

# Take *the* Survey

Maciej Walczakowski  
Jarosław Kulik

## Spis treści

|   |   |
|---|---|
| Temat projektu.....                         | 2 |
| Zespół.....                                 | 2 |
| Opis funkcjonalności.....                   | 2 |
| Wymagania Minimum Viable product (MVP)..... | 3 |
| Diagram przypadków użycia.....              | 4 |
| Wykorzystywane technologie.....             | 5 |
| Kontrola wersji.....                        | 5 |
| Projekt bazy danych.....                    | 7 |
| Kamienie milowe.....                        | 7 |
| Struktura aplikacji.....                    | 8 |

## Temat projektu

Tematem projektu jest stworzenie serwisu internetowego z konfigurowalnymi ankietami dotyczącymi świadczonych usług, produktów, odbywających się wydarzeń itp. Cechą szczególną systemu ma być możliwość tworzenia własnych ankiet z wykorzystaniem bazy gotowych pytań i odpowiedzi.

## Zespół

Maciej Walczakowski

Jarosław Kulik

## Opis funkcjonalności

W systemie przewidziane są następujące funkcjonalności:

- Tworzenie konta użytkownika;
- Logowanie do systemu (podanie loginu/emaila i hasła);
- Dla zalogowanych użytkowników:
  - konfigurowanie nowej ankiety ;
  - wykorzystywanie publicznych pytań z wcześniej dodanych ankiet (przez siebie i innych użytkowników) z możliwością modyfikacji pytania/odpowiedzi;
  - publikacja ankiety (publicznych i dostępnych wyłącznie po zalogowaniu);
  - dodawanie nowych pytań do bazy pytań (prywatnych – tylko dla danej ankiety i publicznych - dostępnych dla konfigurowanych w przyszłości ankiet);
  - tagowanie pytań (kategoria, produkt itp.);
  - wypełnianie wszystkich aktywnych ankiet;
  - usuwanie własnych ankiet (stworzonych i wypełnionych);
  - poprawa danych rejestracyjnych konta własnego (w tym loginu i hasła);
  - usuwanie własnego konta użytkownika;
  - przeglądanie wszystkich aktywnych ankiet;
  - możliwość przeszukiwania/filtrowania ankiet wszystkich(parametryzowanie);
- Przeglądanie listy aktywnych ankiet publicznych (nie wymagających logowania);

- Wypełnianie aktywnych ankiet publicznych (nie wymagających logowania);
- Możliwość przeszukiwania/filtrowania ankiet publicznych (parametryzowanie);
- Panel/ widok administratora umożliwiający:
  - usuwanie kont użytkowników;
  - dodawanie, usuwanie i modyfikacja ankiet dodanych przez użytkowników;
- Automatyczna kontrola terminu aktywności ankiety (dezaktywacja ankiety po ustalonym przez użytkownika terminie);
- W systemie przewidziane są 2 role użytkownik i administrator;

