



Narzędzia i Środowiska Programistyczne

Projekt:
gra „Milionerzy”

Przygotowali:

- Rafał Gębica
- Kamil Marnik
- Jakub Warchoł

Prowadzący: mgr inż. Tomasz Gądek

Tarnów 2019

1. Wykorzystane narzędzia

W celu zaimplementowania serwera dla powyższego projektu, wykorzystano język programowania *Java* wraz z frameworkiem *Spring*. Natomiast przy tworzeniu aplikacji klienckiej, wykorzystano również język *Java*, a dla graficznego interfejsu – *JavaFX*.

W obu przypadkach, wykorzystano *IDE: IntelliJ IDEA*. Do współpracy wykorzystany został system hostingowy *bitbucket*. Do testów poprawności działania serwera, skorzystano z aplikacji *Postman*.

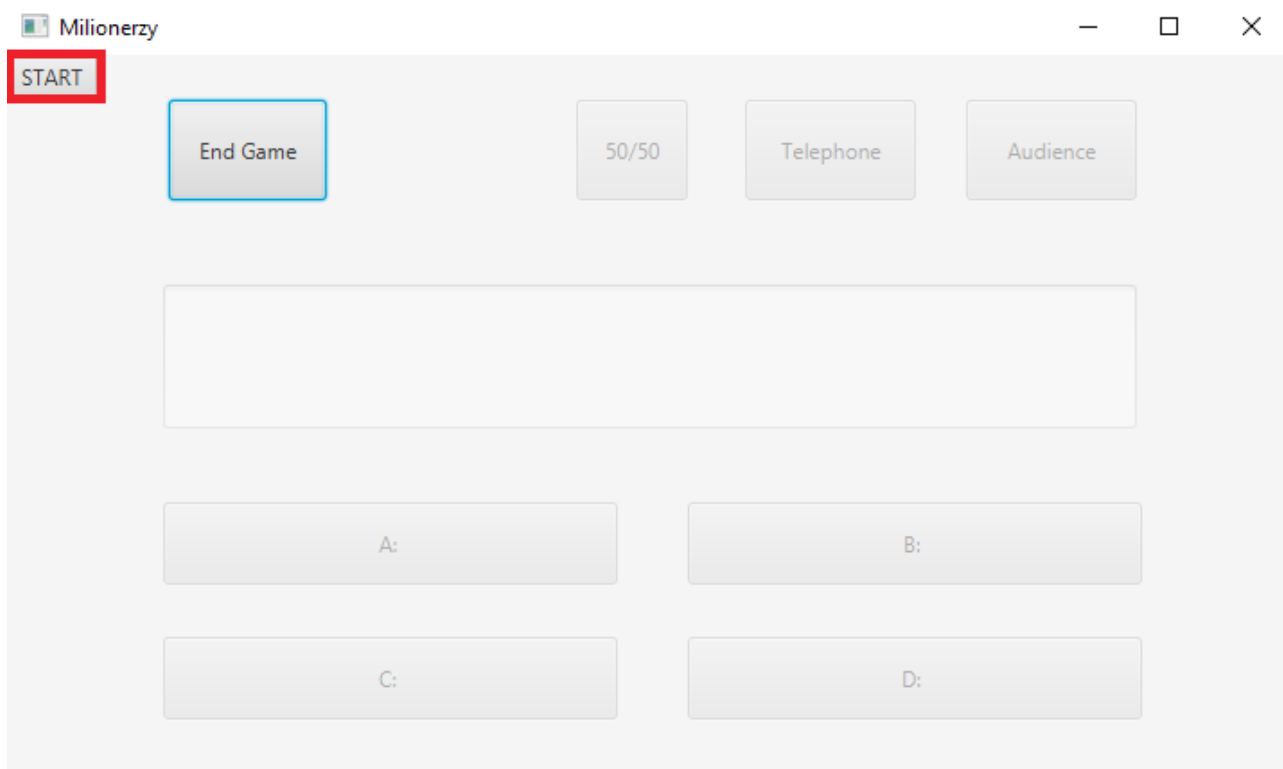
2. Opis funkcjonalny klienta

UWAGA:

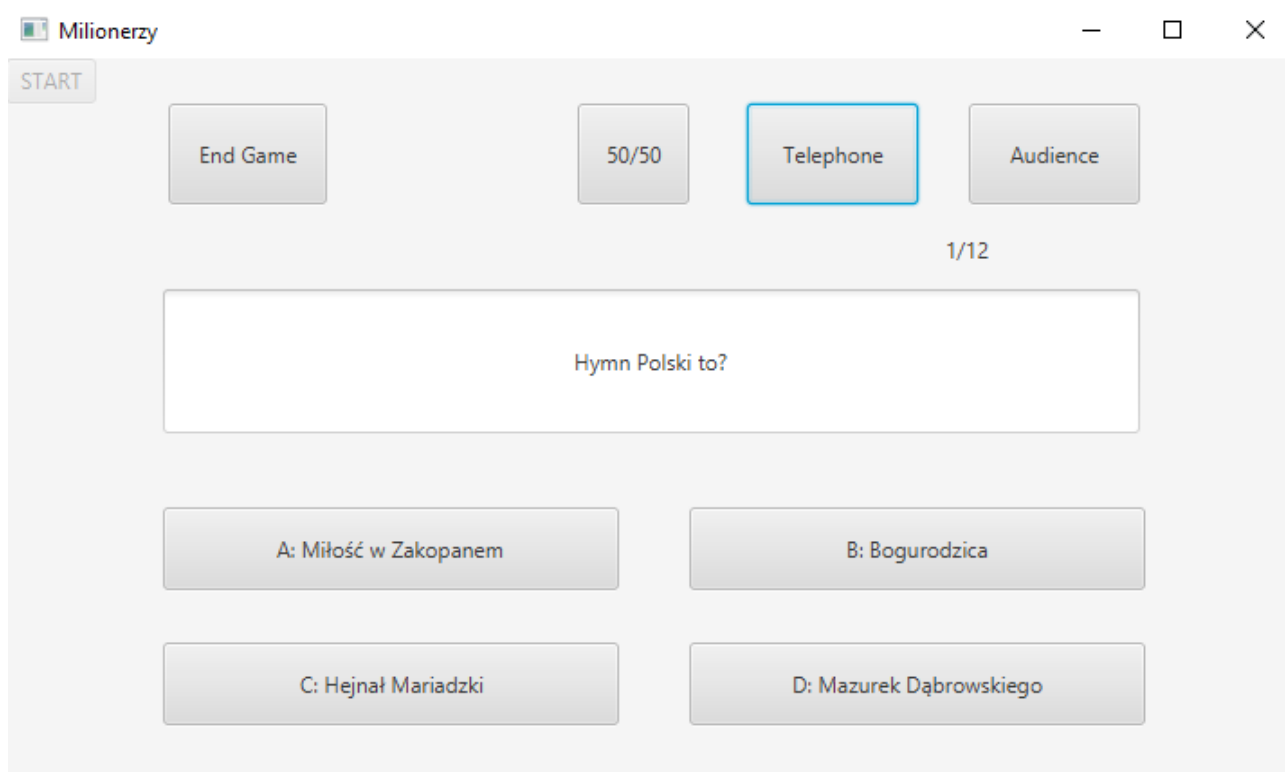
W poniższym opisie, zakłada się, że Czytelnik (dalej nazywany „Graczem”) ma dostęp do serwera, który to został przez zespół zaimplementowany.

