Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Moduly systému

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Vyhotovil: Matúš Cuper, Ondrej Kaščák, Lukáš Manduch, Ivana Mujgošová, Jozef Pazúrik, Ondrej Pitoňák, Ondrej Selecký, Martin Schnürer
Dátum odovzdania: 10.05.2018

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Spracovanie udalostí

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 09.05.2018

1. Analýza

Modul realizuje funkcionalitu, ktorá prijíma udalosti od elektronického obchodu, upravuje ich štruktúru podľa schémy a ukladá do Pio databázy. Zo skúseností vieme že elektronické obchody majú tendenciu sa často meniť a nakoľko je dôležité aby prichádzajúce udalosti dodržiavali presný formát, je nutné ich na začiatku kontrolovať voči dohodnutému formátu. Na druhej strane ale niektoré drobné zmeny vo formáte udalosti nijako neovplyvnia jej spracovanie, podobne nie všetky atribúty udalosti sa pri spracovaní využívajú. Preto je nutné vykonávať kontrolu iba pre prítomnosť konkrétnych atribútov a vždy sa pokúsiť o spracovanie udalosti, nezávisle na výsledku kontroly. Výsledkom kontroly v prípade neplatnej udalosti je poslanie notifikácie do integrovaného klienta.

2. Návrh

Udalosti prijaté od elektronického obchodu musia mať pre ďalšie spracovanie dohodnutú štruktúru. Formát udalostí je pre každý elektronický obchod odlišný, udalosti jedného obchodu sa tiež môžu v malej miere líšiť. Preto je ich nutné modifikovať. Modul obsahuje funkcie na modifikáciu a možnosť nakonfigurovania modifikátorov pre konkrétne typy udalostí prestredníctvom konfiguračných súborov.
Ďalšou funkcionalitou je ukladanie do pio event storage. To je realizované pio klientom, ktorý odošle udalosti na špecifikovanú adresu.

3. Implementácia

Implementácia sa nachádza v komponente *core*. Najdôležitejšie súčasti modulu sú:

- *config*: obsahuje konfiguračné súbory.
    - *Common.py*: definícia modifikátorov pre konkrétne *event_repositories*
    - *development.py, staging.py, production.py* - konfigurácie pre rôzne typy prostredí
- *event_modifiers*: modifikátory. Každý modifikátor je implementovaný ako samostatná trieda odvodená od *EventModifier*. Modifikátory implementujú metódu *modify()*, ktorá modifikuje udalosť na základe parametrov modifikátora zadaných v jeho konštruktore. Modifikátory pre konkrétne repositorie sú zadefinované v konfigurácii.
- *event_repositories*: Repozitáre udalostí, ktoré zabezpečujú ukladanie do databáz a modifikáciu udalostí
- *pio_rest_event_client.py*: zabezpečuje odosielanie udalostí na pio rest api.
- *settings.py*: nastavenia spracovávača podľa konfigurácie. Pri konštrukcii sa vytvoria inštancie repozitárov a vytvoria sa modifikátory pre každý z nich.
- *event_processor*: Spracovávač udalostí. Metóda *process()* pre udalosti na vstupe zavolá ich uloženie zavolaním *store()* pre všetky repozitáre
- *rdb_mapping_repository*: mapovanie identít z prichádzajúcich udalostí do tabuliek *cookies* a *emails*, okrem vkladania nových záznamov poskytuje aj vyhľadávanie *cookie_id* na základe ID používateľa
- *rdb_feedback_repository:* slúži na ukladanie informácií o spätnej väzbe. Medzi ne patrí ukladanie odporúčaní, impresií, klikov a konverzií. Pre jednoduchšiu prácu s dátami sa nad nimi vytvorili indexy.

Spracovanie udalostí prebieha nasledovne: Pri zavolaní metódy *process()* objektu *EventProcessor* sa pre všetky *repositories* zavolá metóda store(), ktorá vykoná modifikáciu udalosti a vykoná uloženie potrebných dát do databázy. Metóda store je vykonávaná asynchrónne.

Kontrola štruktúry:
Implementácia kontroly prijatých udalostí sa nachádza v module *core* v triede *StructureChecking.* Táto trieda je založená na dynamickom vytváraní tabuľky povolených atribútov v prichádzajúcich udalostiach. Táto tabuľka sa ukladá do databázy pod názvom *event_structure.*

V rámci kontroly sa vytvára tabuľka všetkých atribútov udalosti na najvyššej úrovni v prijatom JSON-e. Pre každú kombináciu hodnôt *entityType, event, targetEntityType,* sa uloží zoznam kľúčov atribútov v prijatom JSON-e. Pre každý atribút sa navyše pridá jeho status ktorý môže byť: *'required', 'optional', 'forbidden', 'not_determined'.*
Každý nový detekovaný atribút sa automaticky nastaví na *'not_determined'* a vygeneruje sa upozornenie o detegovaní. Podobne sa vygeneruje upozornenie v prípade detekcie zakázaného atribútu, alebo absencie vyžadovaného atribútu. Hodnoty statusov pre jednotlivé atribúty sa nastavujú cez osobitné grafické rozhranie.

Kontrola kvantity:
Okrem kontroly štruktúry sa vykonáva aj kontrola kvantity udalostí. To znamená, že sa uchováva počet prijatých udalostí v hodinových časových intervaloch pre každý typ prijatej udalosti.

Počas príchodu udalostí sa ich počet ukladá v redise. Z prijatej udalostí sa zistí čas (deň a hodina) a typ udalosti, pre daný čas a typ sa v redise inkrementuje počítadlo. V prípade, že príde udalosť, ktorá obsahuje časový údaj s inou hodinou ako dátum poslednej udalosti, údaje o počtoch sa z redisu vymažú, uložia sa do postgres databázy a vytvoria sa nové počítadlá s časom poslednej udalosti.

Počty prijatých udalostí sa ukladajú v databáze do tabuľky s názvom *event_statistics*. Tabuľka obsahuje atribúty : *date, event_type, quanity, weekday a status.* Weekday je číselný atribút, ktorý reprezentuje deň v týždni a je v rozsahu 0 (pondelok) až 6 (nedeľa). Status je textový atribút, ktorý môže nadobúdať hodnoty *'valid', 'invalid', 'unchecked', 'forced_valid'.*

Každú hodinu je automaticky spúšťaný cron job, ktorý vykoná validáciu udalostí v tabuľke. Najprv sa zapíšu počítadlá do databázy, potom sa vykoná validácia predchádzajúcej hodiny. Vypočítajú sa priemerné hodnoty počtu prijatých udalostí za posledných maximálne *n* dní. Priemer sa počíta osobitne pre pracovné dni a pre víkendy. Ak je priemer mimo prahovej hodnoty nastaviteľnej v konfigurácii, hodnota status v zázname v tabuľke sa nastaví na *invalid*, inak na *valid.* Nastavenia prahovej hodnoty a počtu dní pre ktorý sa počíta priemer, sa nastavujú v spoločnom konfiguračnom súbore common.py. Konkrétne sa jedná o hodnoty *EVENT_COUNT_THRESHOLD* a *EVENT_HISTORY_DAYS*. Prahová hodnota predstavuje predstavuje pomer voči priemernej hodnote o ktorý sa môže líšiť aktuálny počet

eventov. Napr. hodnota 0.2 predstavuje 20% tolerenciu. Ak je rozdiel väčší alebo menší ako uvedený pomer, kontrola odošle upozornenie do sentry.