## **Wymagania Minimum Viable product (MVP)**

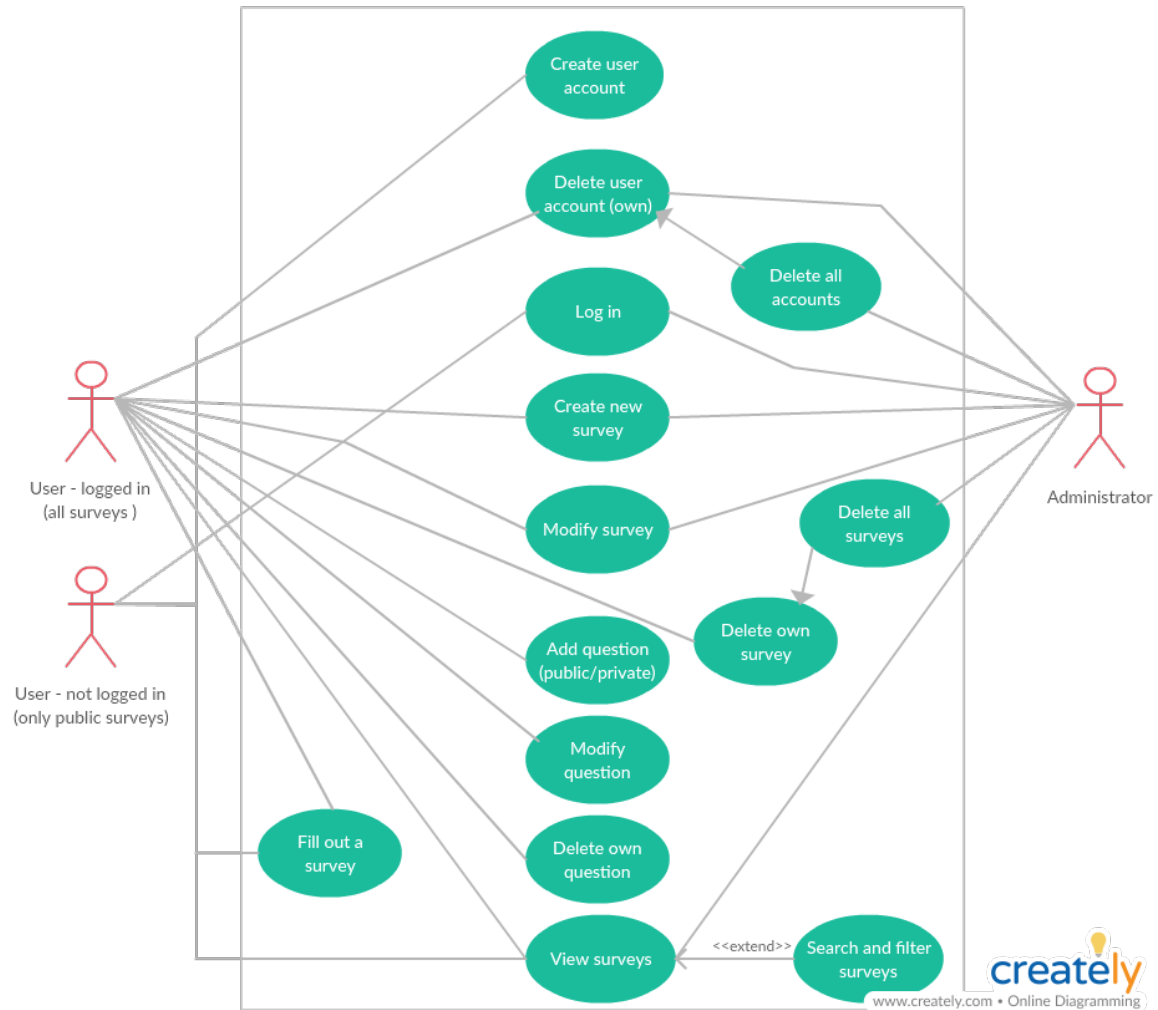
Wymagania odnośnie MVP spełni serwis który umożliwi:

- Rejestracje i logowanie użytkownika;
- Przeglądanie, wypełnianie oraz dodawania własnych ankiet (publicznie)
- Przeglądanie, dodawanie własnych pytań i odpowiedzi (publicznie)

Jest to sensowne minimum funkcjonalności, aby serwis miał wystarczającą funkcjonalność bazową. Na tym etapie można zacząć poszukiwać inwestorów i innych twórców zainteresowanych rozwijaniem aplikacji. Dodatkowe funkcjonalności będą zaimplementowane na późniejszych etapach aplikacji, zgodnie z oczekiwaniami użytkowników i potencjalnych inwestorów.

## Diagram przypadków użycia

TtS



Rys 1. Diagram przypadków użycia

## Wykorzystywane technologie

W projekcie planowane jest użycie poniższych technologii:  
Python (flask), JavaScript, SQL, HTML, CSS (bootstrap).

Celem projektu jest stworzenie responsywnej aplikacji web, korzystającej z wzorca projektowego MVC, MVVC lub pokrewno. Backend strony oparty będzie o mikro-framework Flask. Frontend wykorzystywać będzie HTML, CSS (bootstrap) oraz JavaScript.

Użycie w/w technologii pozwoli na szybkie i w miarę bezproblemowe powstanie pierwszych, w pełni funkcjonalnych wersji serwisu. Jednocześnie Flask, ze względu na swoją stosunkowo niewielką funkcjonalność, nie narzuca programiście sztywnych rozwiązań (tak jak np. Django czy TurboGears). Mimo większego narzutu pracy na stworzenie podstawowych funkcjonalności strony, to jego elastyczność jest w późniejszych etapach dużą zaletą i pozwala na lepsze dostosowanie projektu do wymagań.

