

Politechnika Śląska
Wydział Matematyki Stosowanej
Kierunek Informatyka
Studia stacjonarne I stopnia

Projekt inżynierski

Implementacja i rozwój wieloplatformowych aplikacji na przykładzie aplikacji wspomagającej system rekrutacji

Kierujący projektem:
dr inż. Adrian Kapczyński

Autorzy:
Dawid Bitner
Daniel Broczkowski
Mateusz Kowol

Gliwice 2021

*Wszystkim ludziom branży IT i nie tylko,
którzy oferują swoją bezinteresowną pomoc w sieci
oraz społeczności open source.*

Projekt inżynierski:

Implementacja i rozwój wieloplatformowych aplikacji na przykładzie aplikacji wspomagającej system rekrutacji

kierujący projektem: dr inż. Adrian Kapczyński

1. Dawid Bitner – (33,33%)

Rozwój aplikacji od strony aplikacji przeglądarkowej (część kliencka).

Wspomaganie budowy zaplecza systemu aplikacji, implementacja zabezpieczeń (część serwerowa).

2. Daniel Broczkowski – (33,33%)

Rozwój oraz nadzorowanie prac projektu od strony aplikacji przeglądarkowej (część kliencka).

Wspomaganie budowy zaplecza systemu aplikacji (część serwerowa).

3. Mateusz Kowol – (33,33%)

Rozwój oraz nadzorowanie prac projektu od strony aplikacji mobilnej (część kliencka).

Wspomaganie budowy zaplecza systemu aplikacji (część serwerowa).

Podpisy autorów projektu

1.
2.
3.

Podpis kierującego projektem

.....

6:1927586054

Oświadczenie kierującego projektem inżynierskim

Potwierdzam, że niniejszy projekt został przygotowany pod moim kierunkiem i kwalifikuje się do przedstawienia go w postępowaniu o nadanie tytułu zawodowego: inżynier.

Data

Podpis kierującego projektem

Oświadczenie autorów

Świadomy/a odpowiedzialności karnej oświadczam, że przedkładany projekt inżynierski na temat:

Implementacja i rozwój wieloplatformowych aplikacji na przykładzie aplikacji wspomagającej system rekrutacji

został napisany przez autorów samodzielnie.

Jednocześnie oświadczam, że ww. projekt:

- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (j.t. Dz.U. z 2018 r. poz. 1191, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
- nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów wyższej uczelni lub tytułów zawodowych.
- nie zawiera fragmentów dokumentów kopiowanych z innych źródeł bez wyraźnego zaznaczenia i podania źródła,
- złożona w postaci elektronicznej jest tożsama z pracą złożoną w postaci pisemnej.

Podpisy autorów projektu

1. Dawid Bitner, nr albumu: 275250,(podpis:)
2. Daniel Broczkowski, nr albumu: 275254,(podpis:)
3. Mateusz Kowol, nr albumu: 275282,(podpis:)

Gliwice, dnia

8:9547603602

Spis treści

1. Wstęp	9
1.1. Motywacja do wyboru tematu	9
1.2. Cel pracy	9
1.3. Wspomaganie rekrutacji poprzez systemy informatyczne	10
1.3.1. Istniejące rozwiązania na rynku	11
1.3.2. Rozwiązanie zaproponowane w projekcie inżynierskim	12
1.4. Pomysł na rozwiązanie problemu	13
2. Ogólne informacje dotyczące rekrutacji	15
2.1. Rodzaje rekrutacji	16
2.2. Etapy rekrutacji	16
2.3. Techniki i narzędzia rekrutacyjne	18
2.4. Spojrzenie na problem rekrutacji w tekstach kultury	19
2.5. Przepisy prawne stosowane podczas rekrutacji	21
3. Założenia projektowe	25
3.1. Wstępne założenia dotyczące serwera i bazy danych	25
3.2. Wstępne założenia dotyczące aplikacji webowej	27
3.3. Wstępne założenia dotyczące aplikacji mobilnej	28
4. Metodyka	31
4.1. Faza planowania	31
4.2. Wczesna faza implementacji	32
4.3. Zaawansowana faza implementacji	32
4.4. Faza stabilizacji	33
5. Technologie	35
5.1. Serwer i baza danych	35
5.1.1. PostgreSQL	35
5.1.2. SpringBoot	35
5.1.3. Spring Security	36

5.1.4. Spring Data JPA	36
5.2. Aplikacja webowa	36
5.2.1. Angular	36
5.2.2. MDBootstrap	37
5.3. Aplikacja mobilna	37
5.3.1. Flutter	37
6. Narzędzia	39
6.1. System kontroli wersji	39
6.2. Oprogramowanie do zarządzania projektem	41
6.3. Środowiska programistyczne	45
6.3.1. IntelliJ	45
6.3.2. Visual Studio Code	45
6.3.3. Android Studio	46
6.3.4. PG Admin	47
6.3.5. Postman	48
7. Specyfikacja	49
7.1. Instalacja i konfiguracja środowisk	49
7.1.1. IntelliJ IDEA	49
7.1.2. Visual Studio Code	49
7.1.3. Android Studio	49
7.1.4. Postman	50
7.1.5. GitBash	50
7.1.6. PostgreSQL	50
7.1.7. Java	50
7.1.8. Node.js	51
7.1.9. Angular	51
7.1.10. Flutter	51
7.2. Instrukcja obsługi	52
7.2.1. Część serwerowa	52
7.2.2. Część internetowa	52
7.2.3. Część mobilna	52

