AKADEMIA NAUK STOSOWANYCH W TARNOWIE

Wydział Politechniczny

Kierunek: Informatyka

Specjalność: Informatyka stosowana

2022/2023

Patryk Zaucha

PRACA INŻYNIERSKA

Aplikacja do zarządzania zawodnikami międzynarodowej federacji wspinaczki sportowej (IFSC)

Promotor pracy
mgr inż. Tomasz Gądek

Tarnów, 2023

Spis treści

1. Wstęp

Przedmiotem niniejszej pracy dyplomowej jest aplikacja desktopowa, będąca panelem służącym do pobierania, przechowywania, oraz analizowania osiągnięć i postępów zawodników wspinaczki sportowej. W kolejnych rozdziałach przedstawione zostaną technologie wykorzystane przy jej tworzeniu, wybrane elementy implementacyjne oraz uzyskane efekty.

1.1. Cel pracy

Celem pracy jest zaprojektowanie oraz zaimplementowanie systemu, umożliwiającego przetwarzanie informacji o zawodnikach wspinaczki sportowej. Wymagana jest aplikacja desktopowa, której utrzymanie i użytkowanie nie będzie wiązało się z dodatkowymi nakładami pracy czy też finansów. Głównym zadaniem programu, będzie prezentowanie postępu poszczególnych zawodników na przestrzeni czasu - w tym celu tworzone będą odpowiednie tabele, wykresy oraz diagramy. Użytkownik będzie miał możliwość ręcznego wprowadzania własnych zawodników lub importu pliku CSV (ang. *comma separated values)* z danymi. Zaimplementowane zostanie także pobieranie już istniejących wpisów z oficjalnej strony wspinaczki sportowej. Dane aplikacji będą zapisywane w lokalnej bazie, a także eksportowane w celu przechowywania i udostępniania poza aplikacją.

1.2. Zakres pracy

W ramach pracy inżynierskiej zaimplementowany zostanie system składający się z następujących modułów:

- aplikacja desktopowa,
- lokalna baza danych.

Zrezygnowano tutaj z części serwerowej, która umożliwiłaby m.in. szybkie, bezprzewodowe udostępnianie danych, czy też ich synchronizację na różnych urządzeniach. Decyzja ta motywowana jest koniecznością maksymalnego uproszczenia eksploatacji oraz utrzymania systemu. Aplikacja desktopowa będzie miała możliwość odczytu i prezentacji danych pochodzących ze strony internetowej międzynarodowej federacji wspinaczki sportowej [1]. W tym celu wykorzystana zostanie technika ***web scrapingu***, czyli pozyskiwania z sieci danych w sposób zautomatyzowany, pomimo braku udostępnionego API.

2. Analiza biznesowa

W niniejszym rozdziale zaprezentowane zostaną założenia dotyczące funkcjonalności systemu, będącego przedmiotem pracy.

2.1. Wymagania funkcjonalne

2.1.1. Dodawanie i edycja zawodników

Podstawowym warunkiem funkcjonowania tworzonej aplikacji, jest dostęp do bazy zawodników. W tym celu, użytkownikowi zostanie udostępniona możliwość ręcznego dodawania i edytowania wpisów. Ponadto, utworzony zostanie algorytm umożliwiający odczytywanie i przetwarzanie w czasie rzeczywistym danych sportowców z oficjalnych zawodów wspinaczki sportowej. Uwzględnione zostaną informacje takie jak podstawowe dane personalne, wyniki osiągnięte w kwalifikacjach, czy poszczególnych etapach zawodów.

2.1.2. Import i eksport danych

Aplikacja będzie wspierała odczyt zapisanych wcześniej danych w formacie CSV. Takie rozwiązanie ułatwi tworzenie kopii danych na zewnętrznych nośnikach pamięci oraz ich wymienianie pomiędzy urządzeniami. Format ten jest czytelny dla człowieka, co umożliwia potencjalne zmiany bezpośrednio w pliku.

2.1.3. Analiza wyników

Najważniejszą funkcjonalnością będzie prezentowanie wszelkich informacji pozyskanych o zawodnikach. Jednym z poziomów takiej prezentacji będzie zbiorcze przedstawienie surowych danych, tj. zestawienie w postaci tabeli wszystkich albo wybranych osiągnięć sportowców. Ta najbardziej podstawowa forma prezentacji danych umożliwi poznanie dokładnych wartości liczbowych. Rekordy tabeli będzie można posortować wg dowolnego atrybutu.

Opisany sposób przedstawiania danych nie ułatwia interpretacji wyników w odniesieniu do innych zawodników. Kolejnym poziomem wizualizacji będzie naniesienie danych kilku uczestników na wykres, gdzie zostaną one bezpośrednio porównane. Możliwe będzie zestawienie konkretnych zawodników, średnich wyników przedstawicieli wybranych narodowości lub wszystkich sportowców.

Niewątpliwie przydatną dla trenera funkcją będzie także możliwość analizy postępów

dokonywanych przez zawodnika w czasie. Tutaj przydatny może być wykres z naniesionymi osiągnięciami sportowca na przestrzeni tygodni, miesięcy czy lat.

2.2. Wymagania niefunkcjonalne

2.2.1. Wydajność

Ważnym aspektem jest również wydajność aplikacji. Aplikacja będzie generowała różnorodne statystyki i wizualizacje dla poszczególnych zawodników, stąd istotne jest ich wyświetlanie w taki sposób, aby użytkownik nie był zmuszony do czekania na zakończenie wszystkich obliczeń. W tym celu zaimplementowano mechanizmy, które na bieżąco wyświetlają kolejne analizy, a także zapobiegają wielokrotnemu wykonywaniu tych samych obliczeń.

2.2.2. Przejrzysty interfejs aplikacji

Interfejs aplikacji powinien być przejrzysty oraz intuicyjny dla każdego użytkownika. Poszczególne jego elementy zostały zaprojektowane w taki sposób, aby mogły być użyte ponownie. Dzięki temu zapewniona zostanie spójność poszczególnych widoków.

3. Analiza technologiczna

W rozdziale zostaną opisane najważniejsze technologie wykorzystane do utworzenia aplikacji, będącej przedmiotem tej pracy.

3.1. Git

Git jest darmowym, rozproszonym systemem kontroli wersji. Umożliwia on śledzenie wszystkich zmian dokonywanych na pliku, a także szybki powrót do dowolnej, wcześniej zapisanej wersji. Ułatwia również jednoczesną pracę nad różnymi elementami oprogramowania, poprzez tworzenie dedykowanych gałęzi (ang. *branches*) [2].

3.2. Kotlin

Kotlin to statycznie typowany, obiektowy, międzyplatformowy język programowania. Jest w pełni kompatybilny z Javą, dzięki czemu może wykorzystywać jej liczne biblioteki. Wprowadza wiele udogodnień względem Javy, takich jak wnioskowanie o typach, funkcje wyższego rzędu, czy mechanizmy zabezpieczające przed pojawieniem się niepożądanych wartości typu ***null***. Używany przede wszystkim w produkcji aplikacji na systemy Android, znajduje zastosowanie także w innych dziedzinach, jak tworzenie stron internetowych, rozwój aplikacji serwerowych i desktopowych, czy nawet produkcja gier [3].

3.3. Gradle

Gradle służy do automatyzacji budowania, testowania i wdrażania oprogramowania. Jest oficjalnym narzędziem dla projektów na platformę Android, wspierającym takie języki jak Java, Kotlin, JavaScript, Scala. Używa języka specyficznego dla domeny (ang. *domain-specific language - DSL*) napisanego w Groovy lub Kotlinie, pozwalającego na dostosowanie logiki budowania do potrzeb programisty. Jedną z kluczowych możliwości Gradle jest zarządzanie zależnościami w projekcie, co pozwala w wygodny sposób dodawać nowe biblioteki. Ułatwia on także generowanie dokumentacji, oraz zapewnia wsparcie dla testów [4].

3.4. Compose

Compose jest zestawem narzędzi opartym o język programowania Kotlin, które pozwalają na tworzenie interfejsów użytkownika w sposób deklaratywny - definiowany jest jedynie pożądany stan interfejsu, a narzędzie zajmuje się jego realizacją. Jest to kontrast do klasycznego podejścia imperatywnego, w którym należy przygotować kod jawnie zmieniający stan interfejsu - kod taki jest bardziej podatny na występowanie błędów, a także trudniejszy w utrzymaniu. Compose może być wykorzystany do tworzenia aplikacji mobilnych, webowych i desktopowych [5].

3.5. Realm

Realm to wbudowana, obiektowa baza danych, umożliwiająca tworzenie aplikacji działających w czasie rzeczywistym i w trybie ciągłym. Została zaprojektowana w taki sposób, że nie wymaga pisania złożonych zapytań SQL. Realm zapewnia również szereg zaawansowanych funkcji, takich jak szyfrowanie i kompresja, które mogą być przydatne w zabezpieczeniu wrażliwych danych. Baza danych pozwala programistom na wykorzystanie wbudowanego wsparcia dla relacji danych i zapytań, które mogą być wykorzystane do filtrowania i sortowania danych [6].

3.6. JUnit

JUnit to zestaw narzędzi, umożliwiających testowanie oprogramowania. Służy do pisania i uruchamiania testów jednostkowych dla kodu Java, czyli małych, izolowanych testów, które sprawdzają, czy poszczególne jednostki kodu, takie jak metody czy klasy, działają poprawnie. JUnit dostarcza zestaw adnotacji i asercji, które są używane do definiowania i uruchamiania testów, a także do sprawdzania ich wyników. Testy JUnit są zazwyczaj uruchamiane automatycznie, na przykład jako część procesu ciągłej integracji, budowania, lub w zautomatyzowanym zestawie testów. Pozwala to programistom na wykrycie i naprawienie błędów we wczesnym etapie procesu rozwoju, zanim kod zostanie wdrożony do produkcji [7].

3.7. Selenium

Selenium to projekt służący do automatyzacji przeglądarek - zapewnia rozszerzenia emulujące interakcję użytkownika, pozwalającego między innymi na testowanie

oprogramowania. Napisany kod może być uruchamiany na wielu popularnych przeglądarkach. Selenium umożliwi także wyodrębnienie poszczególnych elementów strony internetowej, a następnie ich odczytywanie [8].

3.8. CRUD

CRUD (ang. ***C****reate,* ***R****ead,* ***U****pdate,* ***D****elete*) - akronim pochodzący od czterech podstawowych funkcji w aplikacjach korzystających z bazy danych [10]. Tworzony model musi być w stanie tworzyć, odczytywać, aktualizować i usuwać zasoby. Przyjmuje się, że jeśli implementowana czynność nie może być opisana przez jedną z tych czterech operacji, to powinna stanowić własny model.

4. Implementacja systemu

4.1. Zakres funkcji

W poniższym podrozdziale zostaną omówione wszystkie funkcje, spełniane przez poszczególne części systemu.

4.1.1. Pobieranie danych

Aby zapewnić dokładne i aktualne informacje o zawodnikach aplikacja pobiera dane z oficjalnej strony internetowej Międzynarodowej Federacji Wspinaczki Sportowej (IFSC). Strona ta zawiera informacje o sportowcach, takie jak imię, nazwisko, narodowość i wiek, a także wyniki zawodów, w których brali udział. Proces ekstrakcji danych odbywa się za pomocą napisanego w języku Kotlin modułu, który wykorzystuje techniki ***web scrapingu***. Odczytane dane są w odpowiedni sposób przetwarzane, w celu zapewnienia ich spójności oraz szybkiego i wygodnego wykorzystywania w procesie działania aplikacji. Pobrane informacje zostają ostatecznie zapisane w lokalnej bazie danych, w celu umożliwienia bezproblemowego dostępu do nich.

