Ankieter+

**Aleksander Kiryk, Emil Markiewicz,
Rafał Piotrowski, Wojciech Wiśniewski**

19 sty 2022

Spis treści:

ROZDZIAŁ 1

Wstęp

1.1 Informacja licencyjna

Oprogramowanie Ankieter+ jest udostępnione na 2-klauzulowej licencji BSD. W celu zdobycia szczegółowych informacji o warunkach użytkowania nałożonych przez tę licencję należy zajrzeć do pliku `LICENSE` udostępnianego razem z kodem źródłowym tego oprogramowania.

1.2 O Ankieterze+

Ankieter+ jest darmowym oprogramowaniem pozwalającym na łatwe tworzenie raportów z ankiet pochodzących z systemu Ankieter będącego częścią USOSa.

Ankieter+ pozwalaja na prezentację danych zebranych w Ankieterze, tj. tworzenie wykresów i tabel oraz wzbogacania ich o dowolne komentarze z użyciem wbudowanego edytora tekstu.

Tworzenie wspomnianych zestawień tabelarycznych oraz wykresów odbywa się na zasadzie zbliżonej do mechaniki tabel przestawnych. Układanie ich odbywa się jednak ze wsparciem prostego w obsłudze interfejsu.

Wraz z Ankieterem+ dostarczona jest mechanika uprawnień, widoczności i udostępniania zarówno projektów raportów jak i zebranych dla nich nich danych. W systemie zawarty jest też panel administratora pozwalający na tworzenie i usuwanie kont oraz przydzielanie użytkowników do określonych grup.

1.3 Jak czytać tę dokumentację

Jeśli czytasz tę sekcję tylko w celu dowiedzenia się, jak przeprowadzić ankietę, polecamy zajrzeć prosto do ostatniej sekcji *Dla użytkownika*.

Dokumentacja Ankietera+ została podzielona na 4. główne części, z których każda trzymana jest w osobnym pliku Markdown. Części te zawierają następujące informacje:

1. *Wstęp*: zawiera podstawowe informacje o systemie oraz strukturze dokumentacji.
2. *Dla użytkownika*: zawiera informacje o obsłudze strukturze interfejsu, instrukcję obsługi edytora ankiet, edytora raportów, opis możliwości udostępniania oraz przewodnik pozwalający na utworzenie ankiety, przeprowadzenie jej oraz analizę danych w niej zebranych.
3. *Dla administratora*: zawiera informacje o instalacji systemu, jego konfiguracji, sposobie, w jaki w systemie działają uprawnienia oraz opis mechanizmu zarządzania użytkownikami i grupami.
4. *Dla programisty*: jest to techniczna część dokumentacji, zawiera opis API, dokumentację silnika, kodu interfejsu oraz struktury bazy danych.