8. Implementacja	53
8.1. Serwer	53
8.1.1. Specyfikacja bazy danych	53
8.1.2. Projekt bazy danych	54
8.1.2.1. Diagram związków encji	55
8.1.3. Spring Security i bcrypt	57
8.1.4. Architektura	60
8.1.5. Metody protokołu HTTP	60
8.1.5.1. PostMapping	60
8.1.5.2. GetMapping	63
8.1.5.3. DeleteMapping	64
8.1.5.4. PutMapping	66
8.1.6. Opis działania najważniejszych punktów końcowych	68
8.1.6.1. Pobieranie informacji o quizie	68
8.1.6.2. Pobieranie pytania quizu	71
8.1.6.3. Wysyłanie odpowiedzi na pytanie z quizu	72
8.1.6.4. Protokół FTP na przykładzie wysyłania CV	73
8.2. Aplikacja webowa	75
8.2.1. Interfejs użytkownika	76
8.2.2. Biblioteki	93
8.2.3. Budowa projektu	95
8.2.4. HTTP interceptors	98
8.2.5. Przechowywanie informacji	99
8.2.6. Wykorzystane szyfrowanie	99
8.2.7. Angular Hooks	102
8.2.8. Przykłady haków cyklu życia aplikacji	104
8.2.9. Powiadomienia w czasie rzeczywistym	105
8.2.10. Budowa stylu aplikacji na przykładzie MDBootstrap	109
8.2.11. RxJS	112
8.3. Aplikacja mobilna	113
8.3.1. Wymagania urządzenia	113
8.3.2. Architektura MVVM	114
8.3.3. Biblioteki	114
8.3.4. Cykle życia aktywności w Androidzie	119

8.3.5. Rodzaje widgetów	120
8.3.6. HTTP interceptors	120
8.3.7. Interfejs użytkownika	122
9. Testy	133
9.1. Testy części serwerowej	133
9.2. Testy części przeglądarkowej	134
9.3. Testy aplikacji mobilnej	139
10. Podsumowanie	143
10.1. Możliwości wykorzystania systemu	143
10.2. Wnioski	144
10.3. Dalsze możliwości rozwoju projektu	144
Literatura	147

1 Wstęp

Jednym z wymogów ukończenia studiów przez zespół projektowy było odbycie trzymiesięcznej praktyki zawodowej. W związku z tym członkowie grupy odbyli wiele rozmów kwalifikacyjnych i na ich podstawie nabyli pewne doświadczenia. Jako że sam okres stażu na studiach wiąże się z pisaniem pracy inżynierskiej, pomysł na aplikację wspierającą system rekrutacyjny, zasugerowany przez promotora pracy dyplomowej, idealnie trafił w warsztat umiejętności, jak i gustu zespołu.

1.1 Motywacja do wyboru tematu

Główną motywacją do stworzenia serwisu była zauważalna luka na rynku oprogramowania rekrutacyjnego. Nieznaczna liczba aplikacji umożliwia bezpośrednie przypisywanie testów dla poszczególnych użytkowników, stąd też decyzja o obraniu takiego kierunku. Dzięki temu, że kody są unikalne, istnieje teoretyczna możliwość stworzenia niepowtarzalnych testów bądź różniących się od siebie w taki sposób, by wykluczyć możliwość „ściągania” odpowiedzi od innych użytkowników, którzy przystąpili wcześniej do podobnego testu. Daje to szansę indywidualnego podejścia do każdego kandydata. Dodatkowym celem projektu, było przedstawienie zwanego rozwoju oprogramowania, z wykorzystaniem znanych narzędzi, wspomagających ten proces. Projekt jest swoistą laboratoryjną symulacją rozwoju aplikacji w warunkach komercyjnych. Posiada wymagany temat, termin, podział na role, zarządzanie projektem, kontrolę wersji oraz podział na rozgałęzienia projektu w repozytorium.

1.2 Cel pracy

Celem pracy inżynierskiej jest opracowanie założeń oraz implementacja z wykorzystaniem znanych wzorców, bibliotek i języków programowania, wieloplatformowego rozwiązania informatycznego wspomagającego rekrutację. Projekt przedstawia informacje o sposobie budowania systemu informatycznego, który z założenia powinien być skalowalny i modularny, aby poszczególne zespoły programistyczne mogły

pracować nad rozwojem aplikacji w sposób równoległy, co posiada odwzorowanie w projektach komercyjnych.

1.3 Wspomaganie rekrutacji poprzez systemy informatyczne

Ostatnie lata przyniosły wiele zmian w procesach rekrutacyjnych, zarówno na stanowiska najniższego szczebla, jak i kierownicze. Dla zwiększenia wydajności, kolejne przedsiębiorstwa starają się usprawnić przebieg zatrudnienia potencjalnych kandydatów, stosując coraz to nowsze sposoby wyboru przyszłych pracowników. W tym przypadku, bardzo dużą popularnością cieszy się Internet, za pośrednictwem którego przeprowadzane są wstępne wybory kandydatów, rozmowy rekrutacyjne, czy nawet końcowe etapy rekrutacji, wraz z zawarciem umowy o pracę.

