# Rozwój narzędzia rozszerzającego usługę Google Forms

(Development of an extension to Google Forms service)

Aleksandra Rozkrut

Praca inżynierska

**Promotor:** dr hab. Jan Otop, prof. UWr

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

Wrocław 2022

**Streszczenie**

Praca jest kontynuacją pracy Agnieszki Pawickiej „Implementacja narzędzia rozszerzającego usługę Google Forms". Zawiera implementację oraz opis rozwijanego rozszerzenia do usługi Google Forms. Rozszerzenie to pozwala na automatyczne generowanie formularzy z odpowiedniego pliku w formacie JSON, konwertowanie wstawek matematycznych napisanych w LaTeX do odpowiednich symboli oraz zarządzanie niektórymi własnościami utworzonych uprzednio formularzy. W pracy znajduje się również omówienie wykorzystanych technologii.

# Spis treści

# Rozdział 1.

# Wprowadzenie

Wraz z rozpoczęciem pandemii problem zdalnego sprawdzania wiedzy i umiejętności zrobił się popularny. Miejsca, w których było to szczególnie uciążliwe to m. in. szkoły, uczelnie, czy procesy rekrutacyjne w różnych firmach. Dobre narzędzie do przeprowadzania zdalnych testów powinno spełniać pewne wymagania, takie jak:

- przejrzystość formularzy,
- łatwość tworzenia nowych testów,
- możliwość skutecznego weryfikowania tożsamości osoby piszącej test.

Dodatkowo dobrze jest, gdy takie narzędzie posiada również takie funkcje jak:

- automatyczne sprawdzanie pytań zamkniętych,
- kontrolowanie czasu potrzebnego poszczególnym osobom na zakończenie testu,
- możliwość losowych zmian kolejności pytań,
- możliwość zapisywania odpowiedzi w popularnym formacie.

Ponadto niektórzy egzaminujący cenią sobie możliwość wstawiania symboli matematycznych, wykresów czy zdjęć w ramach pytań.

We wrześniu 2021 roku Agnieszka Pawicka obroniła pracę inżynierską o tytule „Implementacja narzędzia rozszerzającego usługę Google Forms", w której zaproponowała system umożliwiający generowanie testów Google Form z pliku w formacie JSON. Projekt umożliwiał także generowanie pytań zawierających wstawki w LaTeX, które były automatycznie kompilowane i wstawiane do formularza.
Celem tej pracy jest usprawnienie działania wymienionego wyżej narzędzia.

## 1.1. Wprowadzone usprawnienia

W pracy został zaimplementowany szereg modyfikacji, które ułatwiają korzystanie z narzędzia zarówno przy tworzeniu formularzy, zarządzaniu nimi, jak i ich późniejszym sprawdzaniu. Te usprawnienia to:

- łatwiejszy sposób instalacji narzędzia,
- zmiana interfejsu użytkownika na bardziej intuicyjny,
- zmiana sposobu wgrywania zakodowanego formularza,
- możliwość edycji pliku JSON z zakodowanym formularzem już po wgraniu go,
- możliwość zobaczenia odpowiedzi oraz wyników w formacie JSON,
- możliwość pobrania danych o przesłanych odpowiedziach i wynikach sprawdzania automatycznego w formacie EXCEL,
- utworzone formularze są teraz przechowywane na Dysku Google użytkownika - nie jak w poprzedniej wersji na wspólnym dysku dla wszystkich użytkowników,
- szereg nowych opcji związanych z parametrami formularza tj.:
  - możliwość zmiany kolejności odpowiedzi wewnątrz pytań zamknietych,
  - możliwość dodania opisu testu,
  - odpowiedzi do pytań mogą być teraz w formie symboli matematycznych (LaTeX),
  - możliwe jest dodanie oceniania innego niż zerojedynkowe dla pytań wielokrotnego wyboru.

Wszystkie wymienione zmiany wraz z użytymi narzędziami, instrukcją użytkownika oraz opisem środowsk i napotkanych problemów są opisane w dalszych rozdziałach niniejszej pracy.

# Rozdział 2.

# Środowisko

Praca jest rozszerzeniem innego projektu, wobec tego część użytych bibliotek zostało opisanych w rozdziale „Środowisko" pracy „Implementacja narzędzia rozszerzającego usługę Google Forms". W niniejszym rozdziale skupię się więc na nowo dodanych elementach środowiska. W celu zapoznania się z całością polecam lekturę wymienionej pracy.
Kod projektu można podzielić na cztery główne części:

1. interfejs użytkownika,
2. serwer komunikujący się z zewnętrznymi API oraz bazą danych,
3. kod służący do instalacji zależności i uruchamiania aplikacji na maszynie wirtualnej - korzystający z technologii Vagrant i Ansible
4. skrypt w języku Python konwertujący wstawki w języku LaTeX do obrazów.

Dodatkowo aplikacja korzysta z Google Forms API oraz Imgur API.

## 2.1. Interfejs użytkownika

### 2.1.1. TypeScript

Aplikacja została napisana w języku TypeScript, który jest nadzbiorem języka JavaScript. Aplikacja korzysta z bibliotek: types/node, typescript.

### 2.1.2. React

Technologia służąca do tworzenia interfejsów graficznych aplikacji internetowych. Aplikacja korzysta z bibliotek: emotion/react, emotion/styled, types/react, types/react-dom, react, react-dom, react-scripts.

### 2.1.3. Material UI

Biblioteka udostępniająca wiele gotowych komponentów, których można użyć w aplikacjach korzystających z Reacta.

### 2.1.4. axios

Biblioteka służąca do wysyłania zapytań http i pobierania danych z serwera.

### 2.1.5. date-fns

Biblioteka pomagająca zarządzać datami. Udostępnia wiele użytecznych funkcji.

### 2.1.6. react-code-blocks

Biblioteka z gotowymi komponentami służącymi do wyświetlania bloków kodu.

## 2.2. Serwer

### 2.2.1. Node.js

Serwer został napisany w środowisku Node.js, które służy do tworzenia aplikacji serwerowych w języku JavaScript.

### 2.2.2. Google

Aplikacja używa bibliotek google-cloud/local-auth oraz googleapis/forms odpowiednio do autoryzacji projektu oraz użytkownika w serwerach Google oraz do wysyłania zapytań do Google Forms API, które jest opisane w dalszej sekcji.

### 2.2.3. excel4node

Biblioteka służąca do tworzenia plików Excel.

### 2.2.4. lowdb

Mała, lokalna baza danych przechowująca dane w plikach JSON. Za jej pomocą przechowywane są zakodowane formularze.

## 2.3. npm

npm to menadżer pakietów, który jest używany w aplikacji serwera oraz interfejsu użytkownika. npm zarządza pakietami wykorzystywanymi przez te aplikacje. Listy zależności aplikacji, wersje tych zależności oraz skrypty, w tym skrypt, który uruchamia aplikację interfejsu użytkownika można znaleźć w pliku `package.json`.