4.1.2. Prezentowanie i edycja informacji

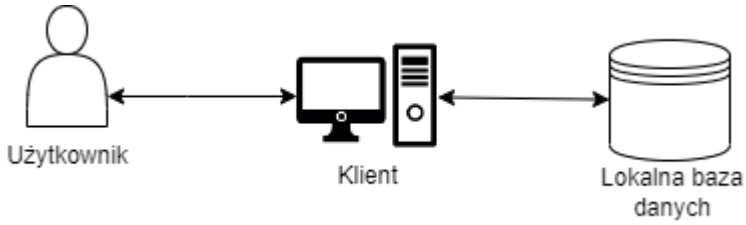
Aplikacja zapewnia przyjazny dla użytkownika interfejs do przeglądania i edycji informacji. Jedną z głównych cech programu jest możliwość wyświetlania danych zawodników w formie tabeli. Tabela wyświetla nazwiska, narodowości i wiek wspinaczy oraz pozwala użytkownikowi na sortowanie i filtrowanie danych w oparciu o dowolny z tych atrybutów. Dodatkowo, możliwe jest odfiltrowanie zawodników w taki sposób, aby widzieć tylko tych, którzy przynajmniej raz brali udział w dowolnych zawodach. Użytkownik może również ręcznie wprowadzić nowego zawodnika, dzięki czemu może on być zaprezentowany w jednej tabeli z innymi sportowcami, ułatwiając porównywanie ich wyników. Oprócz przeglądania danych użytkownik ma również możliwość ich edycji. Dzięki temu aplikacja może być na bieżąco aktualizowana o najnowsze informacje o wspinaczach. Kluczową cechą aplikacji jest możliwość przeglądania wyników wybranego wspinacza. Po wybraniu przez użytkownika rekordu w tabeli wyświetlana jest kolejna tabela, która prezentuje wszystkie wyniki wspinacza. Tabela ta zawiera informacje takie jak data zawodów, miejsce zajęte przez wspinacza oraz osiągnięte rezultaty. Ta funkcja umożliwia użytkownikom przeglądanie szczegółowej historii rozwoju zawodnika. Możliwość przeglądania osiągnięć wybranego

wspinacza w osobnej tabeli ułatwia użytkownikowi znalezienie poszukiwanych informacji, a także pozwala na porównanie ze sobą różnych wspinaczy.

4.1.3. Analiza wyników

Oprócz wyświetlania danych w formie tabeli, aplikacja zapewnia również możliwość generowania wykresów do wizualizacji i analizy wyników wspinaczy sportowych. Wykresy te pozwalają użytkownikom łatwo dostrzec wzorce i trendy w danych, ułatwiając identyfikację obszarów mocnych i słabych dla poszczególnych wspinaczy lub porównanie rezultatów w stosunku do pozostałych zawodników. Jednymi z kluczowych cech wykresów są możliwość przeglądania postępów sportowca w czasie, czy też porównania wyników różnych wspinaczy. Poprzez wygenerowanie wykresu, który pokazuje wyniki wielu zawodników w jednym miejscu, użytkownicy mogą łatwo zobaczyć, jak pod względem efektywności wspinacze prezentują się na tle konkurencji.

4.2. Architektura systemu

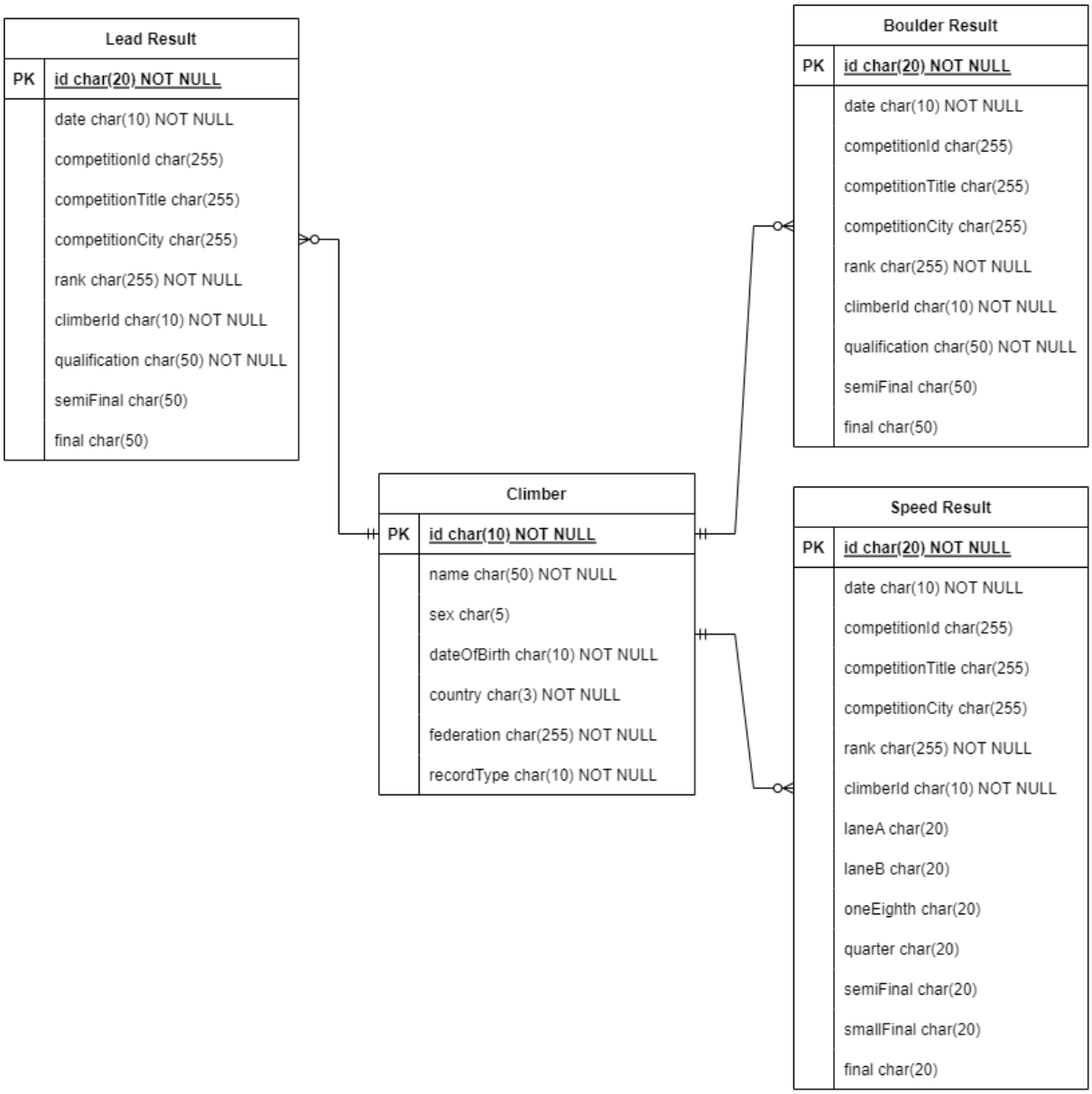


Rys. 4.1 Architektura systemu [opracowanie własne]

Prosta architektura składająca się z klienta dedykowanego na komputer osobisty i lokalnej bazy danych ma kilka zalet. Po pierwsze, eliminuje potrzebę posiadania scentralizowanego serwera i związanych z tym kosztów, takich jak jego utrzymanie, czy bezpieczeństwo. Po drugie, posiadanie lokalnej bazy danych oznacza, że dostęp do danych i ich przetwarzanie jest szybsze, ponieważ nie ma potrzeby wysyłania zapytań przez sieć. Dodatkowo dane są przechowywane na urządzeniu użytkownika, co zapewnia mu pełną kontrolę nad nimi i gwarancję ochrony przed nieautoryzowanym dostępem. Ponadto baza danych ***Realm*** oferuje bezproblemową integrację z klientem desktopowym, zmniejszając złożoność architektury i ułatwiając zarządzanie. W rezultacie ta prosta architektura zapewnia opłacalne i bezpieczne rozwiązanie do przechowywania i przetwarzania rekordów dotyczących wspinaczy i ich wyników.

4.3. Diagram ERD

ERD (ang. Entity Relationship Diagram), czyli diagram związków encji to graficzna reprezentacja struktury bazy, przedstawiająca różne encje i relacje między nimi.



Rys. 4.2 Diagram ERD [opracowanie własne]

Diagram ERD przedstawiony na rysunku 4.2 obrazuje strukturę bazy danych będącej elementem niezbędnym do funkcjonowania całego systemu.

Fundamentalną częścią opisywanej bazy danych jest tablica zawodników wspinaczki (climber). Przechowuje ona podstawowe informacje o zawodnikach, jak na przykład ich płeć, data urodzenia, czy narodowość. Wymienione dane umożliwiają podział sportowców na grupy,

czy też analizę ich osiągnięć w oparciu o wiek w chwili startu w zawodach. Każdy zawodnik posiada swój klucz identyfikacyjny - zawodnicy, o których informacje pobrane zostały ze strony internetowej IFSC, mają przypisany numer pokrywający się z tym, którym są identyfikowani także na wspomnianej stronie. Rozwiązanie takie pozwala na ich proste odszukanie w celu np. weryfikacji poprawności danych, ich aktualizację lub poszerzenie zakresu. W przypadku zawodników dodanych ręcznie przez użytkownika aplikacji klucz zawodnika generowany jest przez aplikację.

Drugą częścią bazy jaka może zostać wyodrębniona są tabele z wynikami uzyskanymi na różnych zawodach. Poza samymi wynikami w poszczególnych konkurencjach każdy rekord zawiera klucz zawodnika, do którego należy wynik, a także informacje o wydarzeniu, podczas jakiego został uzyskany (np. Mistrzostwa Świata, Mistrzostwa Europy). Klucz każdego rekordu w tabelach z wynikami generowany jest z wykorzystaniem **id** wydarzenia, oraz klucza zawodnika, co zapewnia jego niepowtarzalność.

Projekt bazy danych odzwierciedla krytyczny wymóg ustanowienia relacji pomiędzy każdym wynikiem, a odpowiadającym mu wspinaczem. Wynik musi być powiązany z dokładnie jednym wspinaczem, podczas gdy wspinacz może nie mieć żadnych wyników lub mieć ich wiele. Ten typ relacji znany jest jako relacja jeden do wielu i służy jako podstawowy aspekt architektury bazy danych. Implementacja tej relacji pozwala na przejrzyste i zorganizowane przechowywanie danych, umożliwiając tym samym efektywne wyszukiwanie informacji i analizę wyników.

4.4. Baza danych

Głównym zadaniem aplikacji jest pobieranie, przechowywanie i przetwarzanie danych o zawodnikach i ich osiągnięciach. Oczywistym jest więc zapotrzebowanie na prosty i szybki dostęp do nich, a także możliwość utrwalenia tworzonych, czy modyfikowanych rekordów. W projekcie wykorzystana została baza ***Realm***, będąca lekką i wydajną alternatywą dla ***SQLite***, znajdująca zastosowanie w produkcji aplikacji na komputery osobiste, urządzenia mobilne, czy też przeglądarki, co pozwoli na możliwie proste rozszerzenie dostępności aplikacji o wymienione platformy.