2.1 Zaczynamy!

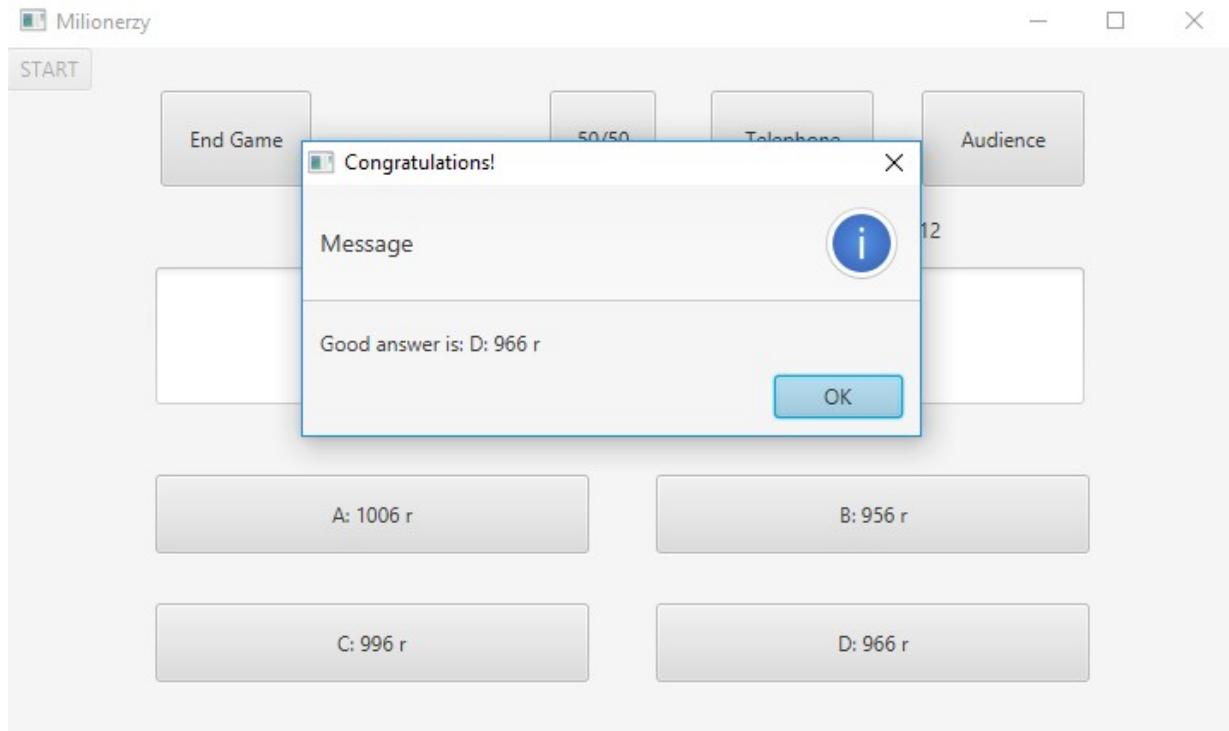
Po uruchomieniu aplikacji, należy kliknąć w przycisk *START*:



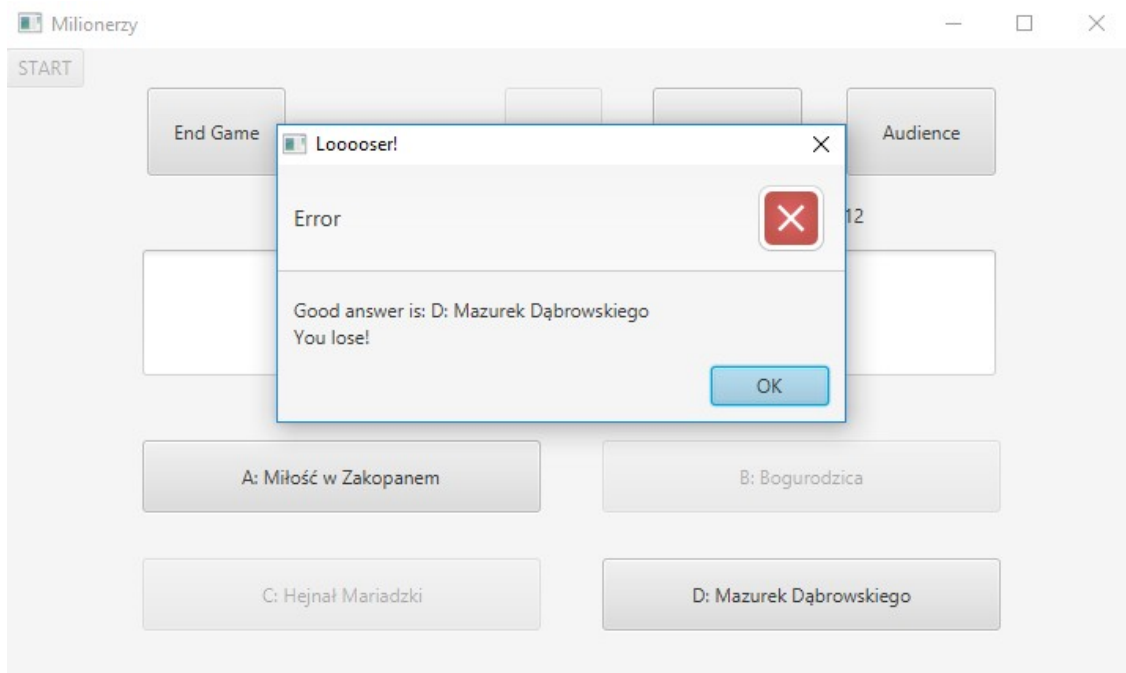
I, jeżeli nie nastąpi błąd odczytu danych z serwera (*patrz: 2.5*), zostanie wyświetlone pierwsze pytanie:



Po udzieleniu poprawnej odpowiedzi, klient wyświetli komunikat i przejdzie do następnego pytania:



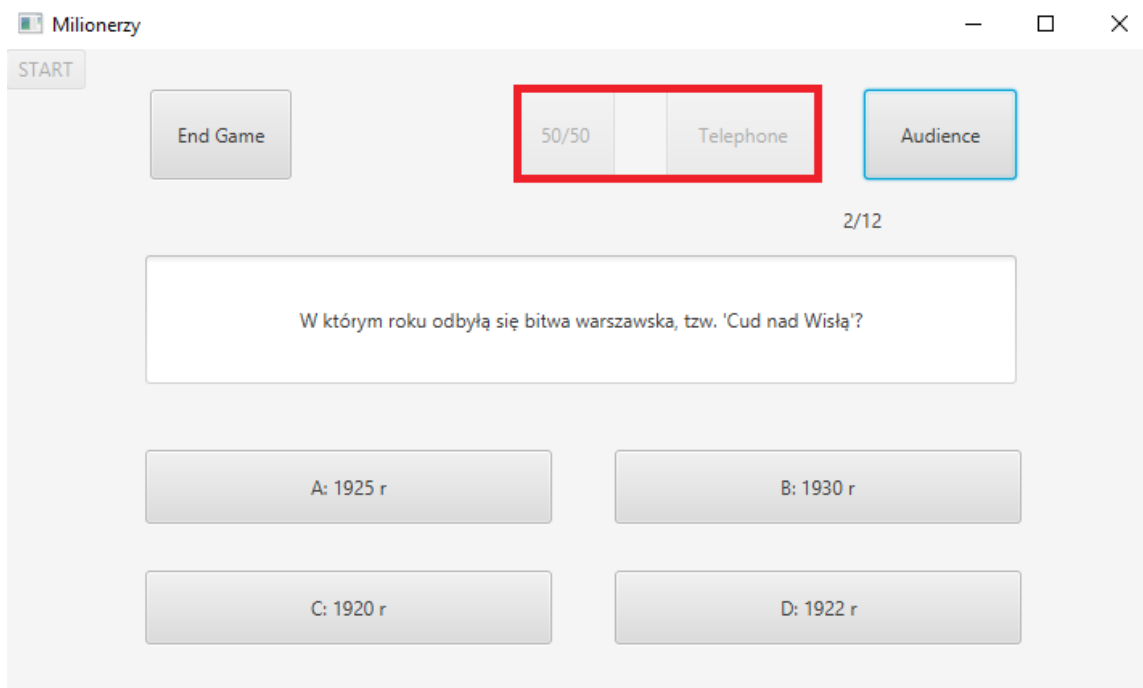
Natomiast, gdy odpowiedź okaże się być błędna:



Po naciśnięciu przycisku *OK*, Gracz powraca do ekranu startowego.

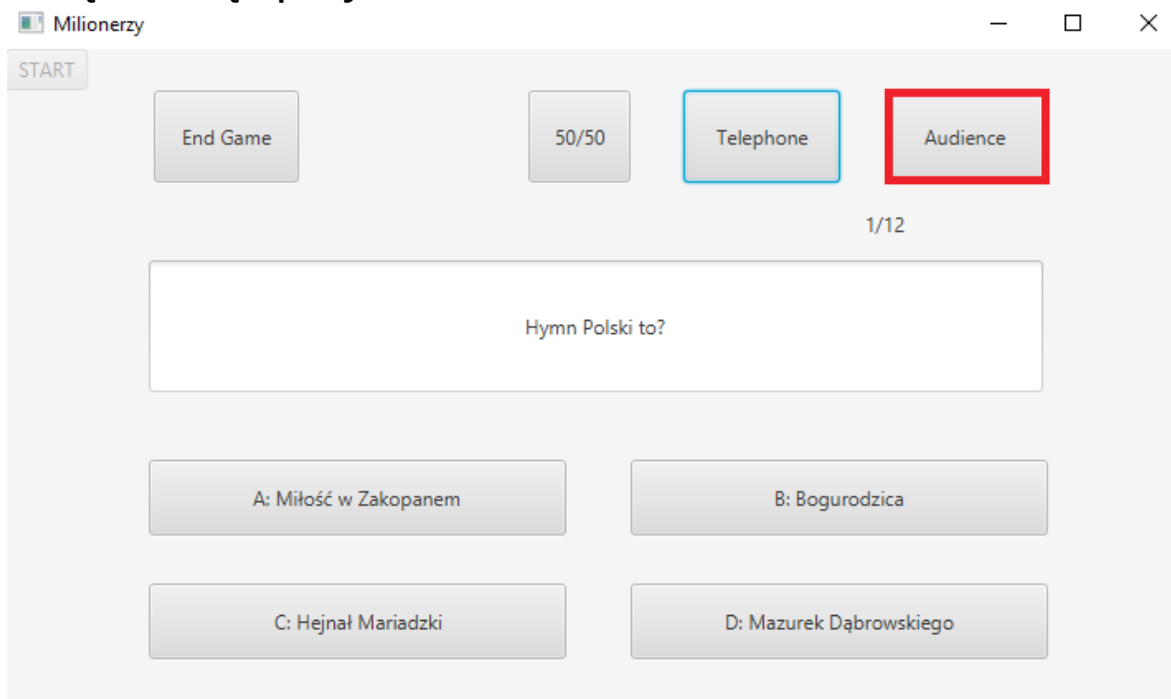
2.2 Koła ratunkowe

Na powyższych screenach zauważyć można przyciski, będące *kołami ratunkowymi* dla Gracza:

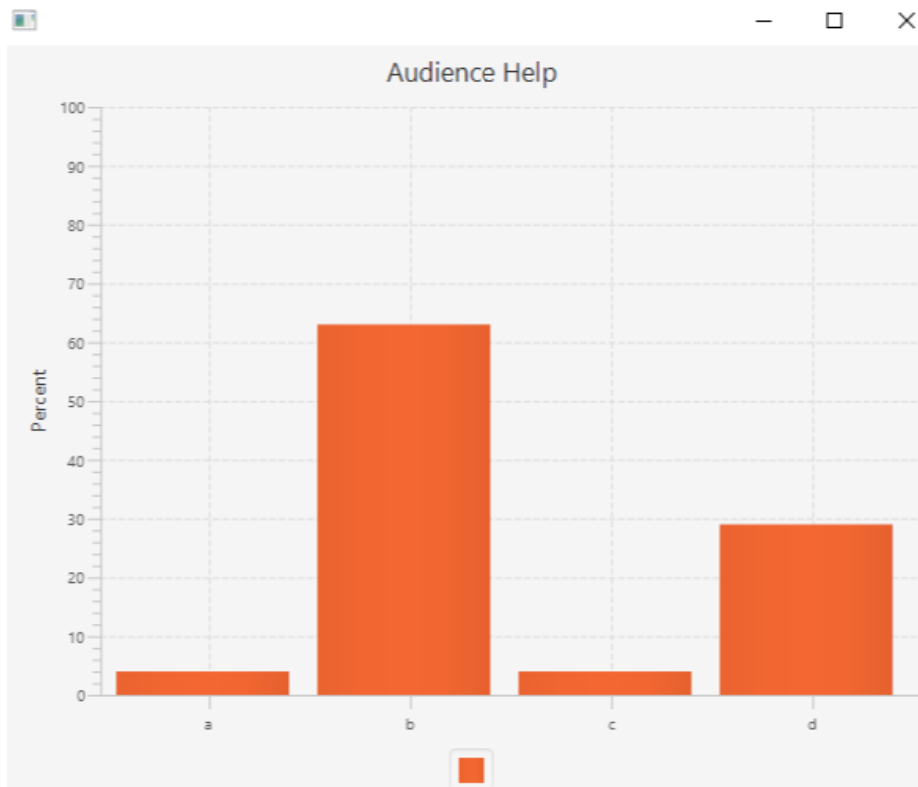


2.2.1 Pytanie do publiczności (Audience)

Proszę kliknąć przycisk *Audience*:



Po kliknięciu na powyższy przycisk, Gracz uzyska wykres z odpowiedziami *publiczności* programu:

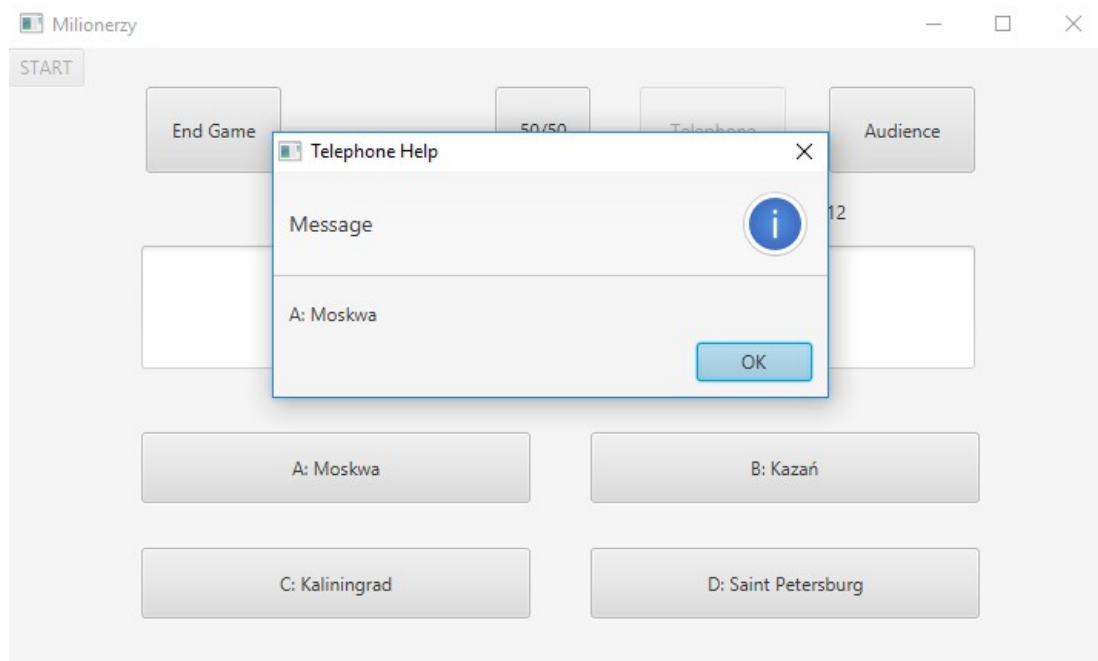


2.2.2 Telefon do przyjaciela (*Telephone*)

Proszę kliknąć przycisk *Telephone*:

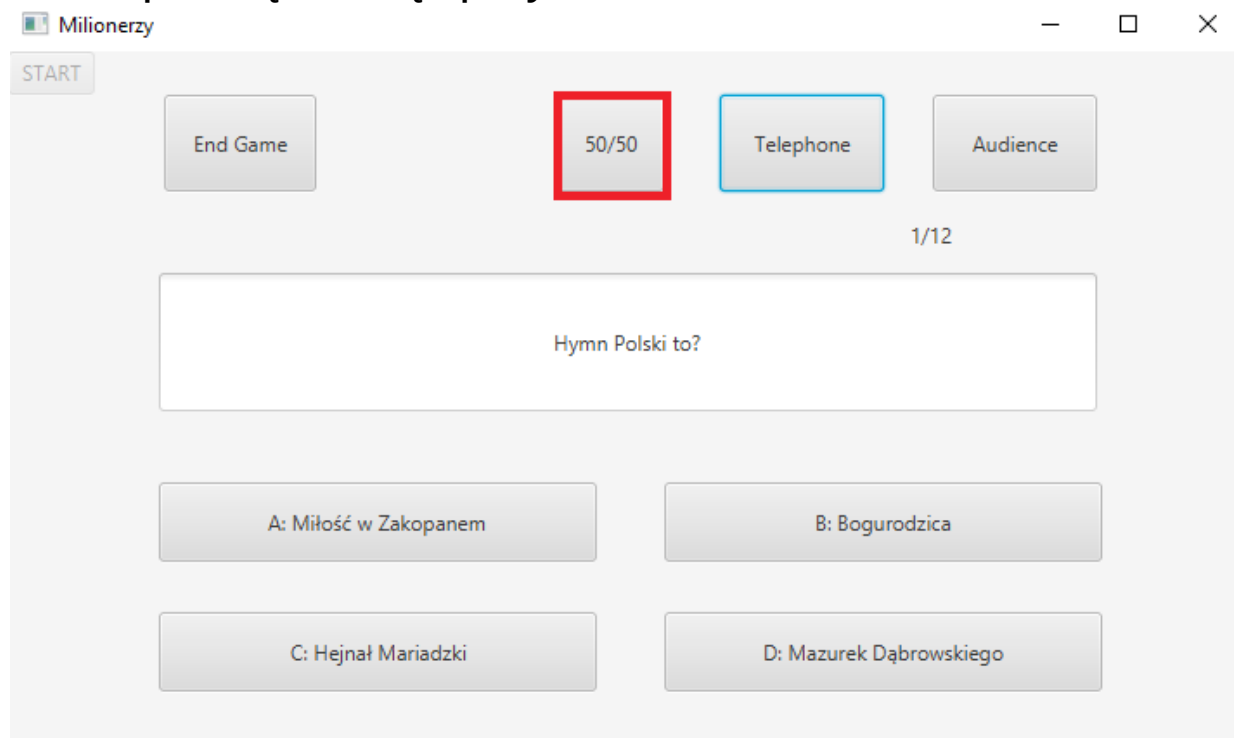
A screenshot of the "Milionery" game interface. At the top, there is a "START" button and four main buttons: "End Game", "50/50", "Telephone" (highlighted with a red box), and "Audience". Below these buttons, the question "Hymn Polski to?" is displayed. At the bottom, there are four answer buttons: "A: Miłość w Zakopanem", "B: Bogurodzica", "C: Hejnał Mariadzki", and "D: Mazurek Dąbrowskiego". The interface also shows a progress indicator "1/12" and standard window controls.