Ďalšou prípustnou hodnotou poľa *status* je *forced_valid*. Táto hodnota je určená pre manuálne nastavenie konkrétnej hodnoty na valid, s tým že tento záznam bude zároveň predstavovať nový základ pre počítanie priemeru. To znamená, že staršie udalosti ako daná udalosť nebudú brané do úvahy pri výpočte priemeru. Tento atribút je určený pre prípad kedy by sa výrazne zmenil počet prichádzajúcich udalostí pre konkrétny typ v dôsledku zmien na strane servera, plánovanej odstávky, alebo nasadenie novej verzie odporúčača, teda nie v dôsledku chyby.

4. Testovanie

Modul obsahuje jednotkové testy, ktoré sú zamerané na testovanie správnosti modifikátorov, pio klienta a *pio_event_repository*. Samostatný test je vytvorený pre kontrolu štruktúry aj pre kontrolu kvantity.

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Kafka konzument

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Vyhotovil: Ondrej Kaščák
Dátum poslednej zmeny: 09.05.2018

Kafka je distribuovaná platforma slúžiaca pre prácu s tokom udalostí. Základná idea je založená na publish/subscribe princípe a dvoch druhoch entít producentovi a konzumentovi. Zjednodušene povedané, producent udalosti zapisuje do základnej dátovej štruktúry nazývanej *topic* a konzument na druhej strane zapísané správy získava.

1 Analýza

Použitie Kafka konzumenta na získavanie udalostí potrebných pre tvorbu odporúčacieho modelu je možné v prípade ak náš zákazník na svojej strane disponuje inštanciou Kafky, do ktorej zapisuje udalosti zo svojej aplikácie. Takéto riešenie je vhodné pre obe strany, pretože zákazník sa nemusí starať o posielanie udalostí na náš externý end-point a navyše zapísané udalosti môže poskytnúť N ďalším konzumentom.

Formát udalostí, ktoré konzument z topic-u prečíta je závislý od producenta, teda od nášho zákazníka. Avšak keďže sa vo väčšine prípadov jedná o internetové stránky, udalosti sú vo formáte JSON, ktoré je možné jednoducho deserializovať.

2 Návrh

Pre tvorbu nášho Kafka konzumenta a samotné pripojenie na Kafka server sme využili Python knižnicu *kafka-python*. Jedná sa o natívnu Python knižnicu, ktorá poskytuje všetku nami požadovanú funkcionalitu.

Konzument bude získané udalosti ďalej predávať modulu zodpovednému za spracovanie udalostí (5.1. Spracovanie udalostí).

3 Implementácia

Konzument sa nachádza v priečinku *kafka_consumer*. Konkrétne sa jedná o súbor *consumer.py*. V tomto súbore nájdeme implementáciu nášho konzumenta, ktorý je reprezentovaný triedou *RecommersConsumer*. Pri vytváraní inštancie konzumenta je potrebné zadať všetky požadované atribúty ako napríklad: Kafka server, názov topic-u, SSL certifikáty.

Po vytvorení inštancie konzument navonok poskytuje dve metódy a to:

- *consume* - zavolaním tejto metódy sa začne vykonávať nekonečný cyklus, v ktorom sa získavajú udalosti z Kafka topic-u a ďalej predávajú modulu zodpovednému za spracovanie udalostí.
- *shutdown* - zavolaním tejto metódy dôjde k vypnutiu konzumenta.

Manažment offsetov:

- offset posledného prijatého eventu sa ukladá do in-memory databázy Redis
- pre chod konzumenta je teda potrebné mať spustenú túto databázu
- pri štarte sa konzumert pokúsi získať posledný offset z Redisu
    - ak sa mu to podarí, nastaví sa jeho pozícia na tento offset v rámci topicu
    - ak sa mu to nepodarí, tak konzument číta všetky eventy od začiatku definovaného topicu.
- počas získavania eventov sa do Redisu vždy zapíše offset posledného eventu

4 Testovanie

Kafka konzument sa momentálne netestuje.

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Odporúčanie

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 11.12.2017

1 Analýza

Pre poskytovanie odporúčania, používame systém PredictioIO. Na to aby sme boli schopný poskytovať odporúčanie, musíme byť schopný prijať požiadavku a rozhodnúť sa ktorý odporúčací systém použijeme.

2 Návrh

Pre pripojenie k pio je použitá knižnica PredictionIO, ktorú je možné jednoducho získať pomocou nástroja pip a ktorá ponúka vhodné rozhranie na komunikáciu s PredictionIO.

3 Implementácia

Kód realizujúci odporúčanie sa nachádza v priečinku app_django/core/recommend.py. Tu sa získavajú odporúčania z odporúčacích systémov, napr. PredictionIO. Django vystavuje endpoint pre získavanie odporúčaní, cez ktorý sa volá implementácia zo súboru recommend.py.

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

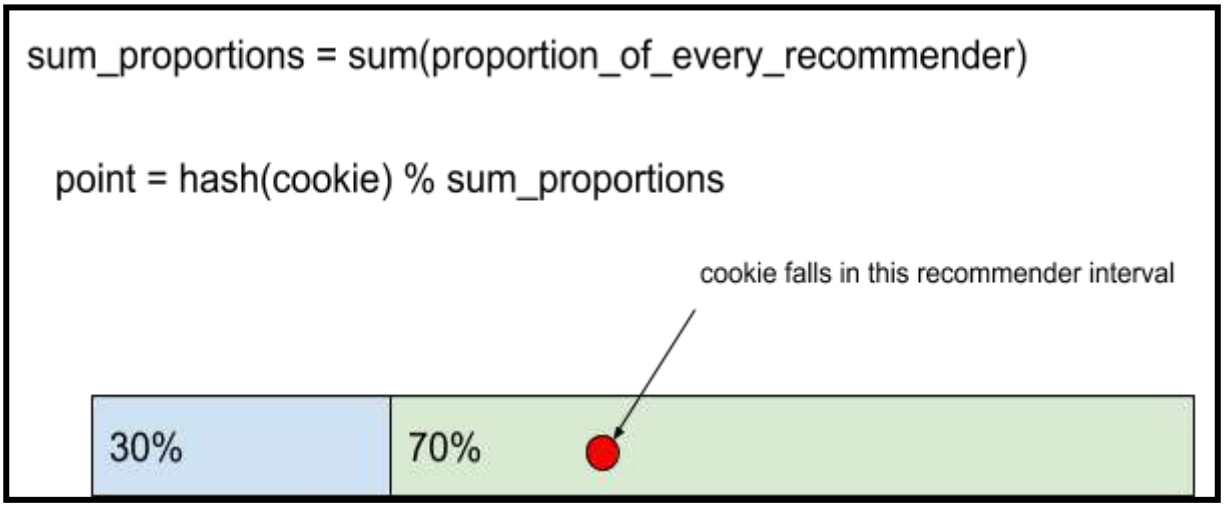
A/B filtrovaniu odporúčačov

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Vyhotovil: Martin Schnurer
Dátum poslednej zmeny: 1.5.2018
Verzia: 1.0