4.5. Wybrane elementy implementacyjne

4.5.1. Wyświetlanie zdjęć

W celu łatwiejszej identyfikacji konkretnego zawodnika, na jednym z ekranów aplikacji wyświetlane jest jego zdjęcie. Na listingu 4.1 przedstawiono fragment kodu, odpowiedzialny za wyświetlanie takiego zdjęcia.

```
fun loadImageOfClimber(imageUrl: String): ImageBitmap? {
    try {
        val url = URL(imageUrl)
        val connection = url.openConnection() as HttpURLConnection
        connection.connect()
        val inputStream = connection.inputStream
        val bufferedImage = ImageIO.read(inputStream)
        val stream = ByteArrayOutputStream()
        ImageIO.write(bufferedImage, "png", stream)
        val byteArray = stream.toByteArray()
        return Image.makeFromEncoded(byteArray).toComposeImageBitmap()
    } catch (e: Exception) {
        return null
    }
}
```

Listing 4.1 Przygotowanie zdjęcia do wyświetlenia [opracowanie własne]

Przedstawiona funkcja jest odpowiedzialna za przygotowanie zdjęcia zawodnika przed jego wyświetleniem. Przyjmuje ona pojedynczy parametr ***imageUrl***, reprezentujący adres URL obrazu, który ma zostać załadowany. Na początku podejmowana jest próba połączenia z danym adresem URL i w przypadku powodzenia strumień wejściowy jest odczytywany, a następnie wykorzystywany do utworzenia buforowanego obrazu. Następnie obraz jest konwertowany na tablicę bajtów, z której ostatecznie powstaje instancja klasy ***Image***. Jeżeli w trakcie generowania obrazu wystąpi błąd związany np. z brakiem internetu lub niepoprawnym adresem URL zwracany jest ***null***.

4.5.2. Pobieranie danych z sieci

Pobieranie danych w celu ich analizy jest fundamentalną funkcją aplikacji, stąd szczególnie istotna była jej staranna i przemyślana implementacja. Spośród wielu dostępnych

narzędzi najwłaściwszym okazało się ***Selenium*** będące popularnym frameworkiem open-source do automatyzacji przeglądarek internetowych.

Przed rozpoczęciem pobierania jakichkolwiek danych, niezbędna jest odpowiednia konfiguracja sterownika. Aby umożliwić działanie Selenium, wymagana jest zainstalowana na urządzeniu przeglądarka internetowa, a konkretnie Google Chrome. Użytkownik musi także pobrać sterownik [9] zgodny z aktualną wersją przeglądarki. Listing 4.2 przedstawia proces inicjowania sterownika, który jest uzależniony od systemu operacyjnego. Dodawany jest także parametr ***headless*** umożliwiający wykonywanie się kodu na przeglądarce otwartej w tle.

```
private fun setupDriverOptions() {
    val os = System.getProperty("os.name").toLowerCase()
    if (os.startsWith("windows")) {
        System.setProperty("webdriver.chrome.driver", "chromedriver.exe")
    } else if (os.startsWith("mac")) {
        System.setProperty("webdriver.chrome.driver", "chromedriver")
    }
    driverOptions.addArguments("--headless")
}
```

Listing 4.2 Przygotowanie sterownika do pracy [opracowanie własne]

Jedną z niewątpliwych zalet wykorzystania Selenium jest to, że może ono wchodzić w interakcję ze stroną internetową, tak jak użytkownik. Jest to szczególnie przydatne np. w iterowaniu po poszczególnych latach czy rodzajach wydarzeń sportowych, ponieważ wymaga to wybrania konkretnej opcji z listy. Na listingu 4.3 zaprezentowany został fragment kodu odpowiedzialny za uzyskanie dostępu do podstawowych danych niezbędnych do pobrania dokładnych wyników.

```
val competitions = mutableListOf<CompetitionData>()
tagsWithDates.forEach { tagsWithDate ->
    tagsWithDate.first.forEach { tag ->
        val href = (tag as RemoteWebElement)
                        .findElementsByTagName("a")
                        .firstOrNull()
                        ?.getAttribute("href")
                ?: return@forEach
        val type = tag.text.split(" ").first()
```

```
        val dateElements = tagsWithDate.second.split(" ")
        val date = dateElements[0] + " "
                        + dateElements[1] + " "
                        + dateElements.last()
        val formatter = DateTimeFormatter
                             .ofPattern("d MMMM yyyy")
                             .withLocale(Locale.ENGLISH)
        val formattedDate = LocalDate.parse(date, formatter).toString()
        competitions.add(CompetitionData(href, type, formattedDate))
    }
}
```

Listing 4.3 Przygotowanie zestawu danych do pobierania wyników [opracowanie własne]

Przedstawiony na listingu 4.3 kod odpowiada za zapis typu i daty rozpoczęcia zawodów oraz linku do strony z tablicą wyników. Data będzie przydatna podczas analizowania postępu konkretnych zawodników lub generalnych zmian zachodzących w dyscyplinie. Link zostanie w dalszej części kodu wykorzystany do pobrania szczegółowych wyników każdego zawodnika. Typ (np. zawody na czas lub prowadzenie) definiuje dokładny sposób pobierania wyników.

```
val results: MutableList<BoulderGeneral> = mutableListOf()
val table = driver.findElementById("table_id")
val rows =
table.findElement(By.tagName("tbody")).findElements(By.tagName("tr"))
rows.forEach { row ->
    val result = fetchBasicResultFromTable(row)
    if (result.climberId != null) {
        results.add(
            BoulderGeneral(
                rank = result.rank,
                climberId = result.climberId,
                qualification = result.qualification,
                semiFinal = result.semiFinal,
                final = result.final
            )
        )
    }
}
val competitionId = generateCompetitionId(url)
```

```
database.writeBoulderResults(results, date, competitionId, eventTitle,
eventCity)
```

Listing 4.4 Pobieranie i zapis wyników z zawodów typu bouldering

```
private fun fetchBasicResultFromTable(row: WebElement): BasicResult {
    val rank = (row as RemoteWebElement)
                .findElementsByClassName("rank")
                .firstOrNull()?.text?.toIntOrNull()
    val climberId = row.findElementByTagName("a").getAttribute("href")
                    .split("=").last().toIntOrNull()
    val scores = row.findElements(By.className("tdAlignNormal"))
    val qualification = scores[0].text
    val semiFinal = scores.getOrNull(1)?.text
                    .takeUnless { it.isNullOrBlank() }
    val final = scores.getOrNull(2)?.text.takeUnless { it.isNullOrBlank() }
    return BasicResult(rank, climberId.toString(), qualification,
semiFinal, final)
}
```

Listing 4.5 Odczytywanie wyników z tabeli [opracowanie własne]

Zaprezentowane na listingach 4.4 i 4.5 części kodu odpowiadają za odczytanie wyników z tabeli oraz ich zapis do bazy w ostatecznej postaci. W pierwszej kolejności poszczególne elementy struktury strony internetowej, takie jak tabela i znajdujące się w niej rekordy są wyodrębniane z wykorzystaniem ich numerów **id** lub nazw tagów HTML. Następnie wartość z każdej komórki zostaje odczytana i rzutowana na odpowiedni typ. Na tym, etapie link wykorzystany wcześniej do otworzenia strony z tabelą ponownie jest przetwarzany - wyodrębniona zostaje z niego informacja o **id** zawodów. Numer ten jest używany do wygenerowania unikalnego identyfikatora rekordu zapisywanego w bazie danych.

W osobnym procesie pobierane są także niezbędne dane zawodników. Do działania systemu wymagane są informacje takie jak ich wiek, narodowość, czy płeć. Proces ich pobierania przedstawiony został na listingach 4.6 i 4.7.

```
suspend fun fetchAllClimbers() {
    val url = "https://www.ifsc-
climbing.org/index.php?option=com_ifsc&task=athlete.display&id="
    var climberId = 0
    val driver = ChromeDriver(driverOptions)
    loop@ while (climberId < MAX_CLIMBER_ID) {
```

```
        try {
            climberId++
            driver.get(url + climberId)
            val climber = getClimberDataFromTable(climberId, driver)
                                ?: continue@loop
            database.writeClimber(climber)
        } catch (_: NoSuchElementException) {
            continue@loop
        }
    }
    driver.close()
}
```

Listing 4.6 Iterowanie po dostępnych zawodnikach [opracowanie własne]

```
private fun getClimberDataFromTable(
climberId: Int,
driver: ChromeDriver
): Climber? {
    val wait = WebDriverWait(driver, 30)
    try {
        wait.until(
ExpectedConditions.visibilityOfElementLocated(By.className("name"))
        )
    } catch (e: Exception) {
        Arbor.e("Climber with id $climberId could not be fetched")
        return null
    }
    val name = driver.findElementByClassName("name").text
                    .takeUnless { it.isBlank() } ?: return null
    val country = driver.findElementByClassName("country").text
    val federation = driver.findElementByClassName("federation").text
    val sex =
        if (driver.pageSource.contains("WOMEN")) Sex.WOMAN
        else if (driver.pageSource.contains("MEN")) Sex.MAN
        else null
    val age =
        (driver.findElementByClassName("age") as RemoteWebElement)
        .findElementByTagName("strong").text.toIntOrNull()
    val dateOfBirth = age?.let {
                            Calendar.getInstance().get(Calendar.YEAR) - it
                        }.toString()
    return Climber(climberId.toString(), name, sex, dateOfBirth, country,
```

```
federation, RecordType.OFFICIAL)
}
```

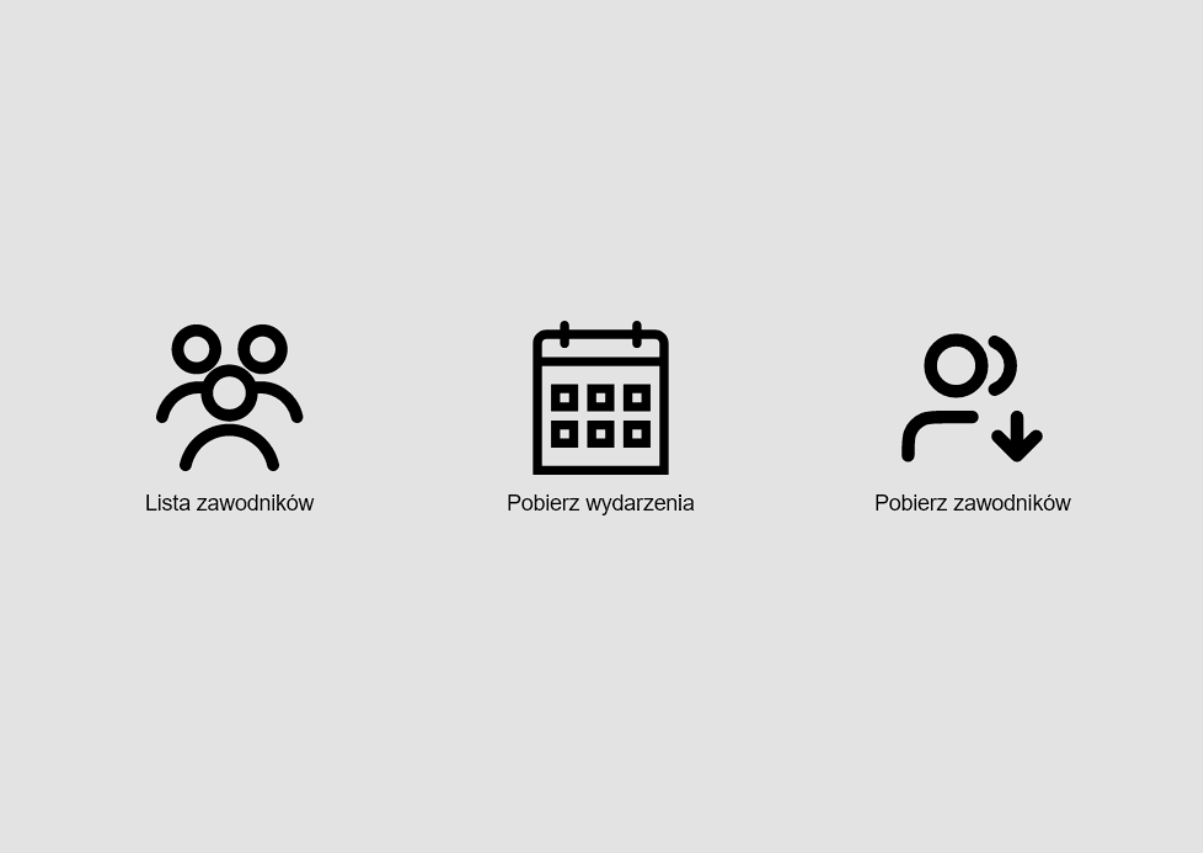
Listing 4.7 Pobieranie danych zawodnika [opracowanie własne]

Funkcja ***fetchAllClimbers*** odpowiedzialna jest za otwieranie kolejnych stron internetowych zawierających dane o sportowcach. W tym przypadku jest to prostsze niż dla wyników, ponieważ do statycznego adresu HTTPS wystarczy dopisać inkrementowany z każdą iteracją parametr **id**. Przygotowana strona zostaje przekazana do funkcji ***getClimberDataFromTable*** razem z **id** zawodnika. Na tym etapie odczytywane są wszystkie dostępne informacje, takie jak narodowość, wiek, czy płeć. Jeżeli zostaną odczytane pomyślnie, tworzony i zwracany jest obiekt wspinacza. Obiekt zapisujemy do bazy danych w funkcji z listingu 4.6. W przeciwnym wypadku funkcja zwraca wartość ***null***, co skutkuje przerwaniem aktualnej iteracji i przejściem do następnej. Cały proces kontynuowany jest do momentu osiągnięcia maksymalnej możliwej wartości **id**.