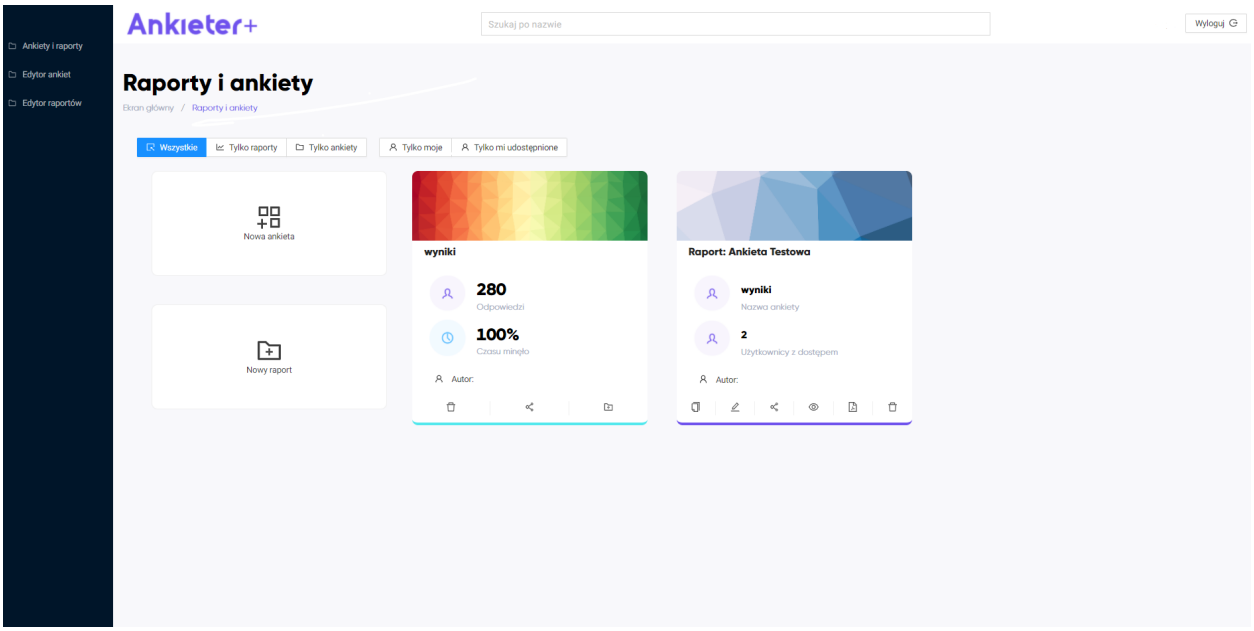
ROZDZIAŁ 2

Dla użytkownika

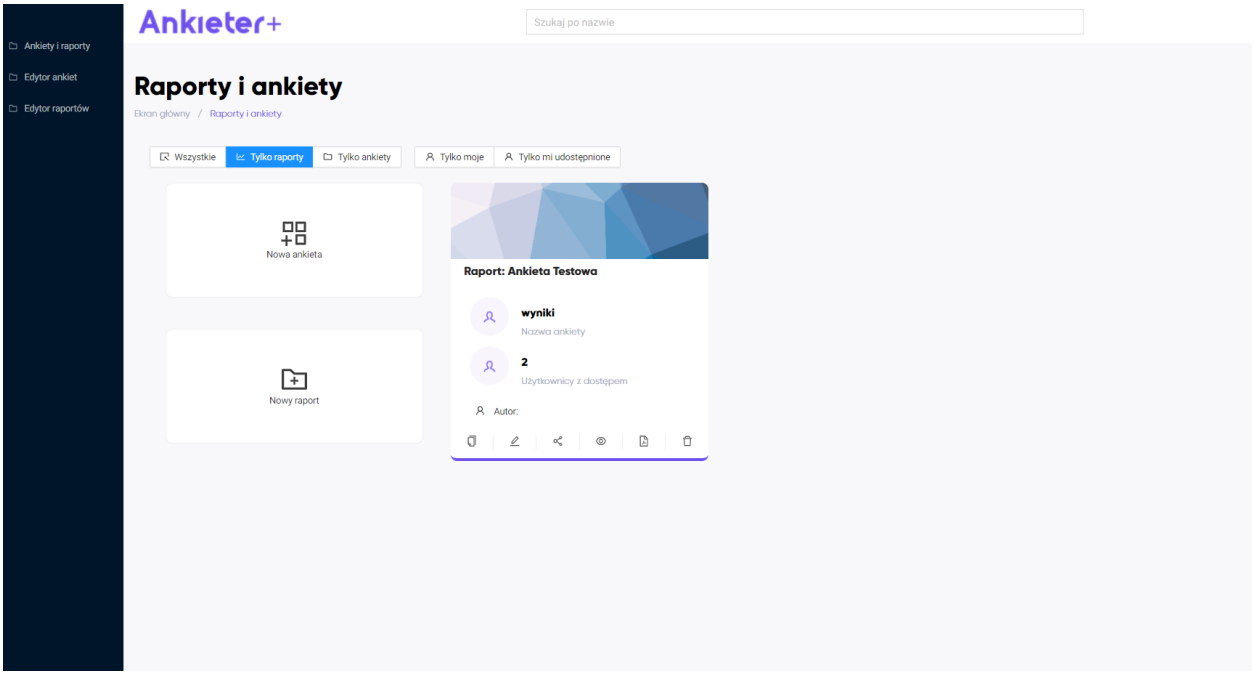
2.1 Opis interfejsu

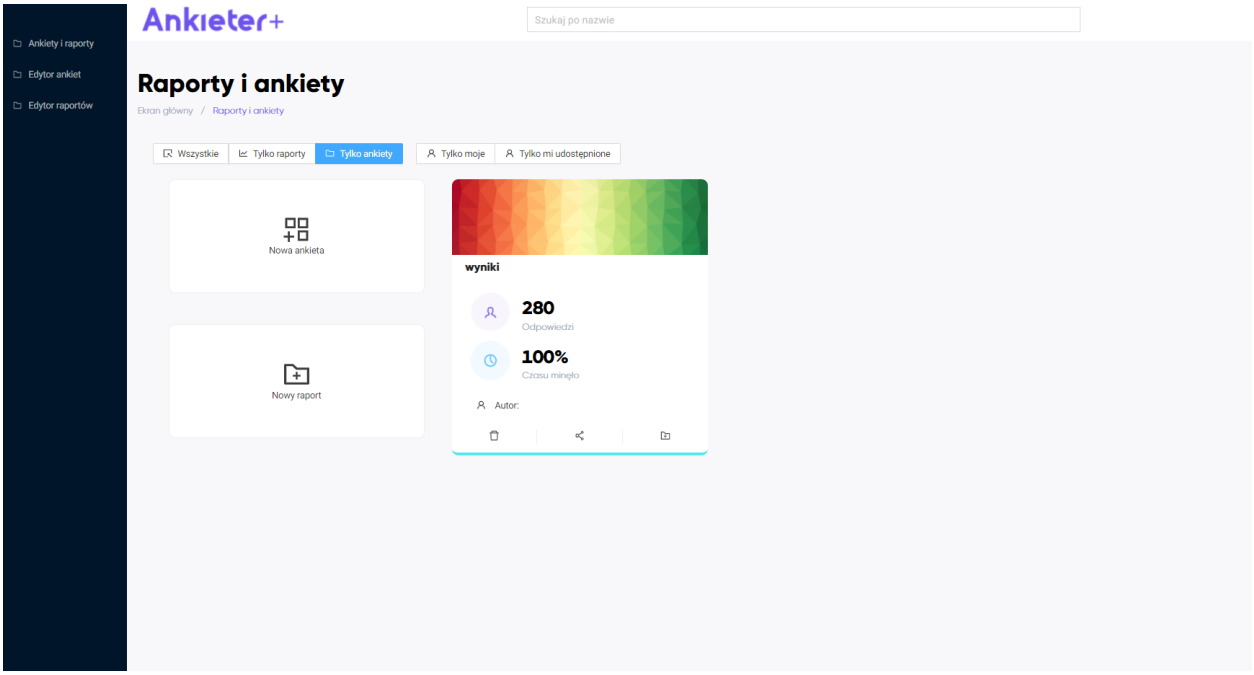
2.2 Panel główny

Po zalogowaniu, w panelu głównym wyświetlają się ankiety oraz raporty, które są utworzone przez użytkownika lub, do których użytkownik ma uprawnienia.

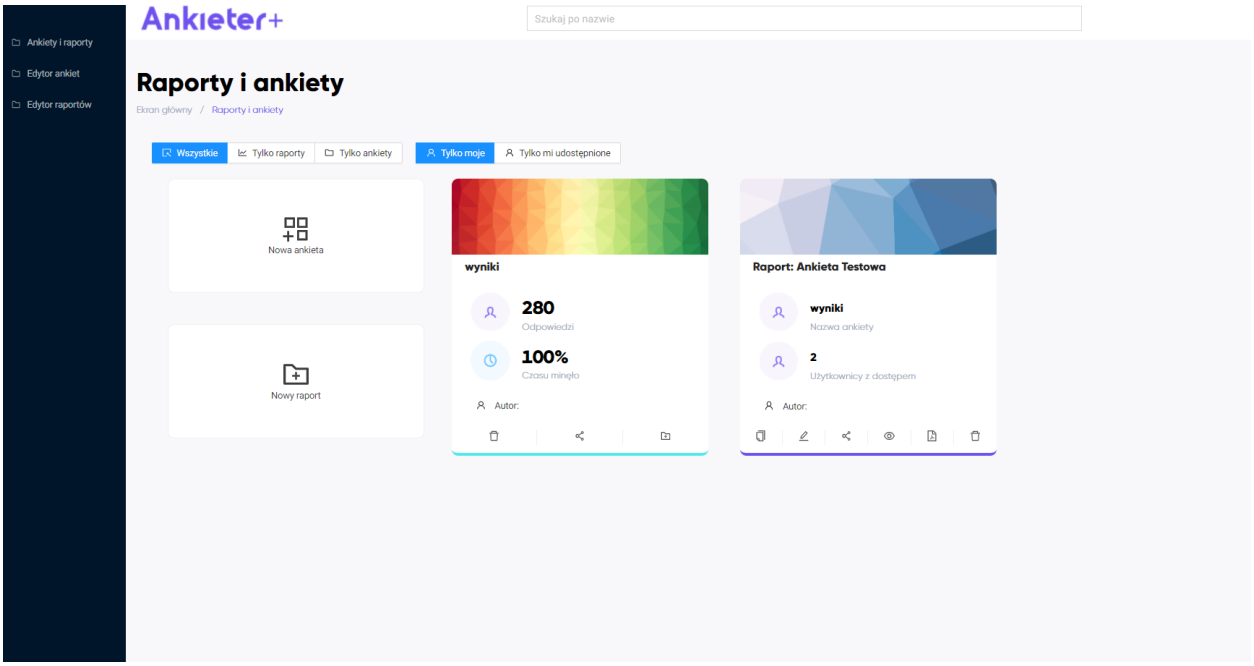


Można zawęzić widok do ankiet lub raportów.

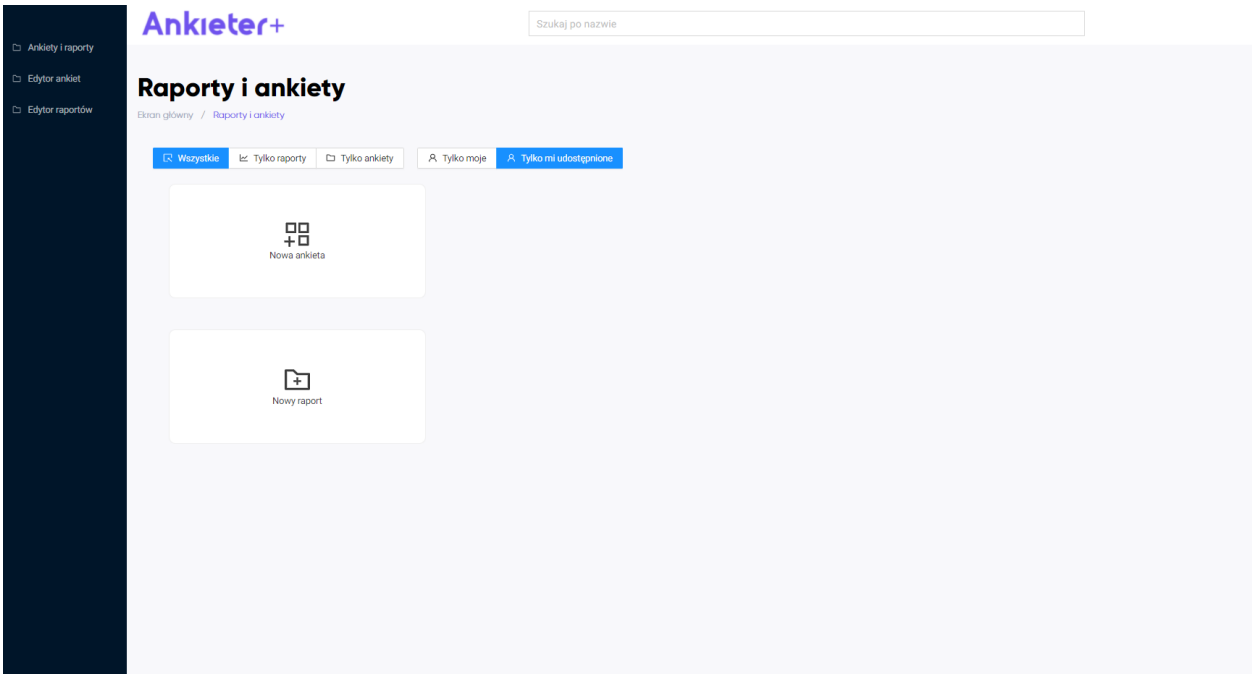




Można wyświetlić te ankiety/raporty, które są utworzone przez użytkownika.