Úvod

Princíp:

- rovnaká cookie padá podľa hashu do rovnakého bodu v intervale - vieme vybrať intervaly odporúčačov 30/70, pri requeste istého používateľa padne hash cookie-ny do rovnakého bodu.



Pri požiadavke na endpoint /recommend musí byť prítomný aj typ eventu a potrebné informácie pre daný odporúčač pre zvolený event.
V jadre nášho programu je uložená definícia odporúčačov = agregované podľa eventu.
V metóde recommend sa zvolia typy podľa daného názvu eventu.

Príklad pre event “show_deal”
Zvolené odporúčače pre event “show_deal”

Rozhodujúci pri každom odporúčači pri evente show_deal je atribút *ab_proportion*.
Algoritmus najprv rozdelí odporúčače podľa typov = agregácia na podľa kľúča *type*.
Teraz pre každý kľúč nastane filtrácia na základe hashu cookieny.

```
recommenders = Array()
for every type_aggregation:
   sum = get_sum_of_all_recommenders_at_type()
   chosen_recommender = get_rec_by_cookie(cookie)
   recommenders.add (chosen_recommender)
```

Pre každý agregovaný kľúč (podľa typu odporúčača) sa vyberie vlastne iba jeden odporúčač = ten, pre ktorý vyhovuje cookie ktorá padla do intervalu v rámci sumy *ab_proportion.*

ab_proportion moze nadobudat v sume aj vacsie hodnoty ako 100. (percentá). Nemali by sa skúšať hodnoty menšie ako 0.

Priklad:

```
show_deal: [
  {
    ‘type’: ‘A’,
    ‘ab_proportion: 70’,
    ….
  },
  {
    ‘type’: ‘A’,
    ‘ab_proportion: 30’,
    ….
  },
  {
    ‘type’: ‘B’,
    ‘ab_proportion: 20’,
    ….
  },
  {
    ‘type’: ‘B’,
    ‘ab_proportion: 80’,
    ….
  },
  {
    ‘type’: ‘C’,
    ‘ab_proportion: 70’,
    ….
  }
]
```

Pre nahodnu cookienu sa potom odporucace prefiltruju podla typov:
(Z kazdeho typu jeden)

```
show_deal: [
  {
    ‘type’: ‘A’,
    ‘ab_proportion: 70’
  },
  {
    ‘type’: ‘B’,
    ‘ab_proportion: 80’
  },
  {
    ‘type’: ‘C’,
    ‘ab_proportion: 70’
  }
]
```

Ak typ obsahuje proportions 80,20, tak to znamená, že pre 80% cookien bude použitý odporúčač s ab_proportion 80 a pre 20% tych istých cookien bude odporúčať vždy odporúčač s ab_proportion 20.

Ak by z jedneho typu existoval iba jeden odporucac a mal nastavene ab_proportion 0, tak sa odfiltruje aj ten = odporucac je **disabled =** nepouzije sa

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Šablóna odporúčacieho systému

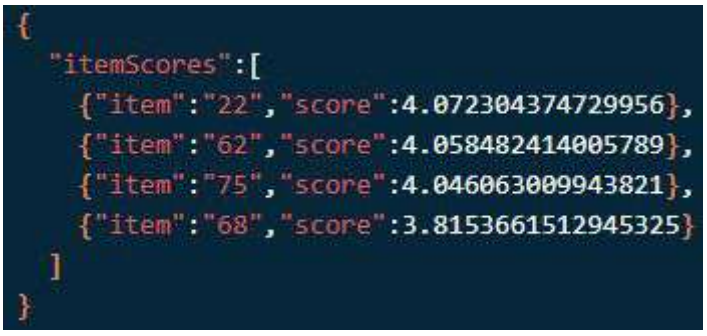
Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Vyhotovil: Ondrej Kaščák
Dátum vypracovania: 6.5.2018

1. Opis

Šablóna odporúčacieho systému sa nachádza v projekte v priečniku **dummy_recommender.**

Jedná sa o jednoduchú Flaskovú aplikáciu, do ktorej stačí doplniť volanie vlastného odporúčacieho systému, ktoré je parametrizované podľa prijatého payloadu z requestu.

Návratová hodnota musí spĺňať nasledujúcu vzorovú štruktúru:



Táto štruktúra je zhodná a vychádza zo štruktúry prediction.io.

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Schema builder

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 28.03.2018

1. Riešenie problému

- Potreba zadefinovat vlastnú schému pre PIO.
- Vlastné fieldy, ktoré potrebuje pio pre nejaký konkrétny odporúčač.

Riešenie predtým: definované schémy pomocou knižnice marshmallow v súboroch umiestnené v core/schemas

Riešenie teraz: Schémy sú uložené v tabuľke recommender_types a podľa nich sa dynamicky vytvára custom schéma zdedená od *Schema* knižnice Marshmallow.

Čo rieši marshmallow ?
- Kvázi funkcia, ktorá transformuje objekt alebo pole dát na určitý typ, ktorý je zadefinovaný v tejto triede Schema.