5. Interfejsy użytkownika

W tym rozdziale omówione zostaną najważniejsze interfejsy użytkownika zamieszczone w aplikacji. Przedstawione ekrany pochodzą z systemu operacyjnego Windows, do którego też odnoszą się używane terminy.

5.1. Strona główna



Rys. 5.1 Strona główna aplikacji [opracowanie własne]

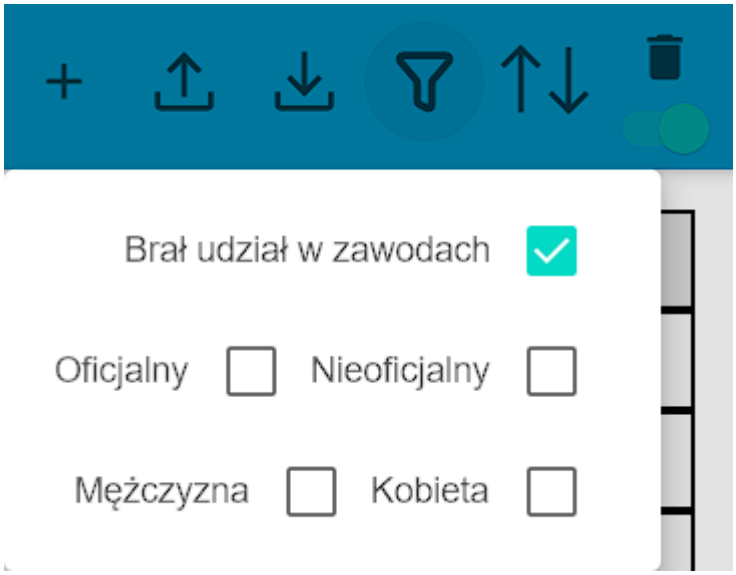
Na rysunku 5.1 przedstawiony został minimalistyczny ekran startowy aplikacji, umożliwiający użytkownikowi nawigację do kolejnych elementów systemu:

- listy dostępnych zawodników,
- pobierania zawodników,
- pobierania wyników z wydarzeń sportowych.

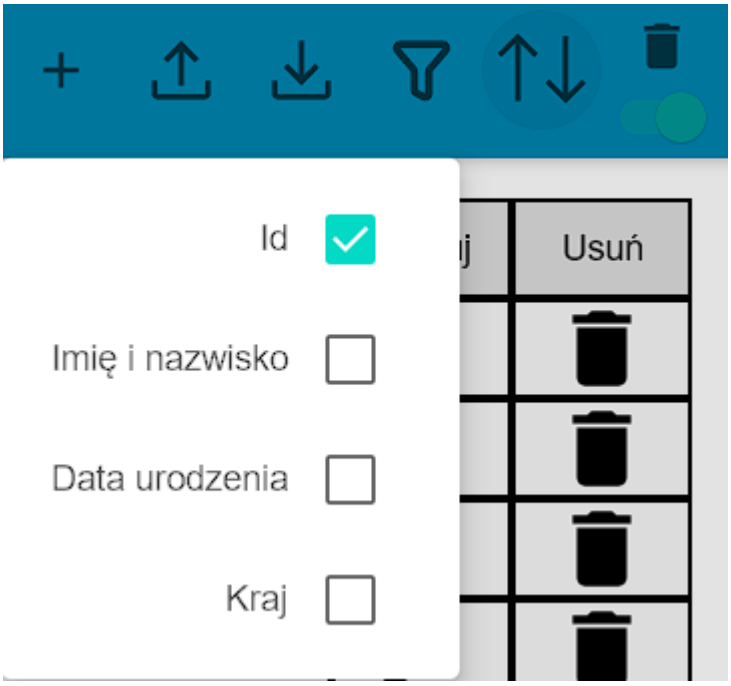
5.2. Lista zawodników

Id	Imię i nazwisko	Płeć	Data urodzenia	Kraj	Edytuj	Usuń
437	ROMAIN DESGRANGES	Mężczyzna	1983	FRA		
455	HÉLÈNE JANICOT	Kobieta	1994	FRA		
462	MANON HILY	Kobieta	1994	FRA		
466	JULIA CHANOURDIE	Kobieta	1997	FRA		
467	SALOMÉ ROMAIN	Kobieta	1997	FRA		
469	FANNY GIBERT	Kobieta	1994	FRA		
470	JEREMY BONDER	Mężczyzna	1992	FRA		
474	GUILLAUME MORO	Mężczyzna	1995	FRA		
479	ANOUCK JAUBERT	Kobieta	1994	FRA		
486	ALBAN LEVIER	Mężczyzna	1995	FRA		
488	AURELIA SARISSON	Kobieta	1998	FRA		
490	MANUEL CORNU	Mężczyzna	1994	FRA		
492	HUGO PARMENTIER	Mężczyzna	1999	FRA		
494	PIERRE REBREYEND	Mężczyzna	1998	FRA		
505	LEO AVEZOU	Mężczyzna	2000	FRA		
511	VICTOIRE ANDRIER	Kobieta	1997	FRA		

Rys. 5.2 Lista zawodników [opracowanie własne]



Rys. 5.3 Lista filtrów [opracowanie własne]

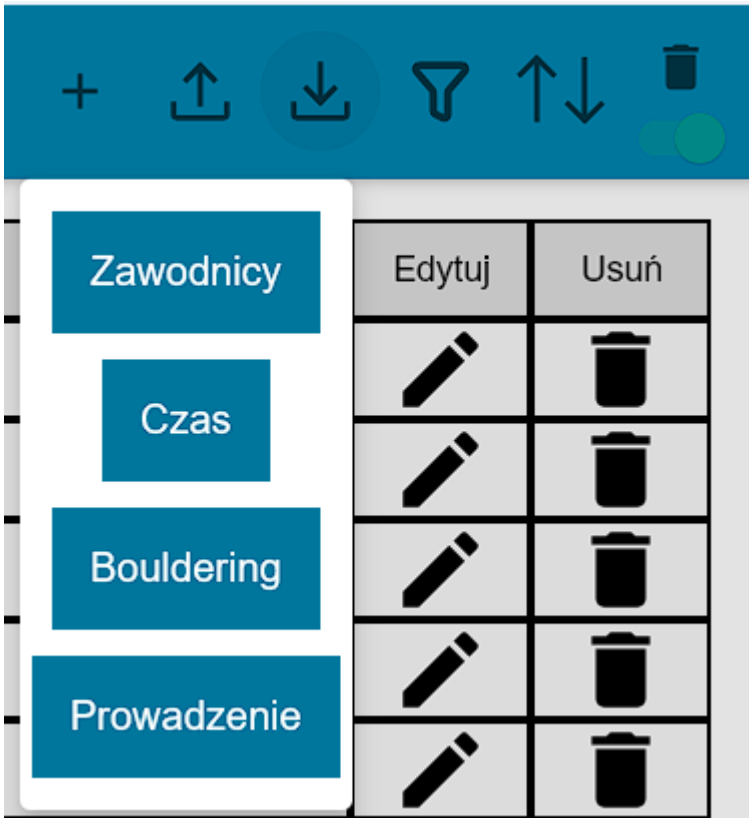


Rys. 5.4 Tryby sortowania [opracowanie własne]

Po wybraniu pierwszej opcji z ekranu głównego użytkownikowi wyświetlana jest lista zapisanych w bazie zawodników. Ma on możliwość dostosowania jej zawartości z wykorzystaniem filtrów przedstawionych na rys. 5.3, a także zmiany kolejności wpisów przy użyciu opcji sortowania zamieszczonych na rys. 5.4. Domyślnie zaznaczona jest opcja filtrowania „Brał udział w zawodach”, dzięki czemu ignorowana jest duża część sportowców, dla których nie są dostępne żadne wyniki.

Z poziomu przedstawionego ekranu użytkownik może usuwać zawodników z bazy. Aby uniknąć przypadkowego usunięcia sportowca należy zaznaczyć domyślnie odznaczony checkbox umieszczony w prawym górnym rogu ekranu, obok ikony kosza. Dopiero wtedy w pobliżu zawodników pojawiają się przyciski umożliwiające ich usunięcie.

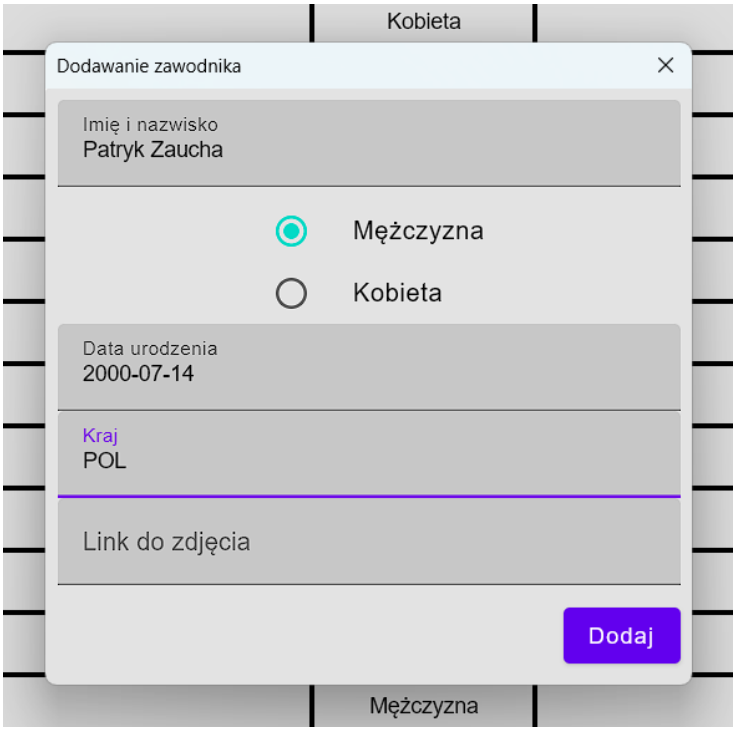
5.3. Import i eksport danych



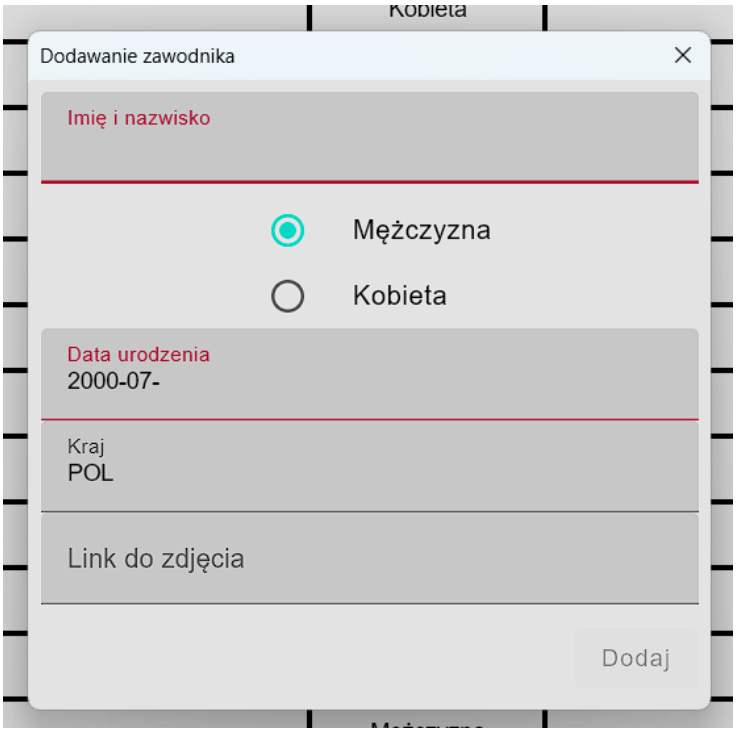
Rys. 5.5 Typy danych do zaimportowania [opracowanie własne]