## Kontrola wersji

Zdalne repozytorium GIT dostępne pod adresem:  
<http://www.github.com/walczakx/TakeTheSurvey>

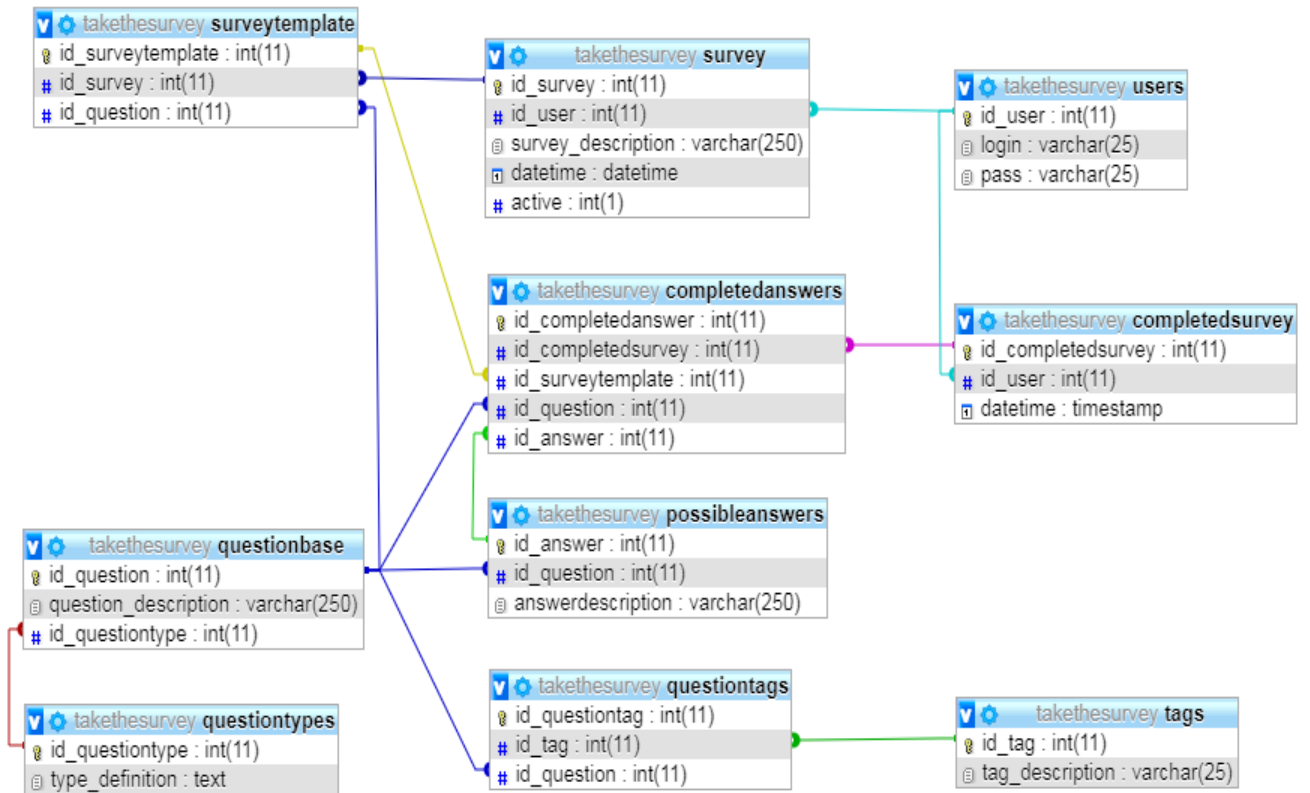
Na rysunku poniżej znajduje się struktura commitów wykonanych podczas trwania projektu.

```

* 57a08c3 - (HEAD -> master, origin/master, origin/HEAD) added about dialog, removing and disabling surveys and more (26 hours ago) <M W>
* 4f75f96 - logout after account delete, corrections layout Settings > Account (33 hours ago) <Jarek>
* fee18b4 - delete account works (34 hours ago) <Jarek>
* c29baa0 - templates user account view, edit, confirm delete (2 days ago) <Jarek>
* e4c9d43 - db improvements: question/answers active/not active (SQL), fixes in specific_survey to fill (2 days ago) <Jarek>
* 17b0170 - pass VAR40 (2 days ago) <Jarek>
* d9a9966 - small corrections (3 days ago) <Jarek>
* 6305c3c - db improvements needed to add completedsurvey (3 days ago) <Jarek>
* ala87fd - sql add survey and surveytemplate (3 days ago) <Jarek>
* 0df7e9b - creating new survey complete w/o sql (3 days ago) <M W>
* ca5d076 - new sql queries adding completed survey and answers (4 days ago) <Jarek>
* 19099a1 - display specific survey to fill v.2 (4 days ago) <Jarek>
* 78daadf - add survey (4 days ago) <M W>
* fa1766c - displaying specific survey with questions v.1 (4 days ago) <Jarek>
* 2b8c04e - new features, need to cleanup & polish" (6 days ago) <M W>
* 71abb79 - many fixes, a lot of thinks now works properly (7 days ago) <M W>
* d77ac3e - changes in database, improvements for completed_survey (9 days ago) <Jarek>
* 9e8bc89 - displaying user data, additional template for completed_surveys (9 days ago) <Jarek>
* d2f83fb - return user data from db (9 days ago) <Jarek>
* 42fa09a - test commit + sql description correction (9 days ago) <Jarek>
* bf95e0d - small fixes (9 days ago) <M W>
* 42afce5 - refactor & whitespace cleanup (9 days ago) <Maciej Walczakowski>
* d717006 - Merge branch 'master' of https://github.com/walczakx/TakeTheSurvey (10 days ago) <Jarek>
| \
| * fa56796 - error/warning page (10 days ago) <M W>
| * b0437ea - new sql queries (10 days ago) <Jarek>
| /
* c906906 - additional sql queries (10 days ago) <Jarek>
* 079dbe1 - small refactor, new shiny things (10 days ago) <M W>