Ako nastava transformacia ?
Potrebujeme dostať tvar

Pred transformaciou	Po transformacii

{
‘a’: ‘3.14’
}
	{
	‘c’: 3.14
	}

Tato Schema by vyzerala nasledovne:
class MySchema (Schema):
c = fields.Float(load_from=’a’, required=True)

Ak by sme vo vstupnom objekte nemali attribut ‘a’, nacitanie schemy by vyhodilo Exception kvoli tomu, ze tento parameter je vyzadovany (required=True)

Ako nastava transformacia na list ?
Ak chceme dostat vo vyslednom attribute list, je ocakavany vstupny attribut string, ktory obsahuje dane itemy oddelene ciarkou.
Napr. {‘a’: ‘1,2,3,4’ } a na vystupe ocakavame {‘a’: [1, 2, 3, 4]}

```
@pre_load
def split_excluded_items(self, item):
    for key, obj in schema.items():
        if 'list_' in obj['type']:
            if 'load_from' in obj:
                if type(item[obj['load_from']]) is str:
                    item[obj['load_from']] = item[obj['load_from']].split(',')
            elif key in item:
                if type(item[key]) is str:
                    item[key] = item[key].split(',')
            else:
                pass
```

Ako funguje schema builder ?
Vstupna funkcia ocakava jeden paramete - dictionary.
/*
{
‘field_a’: {
‘type’: -> typ ktorym bude ‘**field_a**’ possible values: 
‘missing’ -> OPTIONAL, znamena aka bude defaultna hodnota ak chyba atribut v objekte
‘load_from’ -> z akeho attributu sa ma nacitat hodnota, v novom objekte bude nazvany attribut menom **field_a**,
‘required’ -> tento fields sa musi vytvorit, ak sa nevytvori, vyhodi sa **Exception**
}
}
*/

Na zaklade tohto dictionary sa vygeneruje schema, do ktorej mozeme dalej nacitavat data
result = schema.load(data)
result.data # Obsahuje objekt s definovanymi fieldami

Este raz vsetky mozne typy pre field: list_int, list_float, list_boolean, list_str, int, float, boolean, str

-> list_int sa prekonvertuje na **fields.List(fields.Int())**

2.Jednduchy priklad

```
schema_pattern = {
   ‘items’: {
      ‘type’: ‘list_int’,
      ‘required’: True
   },
   ‘name’: {
      ‘type’: ‘str’,
      ‘missing’: ‘user’,
      ‘load_from’: ‘User’
   }
}
schema = generate_schema(schema_pattern)

some_data = {
    ‘items’: ‘1,2,3,4,5’,
    ‘User’: ‘Adam’
}

result = schema.load(some_data)

print(result.data) # { ‘items’: [1, 2, 3, 4, 5], ‘name’: ‘Adam’ }
```

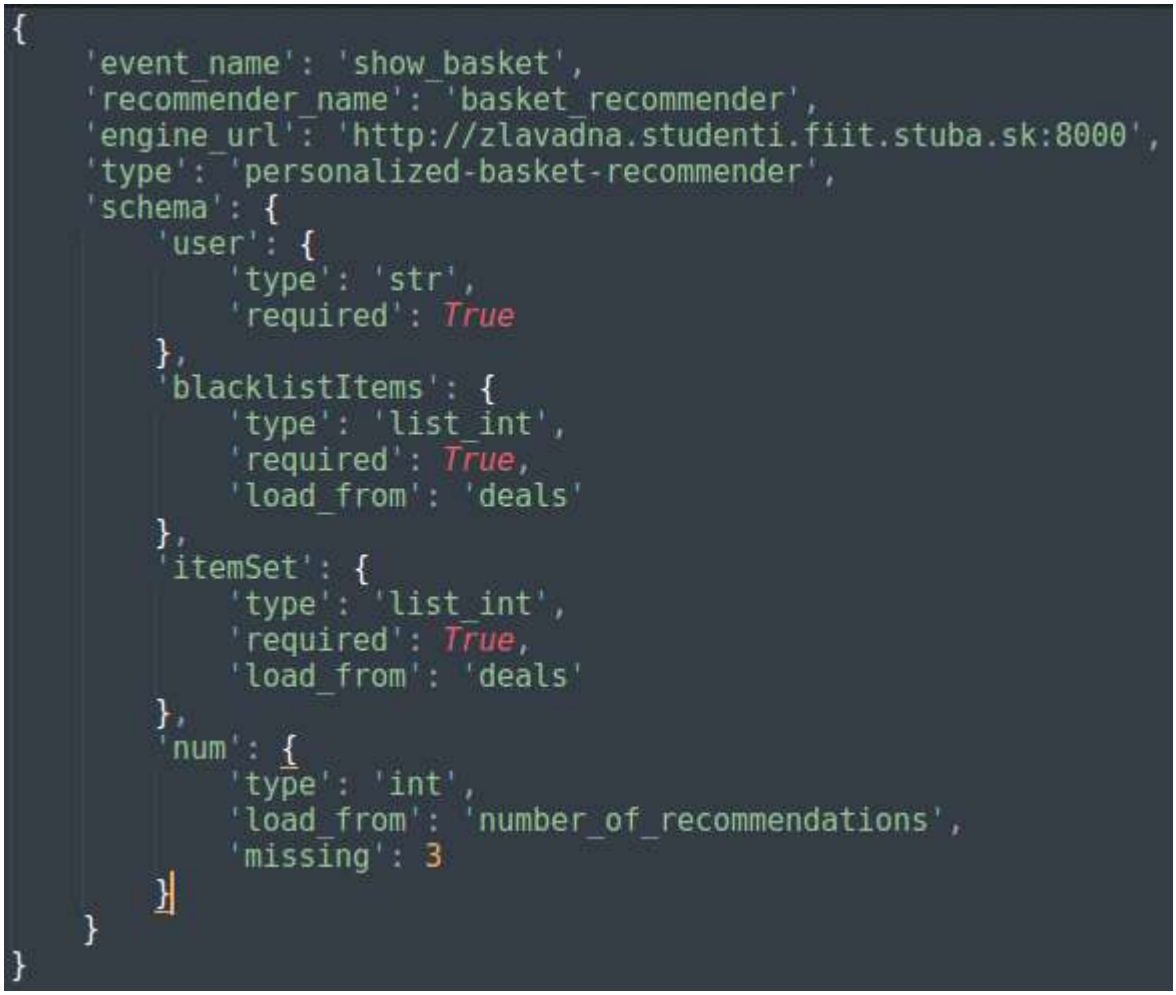
Meta schema builder FE

FE pozostáva z 3 obrazoviek:

1. **index.html** - Obsahuje tabuľku vytvorenú pomocou *datatables.js* obsahujúcu zoznam typov odporúčačov. Tabuľku je možné zoraďovať a filtrovať. Posledný stĺpec obsahuje operáciu *update*
2. **add.html** - Slúži na vytváranie nových typov odporúčačov. Dynamicky vzniká formulár na základe konfiguračného súboru. Pri ukladaní nového typu sa zozbierajú iba dáta, ktoré boli skutočne vyplnené. Ostatné (prázdne) vstupné polia sa neposielajú
3. **update.html** - Rovnaký formulár ako *add* ale obsahuje už predvyplnené vstupné polia. Inak sa správa rovnako ako add

Konfiguračný **config.json** súbor slúži na jednoduchú zmenu formulárov *add* a *update* pri zmene parametrov PIO. Stačí jednoducho pridať alebo zmeniť názov parametra (key) a pridať dátový typ (value).

komunikácia s BE prebieha pomocou JSON-u.
Ukážka štruktúry:



Tento formát sa používa pre všetky 3 obrazovky. Špeciálna je jedine obrazovka update, ktorá spolu s týmto JSON-om posiela ako samostatný parameter aj id upravovaného typu.

