TODO es posible

Seminární práce

Adrian Bartoň

1. Požadavky na aplikaci

Jako semestrální projekt **musíte vytvořit jednoduchý TODO systém – seznam úkolů a jejich správu.**

- Konzultovat problémy spojené s řešením práce můžete na každém cvičení během semestru, či po dohodě i jindy.
- Cvičení cca od 16.11. (bude upřesněno) jsou pak určena pro práci na projektu, zejména na konzultace a code-review vaší práce. Nebude na nich již vypsán žádný další úkol.

Hlavní požadavky na aplikaci (25 bodů)

- **Implementovat buď SPA + backend API nebo server side rendering aplikaci**
- **Klient i server kód v JavaScriptu, HTML5 a CSS3**. Pokud chcete pro server využít jiný jazyk, je to **po domluvě** možné.
- Dodržet **základní bezpečnostní pravidla** – ochrana proti XSS, SQL injection atd.
- Vhodným způsobem využít JavaScript **také pro klienta** – kontrola formuláře, interakce s uživatelem apod.
- Validní HTML a CSS
- persitentní uložení dat na serveru

Cílem je vytvořit jednoduchý TODO list / seznam úkolů.

Aplikace musí umožnit úkol přidat, upravit, označit jako hotový, smazat kompletně a samozřejmě zobrazit samotný seznam aktuálních / hotových úkolů (CRUD operace).

Seznam úkolů musí být k dispozici ve formátech HTML a JSON. K vyřešení tohoto problému vám dobře poslouží právě MVC architektura, případně REST API, což si vysvětlíme a ukážeme na přednášce.

Základem je implementovat kompletní CRUD operace pro úkol a dále jednoduché ověření uživatelského přistupu. Aplikaci nesmí být možné použít anonymně pro zápis, maximálně pro čtení.

Nemusíte ale dělat kompletní správu uživatelů, několik napevno zapsaných v databázi pro testování je dostatečné. Také můžete použít OpenId a podobné služby.

Požadovaná funkčnost (25 bodů):

- zobrazení úkolů
    - filtr splněných / nesplněných úkolů
    - možnost zobrazení výstupu ve formátech HTML a JSON – přepínání formátu parametrem v URL nebo dle vaší volby (zdokumentovat)
    - json výstup je určený pro další aplikace, musí tedy mít správný content-type a být validní dle JSON pravidel
- administrace úkolů – přidání, úprava, smazání, označit jako hotový
- přihlášení uživatele, zabezpečení administrace úkolů před anonymním přístupem

Další funkce jsou vítané, záleží jen na vašich schopnostech. Můžete například přidat více uživatelů a pro každého vytvořit extra seznam. Nebo přidat možnost úkolovat jiné uživatele.

Pro výsledné body je ale důležitá hlavně kvalita kódu, jeho zdokumentování a případné pokrytí testy, ne počet funkcí navíc. I když se může na vaší známce pozitivně projevit.

Aplikaci musíte vypracovat samostatně – týmová práce na takto jednoduchém projektu není možná.

Je povolené použít pro řešení různé frameworky a knihovny, ale musíte chápat co pomocí nich řešíte, jak to řešíte atd. Nelze brát framework jako kouzelnou černou skříňku, která nějak záhadně dělá to co zrovna potřebujete.

Pokud chcete řešit nějaké jiné téma, je to **po domluvě** možné. Stejně tak je možné odezvdat již hotovou aplikaci, například projekt z jiného předmětu apod. Náročnost aplikace by ale v tomto případě měla demonstrovat, že TODO list hravě zvládnete.

2. Design a uživatelská dokumentace

Aplikace nazvaná „TODO es posbile“ neboli „dělat, jak je to možné“ obsahuje čtyři hlavní moduly

- Base – přihlašovací stránka
- Register – stránka pro registraci uživatele
- Dashboard – stránka pro zobrazení úkolů uživatele v HTML
- JSON – stránka obsahující úkoly v podobě JSON formátu

Base

Stránka obsahuje vstupní formulář pro uživatele, kde je požadováno po uživateli pro následní přihlášení jeho email a heslo. Zároveň nabízí uživateli možnost založení účtu. Barvy na webové stránce jsou sladěny do modré a oranžové barvy. Oranžová je kvůli názvu, který je španělsky a modrá už je jen doplňující prvek.

Na první pohled stránka neobsahuje žádné tlačítko pro přihlášení. Tlačítko čeká na zadání emailu a poté se objeví.