* 7a0cd60 - merge (10 days ago) <M W>
* 4fb8525 - further improvements in survey listing, some works with specific survey dispaly (10 days ago) <Jarek>
* 55870e2 - some database improvements - many users can create surveys in the same time (10 days ago) <Jarek>
* ed74be8 - display surveys list improvements (10 days ago) <Jarek>
* 382f8e4 - improvements in showing surveys template - need to keep this footer downgit status! (10 days ago) <Jarek>
* 74ec556 - rendering template show surveys with data frm DB - works (10 days ago) <Jarek>
* 091fd1c - some corrections in adding user sql (11 days ago) <Jarek>
* fd4bef6 - some corrections in adding user sql (11 days ago) <Jarek>
* f041501 - new stuff (11 days ago) <M W>
* d5e8ba1 - merge & refactor (11 days ago) <M W>
* 56d3fe5 - some sql queries (11 days ago) <JK>
* 3758d59 - db corrections (delete relations and role in users table) (11 days ago) <JK>
* f1b3057 - refactor & new things (12 days ago) <M W>
* 2eaab214 - user login/out functions (2 weeks ago) <M W>
* 0ab3de5 - user session scratch (2 weeks ago) <M W>
* 745e2bf - survey templates (2 weeks ago) <M W>
* b184539 - template fix (2 weeks ago) <M W>
* 80063e9 - some sql (2 weeks ago) <M W>
* 6d1569c - Delete forms.py (2 weeks ago) <walczakx>
* fb71730 - database scratch (2 weeks ago) <M W>
* f9c7b24 - login & register forms handling scratch (2 weeks ago) <M W>
* 110d567 - Added column emial in table users (3 weeks ago) <Jarek>
* a53df93 - docs update (8 weeks ago) <M W>
* 6ab0835 - Added /docs and project docs files (8 weeks ago) <M W>
* 9c59134 - added sign up modal (9 weeks ago) <M W>
* 21baff4 - front-end changes (9 weeks ago) <M W>
* d263d4a - Templates schema (9 weeks ago) <M W>
* 4301ac3 - formatka rejestracji (2 months ago) <Jarek>
* b7ca4cd - routing to register.html (2 months ago) <Jarek>
* c50f668 - app routing draft (3 months ago) <M W>
* a7a4d5b - Baza danych aktualna wersja v.3 Mozna robic funkcjonalnosci (3 months ago) <Jarek>
* d31e272 - db v1.1 (3 months ago) <Jarek>
* 734a841 - TakeTheSurvey db v.1 (3 months ago) <Jarek>
* 6034210 - test (3 months ago) <Jarek>
* 371363c - test (3 months ago) <Jarek>
* c745ca2 - some cleanup (3 months ago) <M W>
* 168feb9 - Flask hello world (3 months ago) <M W>