Kolejnymi dostępnymi funkcjami są import oraz eksport danych. Po wybraniu ikony odpowiadającej eksportowi, wszystkie informacje o zawodnikach i zawodach są zapisywane w plikach CSV, po czym otwierany jest zawierający je folder. W przypadku importu, użytkownik najpierw wybiera z listy przedstawionej na rys. 5.5 typ danych, które chce wprowadzić do aplikacji. Następnie wskazuje plik w eksploratorze plików, po czym następuje zapisanie informacji do bazy.

5.4. Dodawanie zawodnika



Rys. 5.6 Wprowadzanie danych nowego zawodnika [opracowanie własne]



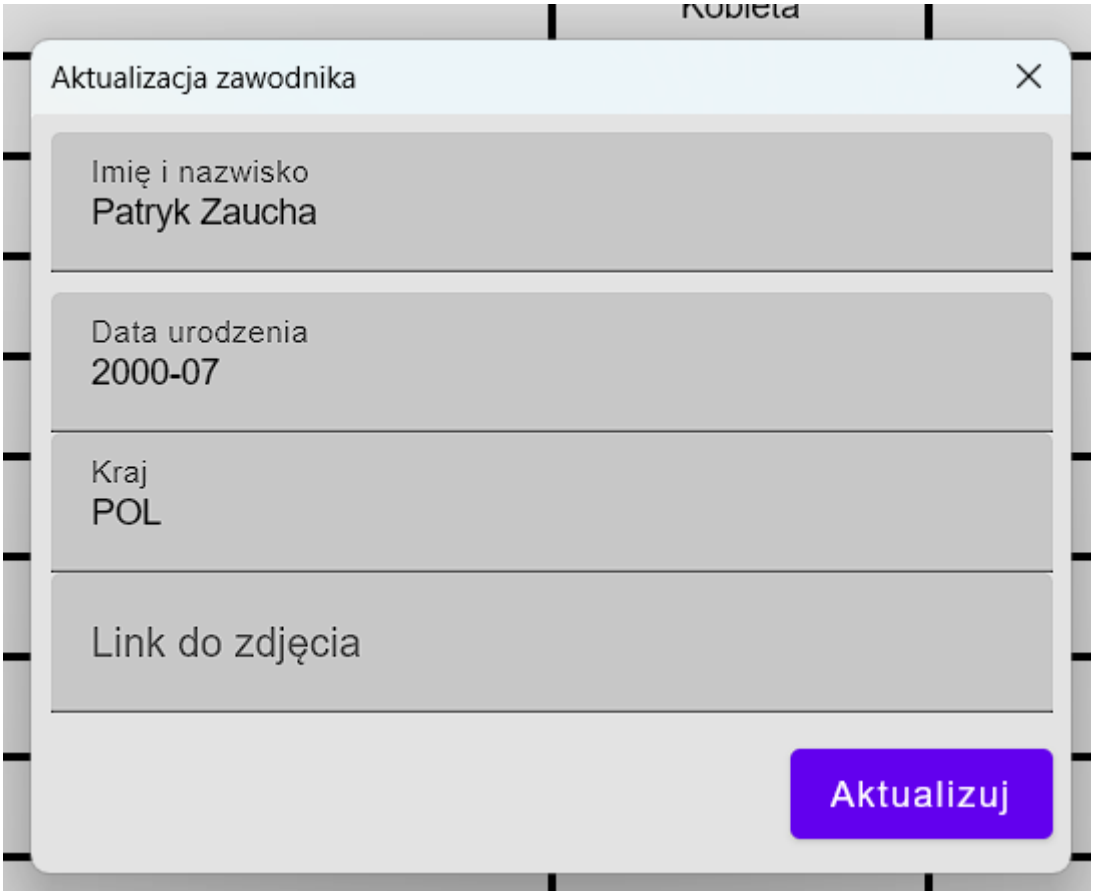
Rys. 5.7 Niepoprawnie wprowadzone dane zawodnika [opracowanie własne]

Możliwe jest także wprowadzenie do systemu nowego zawodnika poprzez ręczne wpisanie jego danych. Na rysunku 5.6 przedstawiony został formularz z częściowo uzupełnionymi informacjami, które pozwalają na zapisanie zawodnika. Natomiast rysunek 5.7 pokazuje sytuację, w której dane są błędne, a przycisk potwierdzający jest nieaktywny. Wymagane w formularzu dane to:

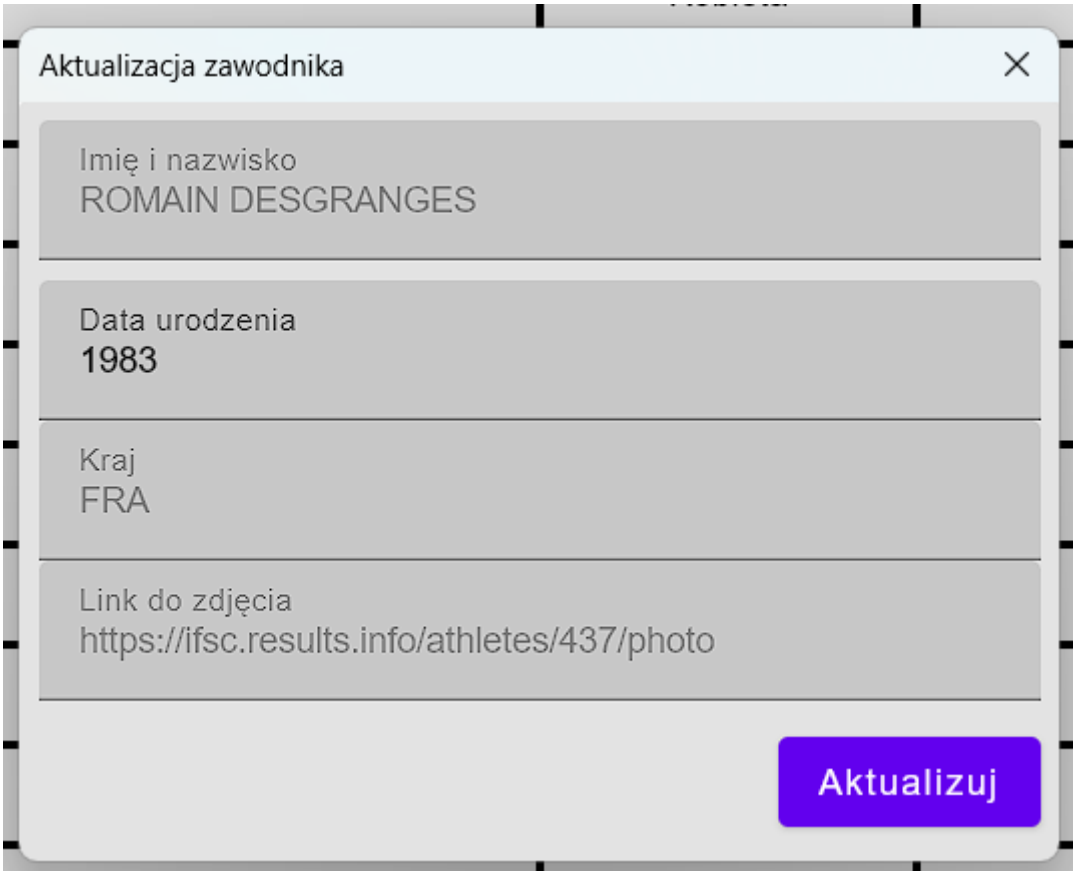
- imię i nazwisko,
- płeć,
- kraj.

Dodatkowo, zaimplementowana została walidacja treści podanej w polu z datą urodzenia zawodnika - musi być ona w formacie **yyyy-mm-dd**, przy czym możliwym jest wprowadzenie tylko roku lub roku i miesiąca bez dokładnego dnia. Rozwiązanie to umożliwia uzyskanie przybliżonego wieku zawodnika, gdy nie jest znana jego dokładna data urodzenia.

5.5. Edycja zawodnika



Rys. 5.8 Formularz edycji zawodnika nieoficjalnego [opracowanie własne]



Rys. 5.9 Formularz edycji zawodnika oficjalnego [opracowanie własne]

Użytkownik aplikacji może w pewnym stopniu edytować dane każdego sportowca. Przedstawione na rysunku 5.8 edytowanie zawodnika nieoficjalnego, czyli wprowadzonego ręcznie, pozwala na zmianę dowolnej informacji poza płcią. Obowiązuje taka sama walidacja pól jak w przypadku dodawania nowego rekordu.

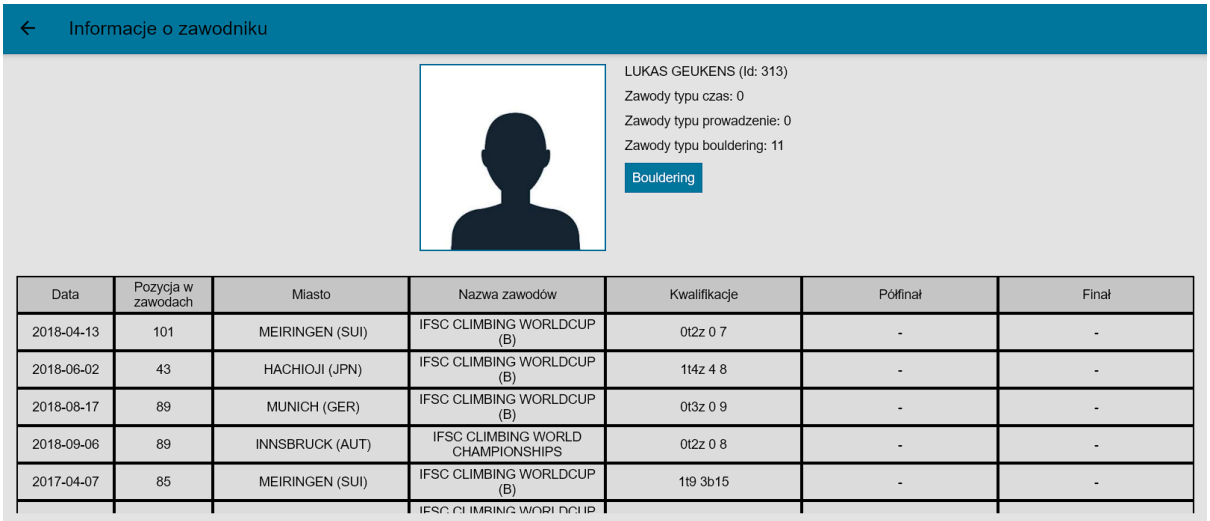
Podczas pokazanego na rysunku 5.9 dodawania zawodnika oficjalnego, czyli pobranego ze strony internetowej IFSC, możliwa jest jedynie zmiana daty urodzenia - pozostałe pola są wyłączone. W aplikacji udostępniono możliwość edycji tego parametru z jednego powodu - ta konkretna informacja nie jest dostępna na stronie w odpowiednio czytelnej formie. Podczas pobierania danych sportowców odczytywany jest wiek sportowca, na bazie którego podaje się rok urodzenia - nie ma tu więc możliwości uzyskania dokładnej daty.