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

reCommers

Webová aplikácia v Djangu

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 11.12.2017

Django aplikácia

Django je webový rámec, ktorý prijíma dopyty od používateľa a odpovedá buď zobrazením pohľadu webovej stránky alebo odoslaním dát vo formáte JSON. Django pozostáva z aplikácií, pričom každá aplikácia zabezpečuje jednu funkcionalitu.

1. Analýza

Vytváraná webová aplikácia potrebuje spracovávať používateľské požiadavky, a odpovedať ako formou zobrazenia pohľadu s vizlualizáciou dát alebo vo formáte JSON.

Samotné Django zabezpečuje hlavne aplikačnú logiku a neobsahuje vizualizácie na dostatočnej úrovni pre potreby nášho projektu. Existujú šablóny ako napríklad AdminLTE2, ktoré sú dostatočne prispôsobiteľné, jednoduché a prehľadné. Vizualizácie databázových dát vieme zabezpečiť buď získavaním dát z databázy a ich vizualizáciou pomocou knižníc ako napríklad D3.js alebo napríklad pomocou vizualizačnej služby Metabase, ktorá sa pripojí na databázu, získava z nej dáta a vizualizuje ich pomocou vopred stanovených modelov. Výsledné vizualizácie je možné integrovať do webovej stránky pomocou vnorených okien.

Odpovede vo formáte JSON vie zabezpečovať ako samotné Django tak aj jeho nadstavba Django REST, ktorá je určená na vytváranie a používanie REST API.

2. Návrh

Využívame administrátorskú šablónu AdminLTE2, ktorú sme zjednoušili a prispôsobili na potrebu nášho projektu.

Vizualizáciu dát z databázy zabezpečujeme pomocou vizualizačnej služby Metabase. V tejto službe sú defionované dopyty na databázu a vizualizácie, ktorá sa na získané dáta aplikujú. Samotné vizualizácie integrujeme webového rozhrania pomocou vnorených okien.

REST rozhranie je postavené na Django REST.

3. Implementácia

Koreňový adresár obsahuje rôzne súbory a priečinky. Medzi dôležité súbory patrí hlavne **manage.py**, ktorý celú aplikáciu spúšťa. Tiež je dôležité prečítať si pokyny v súbore README.md

Priečinky koreňového adresára sú vo všeobecnosti Django aplikácie. Medzi nimi je napríklad priečinok reCommersDjangoo, ktorý obsahuje tieto dôležité súbory:

- **urls.py** - Zabezpečuje smerovanie medzi Django aplikáciami
- **settings.py** - Obsahuje základnú konfiguráciu vrátane zoznamu Django aplikácií alebo priečinku so statickými súbormi

Vizualizácia dát je obsiahnutá v samostatnej Django aplikácii s názvom **admin_dashboard**. Aplikácia využíva šablónu AdminLTE2 v upravenej forme a obsahuje tieto základné priečinky:

- **static** - Obsahuje statické dáta ako napríklad obrázky CSS a JS súbory
- **templates** - Obsahuje šablóny na zobrazovanie dát, ktoré ďalej delíme na
    - **admin_dashboard** - Obsahuje samotnú AdminLTE2 šablónu
    - **includes** - Obsahuje šablóny komponentov, ktoré sa nachdzajú na väčšine pohľadov

Okrem pričinkov aplikácia admin_dashboard obsahuje aj niektoré dôležité súbory

- **urls.py** - Zabezpečuje smerovanie v rámci šablóny
- **views.py** - Zabezpečuje vykresľovanie šablón a pohľadov

Samotná vizualizácia dát prebieha v šablóne **templates/admin_dashboard/data_visualization/index.html**, ktorá integruje Metabase pomocou **iframe**, ktorý musií obsahovať triedu **iframe_date_metabase** a obsahuje atribút **data-graph_id** s ID grafu z Metabase. Je potrebné, aby atribút **src** ostal prázdny. O jeho vytvorenie sa stará JavaScript. Štýl je možné vytvoriť podľa potrieb a nie je centrálne regulovaný. Každý iframe je obalený do Bootstrap tab, ktorým je na začiatku a pri zmene dátumu pridaná trieda **do_refresh**. Obnovujú sa iba vnorené okná, ktorú su v tabe s triedou **active** a ktoré obsahujú triedu do_refresh. Po vyvolaní obnovenia vnoreného okna sa trieda do_refresh odstráni. Tak sa zabráni opakovanému načítaniu rovnakého obsahu.

Pohľad obsahuje aj formulár a tlačidla na výber časového rozpätia, za ktoré sa majú zobraziť dáta. Používajú sa pri tom 2 JS knižnice. Knižnica Bootstrap Datepicker, vďaka ktorej je možné vybrať presné rozpätie dátumov a knižnica Moment, ktorá sa stará o prácu s dátumami.

Django lokalizácia

Django implicitne podporuje lokalizáciu. Využíva pri tom **`i18n_patterns` URL**, ktorým sa automaticky dopĺňa jazyk na základe nastavení prehliadača. Samotné preklady sa vytvárajú v **django.po** súboroch uložených v priečinku **locale** a jeho podpriečinkoch. Tieto súbory vytvára Django automaticky. Následne sa ručne doplnia o preklady a nakoniec sa kompilujú na django.mo, ktoré sa používajú na samotné preklady.

Nastavenie

V **`reCommersDjango/settings.py`** je potrebné nastaviť použitie lokalizácie. V prvom rade sa importuje **from django.utils.translation import ugettext_lazy as _** následne sa medzi middleware doplní **`django.middleware.locale.LocaleMiddleware`** a to presne medzi Session a Common middleware. Ďalej potrebjeme doplniť a pozmeniť nastavenia tak, aby zodpovedali tomuto vzoru. Pričom niektoré sa menia a niektoré dopĺňajú.

```
LANGUAGES = (
      ('en', _('English')),
      ('sk', _('Slovak')),
)

LOCALE_PATHS = (
      os.path.join(BASE_DIR, 'locale'),
)

LANGUAGE_CODE = 'en-us'
USE_I18N = True
USE_L10N = False
```

Nový jazyk sa pridáva do LANGUAGES, kde je potrebné vyhľadať skratku a pridať ju do samostatného riadku.