```

## Projekt bazy danych



Rys. 2. Projekt bazy danych

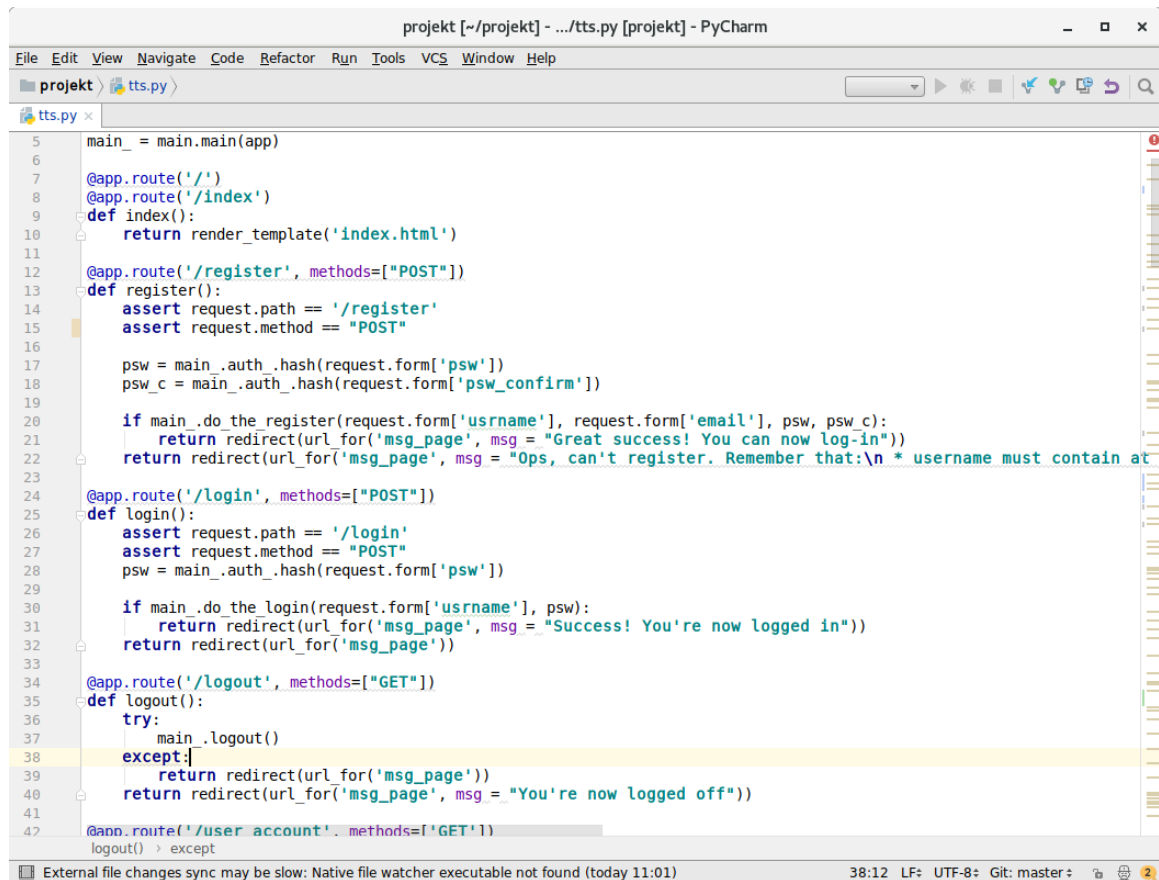
## Kamienie milowe

| Kamień milowy   | Termin |
|---|--------|
| Ogólne założenia, określenie wymagań i MVP                                    | 21.04  |
| Projekt bazy danych, konfiguracja środowiska developerskiego, pierwszy commit | 5.05   |
| Stworzenie szkieletu serwisu  | 19.05  |
| Dodawanie funkcjonalności zgodnie z wewnętrznym backlogiem                    | 30.06  |
| Ostatnie poprawki i dostarczenie serwisu do testów                            | 14.07  |