Wiele podmiotów gospodarczych posiada swoje predefiniowane platformy do prowadzenia i rejestrowania przebiegu rekrutacji nowych pracowników – co z punktu widzenia kandydata może być kwestią uciążliwą, ponieważ często takie rozwiązania dla danej firmy wymagają założenia w niej konta. Biorąc pod uwagę fakt, że osoba poszukująca pracy zazwyczaj nie aplikuje na stanowisko do jednego konkretnego zakładu pracy, a na ogół robi to mniej lub bardziej masowo, to zagadnienie staje się bardziej problematyczne¹. W każdym z systemów potencjalnego pracodawcy, należy posiadać osobny profil, a co za tym idzie – hasło, które – ze względów bezpieczeństwa – powinno być unikalne w każdym z serwisów. Ma to na celu uchronić nas przed włamaniami do innych platform, gdzie używamy współdzielonych² ³ ⁴ danych logowania. Nieautoryzowane przejęcie konta przez osoby trzecie, grozi wyciekiem

¹P. Zdziech, „Nie kupuj Pan cegły», czyli dla kogo platforma do rekrutacji”, <https://erecruiter.pl/blog/dla-kogo-platforma-do-rekrutacji/>, (dostęp 21.11.2020)

²W. Earp, „Why sharing passwords is a bad idea”, <https://swgfl.org.uk/magazine/why-sharing-passwords-is-a-bad-idea/>, (dostęp 21.11.2020)

³Materiały firmy SolarWinds, „Top Seven Reasons Why You Should Not Share Your Passwords”, <https://logicalread.com/top-seven-reasons-why-you-should-not-share-your-passwords/>, (dostęp 21.11.2020)

⁴Materiały firmy SUNY Broome, „Top reasons why you shouldn't share your username and password”, <https://news.sunybroomed.edu/focus/top-reasons-why-you-shouldnt-share-your-username-and-password/>, (dostęp 21.11.2020)

większej ilości danych osobowych, za których wykorzystaniem stoją najczęściej nieuczciwe intencje.

Dodatkowo hasło należy odpowiednio zabezpieczyć, co oznacza, że nie powinno składać się ze słów dostępnych w słowniku, naszych danych personalnych, dat urodzenia członków rodziny itp. Takie elementy narażają użytkownika na włamanie do konta np. poprzez socjotechniki bądź znajomość naszej osoby. W powszechnie uznanych zasadach, przyjmuje się, że silne hasło powinno składać się z niezrozumiałego dla osób trzecich ciągu znaków, cyfr, symboli specjalnych oraz powinno uwzględniać wielkie i małe litery – zabezpiecza to użytkownika przed włamaniami z wykorzystaniem metod słownikowych. Kod dostępu nie powinien być zbyt krótki – najlepiej, aby posiadał minimum 8 znaków. Minimalizuje to możliwość jego złamania poprzez ataki typu *brute force*. Dobrą praktyką jest również korzystanie z generatorów haseł, należy jednak upewnić się, że wygenerowane hasło nie zostaje zapisane na serwerach – co otwiera włamywaczowi furtkę do ataku metodą słownikową⁵ ⁶.

Przykładowe hasło, spełniające wymagań bezpieczeństwa:

Gliwice123!

Przykładowe hasło, spełniające wymogi bezpieczeństwa:

\$GEFjx8oa\$i

1.3.1 Istniejące rozwiązania na rynku

JOBVITE – jest platformą rekrutacyjną w postaci oprogramowania jako usługi (ang. *software as a service, w skrócie SaaS*). Serwis internetowy umożliwia klientom tworzenie zaproszeń do pracy, skierowanych do współpracowników i pracowników, w celu złożenia im przykładowo: propozycji awansu. *Jobvite* integruje się również z serwisami społecznościowymi, takimi jak *Facebook* lub *LinkedIn*. Dzięki temu jest korzystnym dostawcą rekrutacji społecznościowej dla firm, z możliwością publikowania ofert pracy i zarządzania procesem rekrutacji. Wśród klientów platformy możemy znaleźć takie podmioty jak

⁵Materiały Massachusetts Institute of Technology, „Strong Passwords”, <http://kb.mit.edu/confluence/display/istcontrib/Strong+Passwords>, (dostęp 21.11.2020)
⁶Materiały spółki CIS®, „Do Not Share Your Password”, <https://www.cisecurity.org/daily-tip/do-not-share-your-password/>, (dostęp 21.11.2020)

Schneider Electric, czy *Gamesys*. System daje możliwość przeglądania statystyk, które dotyczą np. kliknięć w wysłane zaproszenia, jak również gromadzi informacje o kandydatach.

Więcej informacji: <https://www.jobvite.com/>

Recruitee – oprogramowanie, które działa jako system śledzenia kandydatów na poszczególne stanowiska. Obejmuje system edycji witryn karier dla marki pracodawcy, wtyczkę do pozyskiwania personelu, integrację ze stroną internetową, która dotyczy zatrudnienia; synchronizację poczty elektronicznej i kalendarza. Obecnymi klientami są m.in.: *Vice*, *Usabilla* i *Vlisco*. Pracownicy działów zarządzania zasobami ludzkimi mogą dostosować proces rekrutacji na każdą posadę. Użytkownicy mają możliwość przeciągania i upuszczania profili kandydatów na różnych etapach rekrutacji, za pomocą systemu przeciągnij i upuść (ang. *drag and drop*) – takie rozwiązania możemy zauważać w systemie zarządzania projektem *Jira*, w postaci tablicy *Agile*, która umożliwia zarządzanie przepływem pracy w projektach np.: programistycznych.

Więcej informacji: <https://recruitee.com/>

Accenture Careers – wewnętrzny system korporacji *Accenture*, który wykorzystywany jest do procesów rekrutacyjnych w tymże przedsiębiorstwie. System umożliwia podgląd obecnego statusu kandydata podczas naboru, jak również informacje o terminach i etapach związanych z rekrutacją.