## 2.4. Vagrant i Ansible

Vagrant to narzędzie służące do tworzenia wirtualnych środowisk programistycznych z użyciem na przykład VirtualBox, ale współdziała też z wieloma innymi oprogramowaniami. Ansible to narzędzie służące do automatyzacji wdrażania, konfiguracji i zarządzania. Z pomocą tych dwóch technologi zostały napisane skrypty, które same skonfigurują maszynę wirtualną i następnie uruchomią na niej wszystkie aplikacje potrzebne do działania narzędzia.

## 2.5. Google Forms API oraz Imgur API

### Google Forms API

Google Forms API to interfejs, który umożliwia zarządzania Formularzami Google na dysku użytkownika. W czasie pisania tej pracy jest to wciąż bardzo nowe narzędzie, ponieważ jego pierwsza oficjalna wersja została opublikowana w 2022 roku. API pozwala m.in. zarządzać pytaniami i odpowiedziami, zmieniać ustawienia formularza i włączać automatyczne ocenianie. Jest ono wciąż dynamicznie rozwijane.

### Imgur API

Imgur API pozwala zarządzać plikami użytkownika na platformie Imgur, w tym dodawać je jako zalogowany lub niezalogowany użytkownik. Jeśli pytanie, które ma zostać dodane do formularza, zawiera wstawki w języku LaTeX generowane są obrazy z tekstem pytania. Natstępnie zostająv one wgrane na serwery Imgur, aby potem serwery Google mogły je pobrać używając zewnętrznego URL.

## 2.6. Python

Narzędzie korzysta ze skryptu w języku Python, aby stworzyć obrazy ze wstawkami w języku LaTeX. Jest on dokładnie opisany we wspomnianej wcześniej pracy. Zostało dodane do niego wgrywanie wygenerowanych obrazków na serwery Imgur

za pomocą biblioteki imgurpython. Wszystkie biblioteki z jakich korzysta skrypt to: imgurpython, tex2pix, pdf2image oraz opencv-python.

## 2.7. Poppler

Poppler to biblioteka służąca do renderowania plików PDF. Wspomniany powyżej skrypt używa pakietu poppler-utils, do tworzenia plików PDF. Ten pakiet jest powszechnie używany na systemach operacyjnych opierających się na Debianie.

## 2.8. LaTeX

Jak już wiele razy zostało wspomniane w tej pracy, narzędzie pozwala generować formularze ze wstawkami w języku LaTeX, zatem aby narzędzie działało poprawnie LaTeX musi być zainstalowany na maszynie, na której narzędzie ma zostać uruchomione. Skrypt Ansible dodatkowo instaluje polski pakiet językowy.

# Rozdział 3.

# Opis techniczny

Głównym tematem tego rozdziału jest opis techniczny wszystkich aplikacji i skryptów, które tworzą całe narzędzie. Została tu opisana struktura kodu i schemat komunikacji między aplikacjami. Kod projektu można podzielić na cztery główne części:

1. interfejs użytkownika,
2. serwer komunikujący się z zewnętrznymi API oraz bazą danych,
3. kod służący do instalacji zależności i uruchamiania aplikacji na maszynie wirtualnej - korzystający z technologii Vagrant i Ansible,
4. skrypt w języku Python konwertujący wstawki w języku LaTeX do obrazów.

## 3.1. Serwer

Kod serwera i inne pliki potrzebne do jego działania znajdują się w katalogu `forms-app`.

### 3.1.1. Kod źródłowy

Kod źródłowy serwera znajduje się w katalogu `src`. Serwer udostępnia kilka adresów, pod którymi można się z nim komunikować używając protokołu http:

- `GET /forms`: zwraca wszystkie formularze, które znajdują się w lokalnej bazie danych,
- `GET /forms/:id`: zwraca informacje o formularzu o podanym identyfikatorze,
- `POST /forms`: pod ten adres można wysłać formularz w kodowaniu JSON, żeby stworzyć nowy formularz Google,

- `PUT /forms/:id`: pod ten adres można wysłać zakodowany formularz w formacie JSON, żeby edytować formularz o podanym identyfikatorze; edytowany jest formularz w lokalnej bazie danych oraz formularz Google,
- `GET /forms/:id/answers`: zwraca wszystkie przesłane odpowiedzi,
- `GET /forms/:id/scores`: zwraca wszystkie ocenione przesłane odpowiedzi,
- `GET /forms/:id/scores`: przesyła plik EXCEL z ocenionymi przesłanymi odpowiedziami,
- `DELETE /forms/:id`: usuwa formularz o podanym identyfikatorze z lokalnej bazy danych oraz usuwa wszystkie pytania z formularza Google, ale nie usuwa formularza z dysku Google.

Kod z konfiguracją tych adresów można znaleźć w pliku `src/server.js`. Aplikacja z interfejsem użytkownika używa tych adresów, żeby komunikować się z serwerem. Plik `formsFunctions.js` zawiera funkcje pomocnicze, które przykładowo tworzą zapytania wysyłane do Google Forms API lub oceniają zamknięte pytania w przesłanych odpowiedziach i konwertują je do formatów JSON lub EXCEL. Plik `jsonValidator.js` zawiera walidator zakodowanych formularzy w formacie JSON, który jest dokładnie opisany w pracy „Implementacja narzędzia rozszerzającego usługę Google Forms". W katalogu `tex2png` znajduje się moduł, który służy do uruchamiania skryptu w języku Python, który tworzy obrazki ze wstawkami w języku LaTeX i udostępnia je na serwisie Imgur.

### 3.1.2. Google Forms API

W poprzedniej wersji projekt korzystał z Google Apps Script. Część kodu oraz utworzone formularze Google były przechowywane na jednym konkretnym koncie Google, którego ewentualna zmiana byłaby bardzo skomplikowana. Było to niewygodne w użyciu, może niezbyt bezpieczne oraz utrudniało rozwój narzędzia. Dlatego zdecydowałam się przepisać część komunikującą się serwerami Google na kod wysyłający zapytania do Google Forms API. Google Forms API to niedawno powstałe API, które w łatwy sposób pozwala zarządzać formularzami Google na dowolnym koncie Google i jest wciąż dynamiczne rozwijane, więc w przyszłości można liczyć na dodanie do niego wielu funkcjonalności, które pozwoliłoby na dalszy rozwój narzędzia. Żeby korzystać z Google Forms API wystarczy założyć projekt w konsoli Google Cloud, co jest w pełni darmowe dla każdej osoby posiadającej konto Google i jest dokładnie opisane w następnym rozdziale. Następnie użytkownik może pozwolić aplikacji zarządzać formularzami na swoim dysku Google bez obaw, że ktokolwiek inny miałby do nich dostęp. Serwer wysyła do Google Forms API zapytania i dostaje odpowiedzi w formacie JSON. Na ten moment API pozwala zarządzać formularzami oraz przesłanymi do nich odpowiedziami, co pozwala na napisanie naszego własnego sposobu oceniania tych odpowiedzi. Google Forms API udostępnia automatyczne ocenianie,

ale za każde pytanie można dostać tylko zero albo maksymalną liczbę punktów, co jest niewystarczające. Chcielibyśmy móc za każde pytanie przydzielić przykładowo połowę punktów, jeśli nie zostały zaznaczone wszystkie poprawne odpowiedzi. W tej pracy udało się to zrealizować.