## Struktura aplikacji

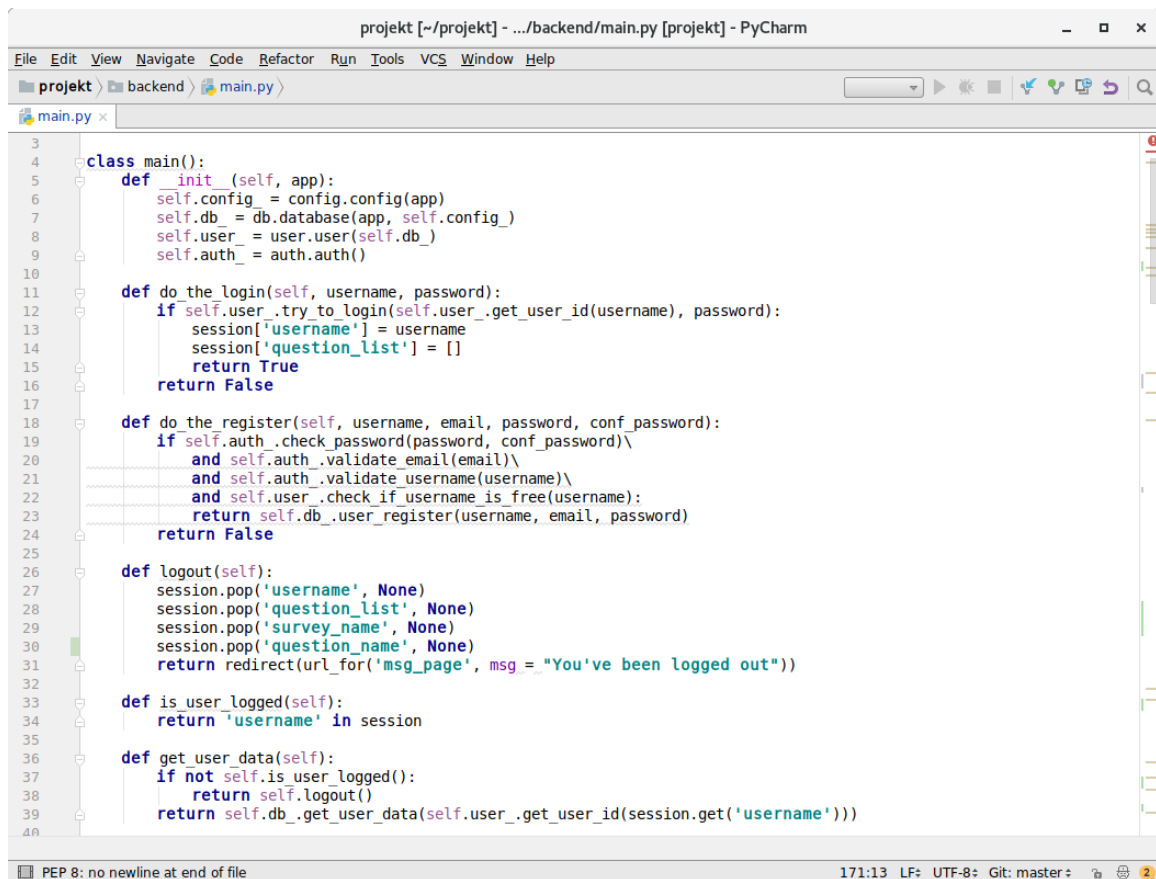
Architektura, jak wspomniano powyżej, wykorzystuje wzorzec MVC. Klient korzystając z widoku, dokonuje odpowiedniego zapytania do kontrolera, który zwraca, zaktualizowany o dane z modelu, nowy widok.



```
5 main_ = main.main(app)
6
7 @app.route('/')
8 @app.route('/index')
9 def index():
10     return render_template('index.html')
11
12 @app.route('/register', methods=["POST"])
13 def register():
14     assert request.path == '/register'
15     assert request.method == "POST"
16
17     psw = main_.auth_.hash(request.form['psw'])
18     psw_c = main_.auth_.hash(request.form['psw_confirm'])
19
20     if main_.do_the_register(request.form['username'], request.form['email'], psw, psw_c):
21         return redirect(url_for('msg_page', msg = "Great success! You can now log-in"))
22     return redirect(url_for('msg_page', msg = "Ops, can't register. Remember that:\n * username must contain at
23
24 @app.route('/login', methods=["POST"])
25 def login():
26     assert request.path == '/login'
27     assert request.method == "POST"
28     psw = main_.auth_.hash(request.form['psw'])
29
30     if main_.do_the_login(request.form['username'], psw):
31         return redirect(url_for('msg_page', msg = "Success! You're now logged in"))
32     return redirect(url_for('msg_page'))
33
34 @app.route('/logout', methods=["GET"])
35 def logout():
36     try:
37         main_.logout()
38     except:
39         return redirect(url_for('msg_page'))
40     return redirect(url_for('msg_page', msg = "You're now logged off"))
41
42 @app.route('/user_account', methods=["GET"])
43 def user_account():
44     pass
```

Rys 3. Główny plik aplikacji 'tts.py'