W obydwu przypadkach stosowana jest walidacja wprowadzanych danych taka sama jak na formularzu dodawania nowego zawodnika.

5.6. Szczegóły zawodnika

Data	Pozycja w zawodach	Miasto	Nazwa zawodów	Kwalifikacje	Półfinał	Finał
2021-04-16	37	MEIRINGEN (SUI)	IFSC CLIMBING WORLD CUP (B)	3t5z 6 10	-	-
2021-06-23	31	INNSBRUCK (AUT)	IFSC CLIMBING WORLD CUP (B,L)	1t4z 1 10	-	-
2021-09-16	25	MOSCOW	IFSC CLIMBING WORLD CHAMPIONSHIPS (B,L,S)	3t4z 5 6	-	-
2020-11-21	8	MOSCOW (RUS)	IFSC EUROPE CONTINENTAL CHAMPIONSHIPS (B,S,L,C)	3t5z 5 10	2t3z 2 4	-
2018-04-21	21	MOSCOW (RUS)	IFSC CLIMBING WORLDCUP (B,S)	3t5z 6 7	-	-

Rys. 5.10 Ekran szczegółów zawodnika [opracowanie własne]

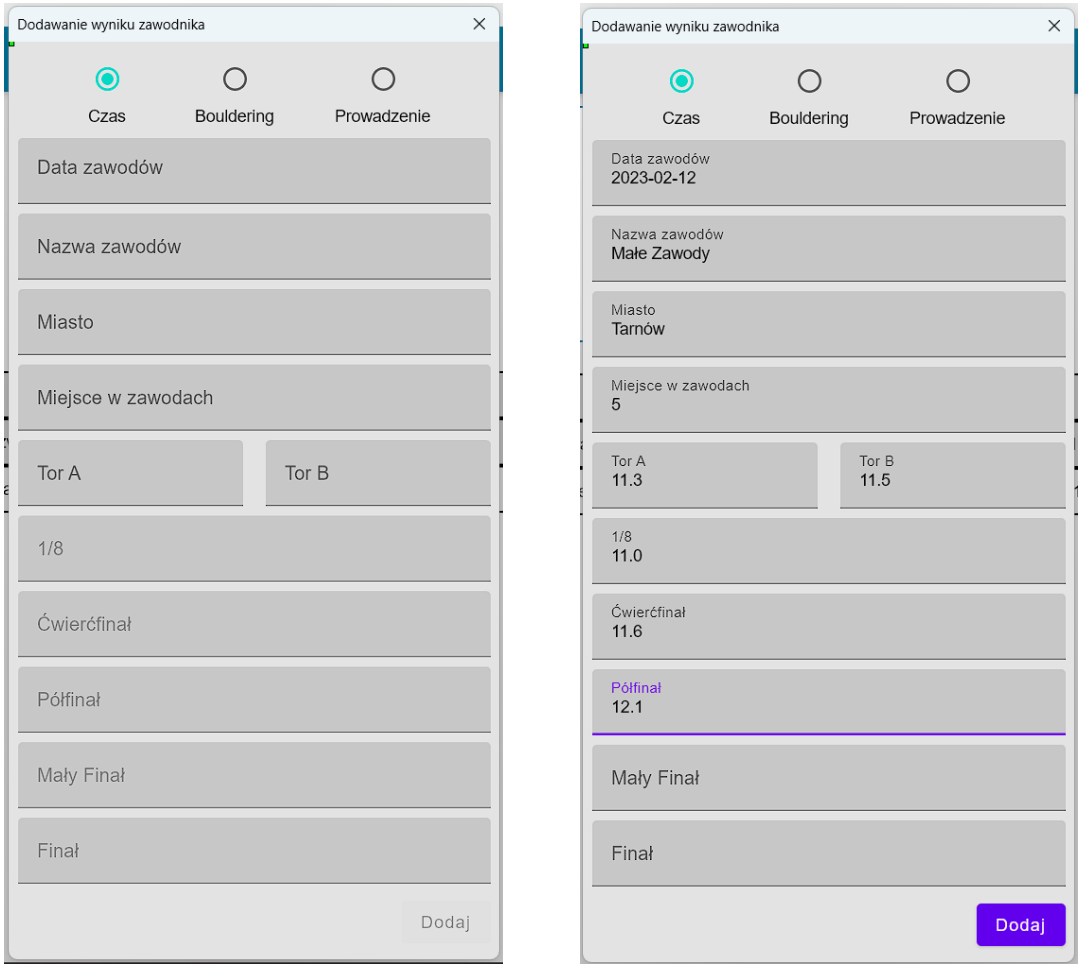


Data	Pozycja w zawodach	Miasto	Nazwa zawodów	Kwalifikacje	Półfinał	Finał
2018-04-13	101	MEIRINGEN (SUI)	IFSC CLIMBING WORLDCUP (B)	0t2z 0 7	-	-
2018-06-02	43	HACHIOJI (JPN)	IFSC CLIMBING WORLDCUP (B)	1t4z 4 8	-	-
2018-08-17	89	MUNICH (GER)	IFSC CLIMBING WORLDCUP (B)	0t3z 0 9	-	-
2018-09-06	89	INNSBRUCK (AUT)	IFSC CLIMBING WORLD CHAMPIONSHIPS	0t2z 0 8	-	-
2017-04-07	85	MEIRINGEN (SUI)	IFSC CLIMBING WORLDCUP (B)	1t9 3b15	-	-

Rys. 5.11 Ekran szczegółów zawodnika bez zdjęcia [opracowanie własne]

Na ekranie z informacjami o zawodniku przedstawiane są takie informacje jak jego imię i nazwisko, czy wyniki ze wszystkich zawodów, w jakich brał udział. Wyświetlane jest również zdjęcie, o ile jest dostępne. Zdjęcia zawodników nie są przechowywane w bazie danych, jedynie linki do nich. W przypadku braku internetu lub po prostu braku zdjęcia, wyświetlane jest zdjęcie zastępcze widoczne na rysunku 5.11. Dla każdego rodzaju zawodów przygotowana została osobna tabela, na którą można się przełączyć używając odpowiadających im przycisków przedstawionych na listingu 5.10. Przyciski są wyświetlane, gdy w danej kategorii zawodnik posiada wpisy. Pod nimi znajduje się rozwijane menu z różnymi analizami wyników, które zostaną omówione w osobnym podrozdziale.

5.7. Dodawanie wyników



Rys. 5.12, 5.13 Dodawanie nowego wyniku dla zawodnika [opracowanie własne]

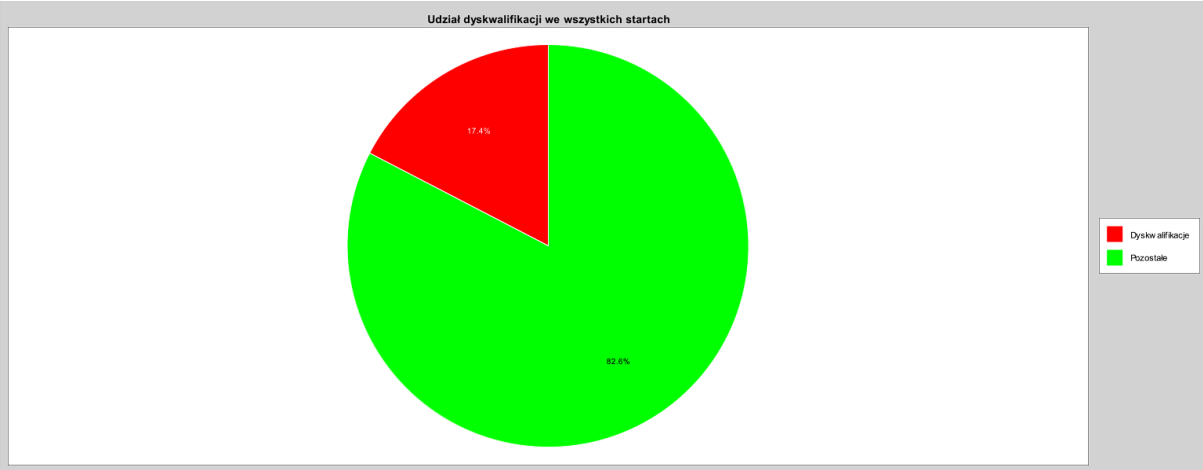
Rysunki 5.12 i 5.13 przedstawiają proces dodawania wyniku uzyskanego przez sportowca w zawodach na czas. Formularz ten otwierany jest przez kliknięcie w ikonę plusa w prawym górnym rogu na ekranie ze szczegółami zawodnika. Dostęp do tej funkcji został ograniczony jedynie do zawodników nieoficjalnych, a na ekranie szczegółów zawodników oficjalnych przycisk nie jest widoczny. Na formularzu zastosowano podobną walidację danych wejściowych jak na tych opisywanych do tej pory, jednak tutaj wymagana jest data z dokładnością do dnia miesiąca. Ponadto, poszczególne pola tekstowe stają się aktywne progresywnie, to znaczy żeby wpisać np. czas uzyskany w ⅛ finału, wcześniej trzeba podać czas na przynajmniej jednym torze z etapu kwalifikacji. Nie można także wypełnić pól małego finału i finału jednocześnie, ponieważ zawodnik nie mógł brać udziału w obu tych etapach.

5.8. Edycja wyników

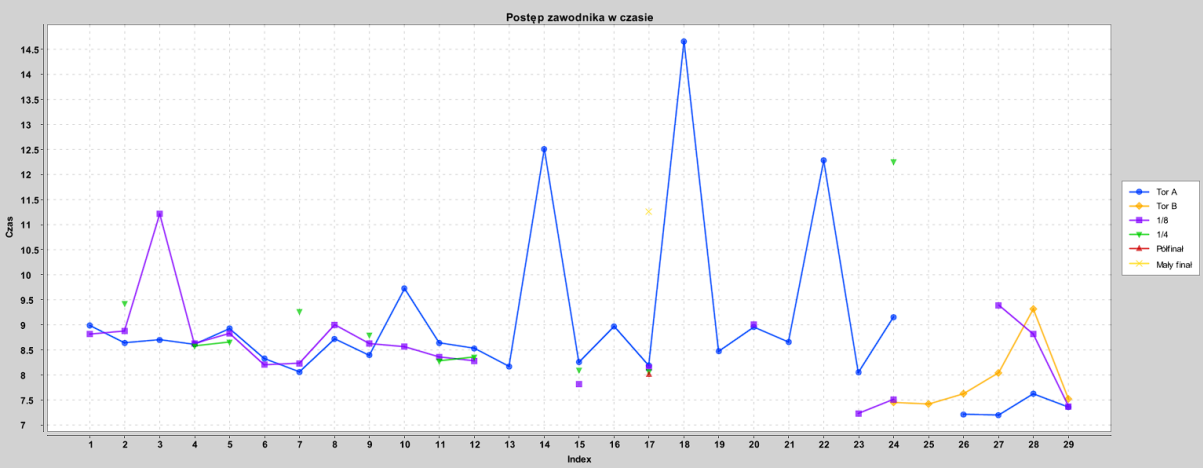
Rys 5.14 Edycja wyniku zawodnika [opracowanie własne]

Na rysunku 5.14 przedstawiony został formularz edycji wyniku zawodnika w kategorii czas, otwierany poprzez kliknięcie na rekord w tabeli wyników. Funkcja ta, analogicznie do dodawania nowego wyniku, dostępna jest tylko w kontekście zawodników nieoficjalnych. Walidacja pól pozostaje dokładnie taka sama jak w przypadku wcześniej wspomnianego formularza.