Po kliknięciu, wyświetlona zostanie odpowiedź, którą wybrałby *przyjaciel* Gracza:

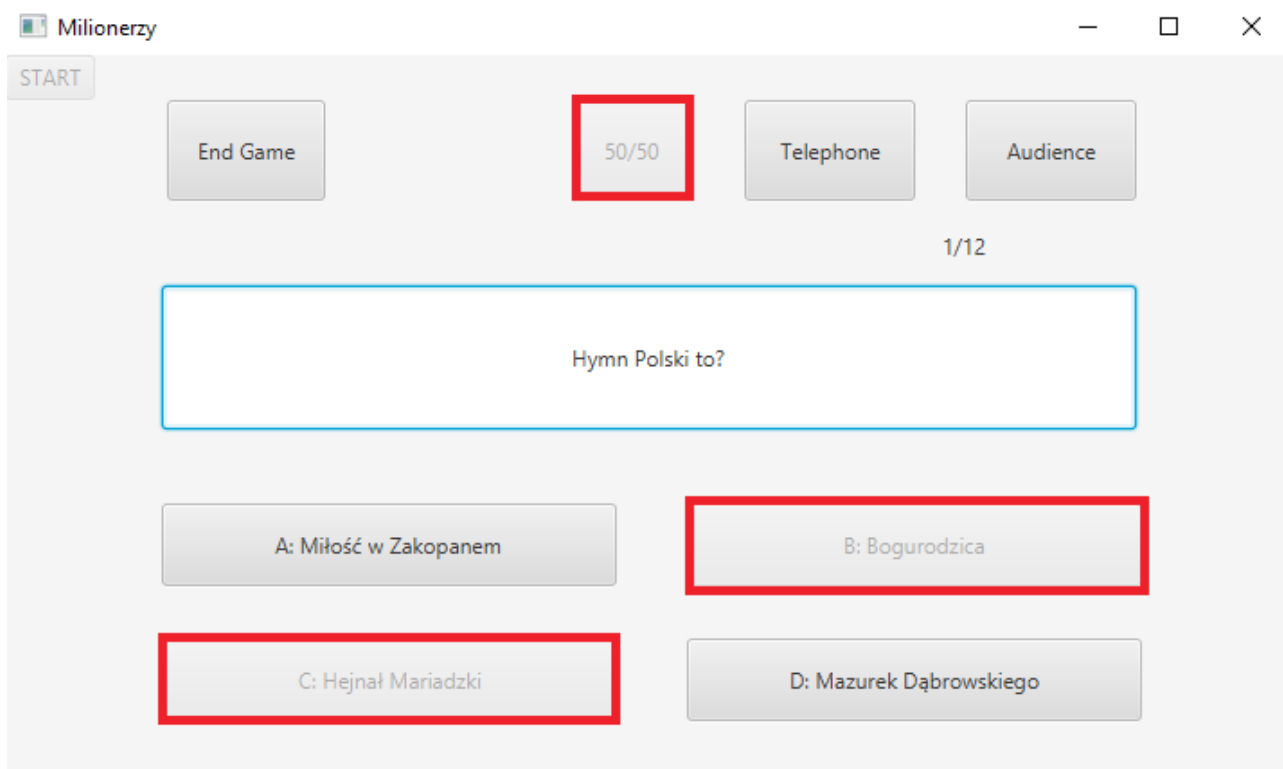


2.2.3 Pięćdziesiąt na pięćdziesiąt (50/50)

Teraz proszę kliknąć przycisk 50/50:

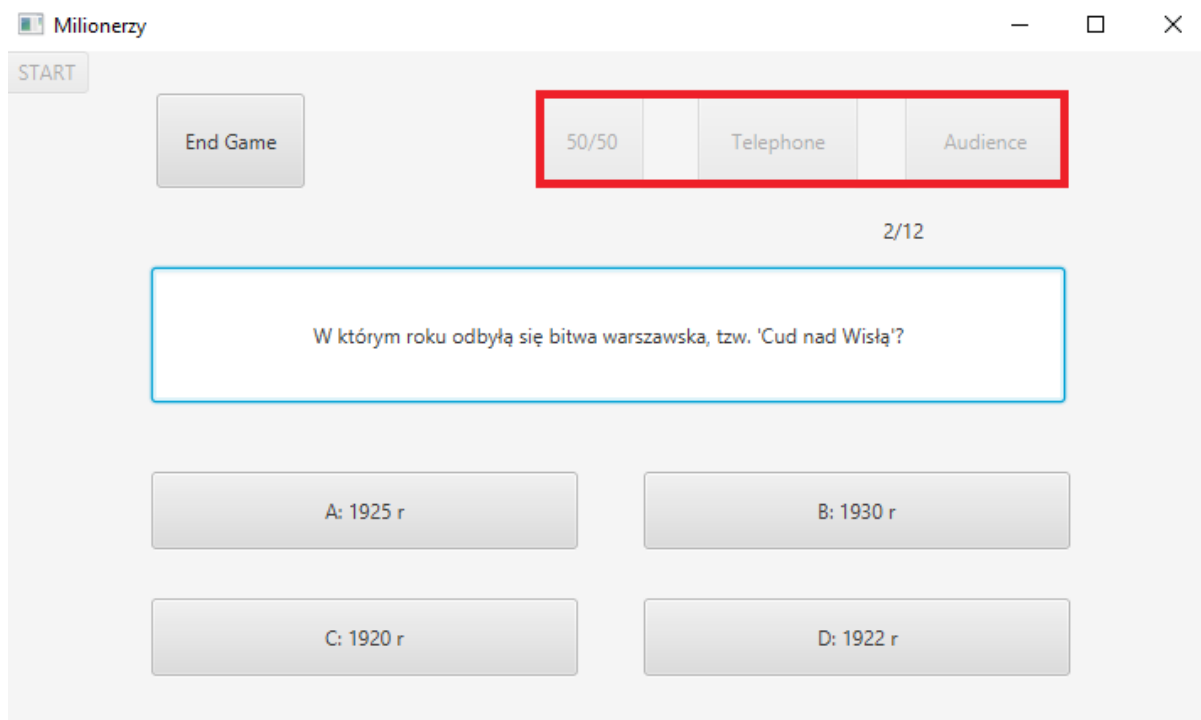


Aplikacja odrzuci losowo dwie błędne odpowiedzi:



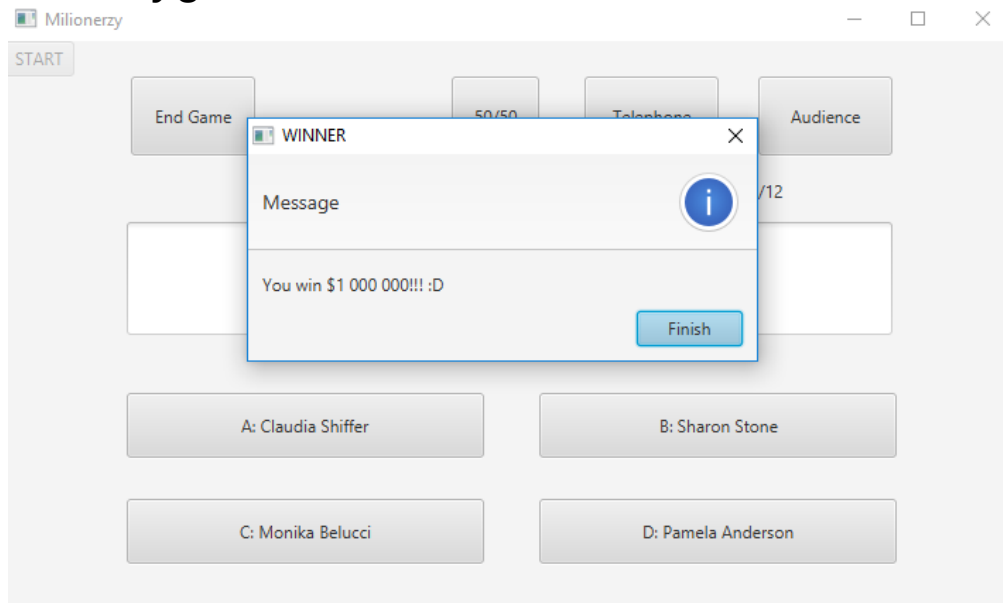
UWAGA 2:

Niestety, ale każdego z powyższych kół można użyć tylko raz:



2.3 Wygrana

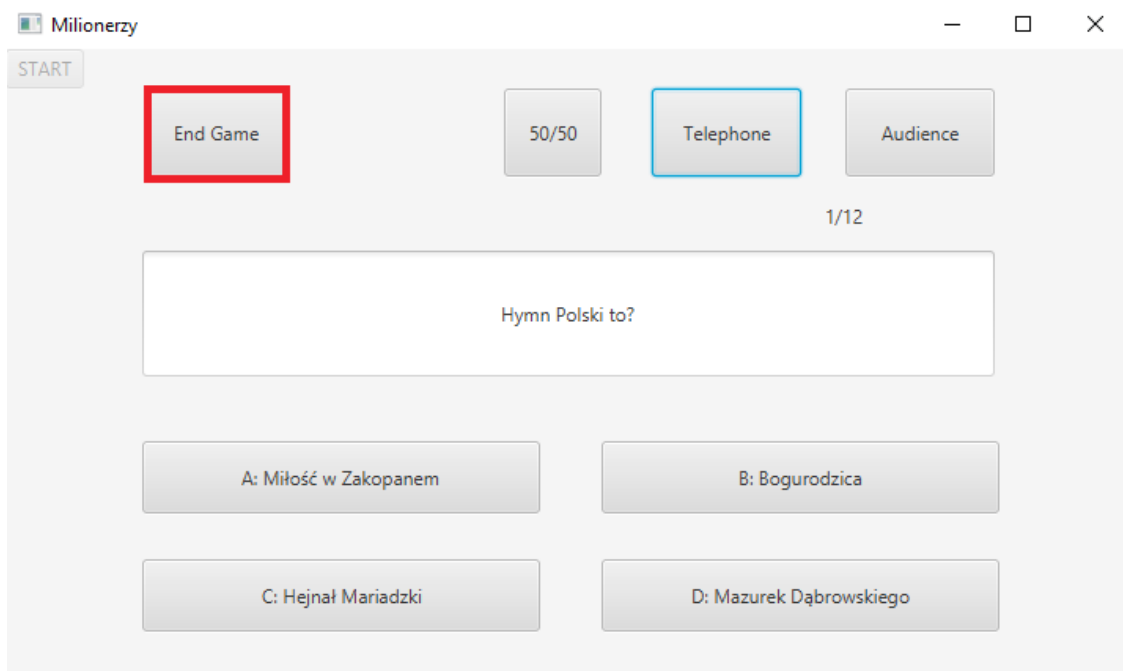
Gdy gracz dojdzie do pytania 12 i odpowie na nie poprawnie, wygra *wirtualne* 1 000 000 \$:



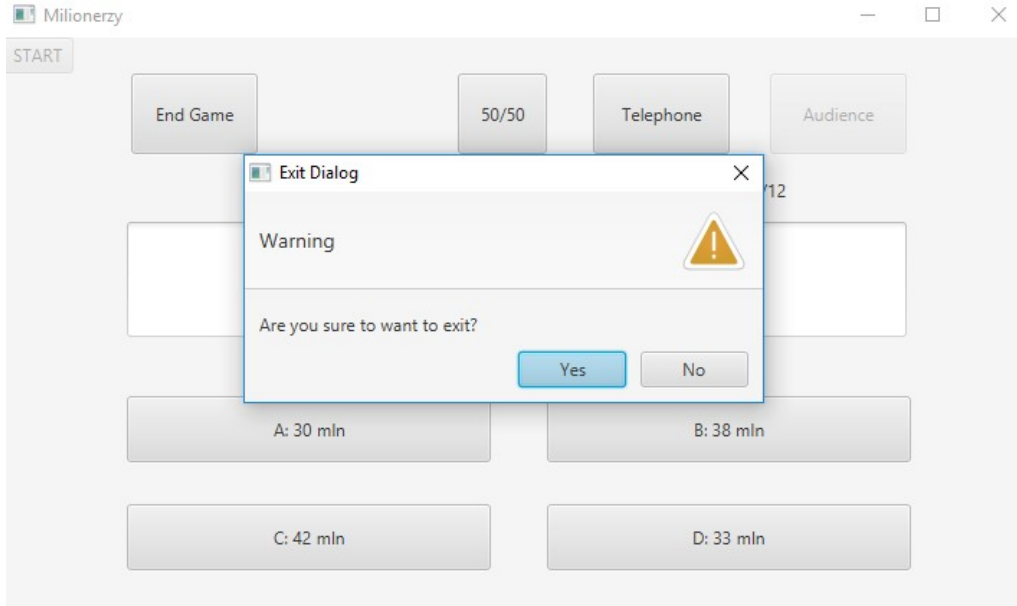
Po kliknięciu na *OK*, następuje powrót do ekranu startowego.