URL

Lokalizáciu je potrebné riešiť v **`reCommersDjango/urls.py`** a nie až priamo v aplikáciách. Dôvod je, že lokalizáciu nie je možné riešiť v importovaných urls.py súboroch. Importuje sa **`from django.conf.urls.i18n import i18n_patterns`** a namiesto **`urlpatterns=()`** sa použije **`urlpatterns=i18n_patterns()`**. Inak všetko ostáva rovnaké. Ak potrebujeme URL, na ktorú sa lokalizácia nebude aplikovať, vyriešime to následovne.

```
urlpatterns = []
urlpatterns += i18n_patterns()
```

Pričom do prvého poľa dáme URL, na ktoré sa nemá aplikovať lokalizácia a do druhého poľa URK, na ktoré sa má aplikovať lokalizácia.

Lokalizácia textov

Každá šablóna, ktorá má byť lokalizovaná musí na začiatku použiť **{% load i18n %}**. Následne každý jeden preklad je potrebné vložiť do **{% trans** "<text>" **%}.** Je potrebné miesto <text> vkladať hlášky v angličtine. Tieto hlášky sa použijú ak sa nezistí jazyk používateľa na základe nastavení prehliadača alebo ak jeho jazyk nie je preložený. Tiež je ľahšie prekladať angličtinu do iných jazykov.

Po vyplnení všetkých textov je potrebné tieto texty vytiahnuť do **django.po** súboru. To sa robí pomocou príkazu **django-admin.py makemessages -l sk**, (odkaz) pričom miesto **sk** je možné použiť aj iné kódy jazykov alebo **django-admin.py makemessages -all**, ktorý vytvorí django.po súbory pre všetky jazyky. Tento súbor je potrebné najprv otvori a doplniť chýbajúce hlášky.

Tiež je potrebné nájsť hlášky, ktoré v komentároch nad sebou obsahuje slovo fuzzy. To naznačuje, že hlášky boli nejakým spôsobom odvodené a nemusia byť správne. Tiež je potrebné všímať si, kde je preklad použitý. Napríklad slovenčina skloňuje slová a preto sa jedno slovo v angličtine môže prekladať rôznymi slovami v slovenčine.

Keď sa vytvorí nový django.po súbor, zachovávajú sa v ňom už vyplnené texty takže je potrebné doplniť iba novo pridané.

Súbor django.po je umiestentný (pre slovenčinu) v **locale/sk/LC_MESSAGES**.

Podstatnou súčasťou django.po je jeho hlavička, ktorá sa nachádza hneď na začiatku a môže vyzerať nejako takto. Najpodtatnejšími riadkami sú posledný a predposledný, ktoré musia vyzerať presne takto, aby fungovala slovenčina. Inak sa použije iba ASCII, čo je pre slovenčinu nevyhovujúce.

```
# Translation of reCommers project into Slovak
# This file is distributed under the same license as the reCommers project.
msgid ""
msgstr ""
"Project-Id-Version: reCommers\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2017-11-30 14:27+0100\n"
"Last-Translator: Ondrej Pitonak\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
```

Posledným krokom kompilácie django.po súborov a vznik django.mo súborov v rovnakom priečinku. to sa vykoná pomocou **django-admin.py compilemessages** (odkaz). Týmto sa kompilujú všetky jazyky.

Schvaľovanie štruktúry udalostí - Vizualizácia

Samotná stránka sa nachádza v **admin_dashboard/event processing/approval/index.html** a je dostupná na adrese **event_processing/approval**. Okrem iného importuje *plugin* ***DataTables***, ktorý vytvára html tabuľky a povoľuje v nich vyhľadávať, zoradzovať a podobne.

Samotnú tabuľku vytvára výhradne JavaScript. Tabuľka vznikne v elemente s id **event_approval_datatable**. DataTables nastavuje:

- vyhľadávanie v dátach tabuľky na case-INsensitive
- preklady hlášok do angličtiny, pričom sa využíva aj Django lokalizácia pomocou i18n, ktorá tieto hlášky vie prekladať do ďalších jazykov
- vlastnosti stĺpcov
    - poradie stĺpca
    - pomenovanie stĺpca pomocou i18n (Django lokalizácia)
    - vyhľadávanie v stĺpci
    - zoraďovanie podľa stĺpca
    - dátový typ, ktorý sa v stĺpci nachádza (num, string, date-euro, html)
    - viditeľnosť stĺpca
    - priraďuje mu triedu hlavne pre účely formátovania
    - niektoré stĺpce zmenšuje na čo najmenšiu šírku

Do vizualizácie vstupujú dáta vo formáte JSON. Tie je potrebné transformovať do 2-rozmerného poľa, ktoré zachováva poradie stĺpcov tak, ako boli definované v pri inicializácii DataTables. Je vhodné všimnúť si stĺpec s *target* 7, ktorý je skrytý. Obsahuje status ktorý je prekladaný pomocou i18n a je podľa neho možné vyhľadávať. Tiež je vhodné všimnúť si posledný stĺpec, ktorý tiež obsahuje status, ale nie je podľa neho možné vyhľadávať. Preto sa zmeny v tabuľke prejavia až po obnovení stránky. Posledný stĺpec obsahuje HTML *select*, ktorým je možné zmeniť status. Zvolenú novú hodnotu je potrebné potvrdiť pomocou tlačidla v príslušnej bunke.

Zmena statusu prebieha pomocou AJAX dopytov. Odosiela sa identifikator a nový status. V databáze sa vyhľadá ID a zmení sa mu status na novú hodnotu. V prípade, že zmena prebehla úspešne, zmení sa farba tlačidla na zeleno. V opačnom prípade na červeno.

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Správa používateľov a autentifikácia

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 09.05.2017

Front End-ová časť

Správa používateľov pozostáva zo 4 obrazoviek

- **Login**
  - Formulár dostupný všetkým používateľom. Ak používateľ nie je prihlásený, je presmerovaný na tento formulár
- **Zoznam používateľov**
  - DataTables tabuľka dostupná iba administrátorovi, ktorá obsahuje zoznam všetkých používateľov. Umožňuje vymazanie používateľa a prechod na úpravu používateľa
- **Vytvorenie nového používateľa**
  - Formulár dostupný iba administrátorovi. Je ho potrebné vyplniť celý, pričom je potrebné dodržať jedinečnosť e-mailovej adresy.
- **Úprava používateľa**
  - Formulár dostupný iba administrátorovi, ktorý sa predvyplní dátami konkrétneho používateľa. V porovnaní s formuárom na tvorbu používateľa absentuje možnosť zmeniť e-mail. Ak by to bolo potrebné, je potrebné vytvoriť nového používateľa. Je tak možné zmeniť používateľské meno, heslo a administrátorské práva.

Celá aplikácia je dostupná až po prihlásení používateľa, pričom sa rozlišuje, či je používateľ administrátor alebo nie. Na základe toho sa mu upraví predovšetkým ľavé menu, no zabráni sa aj prístupu na stránky, na ktoré sa bez administrátorských práv dostať nemá.