Więcej informacji: <https://www.accenture.com/pl-pl/careers>

1.3.2 Rozwiążanie zaproponowane w projekcie inżynierskim

Aplikacja, zrealizowana dla potrzeb projektu inżynierskiego, umożliwia rejestrację użytkowników, nadawanie im uprawnień dyrektora generalnego firmy (*CEO User*), uprawnień pracowników działów zarządzania zasobami ludzkimi (*HR User*). Członkowie działów HR posiadają uprawnienia, by tworzyć nowe ogłoszenia oraz testy, na które z kolei kandydaci, ubiegający się o pożądane stanowisko, mogą aplikować. Po pozytywnym rozpatrzeniu profilu kandydata na podstawie jego danych z życiorysu (łac. *curriculum vitae*), rekruter może przypisać użytkownikowi test, który może rozwiązać. Informacje o nowych zgłoszeniach do testu, jak i o nowych przypisanych testach, są przekazywane użytkownikom działu HR i kandydatom

w formie mailowej, jak również w postaci powiadomienia na stronie internetowej. Serwis wszystkie swoje możliwości oferuje w części webowej. Aplikacja mobilna oferuje zaś podstawowe funkcjonalności, takie jak: rozwiązywanie testów rekrutacyjnych, obsługę konta i ogłoszeń pracy.

1.4 Pomysł na rozwiązanie problemu

W celu realizacji postawionego zadania, wykorzystane zostały popularne środowiska, biblioteki oraz języki predefiniowane do rozwiązań serwerowych i aplikacyjnych od strony serwisu mobilnego i przeglądarkowego.

W realizacji zadania wykorzystana została baza danych *PostgreSQL*, której to przewagą nad konkurencyjnym rozwiązaniem w postaci *MySQL* jest ustawienie indeksowania, które pozwala na wyraźnie szybsze oraz bardziej wydajne przeszukiwanie bazy, która wraz z rozwojem aplikacji, zaczyna nabierać rozmiarów i posiada coraz to więcej rekordów. Warto również zaznaczyć, że bazy *MySQL* w najnowszych wersjach nie są do końca zgodne ze standardem *SQL*. *PostgreSQL* oferuje wysokie prędkości odczytu oraz zapisu, dzięki czemu bardzo dobrze nadaje się do dużych, rozrastających się i skomplikowanych baz danych. *PostgreSQL* to obiektowo-relacyjna baza danych, w której można korzystać z takich abstrakcji, jak dziedziczenie, czy przeciążenie funkcji. Mogą one okazać się bardzo pomocnymi opcjami w większych bazach⁷.

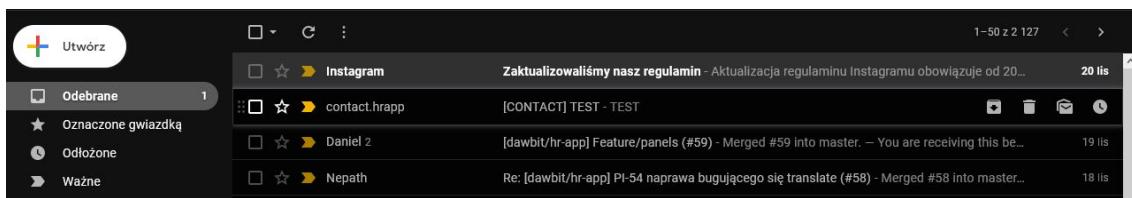
Serwer obsługiwany jest przez technologię *Java Spring Boot*, która zazwyczaj wykorzystywana jest w dużych, komercyjnych projektach, ponieważ jest szeroko znana, stabilna oraz pozwala na skalowalność aplikacji. W *Springu* istnieje wiele modułów, które są wykorzystywane do pracy z np.: chmurami obliczeniowymi, bazami danych i mikroserwisami. Dobrze współpracuje z kontenerami *Tomcat Catalina*, dzięki czemu aplikację można z łatwością przenosić na środowiska produkcyjne⁸.

W rozwoju aplikacji przeglądarkowej wykorzystana została struktura programistyczna (ang. *framework*) *Angular*, który pozwala na modułową (stworzoną

⁷K. Hristozov, „MySQL vs PostgreSQL – Choose the Right Database for Your Project”, <https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres>, (dostęp 21.11.2020)

⁸Materiały spółki VMware, „Spring Framework Documentation” <https://docs.spring.io/spring-framework/docs/5.1.0.RELEASE/spring-framework-reference/>, (dostęp 21.11.2020)

z serwisów, modułów, komponentów i fabryk) budowę stosunkowo łatwo skalowejnej aplikacji. Wykorzystywana technologia *Single Page Application* (SPA)⁹ pozwala na przeładowywanie elementów strony, bez jej całkowitego przebudowania, co daje poczucie poruszania się jak po aplikacji desktopowej. Przykładem takiego rozwiązania może być *Google Gmail* – gdzie po otrzymaniu wiadomości strona nie jest przeładowywana, tylko aktualizowana jest zawartość dwóch komponentów – listy odebranych wiadomości i licznika. Dodatkową zaletą *Angular* jest pisanie aplikacji w języku *TypeScript*, który jest nadzbiorem języka *JavaScript*. Umożliwia programować obiektywnie w prosty sposób, a także zapewnia statyczne typowanie. Każdy kod zapisany w *JavaScript* jest poprawnym programem *TypeScript* – z racji bycia jego podzbiorem¹⁰.



Rysunek 1.1: Przykład SPA w Gmail. Źródło: Opracowanie własne.

Aplikacja mobilna została stworzona z użyciem platformy programistycznej *Flutter*, która korzysta z języka *Dart*. Pozytywną cechą tego rozwiązania jest natywność stworzonej aplikacji. Część kodu można napisać za pomocą odpowiednich metod kanałowych (ang. *channel methods*) np.: w *Kotlinie* lub *Swiftcie* – dzięki czemu aplikacja może doskonale współpracować zarówno z systemem *Android* jak i *iOS*. Po przez wykorzystanie jednego silnika renderowania, framework tworzy dwie natywne aplikacje na oba środowiska – zaoszczędza to czas programistyczny, ponieważ wystarczy, że stworzy jeden kod. Zaletą jest także tzw. *Hotreload*¹¹, który umożliwia wprowadzanie zmian w działającej aplikacji bez konieczności jej całkowitego przeładowywania – wystarczy, że zostanie odświeżony zmodyfikowany komponent. Dzięki temu można ujrzeć zmiany w aplikacji już po chwili.