5.9. Wyniki i osiągnięcia



Rys 5.15 Wykres udziału dyskwalifikacji zawodnika we wszystkich startach [opracowanie własne]



Rys. 5.16 Wykres postępu zawodnika w czasie [opracowanie własne]

Dla zawodników posiadających wyniki z zawodów na czas mogą zostać wygenerowane i wyświetlone analizy w postaci wykresów. Na rysunku 5.15 przedstawiony został wykres kołowy, reprezentujący procentowy udział dyskwalifikacji zawodnika we wszystkich jego zawodach. Dyskwalifikacja następuje zazwyczaj na skutek rozpoczęcia wspinaczki zbyt wcześnie lub przez upadek ze ścianki wspinaczkowej. Podczas generowania wykresu brane są pod uwagę wszystkie wyniki zawodnika.

Drugim przykładowym wykresem, przedstawionym na rysunku 5.16, jest rozwój zawodnika w czasie. W procesie jego generowania analizowane są wyniki uzyskane przez

zawodnika na poszczególnych etapach zawodów, takich jak kwalifikacje, ćwierćfinał itd. na przestrzeni wszystkich lat.

6. Testy integracyjne

Testowanie integracyjne jest rodzajem testowania oprogramowania, które ma na celu zatwierdzenie interakcji i komunikacji pomiędzy wieloma komponentami oprogramowania lub systemami. Celem testów integracyjnych jest zidentyfikowanie wszelkich problemów, które mogą powstać, gdy różne komponenty lub systemy są połączone. Często są to błędy związane z kompatybilnością, spadkiem wydajności, czy utratą danych. Testy integracyjne pomagają upewnić się, że różne komponenty systemu oprogramowania działają razem zgodnie z przeznaczeniem i mogą odkryć potencjalne problemy, pozwalając na ich rozwiązanie we wczesnej fazie rozwoju programu. Testy integracyjne mogą być wykonywane na różnych etapach cyklu życia oprogramowania, od testów jednostkowych do testów systemowych. Mogą być zautomatyzowane lub ręczne i mogą być wykonywane przy użyciu różnych narzędzi i technik. Testy integracyjne pomagają zwiększyć ogólną jakość systemu oprogramowania i zmniejszyć ryzyko wystąpienia problemów, które mogłyby powstać w wyniku interakcji pomiędzy komponentami lub systemami.

6.1. Testy bazy danych

Baza danych umożliwia przechowywanie zróżnicowanych danych, które aplikacja odczytuje i modyfikuje przez znaczną część działania. Jej bezawaryjne działanie jest kluczowe dla funkcjonowania całego systemu.

```
internal class ClimberArgumentProvider : ArgumentsProvider {
    @Throws(Exception::class)
    override fun provideArguments(context: ExtensionContext): Stream<out
Arguments?> {
        return Stream.of(
            Arguments.of(
                Climber(
                    "123",
                    "John Doe",
                    null,
                    null,
                    "USA",
                    "USAC",
                    RecordType.UNOFFICIAL
                )
```

```
            ),
            Arguments.of(
                Climber(
                    "321",
                    "Arkadiusz Justyński",
                    Sex.MAN,
                    "1987",
                    "POL",
                    "Polska Federacja Wspinaczki Sportowej",
                    RecordType.UNOFFICIAL
                )
            ),
        )
    }
}
```

Listing 6.1 Przygotowanie obiektów do testów parametryzowanych [opracowanie własne]

Listing 6.1 przedstawia instancję implementacji ***ArgumentsProvider*** w frameworku testowym JUnit 5. Interfejs ten służy do zapewnienia źródła argumentów wejściowych dla testów sparametryzowanych, które są rodzajem testów wykonujących się wielokrotnie ze zmiennymi zestawami danych wejściowych. Klasa ***ClimberArgumentProvider*** implementuje ten interfejs i dostarcza pojedynczy zestaw argumentów testu w postaci instancji klasy Climber. Klasa ta ma kilka właściwości, które definiują wspinacza, w tym **id**, nazwę, kraj, organizację wspinaczkową i typ rekordu. Poprzez implementację ***ArgumentsProvider*** i dostarczenie instancji wspinacza jako argumentu. Kod pozwala na wykonanie sparametryzowanych testów, które wykorzystują dostarczony argument Climbera. Ta cecha zapewnia elastyczne i wielokrotnego użytku podejście do testowania, pozwalając na zastosowanie tej samej logiki testowej do wielu zestawów wejść.

```
@BeforeEach
fun setUp() {
    database = Database("test.realm")
}
```

Listing 6.2 Przygotowanie nowej instancji bazy do testów [opracowanie własne]

Funkcja z adnotacją ***@BeforeEach*** zostaje wywołana przed każdą metodą testową. Jest ona niezbędna, ponieważ przygotowuje nową, pustą bazę danych, na której można wykonywać

dowolne operacje, nie martwiąc się o integralność danych przechowywanych w głównej bazie aplikacji.

```
@AfterEach
fun tearDown() {
    database.close()
    File("test.realm.management").delete()
    File("test.realm").delete()
    File("test.realm.lock").delete()
}
```

Listing 6.3 Zamknięcie połączenia z bazą danych i usunięcie plików konfiguracyjnych [opracowanie własne]

Adnotacja ***@AfterEach*** użyta na listingu 6.3 działa analogicznie do tej przedstawionej na listingu 6.2, jednak wykonuje się już po zakończeniu działania metody testowej. Przedstawiona funkcja odpowiada za usuwanie testowej bazy danych, dzięki czemu mamy pewność, że każdy test zostanie przeprowadzony na pustej bazie, co umożliwi poprawne zweryfikowanie testowanej metody.

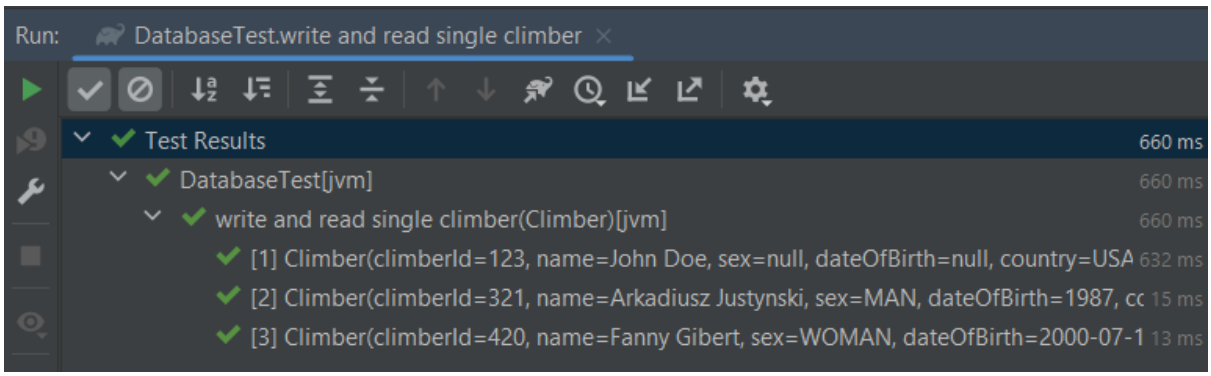
6.1.1. Zapis i odczyt zawodnika z bazy danych

W celu poprawnego działania aplikacja wymaga dostępu do różnych danych, takich jak te o zawodnikach. Aby uniknąć błędów podczas zapisu, skutkujących później pracą na nieaktualnych lub nieprawdziwych informacjach przygotowane zostały odpowiednie testy.

```
@ParameterizedTest
@ArgumentsSource(ClimberArgumentProvider::class)
fun `write and read single climber`(climber: Climber) = runTest {
    // act
    database.writeClimber(climber)
    //assert
    val savedClimber = database.getClimberById(climber.climberId)
    assertEquals(climber, savedClimber)
}
```

Listing 6.4 Test zapisu zawodnika [opracowanie własne]

Przedstawiony kod reprezentuje sparametryzowany test. Adnotacja ***@ParameterizedTest*** wskazuje, że metoda testowa jest testem parametryzowanym, a adnotacja ***@ArgumentsSource*** określa źródło argumentów dla metody testowej, którym jest opisana wcześniej klasa ***ClimberArgumentProvider***. Metoda testowa przyjmuje obiekt ***Climber*** jako argument wejściowy i wykonuje dwie akcje: Zapis obiektu do bazy danych i pobranie go po jego **id**. Pobrany zawodnik jest następnie porównywany z oryginalnym obiektem za pomocą funkcji ***assertEquals*** w celu sprawdzenia, czy operacje zapisu i odczytu zakończyły się sukcesem. Ta metoda testowa zapewnia kompleksowe i zautomatyzowane podejście do testowania operacji zapisu i odczytu pojedynczego obiektu ***Climber*** z bazy bazy danych.



Rys. 6.1 Wynik uruchomienia testu zapisu zawodnika do bazy [opracowanie własne]

6.1.2. Usuwanie wyniku z bazy danych

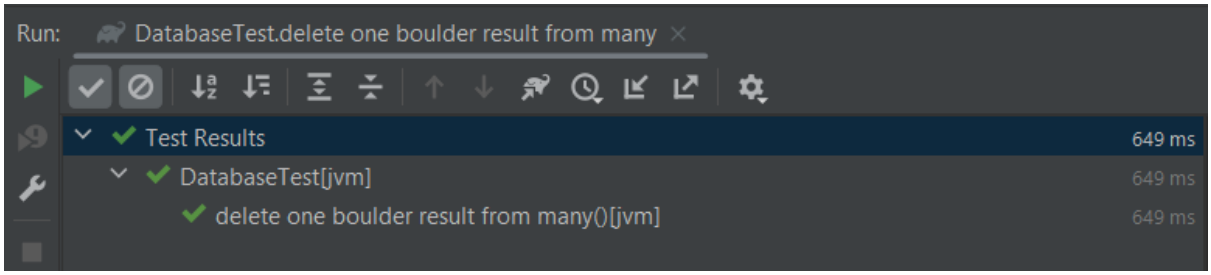
Zapisane w bazie danych wyniki zawodników mogą być usuwane. Aby nie dopuścić do sytuacji, w której w wyniku błędu programu zostałyby usunięte rekordy inne niż zamierzone, funkcja ta powinna być dokładnie sprawdzona pod kątem poprawności.

```
@Test
fun `delete one boulder result from many`() = runTest {
    //arrange
    val boulderResults = listOf(
        BoulderGeneral(
            1,
            "1",
            "12",
            "11",
            null,
        ),
        BoulderGeneral(
```

```
            2,
            "2",
            "12",
            "11",
            null,
        ),
        BoulderGeneral(
            3,
            "3",
            "12",
            "11",
            null,
        )
    )
    database.writeBoulderResults(boulderResults, "2020-06-12", "MS-1",
"IFSC", "Berlin")
    val resultId = database.getAllBoulders()[1].id
    //act
    database.deleteBoulderResult(resultId)
    //assert
    assertAll(
        { assertFalse(database.getAllBoulders().isEmpty()) },
        { assertFalse(resultId in database.getAllBoulders().map { it.id })
},
    )
}
```

Listing 6.5 Test usuwania wyniku z bazy [opracowanie własne]

Metoda najpierw tworzy listę trzech obiektów ***BoulderGeneral***, zapisuje je do bazy danych, a następnie pobiera **id** drugiego wyniku. Drugi wynik jest usuwany z bazy danych przy pomocy metody ***deleteBoulderResult***. Ostatecznie metoda weryfikuje, czy baza danych nie jest pusta oraz **id** usuniętego wyniku nie jest już obecne w kolekcji wyników. Ta metoda testowa zapewnia kompleksowe i zautomatyzowane podejście do testowania funkcjonalności usuwania pojedynczego wyniku boulderowego z bazy danych i pomaga zwiększyć ogólną jakość systemu oprogramowania.