Úprava ľavého menu sa vykonáva pomocou blokov

```
{% if is_admin %}
    // HTML kód , ktorý má byť viditeľný iba Administrátorovi
{% endif %}
```

Pričom do premennej **is_admin** sa v render funkcii priradí booblean hodnota z profilu používateľa.

Presmerovanie na prihlasovaciu obrazovku v prípade ak používateľ nie je prihlásený sa vykoná pomocou tohoto bloku kódu:

```
if not request.user.is_authenticated:
    return redirect('/login')
```

Tento blok kódu je potrebné pridať do všetkých funkcií, ktoré používajú render funkciu

Rozhodnutie, či používateľ je administrátor alebo nie je vykonáva dekorátor:

```
@user_passes_test(user_is_admin, login_url='/')
```

Ten je potrebné pridať nad funkcie, ktoré využívajú render funkcie a sú dostupné iba administrátorovi

Back-end

login - ak sa prihlásenie nepodarí, (nepodarila sa autentifikácia), presmeruje sa na /
logout - presmeruje sa na login

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

.

Zoznam nástrojov v menu

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 09.05.2018

Menu - Zoznam nástrojov

Zoznam nástrojov sa vykresľuje dynamicky podľa aktuálne platného konfiguračného súboru **/app_django/admin_dashboard/templates/includes/tools_config.json**. Tento súbor je potrebné načítať a poslať do každej render funkcie, ktorá zobrazuje menu. napríklad:

```
def get_left_menu_tools_config():
   with open(os.path.dirname(__file__) + '/templates/includes/tools_config.json',
'r', encoding='utf8') as f:
        config = f.read()
        f.close()
   return config


return render(request, 'admin_dashboard/recommenders/statistics/index.html', {
   'tools_config': json.loads(get_left_menu_tools_config())
})
```

samotné vykresľovanie zabezpečuje nasledovný kód v **app_django/admin_dashboard/templates/includes/left_navbar.html**

```
{% for tool in tools_config %}

  <a href="{{ tool.url }}">
     <i class="fa {{ tool.fa }}" aria-hidden="true"></i> {{ tool.name }}
  </a>

{% endfor %}
```

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ladenie histórie používateľa

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 7.5.2018

1. Úvod

Cieľom modulu je poskytnúť administrátorovi prehľad histórie zvoleného používateľa. Náhľad do histórie má spravidla význam pri ladení aplikácie a hľadaní chýb v nej. Administrátor by mal možnosť zistiť históriu na základe *user_id* alebo *cookie_id*. Zaujímajú nás najmä používateľove nákupy, kliky a udalosti, ktoré používa PIO.

2. Implementácia

Implementácia API pre FE sa nachádza v komponente *core* v *debugging_controller*. Samotný FE sa nachádza v *admin_dashboard/debugging/user*. Takto je vytvorený priestor aj pre ďalšie ladenie. Vystavených je 8 endpointov:

- *get_cookies_by_user*: vráti všetky *cookie_id* naviazané s požadovaným *user_id* a zoradí ich od najnovšieho po najstaršie
- *get_cookies_by_cookie*: vráti opäť všetky *cookie_id* rovnako zoradené, *cookie_id* sú naviazané s posledným použitým *user_id* pre požadované *cookie_id*
- *get_conversions_by_user*: vráti všetky konverzie, teda *target_id* pre požadované *user_id*, zoradeného od najnovšieho po najstaršie
- *get_conversions_by_cookie*: vráti opäť všetky *target_id* rovnako zoradené, rovnako *cookie_id* je naviazané s posledným *user_id* pre požadované *cookie_id*
- *get_clicks_by_user*: vráti všetky kliknutia spolu s *target_id* a *recommendation_type*, na základe ktorého kliknutie vzniklo, pre požadované *user_id* sú nájdené všetky *cookie_id*, na základe ktorého sú dopytované kliky, výsledky sú opäť zoradené od najnovšieho po najstaršie
- *get_clicks_by_cookie*: opäť vráti všetky kliknutia rovnako zoradené, pre zvolené *cookie_id* je nájdený posledný používateľ a na základe *user_id* sú nájdené *cookie_id*
- *get_events_by_user:* vráti všetky udalosti, ktoré si uložilo PIO na základe všetkých *cookie_id* pre zvoleného používateľa, udalosti sú zoradené podľa času vytvorenia
- *get_events_by_cookie:* opäť vráti všetky udalosti na základe posledného používateľa pre zvolené *cookie_id*

V princípe sú použité 4 dotazy, ostatné vždy najskôr naviažu požadované *user_id* alebo *cookie_id*. Kvôli modulu bol doplnený nový index pre tabuľku *recommendations* na *id*. Tabuľka je použitá na naviazanie názvu odporúčača.

3. Testovanie

Modul obsahuje jednotkové testy, ktoré testujú dotazy pre používateľov, ktorý majú pridelené jedno alebo viacero *cookie_id* ale aj opačný prípad, kedy viacero *cookie_id* majú pridelené jedno alebo viacero *user_id*.

4. Front-End-ová časť

FE časť pozostáva z jednej obrazovky. Na vrchu sa nachádza vstupný formulár s dvoma vstupmi. Je potrebné zadať buď používateľské ID alebo jeho cookie do príslušného vstupu a vstup potvrdiť. Následne sa z FE posiela na BE JSON s troma parametrami:

- id - zadaný identifikátor (id používateľa alebo cookie)
- id_type - nadobúda hodnoty cookie a user
- operation - nadobúda jednu z hodnôt cookies, conversions, clicks alebo events

Tento JSON pomocou id_type a peration presne identifikuje funkciu na BE, ktorá sa má vykonať a id predstavuje vstup do tejto funkcie. BE v Djangu jednoducho mapuje tieto dopyty na skutočné Python funkcie opísané v časti implementácia a ošetruje prípadné chyby.

Z FE sa odpália 4 POST dopyty a asynchrónne sa čaká na ich dokončenie. Pri ich úspešnom dokončení sa dáta zobrazia v DataTables. Ak dopyt nebol úspešný príslušná tabuľka sa nezobrazí vôbec.

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

CRUD používateľov

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Vyhotovil: Martin Schnurer
Dátum poslednej zmeny: 09.05.2018

Back-end

Model User:

username = meno, not required
email = hlavný identifikátor používateľa, is required
password = is required
is_admin = Boolean, not required, default False

Dostupne metody sú GET, POST, PUT, DELETE

GET - vyberie všetkých možných používateľov z databázy.

POST - vytvorí nového používateľa, ak už je daný email v databáze, na frontend sa postne IntegrityError response, v prípade inej chyby sa pošle všeobecná chyba.

PUT - upraví už existujúceho používateľa, ak id používateľa neexistuje, tak sa pošle chyba na frontend. Upraviteľné položky sú username, password a is_admin flag.
- na backend by sa mali posielat v body iba kľúče, a hodnoty, ktoré chceme zmeniť.
Takže ak chceme zmeniť aj username aj heslo, tak nesmieme poslať žiadny iný kľúč.