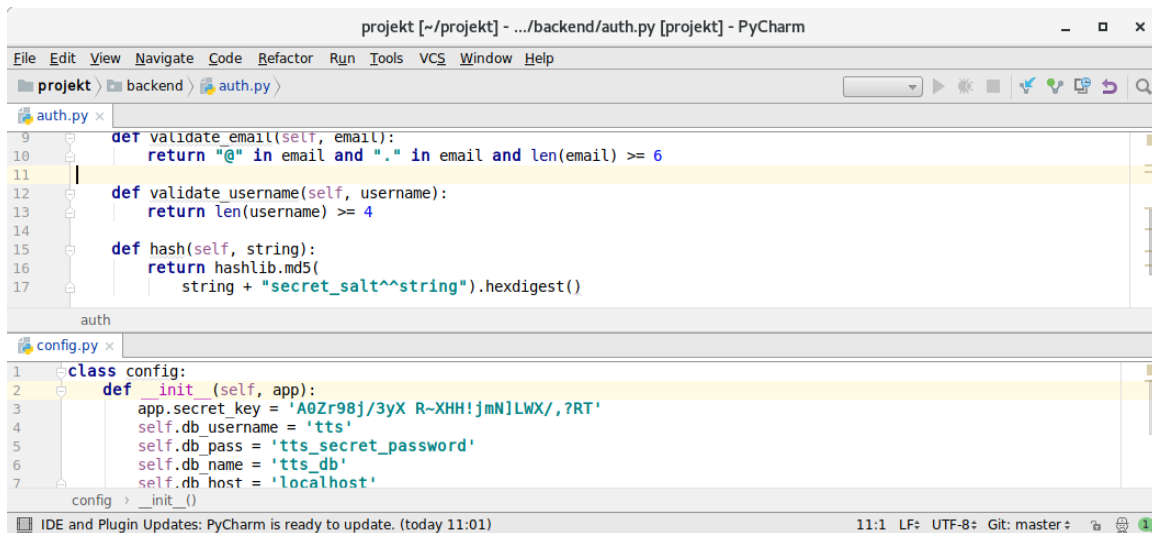
Poniżej przykładowe funkcje odpowiadające za routing całej aplikacji, renderowanie szablonów oraz delegowanie zadań do klasy „main”. Wspomniana klasa „main” wraz z klasami które posiada w swoich strukturach, odpowiada za całą logikę aplikacji. Dla przykładu na poniższym rysunku przedstawiony jest konstruktor tej klasy i przykładowe funkcje odpowiadające za obsługę użytkownika (logowanie, rejestracja, wylogowanie, pobieranie danych użytkownika)



```
3
4 class main():
5     def __init__(self, app):
6         self.config = config.config(app)
7         self.db_ = db.database(app, self.config_)
8         self.user_ = user.user(self.db_)
9         self.auth_ = auth.auth()
10
11     def do_the_login(self, username, password):
12         if self.user_.try_to_login(self.user_.get_user_id(username), password):
13             session['username'] = username
14             session['question_list'] = []
15             return True
16         return False
17
18     def do_the_register(self, username, email, password, conf_password):
19         if self.auth_.check_password(password, conf_password)\
20             and self.auth_.validate_email(email)\
21             and self.auth_.validate_username(username)\
22             and self.user_.check_if_username_is_free(username):
23             return self.db_.user_register(username, email, password)
24         return False
25
26     def logout(self):
27         session.pop('username', None)
28         session.pop('question_list', None)
29         session.pop('survey_name', None)
30         session.pop('question_name', None)
31         return redirect(url_for('msg_page', msg = "You've been logged out"))
32
33     def is_user_logged(self):
34         return 'username' in session
35
36     def get_user_data(self):
37         if not self.is_user_logged():
38             return self.logout()
39         return self.db_.get_user_data(self.user_.get_user_id(session.get('username')))
```