⁹M. Wojciechowski, „Aplikacje SPA, Angular, TypeScript”, <http://www.cs.put.poznan.pl/mwojciechowski/cdv/ria/Angular.pdf>, (dostęp 21.11.2020)

¹⁰Materiały firmy Microsoft, „TypeScript Documentation”, <https://www.typescriptlang.org/docs/>, (dostęp 21.11.2020)

¹¹Materiały firmy Google, „Flutter documentation”, <https://flutter.dev/docs>, (dostęp 21.11.2020)

2 Ogólne informacje dotyczące rekrutacji

Rekrutacja to główny element procesu zarządzania zasobami ludzkimi, najczęściej o charakterze sformalizowanym. Ma za zadanie zachęcić jak największe grono potencjalnych kandydatów do współpracy z pracodawcą. Rozumiana w ten sposób rekrutacja, prowadzi do selekcji petentów, która jest kolejnym etapem procesu wdrożenia nowych pracowników do firmy. Pracodawcy coraz częściej stosują przede wszystkim techniki, mające za zadanie odsiać ochotników, niespełniających wymogów, np. poprzez wstępne testy, jeszcze zanim przejdzie się do etapu bezpośredniej rozmowy kwalifikacyjnej, na której to z kolei coraz częściej sprawdzane są kompetencje miękkie kandydatów. Wraz z postępem techniki, przedsiębiorstwa czy też agencje specjalizujące się w przeprowadzaniu procesów rekrutacyjnych, wykorzystują do tego internet i rozmowy telefoniczne. Często podczas takiej konwersacji sprawdzana jest np.: ogólna znajomość języka obcego kandydata. Natomiast internet wykorzystywany jest do przeprowadzania rekrutacji bezpośredniej – aplikacji na konkretne stanowisko, wysłania życiorysu, wypełnienia wstępnych testów kompetencji, czy też sprawdzania znajomości potrzebnych zagadnień¹².

Taka forma procesu selekcji jest coraz częściej spotykana. Zdalne techniki przeprowadzania procesów rekrutacyjnych przybrały na znaczeniu w dobie pandemii *COVID-19*, gdzie wiele firm przeszło na rekrutację zdalną kandydatów. Obecnie jest to jedyna, a zarazem najlepsza możliwa opcja, która realnie ogranicza kontakty międzyludzkie i minimalizuje ryzyko zakażeniem wirusem *SARS-CoV-2*¹³. Wiąże się to z faktem przejścia przez przedsiębiorstwa w pewnym stopniu, lub często w całości (w zależności od możliwości), na tryb pracy zdalnej.

¹²A. Sulich, „Modele i techniki rekrutacji i selekcji realizowane przez przedsiębiorstwa w województwie dolnośląskim”, [w:] Nurty badawcze w zarządzaniu, Wydawnictwo GSP, Zgorzelec 2015

¹³Materiały firmy Grant Thornton, „Rekrutacja i selekcja nie tylko w czasach SARS-CoV-2 – Purporowy informator”,

https://grantthornton.pl/wp-content/uploads/2020/06/Covid-rekrutacja-i-selekcja_-GrantThornton_-_16062020.pdf, (dostęp 22.11.2020)

2.1 Rodzaje rekrutacji

Biorąc pod uwagę środowisko, w jakim poszukuje się nowych kandydatów, można wyznaczyć dwie główne kategorie:

- **rekrutację wewnętrzną** – potencjalnymi kandydatami na stanowisko są obecni pracownicy firmy, czyli objęcie posady poprzez awans,
- **rekrutację zewnętrzną** – potencjalnymi kandydatami na stanowisko są osoby spoza firmy.

Biorąc zaś pod uwagę rozmiar rynku pracy, w jakim prowadzi się poszukiwania, można także wyznaczyć dwie główne kategorie:

- **rekrutację wąską** – jak sama nazwa wskazuje, grono osób poszukiwanych na dane stanowisko jest wąskie. Wynika to z tego, że poszukiwani są eksperci w określonej dziedzinie. Nie może to być osoba bez doświadczenia, wymaga się od takiej persony najwyższych standardów, które są w stanie zapewnić jedynie nieliczni,
- **rekrutację szeroką** – tutaj grono osób poszukiwanych na dane stanowisko jest szerokie. Zazwyczaj poszukuje się masowo nowych pracowników najniższego szczebla. W tym przypadku często nie wymaga się doświadczenia zawodowego, ani wysokich kompetencji, przez co liczba odbiorców jest stosunkowo duża.

2.2 Etapy rekrutacji

Na proces rekrutacji składają się najczęściej poniższe etapy¹⁴:

Określenie profilu kandydata na podstawie opisu stanowiska pracy – jest to proces przygotowywania ogłoszenia oferty pracy. Proces ten uważany jest za bardzo ważny, ponieważ to od niego zależy, jacy kandydaci będą aplikować na oferowane stanowisko. Dlatego przeprowadzany jest bardzo dokładnie, gdyż należy starannie przeanalizować, a następnie jasno określić, kto i o jakich kompetencjach jest potrzebny w organizacji.