#### Uwierzytelnianie i autoryzacja

Proces uwierzytelniania i autoryzacji z Google przebiega w kilku krokach:

1. uwierzytelnianie aplikacji: kiedy aplikacja się uruchomia, wysyła ona swoje dane uwierzytelniające do Google z informacją, że chciałaby mieć dostęp do formularzy użytkowników Google,

2. ekran zgody: Google zwraca ekran zgody, w którym użytkownik musi wybrać swoje konto Google i zgodzić się na to, że aplikacja będzie mogła zarządzać formularzami na tym koncie. Jeśli użytkownik to zrobi, aplikacja wysyła zapytanie do Google ze swoimi danymi uwierzytelniającymi w celu uzyskania tokena dostępu, którego będzie teraz używać, żeby uzyskać dostęp do wszystkich potrzebnych zasobów,

3. przyznanie tokena dostępu: jeśli wszystko przejdzie pomyślnie, Google odeśle aplikacji żądany token dostępu. Token zawiera informacje o tym, do których zasobów aplikacja uzyskała dostęp,

4. dostęp do zasobów: aplikacja może teraz wysyłać zapytania do Google, w których może zarządzać odpowiednimi zasobami, w naszym przypadku formularzami i przesłanymi do nich odpowiedziami,

5. odświeżanie tokena: tokeny mają czas ważności i jeśli on upłynie, aplikacja musi poprosić o odświeżony token, aby móc dalej działać.

Uwierzytelnianie i autoryzacja w serwisach Google są dokładnie opisane w ich dokumentacji.

### 3.1.3. Lokalna baza danych

Do narzędzia została dodana możliwość automatycznego ocenienia zamkniętych pytań w przesłanych odpowiedziach. Google zapamiętuje przesłane odpowiedzi w postaci:

```
{
  "questionId": string,
  "textAnswers": {
    "answers": [
    {
      "value": string
    }
  ]
  }
}
```

Formularze Google są tworzone z przesłanego do serwera zakodowanego w formacie JSON obiektu, którego opis można znaleźć w rozdziale „Instrukcja użytkownika". Tworząc pytanie w formularzu, Google nadaje mu identyfikator i potem posługuje się nim zwracając przesłane do formularza odpowiedzi. Dla każdego pytania zwraca jego identyfikator w polu `questionId` i odpowiedzi w tablicy `answers`. Trzeba było zatem dodać jakiś sposób, który pozwalałby na rozpoznanie, które pytanie w przesłanych odpowiedziach od Google to pytanie w przesłanym zakodowanym formularzu od użytkownika. Prostym sposobem było dodanie bazy danych, która pamiętałaby zakodowany formularz razem jego identyfikatorem na serwerach Google, identyfikatorami pytań oraz sposobem oceniania tych pytań. Serwer używa małej, lokalnej bazy danych `lowdb`. Tworzy ona pliki, w których przechowuje dane w formacie JSON. W przypadku naszej aplikacji lowdb tworzy plik o nazwie `db.json` w katalogu `src/database`. Zakodowane formularze w bazie danych wyglądają następująco:

```
FormTemplate {
  id: string, // identyfikator formularza
  title: string, // nazwa formularza
  questions: QuestionTemplate[], // tablica z pytaniami
  description?: string, // opis
  startDate?: string, // start testu
  endDate?: string, // koniec testu
  responderUri: string, // adres formularza Google
}
```

```
QuestionTemplate {
  questionId?: string, // identyfikator pytania typu list,
                       // checkBox lub text
  type: QuestionType,  // typ pytania
  text: string, // tekst pytania
  tex: boolean, // pytanie zawiera wstawki matematyczne,
                // gdy pole wynosi true
  answers?: AnswerTemplate[], // tablica odpowiedzi do
                              // pytania
  points?: number, // punkty dla pytania typu list
  pointsArray?: Array<number> // tablica z punktami dla
                              // pytania typu grid
}
```

```
QuestionType {
  list,
  checkBox,
  grid,
  text,
}
```

```
AnswerTemplate {
  questionId?: string, // identyfikator pytania dla typu
                       // grid
  text: string, // tekst odpowiedzi
  tex: boolean, // zawiera wstawki matematyczne,
                // gdy pole wynosi true
  correct: boolean, // poprawna, gdy pole wynosi true
}
```

Pola oznaczone „?:” to pola, którym można przypisać wartość null. Pytania różnych typów różnią się od siebie swoim kodowaniem, ale aby uprościć ich reprezentację, powstał jeden model mogący reprezentować je wszystkie. Jedna z różnic polega na tym, że dla pytań typu grid to odpowiedzi w schemacie formularza są tak naprawdę kolejnymi podpytaniami, więc każde takie podpytanie ma swój własny identyfikator. Zatem ich identyfikatory są przechowywane w polu `questionId` w `AnswerTemplate`. Identyfikatory pozostałych typów pytań są przechowywane w `QuestionTemplate`. Kolejna różnica polega na tym, że pole `points` odnosi się tylko do pytań typu list, a pole `pointsArray` dotyczy pytań checkBox i grid.

Aplikacja automatycznie ocenia przesłane odpowiedzi w kilku krokach:

1. pobiera przesłane odpowiedzi od Google,
2. dla każdej przesłanej odpowiedzi sprawdza, czy czas przesłania jest poprawny - po rozpoczęciu i przed zakończeniem testu,
3. dla każdej przesłanej odpowiedzi:
   (a) identyfikuje, które odpowiedzi dotyczą którego pytania,
   (b) sprawdza, czy odpowiedzi są poprawne,
   (c) przydziela liczbę punktów zgodnie z wprowadzonym przez użytkownika sposobem oceniania, więcej w rozdziale „Instrukcja użytkownika",
4. opcjonalnie generuje plik EXCEL z listą osób, które wzięły udział w teście i ich wynikami.

### 3.1.4. Katalog assets

Wszystkie obrazki z tekstem ze wstawkami w języku LaTeX oraz wszystkie inne pliki, które powstają podczas ich tworzenia są generowane w katalogu `assets`, dlatego ważne jest, żeby go nie usuwać. Zawartość tego katalogu można usunąć (oprócz pliku `.gitignore`), ponieważ wygenerowane obrazki są używane tylko podczas tworzenia lub edytowania formularza.

### 3.1.5. Katalog src/credentials

W tym katalogu należy umieścić dane uwierzytelniające projektu Google. Więcej informacji można znaleźć w rozdziale „Instrukcja użytkownika". Nie należy usuwać tego katalogu.