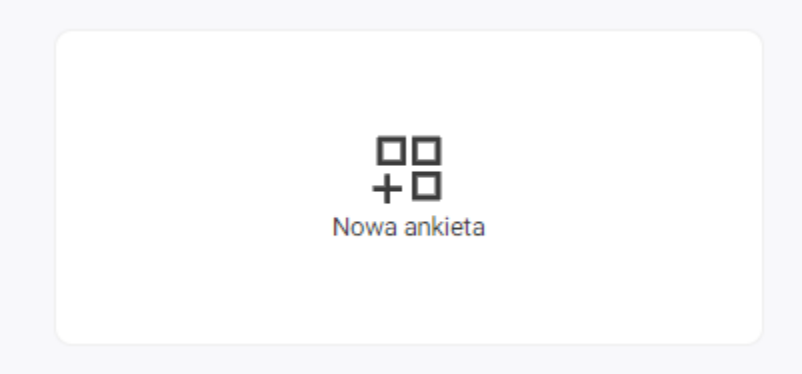
Albo takie, które są użytkownikowi udostępnione.



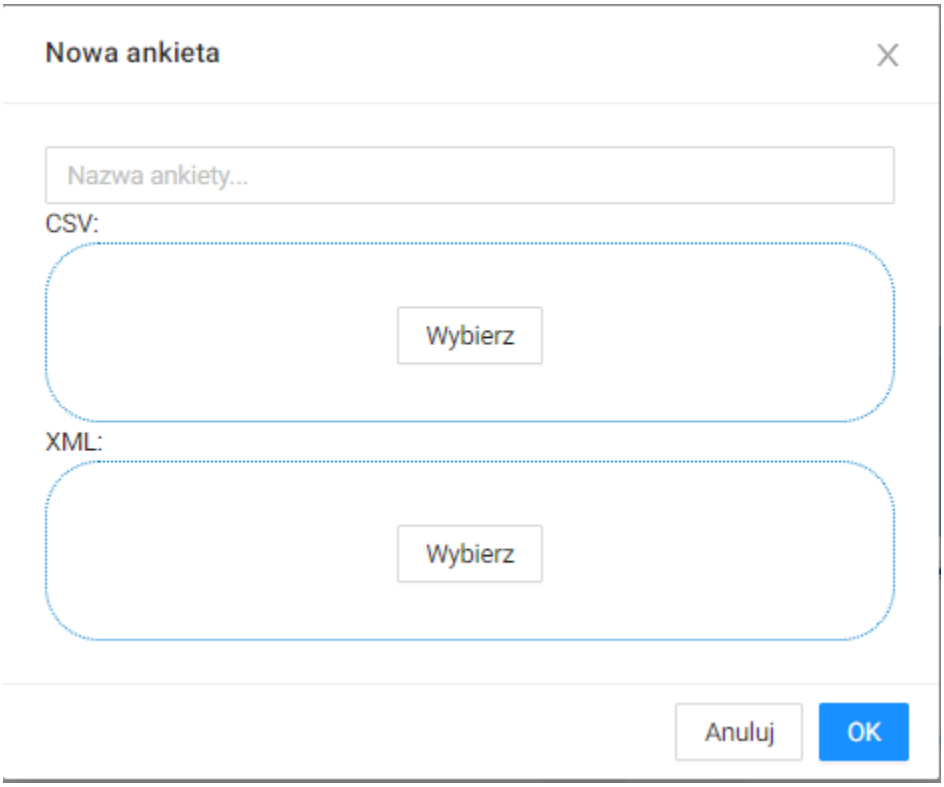
2.3 Edytor ankiet

2.3.1 Dodawanie wyników z ankiety

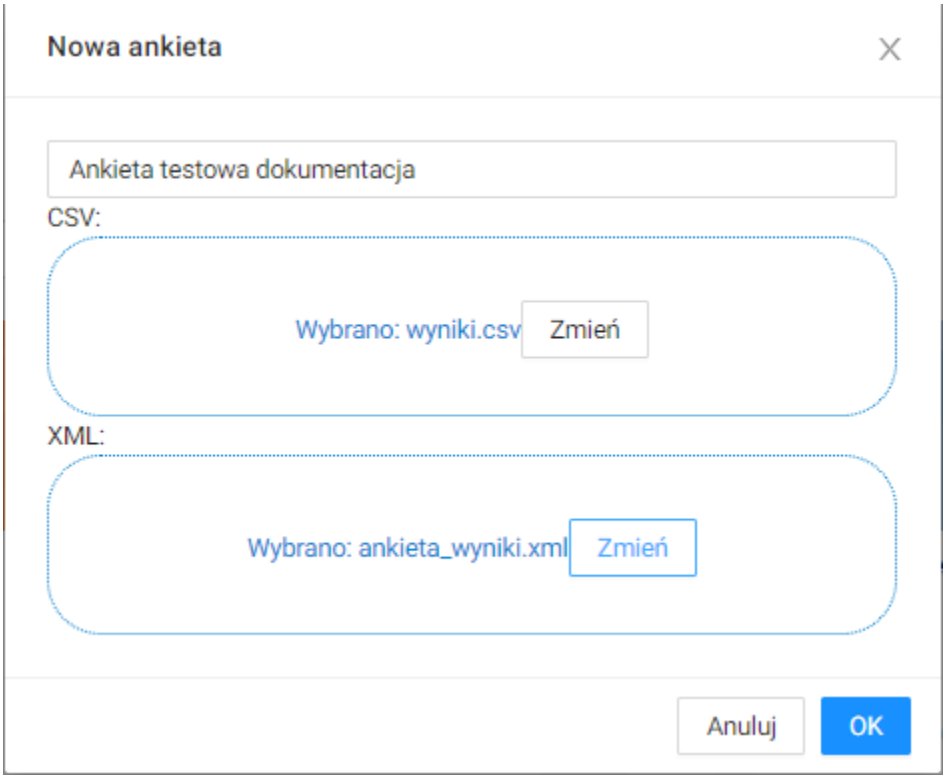
W panelu głównym należy kliknąć `Nowa ankieta`.