¹⁴S. Adam, „Modele i techniki rekrutacji i selekcji realizowane przez przedsiębiorstwa w województwie dolnośląskim”, [w:] Nurty badawcze w zarządzaniu., Wydawnictwo GSP, Zgorzelec 2015

Tworzenie i zamieszczanie ogłoszeń o stanowisku – etap polega na doborze odpowiednich mediów, w zależności od rodzaju rekrutacji (szeroka albo wąska). Ogłoszenie powinno zawierać: opis i zakres wymagań, które są stawiane potencjalnym pracownikom; proponowane formy świadczeń i wynagrodzenie, jakie oferuje pracodawca zatrudnionej osobie.

Tworzenie długiej listy kandydatów – preselekcja. Spośród otrzymanych aplikacji osób, które odpowiedziały na ogłoszenie i spełniają podstawowe wymagania stawiane kandydatom, odrzuca się ochotników, którzy całkowicie nie nadają się na ogłoszony wolny wakat.

Tworzenie krótkiej listy kandydatów – selekcja osób, które najlepiej spełniają określone wymagania oferty pracy.

Zastosowanie technik naboru kandydatów wobec wybranych osób – najczęściej w tym momencie ustala się czas i sposób przeprowadzania rozmowy kwalifikacyjnej. Należy również upewnić się, że dany kandydat nadal jest zainteresowany oferowanym stanowiskiem w organizacji.

Sprawdzenie referencji – często wieloetapowe. Zdarza się, że wstępna rozmowa kwalifikacyjna jest przeprowadzana z pracownikiem działu zarządzania zasobami ludzkimi, natomiast kolejne z osobami na stanowiskach kierowniczych bądź pracującymi na podobnym stanowisku.

Podjęcie decyzji o zatrudnieniu nowego pracownika – poinformowanie potencjalnego pracownika o pozytywnym albo negatywnym przejściu procesu rekrutacyjnego.

Negocjacje dotyczące warunków zawarcia stosunku o pracę – jego formy, wynagrodzenia i innych świadczeń. Najczęściej na dane stanowisko są określone przedziały płacowe. Podczas tego etapu przeprowadzane są również negocjacje konkretnej kwoty z tego zakresu.

2.3 Techniki i narzędzia rekrutacyjne

Rozmowa kwalifikacyjna (wywiad indywidualny) – właściwy etap, w którym organizacja szukająca pracownika ma bezpośredni kontakt z petentem. Podczas rozmowy kwalifikacyjnej pracodawca ustala, czy kandydat jest odpowiednią osobą na stanowisko oraz jaki dodatni wkład może wnieść do firmy¹⁵.

Ośrodek oceny (ang. *Assessment center*) – wielowymiarowy proces oceny kompetencji. Do najczęstszych przykładów wykorzystania należą: selekcja i rekrutacja pracowników (zarówno wewnętrzna, jak i zewnętrzna), audyt personalny, działania rozwojowe, a także planowanie kariery i doradztwo.

Testy wiedzy lub kompetencji – mogą przybierać przeróżne formy, np. testu jednokrotnego wyboru czy z pytaniami otwartymi. Coraz częściej, we wstępnych etapach rekrutacji, testy kompetencji przeprowadzane są z zastosowaniem metod oceny umiejętności kandydata na odległość.

Testy psychometryczne – ustandaryzowany i zobjektivizowany pomiar próbki zachowania. Pozwala na zbadanie postępowania osoby badanej w sytuacji testowej, co pozwala wywnioskować, jakie działania podejmie kandydat w sytuacjach pozatestowych. W rekrutacji na stanowiska ryzykowne, tj.: kierowca zawodowy czy pracownik służb mundurowych, takie sprawdziany są spotykane dużo częściej, a wręcz uważane są za pewną normę.

Studium przypadku – (ang. *case study*) – polega na teoretycznym wykonaniu zadania przez kandydata, najczęściej w określonym czasie. Studium przypadku ma na celu wykazanie, że kandydat posiada umiejętności, o których wspominał w swoim życiorysie, i że potrafi wykonać je w odpowiednim okresie. Zadania w studium przypadku najczęściej rozpoczynają się od słów: zastosuj, oblicz, przygotuj, narysuj, przeanalizuj itp.

Referencje – zadaniem rekrutera jest sprawdzenie autentyczności referencji posiadanych przez kandydata. Najczęściej wymagane jest przedłożenie dowodów na swoje umiejętności, w postaci uprawnień, dyplomów ukończonych kursów,

¹⁵L. Kulczycka, „Jak najlepiej zaprezentować się podczas rozmowy kwalifikacyjnej”, Wydawnictwo Wolters Kluwer Polska SA, Warszawa 2013

czy świadectw pracy. Mają one na celu uwiarygodnić przebieg dotychczasowej kariery.

Wywiady ustrukturyzowane – polegają na zadawaniu pytań, które wcześniej zostały przygotowane przez rekrutującego pracownika i zostają zadane zgodnie ze wcześniej ustaloną kolejnością. Zaletą takiego typu wywiadu jest to, że rekruter otrzyma oczekiwane informacje od kandydata (wywiad prowadzony w luźniejszej konwencji tego nie gwarantuje). Główną wadą wywiadu ustrukturyzowanego jest fakt, że selekcjoner może pominąć istotne kwestie lub nie dowie się o umiejętnościach, które kandydat posiada, a o które nie został zapytany.

Wywiady panelowe – to część rozmowy kwalifikacyjnej, w której kandydat odpowiada na pytania grupy osób, które następnie podejmują decyzję o zatrudnieniu. Zatrudniający menedżerowie korzystają z wywiadów panelowych, aby uzyskać spójny obraz kandydata oraz otrzymać natychmiastową opinię od pozostałych członków zespołu¹⁶. Wywiady panelowe zmniejszają ryzyko zatrudniania niewłaściwej osoby. Celem panelu jest podjęcie możliwie najlepszej decyzji o zatrudnieniu, biorąc pod uwagę dostępne informacje o stanowisku i kandydatach.