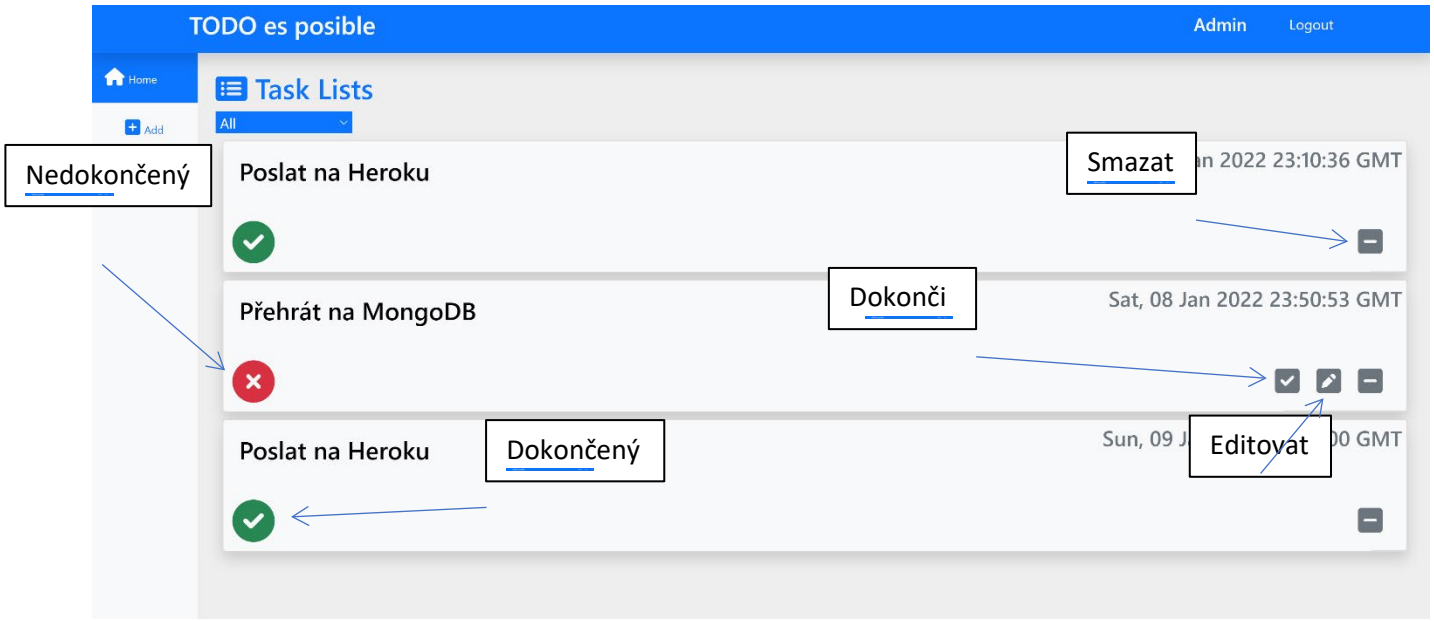
Register

Podobně jako předešlá stránka je konstruovaná i tato registrační stránka, obsahuje registrační formulář se čtyřmi inputy (email, jméno, heslo a potvrzení hesla)

Na první pohled stránka neobsahuje žádné tlačítko pro registraci. Tlačítko čeká na zadání emailu a poté se objeví.

Dashboard

Stránka sloužící pro správu úkolů uživatele, v levém horním rohu je název aplikace, v pravém horním rohu je jméno aktuálně přihlášeného uživatele a také možnost odhlášení. Stránka obsahuje menu, kde je možné si vybrat zobrazení v JSON formátu a filtr, který nabízí uživateli filtrovat dokončené, nedokončené a všechny úkoly. Stránka nabízí uživateli založit úkol.



Uprostřed stránky je list vypsaných úkolů, pro daného uživatele. Ve výchozím neboli přihlašovacím stavu a jakémkoliv jiném stavu, při kterém dochází k opakovanému načtení je filtr nastaven na zobrazení všech úkolů (dokončené i nedokončené). Každý úkol má svou kartičku, která v levém rohu obsahuje text úkolu, v pravém horním rohu datum založení, také obsahuje tři tlačítka na dokončení, upravení textu a smazání úkolu, jak je znázorněno na obrázku. Pokud uživatel odklikne úkol jako dokončený, nemá možnost už jej vrátit, tudíž se v levém dolním rohu objeví zelená fajfka namísto červeného kříže, který značí nedokončený úkol.

JSON

Rozložení stránky je stejné jako na předchozí stránce. Akorát zobrazení úkolů je ve formátu JSON.

3. Technická dokumentace

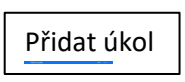
Projekt TODO list využívá NodeJs, EJS, JavaScript a Express. Aplikace využívá metody REST API, tzv. trasování mezi moduly. Struktura aplikace je rozložena do čtyř částí (models, public, routes, views).