### 3.1.6. Katalog src/excels

W tym katalogu można znaleźć wygenerowane podczas oceniania pliki EXCEL z wynikami. Nie należy go usuwać, ale jego zawartość już można.

### 3.1.7. ESLint

Do aplikacji serwera został dodany ESLint, który pomaga pisać przejrzysty kod i na bieżąco analizuje kod i sprawdza, czy nie zawiera on błędów. To zachowanie można dowolnie edytować używając zasad udostępnionych przez to narzędzie. Zasady skonfigurowane dla tego projektu można znaleźć w pliku `.eslintrc.json`. W celu dokładniejszego zrozumienia, jak działa to narzędzie zachęcam do zapoznania się z dokumentacją.

## 3.2. Skrypt w języku Python

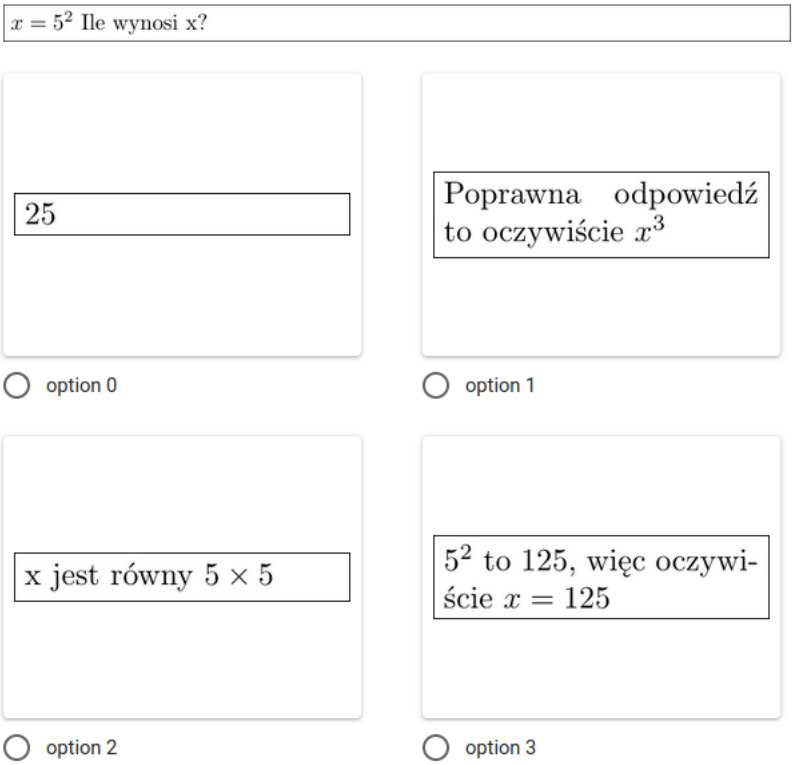
Głównym zadaniem skryptu jest generowanie obrazków ze wstawkami w języku LaTeX i zostało już to napisane w poprzedniej wersji narzędzia. W tej pracy rozwinęliśmy go, dodając do niego wgrywanie utworzonych obrazków do serwisu Imgur oraz opcję generowania obrazków, które będą odpowiedziami do pytania. Poprzednia wersja narzędzia pozwalała wgrywać wygenerowane obrazki tylko w miejsce pytania w formularzu Google.

### 3.2.1. Imgur API

Ważną częścią funkcjonalności narzędzia jest tworzenie pytań ze wstawkami w języku LaTeX. Jest to realizowane przez konwertowanie wstawek do obrazków, a następnie dodawanie ich do formularzy w miejscu pytania lub odpowiedzi. Jak zostało wspomniane wcześniej narzędzie przestało korzystać z Google Apps Script i zamiast tego korzysta teraz z Google Forms API. Wysyłając zapytania, które mają dodać pytanie do formularza z obrazkiem, Google Forms API przyjmuje tylko zewnętrzne URL, z których może pobrać te obrazki. Nie da się ich bezpośrednio przesłać razem z zapytaniem. Pojawił się, więc problem jak udostępnić te obrazki do pobrania. Postanowiłam wykorzystać do tego Imgur i Imgur API. Skrypt w języku Python, który można znaleźć w katalogu `src/tex2png`, tworzy obrazki, a następnie przesyła je do Imgur API, a z powrotem otrzymuje adresy, które serwer przesyła do Google Forms API. Następnie Google pobiera obrazy z podanych adresów i dodaje pytanie do formularza Google.

### 3.2.2. Pliki graficzne w odpowiedziach do pytań

Kolejną dodaną do narzędzia funkcjonalnością jest możliwość generacji odpowiedzi do pytań w formularzu ze wstawkami w języku LaTeX. Poprzednio było to możliwe tylko dla treści samych pytań, ponieważ Google Apps Script nie oferował takiej możliwości, natomiast Google Forms API już na to pozwala. Pojawił się za to inny problem. Formularze Google wyświetlają odpowiedzi, które są plikami graficznymi w dość niefortunny sposób:




Rysunek 3.1: Wygląd przykładowego formularza

Dwie odpowiedzi zawsze zostaną umieszczona obok siebie, więc pliki graficzne z odpowiedziami muszą być wąskie. Inaczej tekst odpowiedzi byłby bardzo mały. Zmiana wyświetlania odpowiedzi z plikami graficznymi jest na ten moment niemożliwa. Dlatego do skryptu Google został dopisany inny sposób generacji plików graficznych z odpowiedziami do pytań.

## 3.3. Interfejs użytkownika

Kod źródłowy aplikacji implementującej interfejs użytkownika znajduje się w katalogu `forms-app-ui`. Aplikacja używa środowiska React i została stworzona używając programu `Create React App`. Pliki `index.tsx` oraz `index.html` tworzą podstawę całej aplikacji. W pliku `index.tsx` jest wstrzykiwany Reactowy komponent `App`, który jest głównym komponentem całej aplikacji. W katalogu `src/components` można znaleźć wszystkie inne komponenty, które tworzą aplikację:

- `FormCard`: implementuje jeden wpis z informacjami o formularzu,
- `FormList`: implementuje listę formularzy, czyli listę komponentów FormCard,
- `CreateDialog`: dialog, za pomocą którego można stworzyć nowy formularz,
- `AddDialog`: dialog, za pomocą którego można dodać już istniejący formularz,

- `EditDialog`: dialog, za pomocą którego można edytować formularz.

Aplikacja używa komponentów funkcyjnych oraz Reactowych hooków takich jak `useState`, `useEffect` i `useMemo`. W katalogu `src/models` znajdują się interfejsy reprezentujące modele danych przesyłanych z serwera:

- `AnswerTemplate`: jedna odpowiedź do pytania w formularzu,
- `QuestionTemplate`: jedno pytanie w formularzu,
- `FormTemplate`: formularz.

Dodatkowo znajduje się tam typ wyliczeniowy `QuestionType` z typami pytań.