2.4 Wyjście z gry

Możliwe jest także wyjście z gry w trakcie jej trwania – należy kliknąć na przycisk *End Game*:



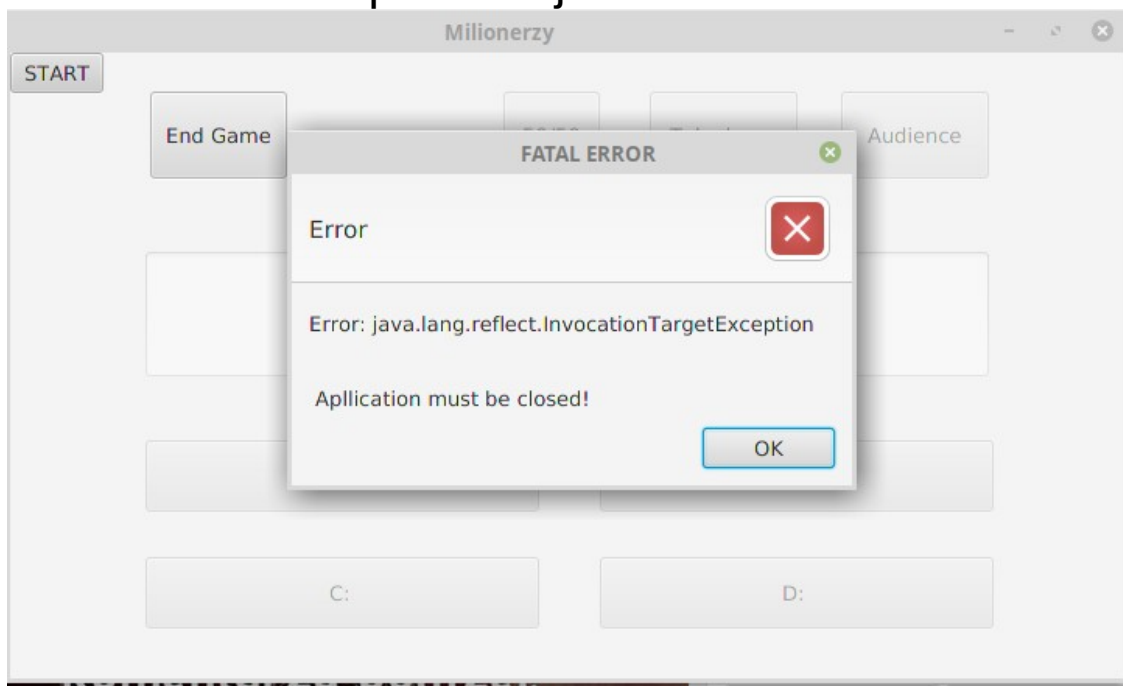
A następnie potwierdzić to przyciskiem Yes:



Kliknięcie przycisku No spowoduje powrót do aktualnego pytania.

2.5 Sytuacje wyjątkowe

W przypadku wystąpienia sytuacji wyjątkowej(brak dostępu do serwera, brak pliku z pytaniem, brak pytania etc.), zostanie rzucony wyjątek, a Gracz zobaczy na swoim ekranie komunikat o podobnej treści:



Kliknięcie przycisku *OK* spowoduje wyjście z programu z kodem błędu 6 (odpowiednik **SIGABRT** w **nixach*).

Szczegóły na temat sytuacji wyjątkowych z systemie, opisane są w rozdziale następnym.

3. Opis funkcjonalny serwera

3.1 Ogólny wgląd na pracę serwera

Swoistym “sercem” serwera jest klasa kontrolera, która odpowiada za kontrolę przepływu danych oraz najważniejsze funkcje, to jest wywoływanie odpowiednich metod oraz wysłanie JSON’a.

Ścieżką dla naszego serwera będzie: <http://127.0.0.1:8080/api/milion/question/{id}>, gdzie {id} jest zależne od numeru pytania, które ma być w danym momencie wyświetlane w aplikacji Milionerzy. Wartość *id* jest pobierana z adresu, a następnie dokonuje się jej sprawdzenia. Poprzez metodę *isIdCorrect* z klasy *QandAServiceImpl* otrzymujemy wczesną informację na temat poprawności wprowadzonego *id*. W przypadku niezgodności ze specyfikacją rzucony jest błąd *ResourceNotFound*.

```

@RequestMapping(value = "/milion/question/{id}", method = RequestMethod.GET)
public ResponseEntity<?> sendQAndA(@PathVariable("id") Long id) throws ResourceNotFoundException {

    if(!qandAService.isIdCorrect(id)) throw new ResourceNotFoundException();

    setUTF();

    QuestionDto questionDto = new QuestionDto();
    qandAService.readQandA(id, questionDto);

    LOGGER.info("Question id: {}", id);
    LOGGER.info("Question: {}", questionDto.getQuestion());
    LOGGER.info("Nr of correct answer: {}", questionDto.getCorrectAnswer());

    return qandAService.isEmpty(questionDto) ?
        new ResponseEntity<>(new ErrorDto( errorMessage: "File doesn't contain correct/any content!"), HttpStatus.NOT_FOUND) :
        new ResponseEntity<>(questionDto, HttpStatus.OK);
}

```

W kontrolerze dbamy również, aby JSON miał możliwość wyświetlania polskich znaków (metoda `setUTF` umożliwia korzystanie z systemu kodowania Unicode UTF-8).