DELETE - vymaže používateľa s daným id, ak neexistuje dané id, tak sa pošle na frontend chyba. ak sa používateľ úspešne vymazal, pošle sa úspešny response na frontend.

V models je vytvorený nový model pre Usera, kde je overridnutý pôvodný model. Terajší model používa email ako svoj hlavný identifikátor, kvoli tomu, že predtým sa ako hlavný identifikátor bral username.

Front-end

FE pozostava z 3 obrazoviek

- **index.html** - využíva Data tables na zobrazenie zoznamu používateľov. Obsahuje tlačidlo upraviť (prechod na update.html) a vymazať pri každom používateľovi. Obsahuje tlačidlo na pridnaie nového používateľa (prechod na add.html)
- **add.html** - obsahuje formulár, ktorý je potrebné celý vyplniť
- **update.html** - Ubsahuje formulár na vykonanie zmien na používateľovi. Ukladajú sa iba reálne vykonané zmeny. Nie je preto potrebné vypĺňať všetky políčka

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Definícia odporúčačov a ich verzií

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 09.05.2018

Front-end

FE pozostava z 3 obrazoviek

- **index.html** - využíva Data tables na zobrazenie zoznamu verzií odporúčačov. Obsahuje tlačidlo upraviť (prechod na update.html) a vymazať pri každom používateľovi. Obsahuje tlačidlo na pridanie novej verize (prechod na add.html)
- **add.html** - obsahuje formulár, ktorý je potrebné celý vyplniť okrem popisu, ktorý je volitelný
- **update.html** - obsahuje rovnaký formulár ako add.html, no je predvyplnený. Platia pre neho rovnaké pravidlá ako pre add.html

views.py obsahuje 3 render funkcie pre každú obrazovku jeden, pričom sa na FE doťahujú dáta:

- recommenders_version_conf
- recommenders_version_conf_add
- recommenders_version_conf_update

Tieto funkcie štandardne získavajú typy odporúčačov

```
rec_types = schema_controller.get()
```

alebo dáta (ktore upravujú aby)

```
data = versions_controller.get()
data = [(dict(row.items())) for row in data]

for d in data:
  d['valid_from'] = d['valid_from'].strftime('%Y-%m-%d')
  d['valid_to']   = d['valid_to'].strftime('%Y-%m-%d')
```

views.py obsahuje aj API metódu, ktorá reaguje na metódy typu PUT, POST, DELETE a slúči na vytváranie záznamov, ich zmenu a mazanie

Back-end

Vytváranie verzií sa sprostredkúva na BE pomocou kontrolera versions_controller.
Možné operácie sú: get, create, update, delete.
Všetky operácie okrem get, berú ako argument dictionary danej inštancie.

Update a Delete nastáva pomocou atribútu *id*.

Get - vracia všetky inštancie z databázy.

Pri update je možné upraviť hodnoty:

- recommender_type_id,
- name

- description
- valid_from
- valid_to

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Zaznamenávanie a monitorovanie chýb

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 11.12.2017

Táto časť zodpovedná za ukladanie udalostí, ktoré môžu nastať počas behu aplikácie. Uchovanie týchto udalostí má veľký význam pri riešení chýb, testovaní a poskytnutí všeobecného prehľadu o chode aplikácie.

5.5.1 Analýza

Udalosti, ktoré môžu počas behu aplikácie nastať možno podľa ich charakteru rozdeliť na dva typy:

- *informačné*
    - prijatie udalostí od zákazníka
    - úspešné spracovania týchto udalostí
    - úspešné uloženia spracovaných udalostí
    - vygenerovanie nových odporúčaní
    - časové informácie o behu aplikácie
- *chybové*
    - chybný formát prijatých udalostí
    - neúspešne uloženie spracovaných udalostí

5.5.2 Návrh

V aplikácii je použité logovanie do viacerých systémov:

- Logstash
- Sentry
- Grafana

5.5.3 Implementácia

Logovanie do Logstash a Sentry prebieha prostredníctvom python modulu *logging*. V *app_django/__init__.py* sa nastavia *handlery* pre tieto *loggery*:

- *events logger* - logovanie prijatých udalostí
- *recommend logger* - logovanie odporúčaní
- *default logger* - logovanie do sentry

Zavolaním loggera sa vykoná logovanie do oboch systémov.

Na logovanie do Grafany je použitý *StatsClient* z knižnice *statsd*. Ten je inicializovaný v *settings.py*.

V konfiguračných súboroch je potrebné nastaviť:

- `LOGSTASH_HOST`
- `LOGSTASH_PORT`
- `SENTRY_DSN`
- `statsd:`
    - `STATSD_HOST`
    - `STATSD_PORT`
    - `STATSD_PREFIX`

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

reCommers

Asynchrónne spracovanie

Tím: 03 - reCommers
Vedúci tímu: Ing. Ivan Srba, PhD.
Dátum poslednej zmeny: 09.05.2018

1. Analýza

Asynchrónne spracovanie je použité v module odporúčania. Cieľom je zrýchlenie celého procesu odporúčania. Tento dokument obsahuje súhrnné informácie o asynchrónnom vykonávaní v aplikácii.

2. Návrh

Asynchrónne spracovanie je prevzaté z existujúcej infraštruktúry, kde bolo testované v produkčnej verzii,

3. Implementácia

Na realizáciu asynchrónneho spracovania je použitá knižnica Celery. Asynchrónne funkcie volané pri ukladaní udalostí do databázy a odporúčaní sú zaradené do fronty, odkiaľ sú vyberané a vykonané jednotlivými vláknami. Fronta je uložená v redise, url adresa redisového servera je uložené v konfigurácii.
Asynchrónne úlohy sú definované v app_django/core/tasks.py. Podľa konvencie má každý *queue* pre *task* v názve '*_*recommers'*. Konfigurácie knižnice celery sa nachádza v app_django/celeryconfig.py. Pred lokálnym spustením aplikácie je nutné spustiť celery worker z adresára app_django príkazom: *celery worker -A core.tasks -l info.*
Cron job, ktorý sa spúšťa každú hodinu, je spúšťaný cez *celery beat*. Ten je tiež potrebné spustiť manuálne cez *celery -A core.tasks beat* z rovnakého adresára.

4. Testovanie

Nie je vytvorený samostatný test na asynchrónne spracovanie, konkrétne asynchrónne úlohy sú testované príslušnými testami.
Knižnica celery od verzie 4.0 nie je oficiálne podporovaná operačným systémom Windows, je však možné ju použiť pri vývoji a testovaní. Pre lokálne spustenie workera vo Windows prostredí je nutné pridať prepínač *--pool=solo.*