### 3.3.1. Material UI

Implementacja korzysta z wiele komponentów z biblioteki Material UI takich jak:

- `Alert` i `Collapse`, żeby wyświetlić ewentualne błędy,
- kontenery `List`, `ListItem`, `Box`, `Stack`, `Accordion`, `Card`,
- `Button`, żeby w łatwy sposób zaimplementować przyciski i ich logikę,
- `CircularProgress`, żeby wyświetlić animację ładowania, kiedy użytkownik musi poczekać na zakończenie jakiejś operacji,
- `Typography`, żeby wyświetlić tekst,
- `Dialog`, żeby zaimplementować dialogi,
- ikony.

Wszystkie te komponenty są niezwykle proste w użyciu oraz wyglądają przejrzyście i łatwo do zrozumienia.

### 3.3.2. Code Block

Jeszcze jednym ważnym komponentem jest `CodeBlock` z biblioteki react-code-blocks. Pozwala on łatwo wyświetlać fragmenty kodu razem z numerami linii i kolorowaniem tekstu w zależności od języka kodu. Jest on wykorzystywany w komponencie FormCard.

### 3.3.3. Daty

Narzędzie pozwala określać rozpoczęcie i zakończenia testu, więc należało zaimplementować zarządzanie datami. Aplikacja używa do tego biblioteki date-fns, która jest lekka i prosta w użyciu. Przykładowo używamy funkcji `format`, żeby ładnie wyświetlić daty we wpisach o formularzach.

### 3.3.4. ESLint

Podobnie jak w serwerze, tutaj również został użyty ESLint. W pliku `.eslintrc.json` zostały skonfigurowane zasady dla projektu korzystającego z języka TypeScript oraz środowiska React.

## 3.4. Vagrant i Ansible

Plik `Vagrantfile` służy do konfiguracji maszyny wirtualnej i środowiska, na której będzie działać narzędzie. Wyjaśnię teraz krótko, jak działa ten skrypt. Pole `config.vm.box` określa jaki system operacyjny zostanie użyty. W naszym przypadku jest to Ubuntu 20.04 LTS (Focal Fossa). Następnie skrypt instaluje Ansible i definiuje ścieżkę do playbooka Ansible, który jest opisany w dalszej części tego rozdziału. W następnej części udostępniamy porty na maszynie wirtualnej przez porty na systemie, na którym działa ta maszyna.

```
config.vm.network :forwarded_port, guest: 80, host: 8001
```

W powyższym przykładzie udostępniamy port 80 na maszynie wirtualnej przez port 8001 na systemie użytkownika. W dalszej części skrypt konfiguruje pamięć, którą będzie mogła wykorzystać maszyna wirtualna i na końcu wypisuje wiadomość z adresami, które należy odwiedzić w przeglądarce po jej uruchomieniu. Przydatne polecenia:

- `vagrant up`: uruchamia maszynę wirtualną,
- `vagrant halt`: zatrzymuje maszynę wirtualną,
- `vagrant reload`: resetuje maszynę wirtualną,
- `vagrant destroy`: usuwa maszynę wirtualną,
- `vagrant provision`: przeprowadza od początku proces konfiguracji maszyny razem z ponownym zainstalowaniem zależności z playbooka Ansible,
- `vagrant ssh`: dodaje dostęp do powłoki maszyny wirtualnej.

### 3.4.1. Playbook

Playbook to plik w formacie yaml, który określa jakie programy i biblioteki powinny zostać zainstalowane oraz jakie usługi powinny zostać uruchomione po uruchomieniu maszyny wirtualnej. Można go znaleźć w katalogu `playbooks` pod nazwą `playbook.yaml`. Zawiera polecenia, które zainstalują wszystkie programy i biblioteki opisane w rozdziale „Środowisko". Oprócz tego są w nim dodane polecenia uruchomienia usług przez menadżera systemu `systemd`. Te usługi uruchamiają aplikacje serwera oraz interfejsu użytkownika i ich konfiguracje można znaleźć odpowiednio w plikach `forms-app.service` i `forms-app-ui.service`.

### 3.4.2. Przydatne polecenia

Po wykonaniu polecenia `vagrant ssh` uzyskuje się dostęp do powłoki maszyny wirtualnej. Pliki projektu znajduję się w katalogu `/vagrant`. Usługa z serwerem nazywa się `forms-app`, a z interfejsem użytkownika `forms-app-ui`. Kilka przydatnych poleceń, które można użyć w powłoce:

- `systemctl status <nazwa_usługi>`: wypisuje status usługi,
- `systemctl list-units -all -type=service | grep forms-app`: wypisuje status wszystkich usług, a następnie wybiera z nich tylko usługi forms-app i forms-app-ui; można w ten sposób sprawdzić, czy usługi są aktywne,
- `systemctl start <nazwa_usługi>`: włącza usługę,
- `systemctl stop <nazwa_usługi>`: zatrzymuje usługę,
- `journalctl -u <nazwa_usługi>`: wypisuje logi usługi razem z logami aplikacji.

# Rozdział 4.

# Instrukcja użytkownika

## 4.1. Początki pracy z narzędziem

Zanim zacznie się pracować z narzędziem, należy zainstalować wszystkie wykorzystywane przez nie zależności. Instalacja została usprawniona wykorzystując Vagrant oraz Ansible, ale można ją przeprowadzić również bez tych technologii. Oba te sposoby oraz późniejsza praca z narzędziem, która zależy od wybranego sposobu instalacji są opisane poniżej.

### 4.1.1. Projekt na Google Cloud

Aplikacja używa Google Forms API do zarządzania formularzami na dysku Google użytkownika i z tego powodu potrzebuje danych uwierzytelniających do komunikacji z tym API. Można je uzyskać zakładając projekt na Google Cloud. Dokładne instrukcje można znaleźć w poradnikach w dokumentacji Google, a w tej pracy spróbuję krótko opisać ten proces:

- załóż nowy projekt
  - zaloguj się do konsoli Google Cloud,
  - znajdź przycisk „Wybierz Projekt”,
  - w nowym oknie naciśnij przycisk „Nowy Projekt”,
  - wpisz nazwę projektu oraz lokalizację, a następnie naciśnij „Utwórz”,
- Dodaj Google Forms API do projektu
  - w menu po lewej stronie wybierz „Wyświetl Wszystkie Usługi”,
  - w sekcji „Zarządzanie” wybierz „Interfejsy API i usługi”,
  - wybierz swój projekt,
  - kliknij „Interfejsy API i usługi”,

  - wyszukaj „Google Forms API”,
  - w szczegółach Google Forms API kliknij „Włącz”,

- wygenereuj Klucz interfejsu API
  - przejdź do strony projektu,
  - w menu po lewej stronie wybierz „Dane logowania”,
  - kliknij „Create Credentials”, a potem „Klucz interfejsu API”,
  - zapisz klucz w pliku i zachowaj na później,