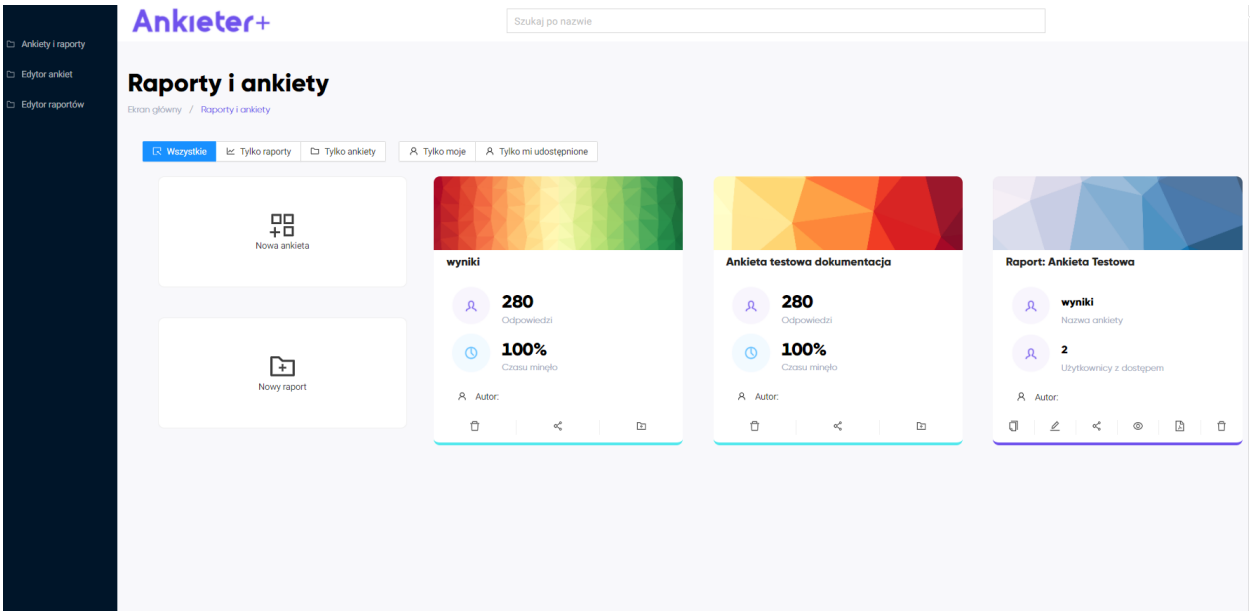
Wyświetli się okno:



W oknie należy podać nazwę nowej ankiety, umieścić plik `.csv` z wynikami ankiety oraz plik `.xml` ze strukturą ankiety.



Należy kliknąć przycisk `OK`. Wyniki ankiety zostaną dodane i wyświetlą się w panelu głównym.



ROZDZIAŁ 3

Dla administratora

3.1 Pobranie nowej wersji

Kod ankietera znajduje się w repozytoriach pod tym adresem.

Repozytoria te, zawierające frontend, backend oraz dokumentację złożone w jedną strukturę folderów gotową do uruchomienia jako aplikacja, znajdują się w repozytorium release.

Aby wykorzystać najnowsze repozytoria źródłowe, należy uruchomić znajdujący się w tym repozytorium skrypt:

```
update.sh -c
```

3.2 Instalacja

Strona serwerowa Ankieter+ do poprawnego działania wymaga języka Python w wersji 3.8. Wymagane są także dodatkowe biblioteki języka Python zawarte w pliku „requirements". W celu zainstalowania tych bibliotek należy w wierszu poleceń, będąc w folderze backend/ wykonać polecenie:

```
python3 -m pip install -r requirements.txt
```

Następnie należy zainstalować potrzebną strukturę katalogów, bazę danych dla aplikacji oraz plik konfiguracyjny. Służy do tego skrypt setup.py:

```
python3 setup.py
```

Podczas działania skryptu administrator zostanie poproszony o podanie adresów URL z których będzie korzystała aplikacja, ścieżki do certyfikatu SSL oraz swojego maila i ewentualnie peselu. Następnie skrypt utworzy konto o podanej nazwie posiadające przywileje administratorskie. Po zakończeniu jego działania w bieżącym katalogu zostaną utworzone następujące katalogi oraz pliki:

- `survey/` - zawierający pliki w formacie XML zawierające informacje o strukturach ankiet źródłowych,
- `raw/` - zawiera pliki w formacie CSV z wynikami ankiet,

- `data/` -zawiera pliki SQLite3 ze wstępnie przetworzonymi danymi zebranymi w ankietach,
- `report/` - zawiera pliki w formacie JSON opisującymi utworzone raporty,
- `master.db` - główna baza danych zawierająca dane na temat użytkowników,
- `config.py` - plik konfiguracyjny zawierający informacje potrzebne aplikacji do startu.

Aby uruchomić serwer należy wykonać następujące polecenia:

```
python3 main.py
```

3.3 Konfiguracja

Plik `config.py` tworzony jest przez skrypt `setup.py`, jednak jego parametry można modyfikować jeszcze po instalacji. Wymaganymi parametrami są:

- `CAS_URL` (str) - adres interfejsu sieciowego wykorzystywanego systemu CAS,
- `CAS_VERSION` (int) - wersja systemu CAS,
- `APP_URL` (str) - adres, na którym hostowany jest Ankieter+, nie powinien on zawierać kończącego ukośnika `/`,
- `APP_PORT` (int) - port wykorzystywany przez aplikację,
- `SSL_CONTEXT` (str, str) - para *(ścieżka do certyfikatu, ścieżka do klucza prywatnego)*

Do niewymaganych należą:

- `GUEST_NAME` (str) - nazwa, którą dostaje konto gościa (domyślnie `'Goście'`),
- `ADMIN_DEFAULT_PERMISSION` (str) - domyślne uprawnienia administratora do *wszystkich* ankiet (domyślnie `'o'`),
- `DAEMONS_INTERVAL` (int) - sekundowy interwał, co który budzone są demony wykonujące dodatkową pracę dla serwera. Obecnie takie demony nie są dostępne, planuje się jednak demon aktualizujący dane ankiet w bazie oraz demon archiwizujący (domyślnie `5*60`),
- `LOCALHOST` (bool) - ustawiony na `True` powoduje, że aplikacja hostowana jest na adresie *localhost* (domyślnie `False`),
- `DEBUG` (bool) - ustawiony na `True` pozwala na logowanie się do systemu z pominięciem autoryzacji (domyślnie `False`).

3.4 Logika uprawnień

3.4.1 Użytkownicy

Każdy użytkownik systemu Ankieter+ ma przypisany jedną z następujących ról:

- „s”: *superuser* - użytkownik z uprawnieniami administratora, ma on wgląd we wszystkie ankiety i raporty utworzone w systemie oraz dostęp do panelu zarządzania użytkownikami i grupami.
- „u”: *user* - zwykły użytkownik, mogący tworzyć, udostępniać i oglądać/edytować udostępnione mu ankiety.
- „g”: *guest* - każdy niezalogowany/niedodany do systemu użytkownik, ma wgląd tylko w raporty udostępnione publicznie lub przez link.

3.4.2 Raporty oraz ankiety

Każdy użytkownik może mieć nadane jedno z następujących rodzajów uprawnień do każdej/każdego z ankiet/raportów w systemie:

- „o”: *own* - właściciel ankiety/raportu, posiada prawa do odczytu, edycji, kopiowania, udostępniania oraz usuwania danego obiektu.
- „w”: *write* - edycja, pozwala na podgląd, edycje oraz kopiowanie raportów
- „r”: *read* - odczyt, zezwala jedynie na podgląd gotowych raportów
- „n”: *none* - oznacza brak jakichkolwiek uprawnień. W bazie danych nie istnieje jawna reprezentacja tego uprawnienia, ponieważ polega ono na braku innych uprawnień.

ROZDZIAŁ 4

Dla programisty

4.1 Wstęp

W poniższym rozdziale przedstawiono dokumentację modułów wykorzystywanych na serwerze.

4.2 Jak dodać nowy wykres?

Wykresy są generowane za pomocą biblioteki apache echarts [https://echarts.apache.org/]. Aby dodać nowy wykres, najłatwiej odwiedzić stronę z przykładami [https://echarts.apache.org/examples/en/index.html] i poszukać wykresu zbliżonego. Po wejściu na jego podstronę można eksperymentować, dostosowując interaktywnie różne opcje. Gdy stworzymy już konfigurację taką jaką chcemy, możemy przejść do integracji nowego wykresu do aplikacji. Zapisz gdzieś obiekt JSON options do użycia później.

Tworzenie wykresu jest kilkuetapowe - najpierw dane do serwowane do klasy generującej `options` - obiekt JSONa, następnie jest on ponowanie modyfikowany przez `ColorsGenerator`. Pełny proces generowania nowego wykresu jest zawarty w `generateChart(series: any, chartElement: ChartReportElement, reportId, namingDictioanry, dictionaryOverrides, localOverrides = undefined,fullQuery=undefined): EChartsOption` w pliku `charts.service.ts`.