Dyskusje grupowe – mają na celu konfrontację kandydata z osobami zatrudnionymi na podobnym stanowisku bądź z kierownictwem. Często są częścią wywiadów panelowych.

2.4 Spojrzenie na problem rekrutacji w tekstach kultury

Dla wielu osób proces rekrutacji jest bardzo stresującym przeżyciem. Kandydaci chcą wypaść dobrze podczas rozmowy kwalifikacyjnej, przedstawić się z jak najlepszej strony. Pracodawca stara się jak najlepiej zbadać potencjalnego pracownika pod kątem jego mocnych i słabych stron. To wszystko sprawia, że u petenta rosnie niepewność. Może on obawiać się o to, czy jego kompetencje są wystarczające.

¹⁶J. Conway, R. A. Jako, D. F. DeLong, „A meta-analysis of interrater and internal consistency reliability of selection interviews”, Artykuł z „Journal of Applied Psychology”, 1995

Ludzie, działając pod wpływem stresu, mogą zapominać podstawowych informacji, związanych ze swoją osobą; dziedziną, w której pracuję, czy udzielać błędnych odpowiedzi na pytania, na które tak naprawdę znają odpowiedź. Dodatkowo, poza stresem, sam proces rekrutacyjny może być bardzo długi i składać się z wielu etapów. Osoba, które pracuje na regularnym etacie z pewnością nie miałaby większych trudności z wyżej wymienionymi problemami. Sytuacja może być zgoła odmienna w przypadku osoby aktualnie bezrobotnej, która przykładowo nie posiada płynności finansowej, jest uzależniona od rodziny, posiada zaciągnięty kredyt lub osoby, która dopiero co wkracza na rynek pracy¹⁷. Rozmowa kwalifikacyjna jest na pewno stresującą sytuacją w życiu. Na temat przebiegu rekrutacji istnieje wiele mitów, które zostały zobrazowane w różnych tekstuach kultury¹⁸:

Test – film reżyserii *Stuarta Hazeldine'a* jest opowieścią o ósemce śmiałków, przystępujących do ostatniego etapu rekrutacji w bardzo tajemniczej korporacji. Każdy z kandydatów otrzymuje jedno pytanie oraz trzy reguły, których nie może złamać. Na udzielenie odpowiedzi każdy z ochotników ma nie mniej i nie więcej niż godzinę i dwadzieścia minut. Bohaterom bardzo zależy na wygranej. Powoduje to, że w pewnym momencie przestają nad sobą panować i zaczynają zachowywać się nieludzko, agresywnie. W pewnym momencie sytuacja zaczyna wymykać się spod kontroli i już nie przypomina standardowej rekrutacji. Film w dosadny sposób opisuje tzw. „wyścig szczurów” i pogoń za pieniędzmi, która w obecnych czasach potrafi przyćmić prawdziwy obraz rzeczywistości.

Metoda – film autorstwa *Marcelo Piñeyro*. Historia przedstawia rywalizację siedmiu kandydatów, ubiegających się o stanowisko międzynarodowego kierownika w jednej z korporacji. Wybrany zostanie tylko jeden spośród nich. Każdy z ochotników reprezentuje inny rodzaj osobowości. Atmosfera panująca podczas procesu rekrutacyjnego zaczyna wzburzać w uczestnikach niepewność i wywoływać paranoję. Grupa poddawana jest serii prób i testów psychologicznych, których zadaniem jest eliminacja i wybór tylko jednego – najlepszego kandydata.

¹⁷ „Stressful experience, brain, and emotions: Developmental, genetic, and hormonal influences”, McEwen (1995)

¹⁸ „Filmy dla rekrutera”, Katarzyna Rojewska

<https://nofluffjobs.com/blog/filmy-dla-rekrutera/> (dostęp 23.11.2020)

2.5 Przepisy prawne stosowane podczas rekrutacji

Proces rekrutacyjny wiąże się z pozyskiwaniem przez potencjalnego pracodawcę danych osobowych, zawartych w dokumentach kandydata, wyrażającego chęć podjęcia zatrudnienia. Należy podkreślić, że pracodawca nie może żądać od petenta danych, które nie są potrzebne do przeprowadzenia rekrutacji, jak np.: stan cywilny, czy orientacja seksualna. Dane osobowe nie mogą być zbierane przez administratora bez powodu zgodnego z prawem, a mogą wyłącznie wtedy, kiedy są niezbędne do realizacji tego celu, jakim jest proces rekrutacji. Podsumowując – administrator danych osobowych podczas rekrutacji nie może zbierać informacji, które nie są przydatne w procesie rekrutacyjnym¹⁹ ²⁰, chyba że kandydat zrobi to dobrowolnie i za specjalnym oświadczeniem – nawet w takiej sytuacji, dodatkowe informacje podane przez kandydata nie mogą wpływać na przebieg rekrutacji (np.: kobieta, nieplanująca posiadania potomstwa, nie może stać na uprzywilejowanej pozycji w stosunku do kandydatki, która ma zamiar w najbliższym czasie zajść w ciążę – pomimo tego, że pracodawca może widzieć w tym korzyści, jak np. dostępność pracownika, czy brak konieczności wypłacania świadczenia podczas urlopu macierzyńskiego). Przyszły pracodawca może jednak żądać przedłożenia dodatkowych, określonych prawem dokumentów czy kwalifikacji, jeśli wynika to z konkretnego charakteru pracy. Takim dokumentem jest np. zaświadczenie o niekaralności – u kandydata do służb mundurowych czy pracownika sądu, prawa jazdy odpowiedniej kategorii u kierowcy, czy uprawnień do obsługi konkretnej maszyny u operatora. Przyszły pracodawca nie ma prawa żądać od kandydata informacji o toczących się i niezakończonych postępowaniach karnych oraz o ich obecnym przebiegu. Dane tego typu nie mieszczą się bowiem w katalogu informacji możliwych do przetwarzania w procesie rekrutacji, gdyż pomimo prawnej możliwości żądania informacji o karalności w określonych przypadkach, postępowanie karne nie musi doprowadzić do prawomocnego skazania danej osoby.

