write_choices_progress_bar(type, correct, wrong, max)` - jako parametry bere typ testu, počet správných a špatných odpovědí a celkový počet otázek.
  - Aktualizuje ukazatele zadaných hodnot.

Test typu "Write"

- `write(count, correct, wrong, answers, current_word_index, all_cards, tag_info)`
  - Jako parametry bere, celkový počet karet, kolik jich už bylo správně, kolik špatně, číslo momentální kartičky, seznam karet a informace o okruhu - pro další kartičku se zavolá znovu tato funkce, akorát s jinými hodnotami
  - Aktualizují se ukazatele správných/špatných odpovědí a zbývajících odpovědí pomocí funkce `update_write_choices_progress_bar("write", correct, wrong, count)`.
  - Po kliknutí na tlačítko s `id=check_write_answer` se uloží hodnota zapsaná do textového pole.
  - Pomocí funkce `check_magic(raw_answer, current_card.card_back.toLowerCase())` se zkontroluje výsledek.
    * Zvýší se počet správných/špatných odpovědí, číslo kartičky a uloží se odpověď.
    * Zobrazí se container, kde se ukáže správnost odpovědi a správná odpověď.
  - Pokud je kartička poslední, zavolá se `summary(tag_info, correct, count, answers)`, jinak se zavolá funkce znovu se změněnými hodnotami
- `update_write_choices_progress_bar(type, correct, wrong, max)` - jako parametry bere typ testu, počet správných a špatných odpovědí a celkový počet otázek.
  - Aktualizuje ukazatele zadaných hodnot.
- `check_magic(raw_answer, correct_answer)` - jako argumenty bere zadanou odpověď v malém fontu a správnou odpověď
  - Správná odpověď se rozdělí podle čárek, v případě, že by bylo více možností (synonyma)
  - Pro každou možnou odpověď se vypočítá Levenshteinova vzdálenost pomocí funkce `levenshtein_distance(raw_answer, correct_answer)`.
  - Když funkce vrátí `true`, vyhodnotí se jako správná, jinak se vyhodnotí jako špatná.
- `levenshtein_distance(string_1, string_2)` - jako parametr bere 2 stringy
  - Spočítá počet operací (přidání, vynechání, záměna), které je potřeba provést, aby se `string_1` změnil na`string_2`
  - Když je (počet chyb)/(délka slova) < 0.3, vyhodnotí se jako správná odpověď (tolerance překlepů).
  - Zvýrazní a doplní chyby ve `string_1`.
  - Vrací `[true/false, string se zvýrazněnými chybami]`

Sumarizace testu

- `summary(tag_info, correct, count, answers)` - jako argumenty bere informace o okruhu, počet správných odpovědí, počet otázek, seznam všech odpovědí
  - Zobrazí container se zhodnocením testu.
  - Vypočítá procentuální úspěšnost a zlepšení/zhoršení oproti minulému testu stejného okruhu.
  - Nabízí seznam všech otázek se zadanými a správnými odpověďmi.
  - Pomocí `post_information("add_tag/", create_tag_object("test, ...))` uloží výsledek testu do databáze.
  - Zavolá funkci `test_type()` a uživatel může začít nový test.

4.5.7 Export

Funkce, která se spustí po kliknutí na tlačítko s `id=export_button` a které se nachází v navigační liště. Vypíše všechny kartičky a okruhy do `.csv` souboru včetně jejich spojení.

- `export_data()` - nemá žádné parametry
  - Zobrazí container s polem, kam se vyplní název souboru bez koncovky.
  - Po stisknutí tlačítka s `id=confirm_export` se uloží obsah pole a zkontroluje se formát názvu (nesmí obsahovat speciální znaky a mezery).
  - Pokud splní všechny podmínky, zavolá se funkce `prepare_data(filename)`.
- `contains_special_symbols(string)` - jako parametr bere jakýkoli string
  - Zkontroluje, že string neobsahuje žádný ze speciálních znaků.
- `prepare_data(filename)` - jako parametr bere název souboru
  - Postupně pomocí `load_information("cards"/"tags")` načte všechny kartičky a okruhy z databáze a vytvoří list objektů.
  - Po načtení všeho zavolá funkci `write_file(filename, export_list)`.
- `write_file(filename, export_list)` - jako parametry bere název souboru a seznam objektů
  - Pro každou položku seznamu vytvoří řádek.
  - Pomocí Node.js balíčku `fs` vytvoří ve složce `./export/` soubor `filename.csv`, kam uloží všechny vytvořené řádky.

4.5.8 Import

Funkce, která se spustí po kliknutí na tlačítko s `id=import_button` a které se nachází v navigační liště. Z `.csv` souboru načte kartičky a uloží je do databáze.

- `import_data()` - nemá žádné parametry
  - Zobrazí container s tlačítkem, které zobrazí průzkumníka souborů, kde uživatel vybere soubor, který chce importovat.
  - Po kliknutí na tlačítko s `id=confirm_import` se přečte soubor a pomocí `post_information("import/", process_data(reader.result))` se uloží importovaná data do databáze.
- `process_data(text)` - jako parametr je text, který obsahuje informace k importu
  - Text se rozdělí na řádky, které se pomocí `sort_tags_to_import(entries)` seřadí na karty a okruhy.
  - Vytvoří se JSON objekty karet a okruhů, přičemž se pomocí indexů seřazeného listu propojí příslušné okruhy a karty.
  - Vrací to list objektů, které projdou funkcí `filter_json(result)`.
- `sort_tags_to_import(entries)` - jako parametr bere seznam řádků
  - Seřadí je tak, aby se karty a okruhy správně propojily.
- `filter_json(json_list)` - jako parametr bere seznam JSON objektů
  - Projde seznam a vyfiltruje objekty, které jsou stejné, nebo obsahují nepovolená data (např. není tam přední strana kartičky, . . .).

4.5.9 Další funkce

- `hide_all()` - skryje úplně vše, kromě navigační lišty (ta skrýt nelze).
- `reset()` - zavolá `hide_all()` a zviditelní titulní stranu
- `show_one_item(item)` - bere `id` objektu jako argument; zavolá `hide_all()` a zobrazí daný objekt

- `post_information(suffix, data)` - bere příponu za základní adresu k serveru, kam se pošle request, a data k odeslání (v JSON formátu) jako argumenty; vytvoří **ajax POST** request
- `load_information(suffix)` - bere příponu za základní adresu k serveru jako argument; vytvoří **ajax GET** request a vrátí příchozí data
- `create_card_object(type, id, card_front, card_back, tags)`
  - vytvoří string z JSON objektu (kartička) s hodnotami z argumentů
- `create_tag_object(type, id, tag_name, success_rate, card_count, cards)`
  - vytvoří string z JSON objektu (okruh) s hodnotami z argumentů

4.5.10 Click events

- Každý **click event** musí mít za sebou metodu `.unbind()`, jinak by se příkazy sčítaly a po druhém zmáčknutí daného tlačítka by se funkce spustila dvakrát a tak dále.
- Uvnitř každého **click event**u musí být zavolána funkce `event.preventDefault();`. Jelikož tato aplikace používá JQuery pro správu kliknutí, je potřeba zabránit HTML, aby se snažilo reagovat na kliknutí. Stránka by se obnovila, protože HTML by nevědělo, co dělat, a byla by vidět jen titulní strana.

4.6 package.json

Soubor popisující aplikaci jako celek. Nalezneme zde název, popis, autora, licenci, ... Dále se zde definují scripty, které usnadňují některé operace jako spouštění z terminálu nebo zabalení aplikace. Nejdůležitější jsou závislé položky, které se zahrnou do finálního package.