Klasa generująca `options` musi dziedziczyć po `AbstractChartGenerator`. Każda taka klasa pochodna musi zostać także zarejestrowana w funkcji: `getGenerator` w `charts.service.ts` przez dopisanie do słownika `strategyType` nowododanej klasy jako wartość. Kluczem niech będzie krótka nazwa `string` używana do identyfikacji klasy. Taki słownik jest workaroundem bo przez niektóre ograniczenia Angulara nie można podać klasy jako first class object w template komponentu.

4.2.1 Tutorial

1. Tworzymy nowy plik o nazwie MyChart.ts.
2. Wklejamy tam szablon klasy generującej `options` wykresu

```
export class MyChartGenerator extends AbstractChartGenerator {

  constructor(series: any, chartElement: ChartReportElement, namingDictionary, public␣
↪reportsService: ReportsService, dictionaryOverrides) {
    super(series, chartElement, namingDictionary, reportsService, dictionaryOverrides);
  }

  generate(): AbstractChartGenerator {

      return this
  }

  getAllCount(reportId, complimentedQuery: SurveyQuery = undefined) {
  }

  asJSONConfig(): EChartsOption {
    return {} as EChartsOption
  }
}
```

`generate()` jest wywoływane jednorazowo przy każdej zmianie ustawień wykresu w UI przez użytkownika. `asJSONConfig()` zwraca `options` wykresu. Należy wkleić to co wygenerowaliśmy w kreatorze na stronie echarts z dodatkowym uzupełnieniem o serie danych. Serie danych pochodzą bezośrednio z response z serwera i są dostępne pod `this.rawSeries`

1. Wybieramy tekstowy identyfikator klasy - może to być jej nazwa zapisana w klasie jako `static name = "MyChart"`
2. Otwieramy plik `reports/editor/chart-editor-view.component.ts`
3. Szykujemy obrazek-miniaturkę wykresu i umieszczamy ją w `assets/`
4. W `reports/editor/chart-editor-view.component.ts` ctrl+f `<nz-tab nzTitle="Wygląd i układ">` W elemencie `<section class="query-marker"></section>` dodajemy markup

```
<div class="spacer"></div>
<figure class="indicator-card indicator-card-velvet preset"
       nz-tooltip="Użyj tego wykresu aby przedstawić wyniki z pytań wielokrotnego wyboru.
↪ Na przykład: dlaczego poleciłbyś UAM"
       (click)="pickPreset('multipleChoice');">
 <div class="indicator-card-inner">
   <div class="indicator-card-header">Wielokrotny wybór</div>
   <div class="indicator-card-content"><img src="./assets/preset2.png" style="width: 100%
↪"></div>
 </div>
</figure>
```

Do uzupełniania: tooltip, miniaturka, nazwa i `(click)="pickPreset('MyChart');"`, gdzie MyChart to nazwa naszego nowego wykresu identyczną z ustaloną wcześniej.

1. Do funkcji pickPreset w tym samym pliku dopisujemy klucz do słownika `fun`. Wartością jest funkcja

bez argumentów która ustawia różne parametry charakterystyczne dla tego rodzaju wykresu i sposób w jaki układane jest zapytanie do bazy danych. Możemy schować prawą kolumnę `Grupuj przez` ustawiając `this.hideGrupBy=true;`, schować panel wybierania rodzajów agregeacji `this.hideData=true;`. Zmienić zachowania po kliknięciu jakiegoś elementu interfejsu: .. `this.onPickQuestion=(question)=>{}` `this.byPickerClick = (by) => {}` Ustawić domyślny typ agregacji: `this.chartData.dataQuery.as[0]='share'` Ustawić domyślny typ grupowania `this.chartData.dataQuery.by[0] = "*"` Lub zrobić cokolwiek innego co jest potrzebne aby umożliwić użytkownikowi wybranie wszystkich potrzebnych opcji.

2. Teraz należy wyświetlić gotowy wykres. Do fragmentu `<section class="chart-area" *ngIf="['multipleChoice', 'groupedBars', 'multipleBars', 'linearCustomData', 'summary','groupSummary', 'multipleBarsOwnData'].includes(chartData.config.type) && this.echartOptions">` należy dopisać nasz typ wykresu żeby się wyświetlał w domyślnym trybie - jako prosty wykres generowany na podstawie `options`

3. (Opcjonalnie) Ostatnią czynnością jest kolorowanie wykresu. W `ColorsGenerator.ts` dodajemy do słownika x nowy wpis `[MyChartGenerator.name] = (o) => this.myChartGenerator(o);` oraz nową funkcję do klasy

```
myChartGenerator(options: EChartsOption): EChartsOption {
    return options;
  }
```

która modyfikuje i zwraca obiekt options w celu nadania kolorów i kosmetyki konkretnym seriom lub elementom. Jeżeli nie jest to potrzebne można nic nie dodawać.

4.3 convert module

convert.**antimode**(*vals: pandas.core.series.Series*)
Return one of the rarest of the values in a series.

Parametry **vals** (*pandas.Series*) – The series

Zwraca One of the rarest values

convert.**csv_to_db**(*survey:* database.Survey, *filename: str*, *defaults: Dict = {}*)
Read the source CSV file and save it to a new database

Parametry

- **survey** (Survey) – The Survey
- **filename** (*str*) – Name of the source CSV file in the raw/ directory
- **defaults** (*Dict*) – A dict with default values set for each column name

convert.**db_to_csv**(*survey:* database.Survey)
Convert db data to csv and write csv file into temp directory

Parametry **survey** (Survey) – The Survey

convert.**detect_csv_sep**(*filename: str*) → str
Detect the separator used in a raw source CSV file.

Parametry **filename** (*str*) – The name of the raw CSV in the raw/ directory

Zwraca The separator string

Typ zwracany str

convert.**get_column_mismatches**(*survey:* database.Survey, *df: pandas.core.frame.DataFrame*) → tuple
Check which columns in the schema are not present in the data, and also which columns in the data are not present in the schema.

Parametry

- **survey** (database.Survey) – Survey object for which the data was gathered
- **df** (*pandas.DataFrame*) – DataFrame containing the compacted data

Zwraca A pair of lists of columns that the data lacks, and the extra ones

Typ zwracany tuple

convert.**get_default_values**(*survey:* database.Survey) → Dict
Get default value string for every question in the survey database

Parametry **survey** (database.Survey) – The survey

Zwraca A dict from question names to a set of its defaults

Typ zwracany Dict

convert.**json_to_xml**(*survey:* database.Survey, *survey_json*)
Convert survey from JSON format to Ankieter xml format.

Parametry

- **survey** (Survey) – The Survey that is edited or created
- **survey_json** (*Dict*) – Survey JSON to be converted

convert.**nodefaults**(*defaults: Dict*, *name: str*)
Creates a pandas row aggregator that skips default column values and leaves a value from the first column with a non-default. It's used when a few columns represent the same variable, in that case the value chosen by the user is saved in one column, and the others contain defaults. Such a group of columns can be joined into one columns with a use of aggregator functions returned by this function.

Parametry

- **defaults** (*Dict*) – Default column values as returned from get_default_values
- **name** (*str*) – A name of the group of columns column to be aggregated

Zwraca A pandas row aggregator

Typ zwracany function

convert.**raw_to_compact**(*survey:* database.Survey, *df: pandas.core.frame.DataFrame*, *defaults: Dict = {}*) → pandas.core.frame.DataFrame
Convert a raw Ankieter DataFrame into a compact format suitable for data analysis. The change is mainly about joining separate columns that in fact represent the same question.