Rys. 4. Klasa main

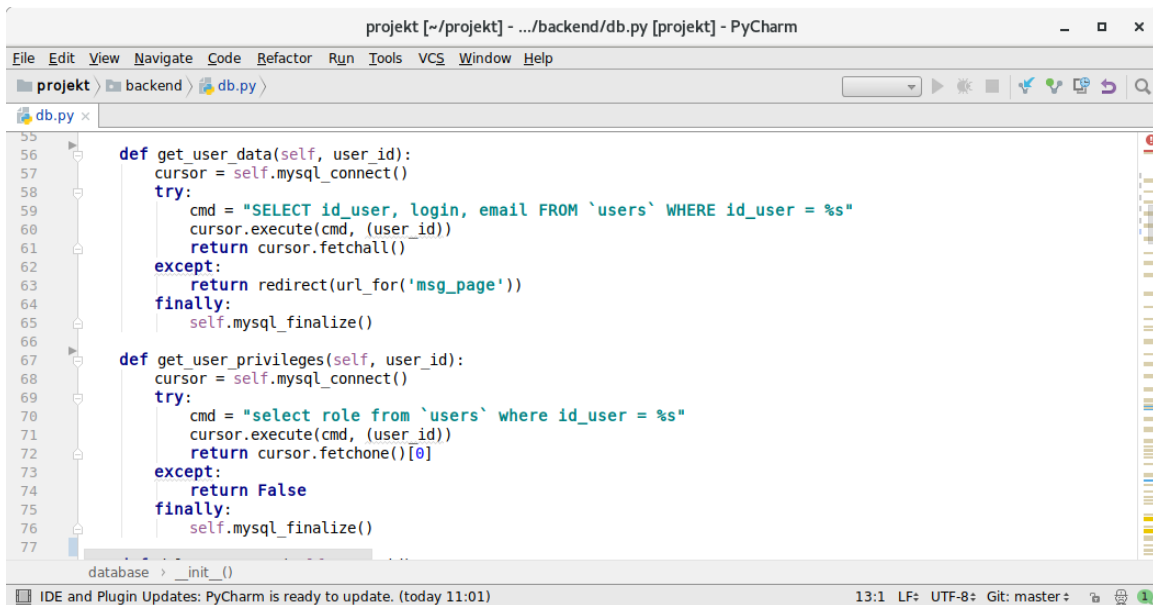
Jedną z klas, która jest inicjowana w klasie 'main', jest klasa 'auth', której zadaniem jest uwierzytelnienie użytkownika. Dla przykładu, na poniższym rysunku przedstawiona została funkcja haszująca hasła. Do haseł dodawany jest dowolny ciąg znaków (tzw. Salt), co dodatkowo zapobiega przed atakami z wykorzystaniem „tęczowych tablic” (rainbow tables). Dodatkowo, ponieważ przy każdym logowaniu, tworzona jest nowa szyfrowana sesja użytkownika, wymagane jest aby aplikacja posiadała swój „secret key”. W naszym przypadku znajduje się on w konfiguracji aplikacji (klasa config). Znacząco utrudnia to popularne ataki (takie jak np. XSS – Cross-Site Scripting). Zapewnia to sensowne minimum bezpieczeństwa dla naszej aplikacji.



```
projekt [~/projekt] - .../backend/auth.py [projekt] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
projekt backend auth.py
auth.py
9 def validate_email(self, email):
10     return "@" in email and "." in email and len(email) >= 6
11
12 def validate_username(self, username):
13     return len(username) >= 4
14
15 def hash(self, string):
16     return hashlib.md5(
17         string + "secret_salt^^string").hexdigest()
18
19 auth
20
21 config.py
22 class config:
23     def __init__(self, app):
24         app.secret_key = 'A0Zr98j/3yX R-XHH!jmN]LWX/,?RT'
25         self.db_username = 'tts'
26         self.db_pass = 'tts_secret_password'
27         self.db_name = 'tts_db'
28         self.db_host = 'localhost'
29
30 config > __init__()
IDE and Plugin Updates: PyCharm is ready to update. (today 11:01) 11:1 LF: UTF-8: Git: master
```

Rys. 5 Metoda haszująca oraz config

Za komunikację z bazą danych, oraz wykonywanie zapytań SQL odpowiada klasa 'database'. Jej całkowitą własnością jest połączenie z bazą danych, inne komponenty tylko za jej pomocą mogą wykonywać procedury i zapytania SQL. Na rysunku poniżej przedstawiono dwie przykładowe funkcje.



```
projekt [~/projekt] - .../backend/db.py [projekt] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
projekt backend db.py
db.py
55
56 def get_user_data(self, user_id):
57     cursor = self.mysql_connect()
58     try:
59         cmd = "SELECT id_user, login, email FROM `users` WHERE id_user = %s"
60         cursor.execute(cmd, (user_id))
61         return cursor.fetchall()
62     except:
63         return redirect(url_for('msg_page'))
64     finally:
65         self.mysql_finalize()
66
67 def get_user_privileges(self, user_id):
68     cursor = self.mysql_connect()
69     try:
70         cmd = "select role from `users` where id_user = %s"
71         cursor.execute(cmd, (user_id))
72         return cursor.fetchone()[0]
73     except:
74         return False
75     finally:
76         self.mysql_finalize()
77
78 database > __init__()
IDE and Plugin Updates: PyCharm is ready to update. (today 11:01) 13:1 LF: UTF-8: Git: master
```

Rys. 6 Klasa obsługująca bazę danych – przykładowe zastosowanie

Jak wspomniano na początku, wynikiem żądania użytkownika jest wygenerowany kod HTML/CSS gotowej strony, zaktualizowany o dane dostarczone przez model. Na rysunku przykładowy kod HTML, gotowy do wyświetlenia danych. Dodatkowo, w szablonie zostały wykorzystane funkcje pomocnicze, zmieniające widok odpowiednio jeśli użytkownik posiada uprawnienia administratora lub jest posiadaczem jakiś ankiet (funkcje is\_surevey\_owner oraz has\_admin\_privileges)