- skonfiguruj ekran zgody
  - przejdź do strony projektu,
  - ponownie kliknij „Create Credentials”, a potem „Identyfikator Klienta OAuth”,
  - kliknij „Skonfiguruj Ekran Zgody”,
  - jako typ użytkownika wybierz „Zewnętrzny” i kliknij „Utwórz”,
  - wpisz nazwę aplikacji, adresy e-mail i kliknij „Zapisz i kontynuuj”,
  - kliknij „Dodaj lub usuń zakres”,
  - w „Filtruj” wpisz „Google Forms API”,
  - zaznacz zakresy „forms.body” i „forms.responses.readonly”,
  - kliknij „Zaktualizuj”, następnie „Zapisz i kontynuuj”,
  - dodaj swój adres konta Google do użytkowników testowych, kliknij „Zapisz i kontynuuj” i na końcu „Powrót do panelu”,

- wygeneruj identyfikator klienta OAuth
  - przejdź do strony projektu,
  - wybierz ponownie „Dane Logowania”, „Create Credentials” i „Identyfikator Klienta OAuth”,
  - jako typ aplikacji wybierz „Aplikacja internetowa”, dodaj URI `http://localhost:3000/oauth2callback` do sekcji „Autoryzowane identyfikatory URI przekierowania” i kliknij „Utwórz”,
  - wybierz „Pobierz JSON”.

### 4.1.2. Imgur API

Aplikacja używa Imgur API, żeby wygenerować linki zewnętrzne obrazków z formułami matematycznymi, z których pobiera je serwer Google. Do komunikacji z Imgur API również potrzebujemy danych uwierzytelniania:

- zaloguj się do Imgur,

- odwiedź stronę Imgur API,
- znajdź „Register an application”,
- wypełnij dane, jako typ autoryzacji wybierz „OAuth2 authorization without a callback URL”,
- skopiuj Client ID i Client secret i zachowaj na później.

### 4.1.3. Repozytorium

Teraz należy sklonować repozytorium projektu — github.com/arozkrut/forms-app. Następnie w pobranym repozytorium w katalogu `forms-app/src/credentials` stwórz pliki:

- `api_key.txt`: wklej tu klucz interfejsu API, tak żeby plik był postaci

  ```
  key=<uzyskany klucz>
  ```

  Klucz służy do identyfikacji aplikacji podczas wysyłania zapytań do Google Forms API. Proces generacji klucza interfejsu API został opisany wczesniej w tym rozdziale.

- `credentials.json`: wklej tu zawartość pobranego pliku JSON. Plik zawiera dane uwierzytelniające, które również są używane przy wysyłaniu zapytań do Google Forms API. Proces generacji pliku JSON został opisany wcześniej w tym rozdziale.

- `imgur_credentials.txt`: wklej tu Client ID i Client secret, tak żeby plik był postaci:

  ```
  ClientID: <Client ID>
  ClientSecret: <Client secret>
  ```

  Identyfikator oraz sekret służą do identyfikacji aplikacji podczas wysyłania zapytań do Imgur API. Proces generacji identyfikatora oraz sekretu został opisany wcześniej w tym rozdziale.

### 4.1.4. Instalacja z Vagrant

Jeśli masz już zainstalowany Vagrant wystarczy, że w głównym katalogu projektu (tym z `Vagrantfile`) użyjesz polecenia `vagrant up`. Skrypt Ansible zainstaluje wszystkie potrzebne zależności i uruchomi wszystkie aplikacje. Ważne jest, że projekt

potrzebuje portów 3000, 9090 i 9091. Jeśli któryś z nich jest już zajęty narzędzie nie uruchomi się. Porty 9090 i 9091 można zmienić na inne, ale portu 3000 niestety nie. Opis jak to zrobić można znaleźć w sekcji „Porty". Instalacja może chwilę potrwać. Po jej zakończeniu wyświetli się wiadomość z linkiem prowadzącym do strony uwierzytelniania Google, gdzie trzeba wybrać konto Google, na którym będą przechowywane formularze oraz wyrazić zgodę na nadanie aplikacji dostępu do formularzy na tym koncie. Następnie w przeglądarce należy odwiedzić adres http://localhost:9091, gdzie działa interfejs użytkownika. Narzędzie jest gotowe do pracy.

### 4.1.5. Instalacja bez Vagrant

Jeśli użytkownik chce korzystać z narzędzia bez zainstalowanego Vagrant, musi on sam zainstalować wszystkie zależności: Python 3, LaTeX, poppler-utils, biblioteki do języka Python opisane w poprzednim rozdziale, nodejs, npm. Następnie w katalogach `forms-app` oraz `forms-app-ui` należy wydać polecenie `npm i`. Teraz należy włączyć serwer NodeJS poleceniem `node src/server.js` (koniecznie w katalogu `forms-app`). Serwer korzysta z portów 9090 i 3000, więc przed uruchomieniem należy upewnić się, że są one niezajęte. Port 9090 można zmienić na inny, ale 3000 już nie. Opis jak to zrobić można znaleźć w sekcji „Porty". Po uruchomieniu serwera w przeglądarce powinna otworzyć się strona uwierzytelniania Google, gdzie trzeba wybrać konto Google, na którym będą przechowywane formularze oraz wyrazić zgodę na nadanie aplikacji dostępu do formularzy na tym koncie. Po tym w katalogu `src/forms-app-ui` należy włączyć serwer interfejsu użytkownika poleceniem `npm start`, który uruchomi się pod adresem http://localhost:9091. Port 9091 na którym działa aplikacja można również zmienić na inny. Po odwiedzeniu http://localhost:9091 w przeglądarce można zacząć pracować z narzędziem.

## 4.2. Porty

Narzędzie korzysta z trzech portów: 9090, 9091 oraz 3000. Dwa pierwsze można zmienić edytując kilka linijek w kodzie, ale portu 3000 niestety nie da się zmienić na inny. Jest on wykorzystywany przez bibliotekę, służącą do autoryzacji aplikacji i użytkownika w serwerach Google i ta biblioteka nie udostępnia sposobu na zmianę tego portu. Niezależnie, czy narzędzie jest instalowane używając Vagranta, czy nie pierwsze kroki, żeby zmienić porty 9090 i 9091 są takie same:

- zmiana portu 9090:
  - w pliku `forms-app/src/server.js` znajdź zmienną `port` i zmień jej wartość,
  - w plikach
    - `forms-app-ui/src/App.tsx`

  * `forms-app-ui/src/components/FormCard.tsx`

  znajdź zmienną `SERVER_BASE_URL` i zmień port na inny,

- zmiana portu 9091:

  – w pliku `forms-app-ui/package.json` w `scripts`, `start` zmień `PORT=9091` na inny port.

Jeśli narzędzie jest instalowane z użyciem Vagranta w pliku `Vagrantfile` należy dodać nowe porty. Pod linijkami:

```
config.vm.network :forwarded_port, guest: 80, host: 8001
config.vm.network :forwarded_port, guest: 9090, host: 9090
config.vm.network :forwarded_port, guest: 3000, host: 3000
config.vm.network :forwarded_port, guest: 9091, host: 9091
```

dla każdego nowego portu należy dodać linijkę:

```
config.vm.network :forwarded_port, guest: <NOWY_PORT>, host: <NOWY_PORT>
```

Po wykonaniu tych kroków należy zresetować obie aplikacje lub maszynę wirtualną poleceniem `vagrant reload`.