Parametry

- **survey** (database.Survey) – Survey object for which the data was gathered
- **df** (*pandas.DataFrame*) – DataFrame containing the raw data
- **defaults** (*Dict*) – A dict with default values set for each column name

Zwraca The compacted data

Typ zwracany pandas.DataFrame

convert.**xml_to_json**(*survey:* database.Survey)
Convert survey from Ankieter xml format to json format

Parametry **survey** (Survey) – The Survey that is edited or created

Zwraca The survey in json format

Typ zwracany Dict

4.4 database module

class database.**Link**(***kwargs*)

Klasy bazowe: sqlalchemy.orm.decl_api.Model

ObjectId
Id of the object the permission is to

ObjectType
Type of the object the permission is to

PermissionType
Perission granted by the link

Salt
The salt of the link

id
Link Id

class database.**Report**(***kwargs*)

Klasy bazowe: sqlalchemy.orm.decl_api.Model

AuthorId
Id of the user who created the report

BackgroundImg
Filename of the report's background image in the menu

Name
Title of the report

SurveyId
Id of the source survey

id
Report Id

class database.**ReportGroup**(***kwargs*)

Klasy bazowe: sqlalchemy.orm.decl_api.Model

Group
The name of the group

ReportId
Id of the report that belongs to a group

class database.**ReportPermission**(***kwargs*)

Klasy bazowe: sqlalchemy.orm.decl_api.Model

ReportId
The Id of the report the permission is to

Type
The type of the permission

UserId
The Id of the user that holds the permission

class database.**Survey**(***kwargs*)
Klasy bazowe: sqlalchemy.orm.decl_api.Model

AnkieterId
Id of the Survey in USOS Ankieter

AuthorId
Id of the user who created the survey

BackgroundImg
Filename of the survey's backgroun image in the menu

EndsOn
End date of the survey

IsActive
No use of this value is implemented yet

Name
Title of the survey

QuestionCount
Number of questions in the survey

StartedOn
Start date of the survey

id
Survey Id

class database.**SurveyGroup**(***kwargs*)
Klasy bazowe: sqlalchemy.orm.decl_api.Model

Group
The name of the group

SurveyId
Id of the survey that belongs to a group

class database.**SurveyPermission**(***kwargs*)
Klasy bazowe: sqlalchemy.orm.decl_api.Model

SurveyId
The Id of the survey the permission is to

Type
The type of the permission

UserId
The Id of the user that holds the permission

class database.**User**(***kwargs*)
Klasy bazowe: sqlalchemy.orm.decl_api.Model

CasLogin
CAS Login

FetchData
No use of this value is implemented yet

Pesel
PESEL number of the user

Role
The user's role in the system

as_dict()

id
User Id

class database.**UserGroup**(***kwargs*)
Klasy bazowe: sqlalchemy.orm.decl_api.Model

Group

UserId

database.**create_report**(*user:* database.User, *survey:* database.Survey, *name: str*, *author: int*) → *database.Report*
Create report for a given user

Parametry

- **user** (User) – The creator of the report
- **survey** (Survey) – The source survey of the report
- **name** (*str*) – The name of the new report
- **author** (*int*) – The database id of the creator

Zwraca The newly created report

Typ zwracany *Report*

database.**create_survey**(*user:* database.User, *name: str*) → *database.Survey*
Create survey by given user

Parametry

- **user** (User) – The creator of the new survey
- **name** (*str*) – Name of a survey

Zwraca The object of the new survey

Typ zwracany *Survey*

database.**create_user**(*cas_login: str*, *pesel: str*, *role: str*) → *database.User*
Create a new user.

Parametry

- **cas_login** (*str*) – New user's cas login
- **pesel** (*str*) – New user's PESEL number
- **role** (*Role*) – New user's role (values: «s»,»u»,»g»)

Zwraca The new user's User object

Typ zwracany *User*

database.**delete_group**(*group: str*)
Delete a group

Parametry **group** (*str*) – The name of the group

database.**delete_report**(*report:* database.Report)
Delete report

Parametry **report** (Report) – The report to be deleted

database.**delete_survey**(*survey:* database.Survey)
Delete survey

Parametry **survey** (Survey) – The survey to be deleted

database.**delete_user**(*user:* database.User)
Delete user from Users database and their permissions from SurveyPermissions and ReportPermissions.

Parametry **user** (User) – The user to be deleted

database.**get_all_users**() → dict
Get all users

Zwraca Cas logins and users id.

Typ zwracany dict

database.**get_answers**(*survey_id: int*) → Dict
Get answers for given survey

Parametry **survey_id** – Id of the survey

Zwraca Answers in the survey

Typ zwracany Dict

database.**get_answers_count**(*survey:* database.Survey) → int
Get number of answers in the database for a given survey.

Parametry **survey** (Survey) – The survey

Zwraca The number of answers

Typ zwracany int

database.**get_columns**(*conn: sqlite3.Connection*) → List[str]
Get column names in the order just like it is returned from the DB.

Parametry **conn** (*sqlite3.Connection*) – Connection to the database

Zwraca A list of column names in the database.

Typ zwracany Lis[str]

database.**get_dashboard**() → Dict
Get dashboard for user

Zwraca Returns dictionary with surveys and reports

Typ zwracany Dict

database.**get_group_users**(*group: str*) → List[*database.User*]
Get users assigned to given group.

Parametry **group** – Name of a group

Rtype group str

Zwraca Returns List of User objects

Typ zwracany List[*User*]

database.**get_groups**() → List[str]
Get all groups from UserGroups

Zwraca List of all groups

Typ zwracany List[str]

database.**get_link_details**(*tag: str*) → *database.Link*
Get link details

Parametry **tag** (*str*) – Salt and id string from the link

Zwraca Returns a Link object

Typ zwracany *Link*

database.**get_permission_link**(*permission: Literal['o', 'w', 'r', 'n']*, *object_type: Literal['s', 'r']*, *object_id: int*) → str
Create and obtain a permission link.

Parametry

- **permission** (*Role*) – Permission type (values: «o», «w», «r», «n»)
- **object_type** (*Literal['s', 'r']*) – Type of the object shared by the link
- **object_id** (*int*) – Id of the object

Zwraca A concatenated salt and link id as a string

Typ zwracany str

database.**get_report**(*report_id: int*) → *database.Report*
Get report by given id.

Parametry **id** (*int*) – Id of a report

Wyrzuca ***error.API*** – no such report

Zwraca Requested report object

Typ zwracany *Report*

database.**get_report_permission**(*report:* database.Report, *user:* database.User) → Literal['o', 'w', 'r', 'n']
Get permission of given user for the report.

Parametry

- **report** (Report) – The report
- **user** (User) – The user whose permissions are to be checked

Zwraca The user's permissions for the report

Typ zwracany Permission

database.**get_report_survey**(*report:* database.Report) → *database.Survey*
Get survey assigned to the given report

Parametry **report** (Report) – Report object

Zwraca The source survey of the report

Typ zwracany *Survey*

database.**get_report_users**(*report:* database.Report) → dict
Get users having permission to the given report

Parametry **report** (Report) – The report

Zwraca Returns a dict with user ids as keys and their permissions under them

Typ zwracany dict

database.**get_survey**(*survey_id: int*) → *database.Survey*
Get survey by given id.

Parametry **survey_id** (*int*) – Survey's id

Wyrzuca ***error.API*** – no such survey

Zwraca Returns survey

Typ zwracany *Survey*

database.**get_survey_permission**(*survey:* database.Survey, *user:* database.User) → Literal['o', 'w', 'r', 'n']
Get permission of given user for the survey.

Parametry

- **survey** (Survey) – The survey
- **user** (User) – The user whose permissions are to be checked

Zwraca The user's permissions for the survey

Typ zwracany Permission

database.**get_survey_users**(*survey:* database.Survey) → dict
Get users having permission to given survey

Parametry **survey** (Survey) – The survey

Zwraca Returns a dict with user ids as keys and their permissions under them

Typ zwracany dict

database.**get_types**(*conn: sqlite3.Connection*) → Dict[str, str]
Get types for each column in the database.

Parametry **conn** (*sqlite3.Connection*) – Connection to the database

Zwraca A dictionary mapping names of columns to SQL names of their types

Typ zwracany Dict[str, str]

database.**get_user**(*login: Any = ''*) → *database.User*
Get a user object from DB.

Parametry **login** – User's CAS login, id or guest if empty string (default: „")

Wyrzuca ***error.API*** – no such user

Zwraca User object

Typ zwracany *User*

database.**get_user_groups**(*user:* database.User) → List[str]
Get all groups for given user

Parametry **user** (User) – Given user

Zwraca List of user's groups names

Typ zwracany List

database.**get_user_reports**(*user:* database.User) → List[*database.Report*]
Get reports for which the user has permissions. For administrators it returns all reports.