Aplikace využívá noSql MongoDB, tudíž nepotřebuje žádné SQLInjection, které je požadováno v první časti. Obsahuje, ale jednotlivé validátory proti XSS injection, které omezující uživatele na vstupu, nebo jej samotný vstup upravují.

Pro validace emailu, textu a uživatelského jména jsou také napsány Unit testy.

Jednotlivé rozdělení adresářů

- **Models**
  - Obsahuje modely, které jsou využívány databázi
    - Task.js – obsahuje model pro řízení dat z databáze.
    - User.js – obsahuje model s datovými typy, které pak ukládá a čte z databáze.
- **Public**
  - Obsahuje veřejné assety, které v tomto případě není moc využito, jelikož při zobrazování snímků v HTML jsou použity URL odkazy. Navíc public obsahuje CSS soubor stylů pro EJS stránky.



- **Views**

    - Obsahující jednotlivé části stránky, které se pak naimportují a složí jednotnou HTML stránku.
        - Header – obsahuje začínající parametry kostry pro HTML
        - Footer – obsahuje koncové parametry kostry pro HTML
        - Base, Dashboard, Register a JSON
- **Routes**
    - **Nodejs** moduly obsahující směrování a chování aplikace.
        - **Route** řídící smyčka která obsluhuju chování pro celou aplikaci, od přihlášení po odhlášení
- **Server.js**
    - Základní nastavení pro aplikaci, propojení modulů a html. Definice volajících metod. Zde je definován port a databázové připojení.
- **User.test.js**
    - Javascriptový soubor obsahující testy na validaci emailu a uživatelského jména

Uživatelské akce

- **Přidání uživatele**
    - Samotné přidání uživatele je možné pro prokliknuti odkazu na hlavní přihlašovací stránce. Vyplnění čtyř inputů, se zobrazí tlačítko, která obsahuje atribut volající do backendové části programu o založení uživatele, kde se stáhne příslušný model uživatele z adresáře /Models a také vstupy od uživatele (email, jméno a heslo), podle daných parametrů se vytvoří uživatel Mongoose.Save( ) se zaheschovaným heslem. Po kliknutí na tlačítko „Registrace“ se uživatel vrátí na přihlašovací stránku, kde dostane zprávu zda pokus o vytvoření byl úspěšný zda nikoli. Využitá async a await, kde program čeká na eventuální dokončení.
- **Přihlášení uživatele**

    - Uživatel zadá na hlavní stránce svůj email a heslo, které zase pomocí akce zavolá metodu v routu a ověří si zda uživatel existuje, pokud ano tak jej přesměruje na /dashboard s daty, které stáhne podle uživatelského ID z tabulky úkolů (stáhne pouze data, které obsahují ID uživatele) a pokud ne, tak vypíše po názvem aplikace chybovou hlášku.

- **Vytvoření úkolu**
    - Úkol se tvoří podle zadaných parametrů uživatele. Tedy uživatel klikne na tlačítko v dashboardu, zobrazí se formulář se vstupem, kam uživatel zadá text úkolu. Úkol se poté vytvoří s aktuálním datem a časem a pošle se do databáze, stránka se aktualizuje a při aktualizace si stáhne data z databáze a zobrazí úkoly uživatele.

- **Odebrání**
    - Funguje na principu smazání z databáze, kde si podle ID úkolu, které je poslané v přes přesměrování, najde daná úkol s ID a smaže ho. Poté aktualizuje stránku a tím zajistí aktualitu dat.

- **Dokončení úkolu**
    - Funguje podobně jako odebírání, ID úkolu se předává spolu s přesměrováním, poté si najde úkol s daným ID a změní stav z „uncompleted“ na „completed“

- **Načítání úkolů uživatele**
    - První metoda, která proběhne po načtení stránky. Stáhne z tabulky úkolů („tasks“) všechny úkoly s uživatelským ID, které se předává v Cookies. Po dosažení všech úkolů uživatele se spustí v js script, který provede tzv. založení s parametry, na rozdíl od zakládání, které má výchozí parametry, tento objekt se načítá s parametry, které jsou zadány v databázi.

Struktura Databáze

- Databáze obsahuje dvě tabulky, uživatelé a úkoly.
    - Uživatelé(users)
        - Id uživatele – typu mongoose.IdObject, primární klíč

        - Jméno uživatele – typu string
        - Email – typu string
        - Heslo – typu hashed string
    - Úkoly(tasks)
        - Id úkolu - typu mongoose.IdObject, primární klíč
        - Id uživatele - typu mongoose.IdObject, unikátní klíč
        - Nadpis – typu string, zpráva úkolu
        - Stav úkolu – typu string, zda je úkol dokončen
        - Datum – typu Date, datum vytvoření úkolu