## 4.3. Praca z narzędziem

### 4.3.1. Schemat pliku kodującego (JSON)

Poniżej znajduje się rozpisany schemat kodowania:

```
{
  "title": "string",
  "description": "string",
  "startDate": "string",
  "endDate": "string",
  "shuffleAnswers": "boolean",
  "questions": [{
    "type": "string",
    "text": "string",
    "tex" : "boolean",
    "answers": [{
      "text" : "string",
      "tex" : "boolean",
      "correct" : "boolean"
    }],
    "points": "number"
    "pointsArray": "array"
  }]
}
```

Jest to opis tych wartości, które mogą być użyte — nie wszystkie są jednak wymagane. Poniżej znajduje się szczegółowy opis pól i wartości:

- title – wartość tekstowa, odpowiada nagłówkowi formularza — **pole wymagane**,
- description – wartość tekstowa, krótki opis formularza, który pojawi się pod tytułem — **pole opcjonalne**,
- startDate – wartość tekstowa, data planowanego rozpoczęcia testu w formacie ISO. To pole służy tylko do odrzucania odpowiedzi, które zostały przesłane przed podaną datą podczas oceniania — **pole opcjonalne**,
- endDate – wartość tekstowa, data planowanego zakończenia testu w formacie ISO. To pole służy tylko do odrzucania odpowiedzi, które zostały przesłane po podanej dacie podczas oceniania — **pole opcjonalne**,
- shuffleAnswers – wartość boolowska, informacja o tym, czy odpowiedzi dla każdego pytania powinny zostać przetasowane dla każdego wyświetlenia formularza. Niestety na ten moment Google Forms API nie udostępnia opcji tasowania pytań, można ją jedynie włączyć ręcznie w ustawieniach formularza na stronie Google: settings -> presentation -> Shuffle question order — **pole opcjonalne**,

- questions – tablica, której każde pole zawiera informacje dotyczące jednego pytania — **pole wymagane**.
  Poniżej znajdują się wartości kodujące pojedyncze pytanie:

  – type – pole tekstowe dotyczące typu kodowanego pytania. Dopuszczalne wartości:

    * „checkBox" – pytanie zamknięte wielokrotnego wyboru,
    * „grid" – pytanie typu „prawda/fałsz",
    * „list" – zamknięte jednokrotnego wyboru,
    * „text" – otwarte.

    **pole wymagane**,

  – text – zawiera treść pytania w formie tekstowej. Może zawierać wstawki z LaTeX'a. Należy jednak pamiętać, że JavaScript traktuje symbol „\" jako specjalny — wszystkie wystąpienia „\" należy więc zastąpić „\\" — **pole wymagane**,

  – tex – wartość boolowska, jeśli **true** treść pytania (text) będzie konwertowana do obrazu z zachowaniem konwersji symboli matematycznych i innych wstawek z LaTeX'a z biblioteki standardowej — **pole wymagane**,

  – answers – tablica, każde pole zawiera jedną z możliwych odpowiedzi w następującej formie:

    * text – pole tekstowe, treść danej odpowiedzi; podobnie jak w treści pytania wszystkie wystąpienia „\" należy zastąpić „\\",
    * tex – wartość boolowska, jeśli **true** treść odpowiedzi (text) będzie konwertowana do obrazu z zachowaniem konwersji symboli matematycznych i innych wstawek z LaTeX'a z biblioteki standardowej, tylko dla pytań typu list i checkBox,
    * correct – wartość boolowska wskazująca czy dana odpowiedź jest prawidłowa. Domyślna wartość: false.

    — **pole opcjonalne**,

  – points – wartość numeryczna, odpowiada liczbie punktów przyznawanej za poprawą odpowiedź na pytanie (tylko dla pytań typu list) — **pole opcjonalne**,

  – pointsArray – tablica z wartościami numerycznymi, odpowiada liczbie punktów przyznawanej za poprawne odpowiedzi na pytanie (tylko dla pytań typu checkBox i grid), przykładowo wartość $[0, 1, 3]$ wskazuje, że jeśli nie zostanie wybrana żadna dobra odpowiedź, zostanie przyznane 0 punktów, jeśli zostanie udzielona jedna dobra odpowiedź, zostanie przydzielony 1 punkt i jeśli zostaną udzielone dwie dobre odpowiedzi, zostaną przydzielone 3 punkty — **pole opcjonalne**.

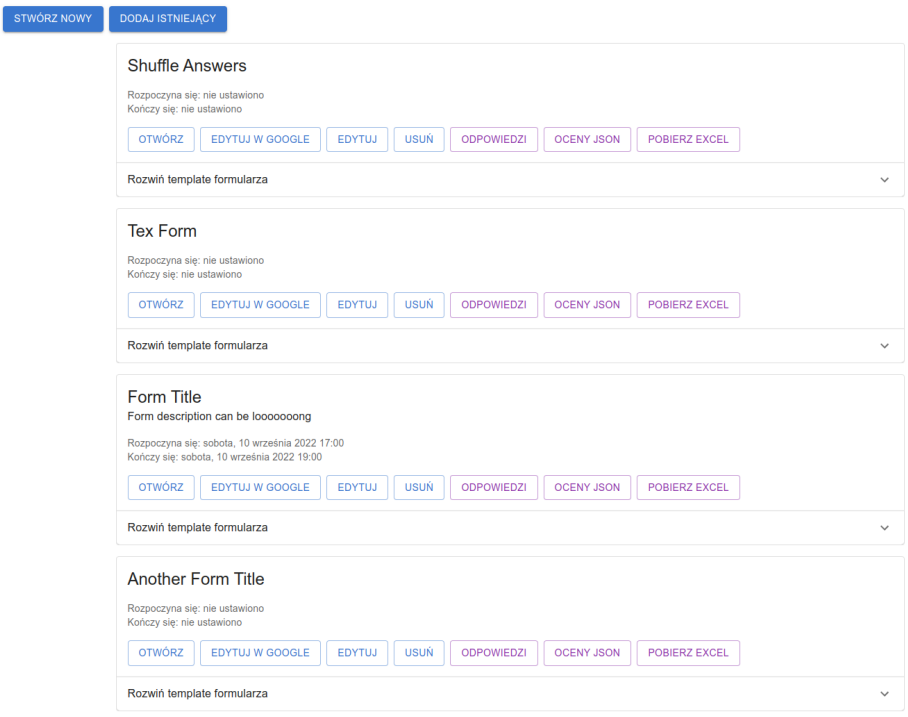
### 4.3.2. Automatyczne ocenianie

Pytania typu list, checkBox i grid mogą zostać automatycznie ocenione i wyniki mogą zostać pobrane w formacie JSON lub pliku Excel. Jeśli chcemy, żeby odpowiedzi zostały ocenione musimy ręcznie ustawić zbieranie adresów email w ustawieniach formularza. Google Forms API niestety na ten moment nie udostępnia możliwości zrobienia tego automatycznie. Sposób oceniania zależy od typu pytania:

- **list**: jeśli zostanie zaznaczona poprawna odpowiedź, odpowiadający uzyska całość punktów z pola `points`, w przeciwnym przypadku otrzyma 0 punktów,
- **checkBox**: zliczane są poprawnie zaznaczone odpowiedzi i od tej liczby odejmuje się liczbę niepoprawnie zaznaczonych odpowiedzi. Jeśli uzyskana liczba jest mniejsza od zera, podstawiamy za nią 0. Następnie z tablicy `pointsArray` z indeksu równego poprzednio uzyskanej liczbie odczytuje się liczbę uzyskanych punktów,
- **grid**: zliczane są poprawnie udzielone odpowiedzi i z indeksu równego tej liczbie odczytuje się liczbę uzyskanych punktów w tablicy **pointsArray**.

### 4.3.3. Obsługa narzędzia

Po uruchomieniu użytkownik widzi stronę w przeglądarce, jak na zdjęciu poniżej:




Rysunek 4.1: Interfejs aplikacji

Przycisk „Stwórz nowy" otworzy dialog, w którym należy wprowadzić kodowanie formularza i ewentualnie czas rozpoczęcia i zakończenia testu. Po kliknięciu „Stwórz" na dysku Google zostanie stworzony nowy formularz i wpis o nim pojawi się na liście na głównej stronie aplikacji. Przycisk „Dodaj istniejący" otworzy dialog, gdzie możemy podać identyfikator istniejącego już formularza, który chcemy dodać do listy formularzy zarządzanych przez aplikację. Musimy również podać jego kodowanie. Po kliknięciu „Dodaj" formularz zostanie edytowany. Na liście formularzy każdy wpis odpowiada jednemu formularzowi na dysku Google użytkownika, jednak nie wszystkie formularze na dysku się tu pojawią, jedynie te stworzone przez aplikację lub dodane używając przycisku „Dodaj istniejący". W każdym wpisie możemy znaleźć przyciski:

- „Otwórz": otwiera formularz na stronie Google w nowej karcie,
- „Edytuj w Google": otwiera stronę edycji formularza na stronie Google w nowej karcie,
- „Edytuj": otwiera dialog, gdzie można edytować kodowanie formularza,
- „Usuń": usuwa formularz z listy formularzy zarządzanych przez aplikację i usuwa całą jego zawartość, jednak pusty formularz tylko z tytułem pozostanie na dysku Google użytkownika,
- „Odpowiedzi": pokazuje wszystkie przesłane odpowiedzi, łącznie z tymi przesłanymi przed początkiem lub po końcu ustawionych w kodowaniu formularza,
- „Oceny JSON": wyświetla ocenione odpowiedzi bez pytań otwartych w formacie JSON,
- „Pobierz Excel": pobiera plik Excel z ocenionymi zamkniętymi pytaniami,
- „Rozwiń template formularza": podgląd kodowania formularza,

### 4.3.4. Przeprowadzanie egzaminów

Jeśli narzędzie ma być użyte do przeprowadzenia egzaminu na początku należy stworzyć z jego pomocą pusty formularz tylko z tytułem, ewentualnie krótką informacją o czasie startu. Przed rozpoczęciem należy ręcznie włączyć w ustawieniach formularza na stronie Google zbieranie adresów email i ewentualnie tasowanie pytań. Następnie w momencie rozpoczęcia egzaminu należy go zaktualizować dodając do niego pytania. Po czasie zakończenia egzaminu można go ręcznie zamknąć, ale jeśli ma zostać użyte automatyczne ocenianie, nie wolno zmieniać jego kodowania w aplikacji. Narzędzie przefiltruje przesłane odpowiedzi i automatycznie je oceni po kliknięciu odpowiedniego przycisku.

# Rozdział 5.

# Podsumowanie

Celem pracy był rozwój narzędzia służącego do zarządzania formularzami Google i pomagającego przeprowadzać testy oraz ankiety, używając tych formualarzy. Udało się dopisać kilka nowych funkcjonalności i ulepszyć te już istniejące, ale wciąż istnieje wiele ulepszeń, które można dodaći funkcjonalności, które można ulepszyć. W dalszej częsci podsumowania opiszę kilka z nich.

### Automatyczne rozpoczynanie testów w podanym terminie

W tej wersji narzędzia użytkownik musi ręcznie utworzyć formularz lub go zaktualizować dokładnie w tym momencie, w którym chciałby, żeby inni użytkownicy mogli zobaczyć dany formularz Google i wysyłać do niego odpowiedzi. Z pewnością wygodne byłoby dodanie możliwości podania daty, kiedy formularz ma zostać utworzony lub zaktualizowany.

### Automatyczne kończenie testów w podanym terminie

Podobnie wygodne byłoby kończenie testów w podanym terminie. Użytkownik musi ręcznie kliknąć, że nie chce akceptować odpowiedzi w ustawieniach formularza na stronie Google, kiedy chciałby zakończyć test, ale w przszłości, kiedy Google Forms API doda taką możliwość, można to zaprogramować.

### Zapamiętywanie poprzednich wersji formularzy

Jeśli użytkownik chce automatycznie oceniać przesłane odpowiedzi, nie może on edytować danego formularza po rozpoczęciu testu, ponieważ nawet pytania, które już istnieją dostałyby nowe identyfikatory w bazie danych i niemożliwa byłaby identyfikacja pytań w poprzednio przesłanych odpowiedziach. Ten problem można rozwiązać poprzez zapisywanie w bazie danych poprzednich wersji formularza razem z datami

edycji. Baza danych Google pamięta wszystkie przesłane odpowiedzi, nawet te odpowiadające na już usunięte pytania, więc wystarczyłaby zmiana modelu w lokalnej bazie danych. Wtedy można by też po zakończeniu testu usunąć pytania i tylko wyświetlać informację, że test się już zakończył i automatyczne ocenianie wciąż by działało.

### Automatyczne włączenie opcji losowej kolejności pytań i zbierania adresów email

Google Forms API wciąż nie oferuje wielu przydatnych funkcji takich jak włączenie losowej kolejności pytań w formularzu i zbierania adresów email. Jednak wciąż jest bardzo dynamicznie rozwijane, więc możliwe, że taka opcja pojawi się w przyszłości. Można by wtedy dodać te dwie rzeczy do narzędzia.

### Usuwanie formularzy z dysku Google

Na ten moment po usunięciu formularza w narzędziu, plik z formularzem Google nie jest usuwany z dysku Google użytkownika. Jest to spowodowane tym, że Google Forms API nie udostępnia takiej opcji, ale za to Google Drive API już to robi. Trzeba by więc dodać Google Drive API do projektu i zaimplementować usuwanie podanego pliku.