Parametry **user** (User) – User object

Zwraca List of Report objects

Typ zwracany List[*Report*]

database.**get_user_surveys**(*user:* database.User) → List[*database.Survey*]
Get surveys for which the user has permissions. For administrators it returns all surveys.

Parametry **user** (User) – User object

Zwraca List of Survey objects

Typ zwracany List[*Survey*]

database.**open_survey**(*survey:* database.Survey) → sqlite3.Connection
Open an SQLite3 connection to the survey database

Parametry **survey** (Survey) – The survey

Zwraca A connection to the DB of the survey

Typ zwracany sqlite3.Connection

database.**rename_report**(*report:* database.Report, *name: str*)
Rename report.

Parametry

- **report** (Report) – The Report object
- **name** (*str*) – New report name

database.**rename_survey**(*survey:* database.Survey, *name: str*)
Rename survey.

Parametry

- **survey** (Survey) – The Survey object
- **name** (*str*) – New survey name

database.**set_permission_link**(*tag: str*, *user:* database.User)
Set permission using link.

Parametry

- **tag** (*str*) – Salt and id string from the link
- **user** (User) – User that will gain the permission

Zwraca Returns permission type, object name and object id

Typ zwracany Permission, object, int

database.**set_report_permission**(*report:* database.Report, *user:* database.User, *permission: Literal['o', 'w', 'r', 'n']*, *bylink=False*)
Set permission of given user for report.

Parametry

- **report** (Report) – The report
- **user** (User) – The user whose permissions are to be set
- **permission** (*Permission*) – The user's permissions for the report
- **bylink** – Is the permission set because of a link? (default: False)

database.**set_survey_meta**(*survey:* database.Survey, *name: str*, *question_count: int*, *meta: dict*)
Add meta information of a given survey.

Parametry

- **survey** (Survey) – The survey to be modified
- **name** (*int*) – The new name of a survey
- **question_count** (*int*) – Number of questions
- **meta** (*dict*) – Other information (started_on, ends_on, is_active)

database.**set_survey_permission**(*survey:* database.Survey, *user:* database.User, *permission: Literal['o', 'w', 'r', 'n']*, *bylink=False*)
Set permission of given user for survey.

Parametry

- **survey** (Survey) – The survey
- **user** (User) – The user whose permissions are to be set
- **permission** (*Permission*) – The user's permissions for the survey
- **bylink** – Is the permission set because of a link? (default: False)

database.**set_user_group**(*user:* database.User, *group_name: str*)
Set group for user. If already exists do nothing.

Parametry

- **user** (User) – User
- **group_name** (*str*) – Name of a group

database.**unset_user_group**(*user:* database.User, *group: str*)
Unset user from a group.

Parametry

- **user** (User) – User object
- **group** (*str*) – Group name

4.5 error module

exception error.**API**(*message: str*)
Klasy bazowe: *error.Generic*

as_dict() → dict
Return message as a dict.

Return value: returns dict object

exception error.**Generic**(*message: str*)
Klasy bazowe: Exception

add_details(*message: str*)
Add details to the error message.

Keyword arguments: message – details to be concatenated

Return value: returns self

4.6 grammar module

grammar.**analyze**(*tp: Any*, *obj: Any*) → str
Analyze object structure.

Keyword arguments: tp – expected object structure obj – given object

Return value: returns message after analyze

grammar.**check**(*tp: Any*, *obj: Any*)
Validate object structure.

Keyword arguments: tp – expected object structure obj – given object

4.7 setup module

setup.**ask**(*question*, *choices*)

setup.**setup**(*admin*, *pesel*)

4.8 table module

class table.**Aggregator**(*func*, **types*)
Klasy bazowe: object

class table.**Filter**(*symbol*, *func*, *arity*, **types*, *beg=''*, *end=''*, *sep=', '*)
Klasy bazowe: object

table.**aggregate**(*query*, *data*)
Aggregate survey data

Keyword arguments: query – survey data data – dataframe object

Return value: returns aggregated data

table.**applymacros**(*query*)

table.**columns**(*query*, *types*, *conn: sqlite3.Connection*)
Obtain dataframe required to compute the query

Keyword arguments: query – survey data conn – sqlite3.Connection

Return value: returns dataframe object

table.**count**(*s*)

table.**create**(*query*, *conn: sqlite3.Connection*)
Create data from survey

Keyword arguments: query – survey data conn – sqlite3.Connection

Return value: returns survey data

table.**get_pandas_filter_of**(*json_filter*, *ctype*)

table.**get_sql_filter_of**(*json_filter*, *types*)

table.**mode**(*s*)

table.**reorder**(*data*)
Reorder survey data

Keyword arguments: data – survey data

Return value: returns survey reordered data

table.**rows**(*s*)

table.**share**(*s*)

table.**tobasetypes**(*s*)

table.**typecheck**(*query*, *types*)
Check types of survey data

Keyword arguments: query – survey data types – column types

4.9 main module

main.**copy_report**(*report_id*)
Create new duplicated report

Route /api/report/<int:report_id>/copy

Methods GET

Roles s, u

Parametry **report_id** (*int*) – Report's id

Zwraca {„reportId": report.id}

Typ zwracany Dict

main.**create_report**()
Create report

Route /api/report/new

Methods POST

Roles s, u

Zwraca {„reportId": report.id}

Typ zwracany dict

main.**create_survey**()
Create survey

Route /api/survey/new

Methods POST

Roles s, u

Zwraca {„id": survey_id}

Typ zwracany Dict

main.**create_user**()
Create a new user

Route /api/user/new

Methods POST

Roles s

Returns Dict

Return parameters

Parametry **id** (*int*) – user id

main.**debug_login**(*username*)
Login in debug mode without cas

Route /api/login/<string:username>

Methods GET

Parametry **username** (*str*) – cas login

main.**delete_group**()
Delete group

Route /api/dashboard

Methods DELETE

Roles s

Zwraca {«message»: «group deleted»}

Typ zwracany Dict

main.**delete_report**(*report_id*)
Delete report

Route /api/report/<int:report_id>

Methods DELETE

Roles s, u

Parametry **report_id** (*int*) – Report's id

Zwraca {«message»: «report has been deleted»,»reportId»: report_id}

Typ zwracany Dict

main.**delete_survey**(*survey_id*)
Get dashboard for user

Route /api/survey/<int:survey_id>

Methods DELETE

Roles s, u

Parametry **survey_id** (*int*) – Survey's id

Typ zwracany Dict

Zwraca {«message»: «survey has been deleted»,»surveyId»: survey_id}

main.**delete_user**(*user_id*)
Delete user from Users database and their permissions from SurveyPermissions and ReportPermissions.

Route /api/user/<int:user_id>

Methods DELETE

Roles s

Parametry **user_id** – user id

Return dict {„delete”: user_id}

main.**download_report**(*report_id*)
Download report as json

Route /api/report/<int:report_id>/download

Methods GET

Roles s, u

Parametry **report_id** (*int*) – Report's id

Zwraca json file

Typ zwracany File

main.**download_survey_csv**(*survey_id*)
Download survey csv

Route /api/data/<int:survey_id>/download

Methods GET

Roles s, u

Parametry **survey_id** (*int*) – Survey's id

Zwraca Survey csv data

Typ zwracany File

main.**download_survey_xml**(*survey_id*)
Get survey schema xml

Route /api/survey/<int:survey_id>/download

Methods GET

Roles s, u

Parametry **survey_id** (*int*) – Survey's id

Returns xml file witch survey schema

Typ zwracany File

main.**for_roles**(**roles*)

main.**get_all_users**()
Get all users

Route /api/user/all, /api/users

Methods GET

Roles s, u, g

Returns Dict

Return

Parametry

- **casLogin** (*str*) – cas login
- **id:** (*int*) – user id

main.**get_bkg**(*path*)
Get background image

Route /bkg/<path:path>

Methods GET

Parametry **path** (*str*) – path to the image

main.**get_dashboard**()
Get dashboard for user

Route /api/dashboard

Methods GET

Roles s, u, g

Returns Dict

Return

Parametry

- **answersCount** (*int*) – number of survey answers,
- **startedOn** (*timestamp*) – date when survey started
- **endsOn** (*timestamp*) – date when survey ended,
- **authorId** (*int*) – author's user_id,
- **authorName** (*str*) – author's name,
- **backgroundImg** (*str*) – filename of a background image,
- **id** (*int*) – survey/report id,
- **name** (*str*) – survey/report name,
- **sharedTo** (*dict*) – {user_id: permission_type}
- **type** (*string*) – object type (survey/report)
- **userId** (*int*) – logged user id
- **connectedSurvey** (*dict*) – {survey_id: survey_name}

main.**get_data**(*survey_id*)
Get survey data

Route /api/data/<int:survey_id>

Methods POST

Roles s, u, g

Parametry **survey_id** (*int*) – Survey's id

Zwraca

Typ zwracany Dict

main.**get_data_types**(*survey_id*)
Get types for each column in the database.