Wywołuje się główną metodę serwisu odpowiedzialną za odczytywanie pytań oraz odpowiedzi, a na końcu zwraca się:

- JSON'a wraz z wartościami:

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:8080/api/milion/question/1`
- Method:** GET
- Status:** 200 OK
- Time:** 226 ms
- Body (JSON):**

```

{
  "question": "Java to?",
  "answer1": "kawa",
  "answer2": "język programowania",
  "answer3": "kot",
  "answer4": "pies",
  "correctAnswer": 2
}

```

- Wyjątek mówiący o niekompletności danych wraz z kodem błędu 404 Not Found:

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:8080/api/milion/question/4`
- Method:** GET
- Status:** 404 Not Found
- Time:** 82 ms
- Body (JSON):**

```

{
  "errorMessage": "File doesn't contain correct/any content!"
}

```

3.2 Cykl życia danych

Klasa *QuestionDto* służy jako klasa przechowująca zmienne, które są przesyłane przez cały “cykl” Springa, poprzez *QuestionDao*, czyli pierwotny dostęp do “bazy danych”, to jest klasy Dto. Otrzymujemy ilość pytań w pliku, a także numer pytania. Następnie dane przesyłane są do serwisu *QandAServiceImpl*, gdzie dokonywane jest odczytanie danych z plików txt, w których przetwarzamy pytania wraz z odpowiedziami.

```
@Override
public void readQandA(Long questionId, QuestionDto questionDto) {
    String line;

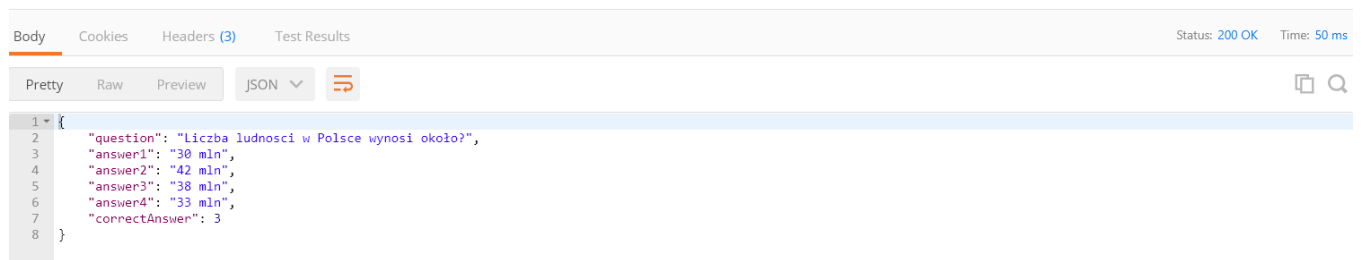
    try {
        int randNrQuestion = questionDAO.randQuestionNr(questionId);

        BufferedReader reader = new BufferedReader(new FileReader("file:" + Long.toString(questionId) + ".txt"));
        try {
            while ((line = reader.readLine()) != null) {
                if (line.startsWith(Integer.toString(randNrQuestion))) {
                    setVariables(questionId, line, questionDto);
                }
            }
        } finally {
            reader.close();
        }
    } catch (FileNotFoundException ex) {
        LOGGER.error("File not found!");
    } catch (IOException ex) {
        LOGGER.error("I can not read this file!");
    } catch (Exception ex) {
        LOGGER.error("File is empty!");
    }
}
```

Kolejna metoda *setVariables* ustawia te dane przygotowując je do wysłania poprzez JSON'a. Informacje przechowywane są w 12 plikach txt, numerowo odpowiadając kolejnym pytaniom, przechodząc przez repozytorium i serwis, są zapisywane, a na końcu wysyłane za pomocą kontrolera.

3.3.1 Użycie aplikacji Postman

Aplikacja pozwoliła nam na testowanie żądań i zobaczenie wartości przesyłanych w JSON'ie. Poprawne pytanie i odpowiedzi:



```
1 {
2   "question": "Liczba ludności w Polsce wynosi około?",
3   "answer1": "30 mln",
4   "answer2": "42 mln",
5   "answer3": "38 mln",
6   "answer4": "33 mln",
7   "correctAnswer": 3
8 }
```

3.3.2 Błędy i wyjątki

Serwer odpowiednio reaguje na wszelakie błędy znalezione podczas zapisu czy wprowadzenia adresu. W rozdziale 3.1 pokazana została główna obsługa w kontrolerze “łapiąca” inne wyjątki, między innymi przypadek błędnego przypadku zapisu do klasy *QuestionDto* (niekompletność danych), a także sprawdzenie, na samym początku, czy w ogóle wprowadzone zostało poprawne *id* w adresie oraz wyjątków odnośnie systemu kodowania Unicode (UTF-8). Sprawdzane są również błędy wynikłe podczas odczytu pliku:

```
}
} catch (FileNotFoundException ex) {
    LOGGER.error("File not found!");
} catch (IOException ex) {
    LOGGER.error("I can not read this file!");
} catch (Exception ex) {
    LOGGER.error("File is empty!");
}
```

Wynik przykładowego błędu wraz z kodem odpowiedzi serwera 404 Not Found i wiadomością opisującą przyczynę w aplikacji Postman:

```
Body Cookies Headers (3) Test Results Status: 404 Not Found Time: 47 ms

Pretty Raw Preview JSON

1 {
2   "timestamp": "2019-01-13T11:02:42.297+0000",
3   "status": 404,
4   "error": "Not Found",
5   "message": "Path is wrong- can not resolve that id",
6   "trace": "com.pwsz.project26_server.domain.dto.ResourceNotFoundException\r\n\tat com.pwsz.project26_server.rest.MilionApiController.sendQAndA(MilionApiController.java:41)\r\n\tat sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)\r\n\tat sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)\r\n\tat sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\r\n\tat java.lang.reflect.Method.invoke(Method.java:498)\r\n\tat org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:189)\r\n\tat org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:138)\r\n\tat org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:102)\r\n\tat org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:895)\r\n\tat org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:800)\r\n\tat org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87)"
```

4. Podział obowiązków w zespole

Rafał Gębica	Kamil Marnik	Jakub Warchoł
<ul style="list-style-type: none">• Utworzenie layoutu aplikacji klienckiej• Implementacja funkcjonalności „kół ratunkowych”• Generowanie wykresu odpowiedzi „publiczności”• Testowanie działania systemu• Przygotowanie dokumentacji klienta	<ul style="list-style-type: none">• Implementacja funkcjonalności serwera• Przygotowanie pytań• Testowanie usługi za pomocą POSTMAN’a• Przygotowanie loggerów do testów działania serwera• Obsługa wyjątków• Opracowanie dokumentacji dla serwera	<ul style="list-style-type: none">• Utworzenie repozytoriów• Przygotowanie wstępnego wyglądu aplikacji• Realizacja przejść do następnego pytania• Sprawdzenie poprawności odpowiedzi oraz czy Gracz nie wygrał• Implementacja algorytmu pobierania JSONa z serwera• Pomoc przy opracowaniu dokumentacji