Rys. 6.2 Wynik uruchomienia testu usuwania wyniku z bazy [opracowanie własne]

6.2. Testy importu i eksportu danych

Kolejnym rodzajem akcji wykonywanym na danych przechowywanych przez aplikację jest ich eksport do zewnętrznych, dostępnych dla użytkownika plików tekstowych, a także analogiczny import.

6.2.1. Generowanie pliku CSV

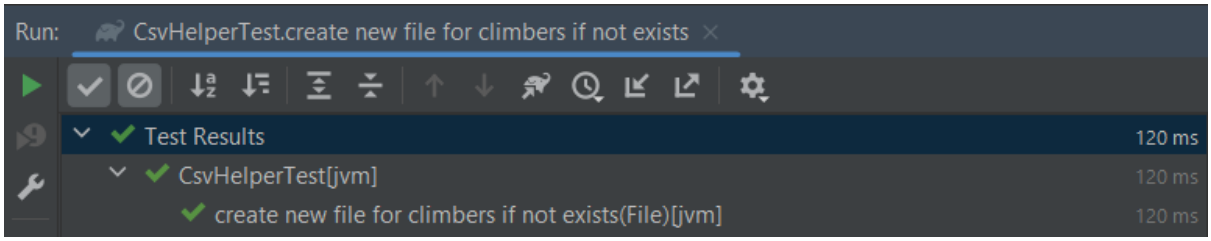
Przed rozpoczęciem eksportu danych należy upewnić się, że istnieje odpowiedni plik docelowy. W tym podrozdziale przedstawiony zostanie test sprawdzający wymienioną funkcję systemu.

```
@Test
fun `create new file for climbers if not exists`(@TempDir tempDir: File) {
    // arrange
    val id = "123"
    val name = "John Doe"
    val sex = Sex.MAN
    val dateOfBirth = "1998"
    val country = "USA"
    val federation = "USAC"
    val recordType = RecordType.OFFICIAL
    val climbers = listOf(
        ClimberRealm().apply {
            this.id = id
            this.name = name
            this.sex = sex.name
            this.dateOfBirth = dateOfBirth
            this.country = country
            this.federation = federation
            this.recordType = recordType.name
        }
```

```
    )
    val fileName = "climbers"
    val fullPath = "${tempDir.path}\\$fileName.csv"
    // act
    csvHelper.writeClimbers(climbers, "${tempDir.path}\\", fileName)
    //assert
    assertTrue(File(fullPath).exists())
}
```

Listing 6.6 Test generowanie nowego pliku jeśli taki nie istnieje [opracowanie własne]

Segment kodu jest kolejną metodą testową JUnit 5, wskazaną przez adnotację ***@Test***. Metoda, *`create new file for climbers if not exists`*, testuje funkcjonalność zapisu kolekcji wspinaczy do pliku CSV. Metoda testowa najpierw tworzy listę jednego obiektu ***ClimberRealm***, która następnie jest przekazywana do metody ***writeClimbers*** obiektu ***csvHelper***. Metoda określa katalog tymczasowy, tworzony automatycznie dzięki wykorzystaniu adnotacji ***@TempDir*** jako miejsce zapisu pliku CSV, a jako nazwę pliku ustawia „climbers”. Pełna ścieżka do pliku jest konstruowana przez połączenie ścieżki katalogu tymczasowego i nazwy pliku. Na koniec metoda weryfikuje, czy plik istnieje w oczekiwane lokalizacji.



Rys. 6.3 Wynik uruchomienia testu generowania pliku CSV [opracowanie własne]

6.2.2. Zapis danych do pliku CSV

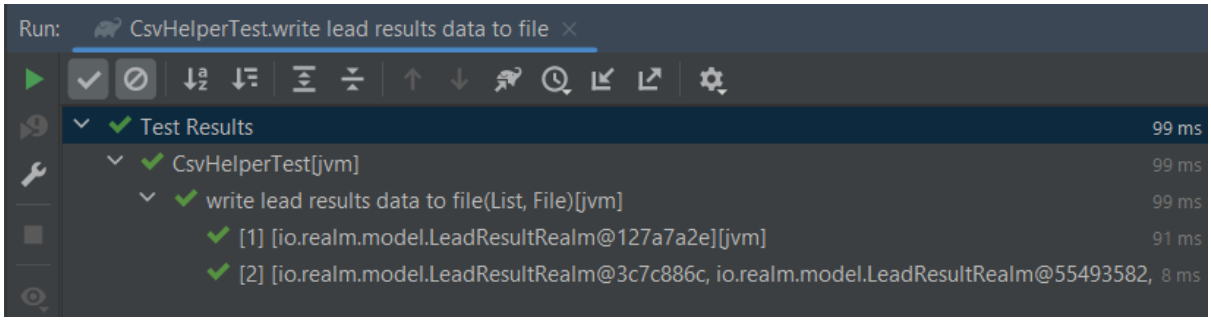
Test przedstawiony na listingu 6.7 weryfikuje poprawność zapisu wyników z zawodów typu ***lead*** do pliku CSV.

```
@ParameterizedTest
@ArgumentsSource(LeadResultRealmArgumentProvider::class)
fun `write lead results data to file`(results: List<LeadResultRealm>,
@TempDir tempDir: File) {
```

```
    // arrange
    val pathName = "$tempDir/"
    val fileName = "leads"
    // act
    val fullPath = csvHelper.writeLeads(results, pathName, fileName)
    // assert
    var expectedContent = ""
    results.forEach { result ->
        with(result) {
            expectedContent +=
"$id,$date,$competitionId,$competitionTitle,$competitionCity,$rank,$climber
Id,$qualification,$semiFinal,$final\n"
        }
    }
    val writtenContent = File(fullPath.toString()).readText()
    assertEquals(
        expectedContent,
        writtenContent
    )
}
```

Listing 6.7 Test zapisu wyników do pliku CSV [opracowanie własne]

Następną testowaną funkcją systemu jest zapis danych do pliku CSV. Przedstawiony fragment kodu operuje konkretnie na danych typu ***LeadResultRealm***. Test przyjmuje jako argumenty wejściowe listę wyników oraz katalog tymczasowy. Lista wyników przekazywana jest poprzez ***LeadResultRealmArgumentProvider***, który służy jako źródło argumentów dla testu. W pierwszym kroku ustawiana jest ścieżka katalogu i nazwa pliku CSV. Następnie wywoływana jest metoda ***writeLeads*** z obiektu ***csvHelper*** z listą wyników, ścieżką katalogu i nazwą pliku jako parametrami. Zwracana jest pełna ścieżka do nowo utworzonego pliku. Ostatni krok polega na sprawdzeniu zawartości pliku. Oczekiwana zawartość jest konstruowana przez połączenie wartości pól każdego obiektu LeadResultRealm. Rzeczywista zawartość pliku jest odczytywana i porównywana z oczekiwaną zawartością. Jeśli zawartość się zgadza, test przechodzi pomyślnie, w przeciwnym razie kończy się niepowodzeniem.



Rys. 6.4 Wynik uruchomienia testu zapisu danych do pliku CSV [opracowanie własne]

7. Testy sprzętowe

Ważnym aspektem z punktu widzenia końcowego odbiorcy systemu jest możliwość jego wykorzystania na komputerach z systemami MacOs oraz Windows. Na przestrzeni całego procesu implementacji, poszczególne rozwiązania były testowane na tych systemach równolegle. Również po zakończeniu prac, poprawność działania całego systemu została zweryfikowana. Wszystkie namierzone problemy zostały rozwiązane, skutkując wysoką wygodą użytkowania i bezawaryjnością na obu systemach. Wygenerowane zostały także pliki wykonywalne, umożliwiające dystrybucję programu bez konieczności udostępniania kodu źródłowego, a także ułatwiające proces uruchamiania go.

8. Podsumowanie

Celem pracy dyplomowej było utworzenie systemu, który umożliwiłby użytkownikowi utworzenie bazy zawodników i ich osiągnięć, a następnie ich analizę. Wszystkie z wymagań funkcjonalnych i niefunkcjonalnych opisanych w drugim rozdziale zostały zrealizowane:

- aplikacja umożliwia pobranie zawodników i ich wyników z oficjalnej strony internetowej z różnych lat i rodzajów wydarzeń sportowych,
- użytkownik może dodawać ręcznie własnych zawodników oraz ich wyniki,
- informacje o zawodnikach mogą być aktualizowane oraz usuwane,
- użytkownik może filtrować i sortować przeglądanych sportowców,
- aplikacja umożliwia import i eksport danych w wygodnej formie,
- generowane i prezentowane są analizy wyników w postaci wykresów.

System nadal posiada wiele możliwości przyszłego rozwoju. Dzięki zastosowaniu technologii Kotlin Multiplatform możliwe jest rozszerzenie dostępu do aplikacji o nowe platformy, szczególnie urządzenia mobilne. Dostęp do bazy zawodników z poziomu urządzenia przenośnego byłby szczególnie przydatny dla trenerów zawodników wspinaczki sportowej w czasie treningów.

Moduł odpowiedzialny za pobieranie danych z sieci może zostać w przyszłości usprawniony i rozbudowany. Możliwe jest rozszerzenie zakresu jego działania w taki sposób, aby informacje były pobierane z różnych stron internetowych, a następnie porównywane i scalane. Rozwiązanie to pozwoli na uzyskanie dostępu do większej ilości dokładniejszych danych, czego efektem będą trafniejsze analizy.

Omówiona praca umożliwiła utrwalenie oraz poszerzenie wiedzy i umiejętności uzyskanych w trakcie studiów, jak i poznanie nowych technologii. Pozwoliła także na doskonalenie procesu zdobywania specjalistycznej wiedzy, jaką jest wspinaczka sportowa, a następnie jej wykorzystywanie w procesie implementacji funkcji systemu.

Bibliografia

[1] Strona międzynarodowej federacji wspinaczki sportowej IFSC
[Online] https://www.ifsc-climbing.org/

[2] Git - dokumentacja
[Online] https://git-scm.com/

[3] Kotlin - dokumentacja
[Online] https://kotlinlang.org/

[4] Gradle - dokumentacja
[Online] https://docs.gradle.org/

[5] Compose Desktop - dokumentacja
[Online] https://www.jetbrains.com/lp/compose-desktop/

[6] Realm - dokumentacja
[Online] https://www.mongodb.com/docs/realm/

[7] JUnit - dokumentacja
[Online] https://junit.org/

[8] Selenium - dokumentacja
[Online] https://www.selenium.dev/documentation/

[9] Sterownik do Selenium, umożliwiający obsługę Google Chrome
[Online] https://chromedriver.chromium.org/downloads

[10] Mark Massé, REST API Design Rulebook, O'Reilly Media, Inc., 2012

Spis rysunków

Spis listingów