Route /api/data/<int:survey_id>/types

Methods GET

Roles s, u, g

Parametry **survey_id** (*int*) – Survey's id

Zwraca A dictionary mapping names of columns to SQL names of their types

Typ zwracany Dict

main.**get_dictionary**()
Returns dictionary with labels

Route /api/dictionary

Methods GET

Roles s, u, g

Zwraca id and label

Typ zwracany dict

main.**get_docs**(*filename*)
Redirect to documentation

Route /docs

Methods GET

Parametry **filename** (*str*) – filename of docs main page, default index,html

main.**get_group_users**()
Get users assigned to given group.

Route /api/group/users

Methods POST

Roles s, u

Zwraca Returns dict of user_id

Typ zwracany dict

main.**get_groups**()
Get list of groups

Route /api/dashboard

Methods GET

Roles s, u, g

Zwraca

Typ zwracany Dict

main.**get_questions**(*survey_id*)
Get column names in the order just like it is returned from the DB.

Route /api/dashboard

Methods GET

Roles s, u, g

Parametry **survey_id** (*int*) – Survey's id

Zwraca A list of column names in the database. {«questions»: questions}

Typ zwracany Dict

main.**get_report**(*report_id*)
Get report data

Route /api/report/<int:report_id>

Methods GET

Roles s, u, g

Parametry **report_id** (*int*) – Report's id

Zwraca report data in json format

Typ zwracany Dict

main.**get_report_answers**(*report_id*)
Get answers for Report's survey

Route /api/report/<int:report_id>/answers

Methods GET

Roles s, u, g

Parametry **report_id** (*int*) – Report's id

Zwraca Answers in the survey

Typ zwracany Dict

main.**get_report_data**(*report_id*)
Get data for chart

Route /api/report/<int:report_id>/data

Methods POST

Roles s, u, g

Parametry **report_id** (*int*) – Report's id

Zwraca Parsed data

Typ zwracany Dict

main.**get_report_survey**(*report_id*)
Get survey assigned to the given report

Route /api/report/<int:report_id>/survey

Methods GET

Roles s, u, g

Parametry **report_id** (*int*) – Report's id

Zwraca {„surveyId": survey.id }

Typ zwracany Dict

main.**get_report_users**(*report_id*)
Get dashboard for user

Route /api/report/<int:report_id>/users

Methods GET

Roles s, u

Parametry **report_id** (*int*) – Report's id

Zwraca Returns a dict with user ids as keys and their permissions under them

Typ zwracany Dict

main.**get_static_file**(*path*)

main.**get_survey**(*survey_id*)
Get survey json by given id

Route /api/survey/<int:survey_id>

Methods GET

Roles s, u

Parametry **survey_id** (*int*) – Survey's id

Zwraca survey json

Typ zwracany dict

main.**get_user_details**()
Get logged user details

Route /api/user

Methods GET

Roles s

Parametry **user_id** – user id

Zwraca User object

Typ zwracany *User*

main.**get_user_groups**(*user_id*)
Get all user groups with users

Route /api/user/<int:user_id>/group

Methods GET

Roles s, u

Parametry **user_id** (*int*) – user id

Zwraca List[group_name]

main.**get_user_id_details**(*user_id*)
Get user details

Route /api/user/<int:user_id>

Methods GET

Roles s

Parametry **user_id** – user id

Zwraca User object

Typ zwracany *User*

main.**index**(*text=None*)

main.**link_to_report**(*report_id*)
Create and obtain a permission link for report

Route /api/report/<int:report_id>/link

Methods POST

Roles s, u

Parametry **report_id** (*int*) – Report's id

Zwraca {«link»: link}

Typ zwracany Dict

main.**link_to_survey**(*survey_id*)
Get permission link to survey

Route /api/survey/<int:survey_id>/link

Methods POST

Roles s, u

Parametry **survey_id** (*int*) – Survey's id

Zwraca {«link»: link}

Typ zwracany dict

main.**login**()
Login with cas

Route /api/dashboard

Methods GET

main.**logout**()
Logout

Route /api/logout

Methods GET

Roles s, u, g

main.**on_errors**(*details*)

main.**rename_report**(*report_id*)
Rename report

Route /api/report/<int:report_id>/rename

Methods POST

Roles s, u

Parametry **report_id** (*int*) – Report's id

Zwraca {«message»: «report name has been changed», «reportId»: report.id, «title»: new_name}

Typ zwracany Dict

main.**rename_survey**(*survey_id*)
Rename survey

Route /api/survey/<int:survey_id>/rename

Methods POST

Roles s, u

Parametry **survey_id** (*int*) – Survey's id

Zwraca {«message»: «survey name has been changed», «surveyId»: survey.id, «title»: new_name}

Typ zwracany dict

main.**set_group**()
Add users to group

Route /api/group/change

Methods POST

Roles s

Zwraca {«message»: «users added to groups»}

Typ zwracany Dict

main.**set_permission_link**(*hash*)
Set permission using link.

Route /api/link/<hash>

Methods GET

Roles s, u, g

Parametry **hash** (*str*) – Salt and id string from the link

Zwraca {«permission»: perm, «object»: object,»id»: id,}

Typ zwracany Dict

main.**set_report**(*report_id*)
Save report changes

Route /api/report/<int:report_id>

Methods POST

Roles s, u

Parametry **report_id** (*int*) – Report's id

Zwraca {„reportId": report.id}

Typ zwracany Dict

main.**share_report**(*report_id*)
Share report for given users

Route /api/report/<int:report_id>/share

Methods POST

Roles s, u

Parametry **report_id** (*int*) – Report's id

Zwraca {„message": „permissions added"}

Typ zwracany Dict

main.**share_survey**(*survey_id*)
Share survey for given users

Route /api/survey/<int:survey_id>/share

Methods POST

Roles s, u

Parametry **survey_id** (*int*) – Survey's id

Zwraca {„message”: „permissions added”}

Typ zwracany str

main.**unset_group**()
Remove users from groups

Route /api/group/change

Methods DELETE

Roles s

Zwraca {«message»: «users removed from groups»}

Typ zwracany Dict

main.**upload_results**(*survey_id*)
Upload survey results

Route /api/data/new

Route /api/data/new/<int:survey_id>

Route /api/data/<int:survey_id>/upload

Methods POST

Roles s, u

Parametry **survey_id** (*int*) – Survey’s id

Zwraca {„id”: survey.id,”name”: name}

Typ zwracany Dict

main.**upload_survey**(*survey_id*)
Upload survey

Route /api/survey/<int:survey_id>, /api/survey/<int:survey_id>/upload

Methods POST

Roles s, u

Parametry **survey_id** (*int*) – Survey’s id

Zwraca {„id”: survey_id}

Typ zwracany Dict

ROZDZIAŁ 5

Indices and tables

- genindex
- modindex
- search

Indeks modułów Pythona

Indeks

I

J

L

M

N

O

P

Q

R

S

T

U

X