¹⁹Rozporządzenie Parlamentu Europejskiego i Rady (UE) 2016/679 z dnia 27 kwietnia 2016 r. w sprawie ochrony osób fizycznych w związku z przetwarzaniem danych osobowych i w sprawie swobodnego przepływu takich danych oraz uchylenia dyrektywy 95/46/WE (ogólne rozporządzenie o ochronie danych)

²⁰Rozporządzenie Ministra Rodziny, Pracy i Polityki Społecznej z dnia 10 grudnia 2018 r. w sprawie dokumentacji pracowniczej (Dz. U. poz. 2369)

Zgodnie z art. 221²¹ § 1 Kodeksu pracy²¹, pracodawca może żądać od kandydata do pracy podania danych osobowych, które obejmują poniższe informacje:

- imię (imiona i nazwisko),
- datę urodzenia,
- dane kontaktowe wskazane przez kandydata,
- wykształcenie,
- kwalifikacje zawodowe,
- przebieg dotychczasowego zatrudnienia.

Prowadzenie dokumentacji pracowniczej (po zatrudnieniu) odbywa się na podstawie odrębnych przepisów prawa pracy. Rozporządzenie Ministra Rodziny Pracy i Polityki Społecznej z 10 grudnia 2018 r. określa zakres oraz sposób prowadzenia i przechowywania akt pracowników. Nie można zatem mówić o dobrowolności podczas przetwarzania danych o zatrudnionych pracownikach, ponieważ ta kwestia jest regulowana przez powyższe rozporządzenie.

Pracodawca musi mieć na uwadze możliwość wycieku danych osobowych oraz musi stosować środki zapobiegawcze, jak np. przetrzymywanie informacji w systemach elektronicznych o nienagannym poziomie zabezpieczeń i autoryzacji, czy zabezpieczonego fizycznego archiwum. W przypadku wycieku danych, pracodawca musi niezwłocznie (w przeciągu 72 godzin od momentu wykrycia incydentu) poinformować osoby, których dane zostały wykradzione. Dodatkowo musi zgłosić ten fakt do odpowiedniego organu nadzorczego – wskazuje na to Art. 33 RODO. Również na pracowniku, który zauważy wyciek, ciąży powinność niezwłocznego poinformowania o tym pracodawcy.

²¹Ustawa z dnia 26 czerwca 1974 r.- Kodeks Pracy (Dz. U. z 2019 r. poz. 1040, 1043 i 1495)

Podsumowując, rekrutacja to bardzo trudny i złożony proces, do którego różne przedsiębiorstwa podchodzą w zindywidualizowany sposób, dodatkowo pod uwagę muszą byćbrane często kwestie prawne, związane zarówno z kodeksem karnym, jak i *RODO*. Sam proces rekrutacji bywa stresującym przeżyciem dla kandydatów, z tego też powodu jest to częsty motyw w przeróżnych teksthach kultury np. produkcjach filmowych. Brak tutaj standardu, który zapewniałby klarowność tego procesu od samego początku. Wieloplatformowa aplikacja, która wspierałaby przeprowadzanie wspomnianego procesu, z pewnością byłaby bardzo przydatna w wielu średnich firmach, jak i korporacjach. Przyczyniłaby się także do redukcji kosztów, związanych z prowadzeniem działalności gospodarczej, poprzez zmniejszenie nakładu czasowego potrzebnego na przeprowadzanie procesu rekrutacji.

3 Założenia projektowe

W rozdziale zostały omówione w zwięzły sposób początkowe założenia, dotyczące realizacji projektu inżynierskiego. Po wstępnej analizie problemu, zostały sporządzone założenia wyjściowe, które dotyczą poszczególnych części serwisu.

3.1 Wstępne założenia dotyczące serwera i bazy danych

Podczas rozważania założeń dotyczących części zaplecza aplikacji, zostały wysegregowane następujące punkty, które projekt powinien spełniać:

- kod aplikacji powinien działać stabilnie, szybko i sprawnie (najlepiej skorzystać z popularnego i rozwijanego frameworku, co zapewni dodatkowo dużą bazę wsparcia technicznego),
- odporność na skomplikowane zapytania,
- szybkość odpowiedzi dla dużych ilości informacji,
- serwer i baza powinny być łatwe w rozbudowaniu,
- wsparcie dla wzorców projektowych (np. *Model View Controller*),
- operacje głównie na obiektach,
- łatwość konfiguracji,
- tworzenie tabel i zależności w bazie danych na poziomie kodu programu,
- wsparcie dla modułów bezpieczeństwa.

Na podstawie powyższych wymagań, jako środowisko dla *backendu* zostało wybrane połączenie bazy danych *PostgreSQL* oraz serwera *Spring*. Baza danych *PGSQL* jest uznawana w środowisku programistycznym za najlepsze rozwiązanie od strony baz danych dla aplikacji, która ma potencjał się rozrastać. *Spring* natomiast jest technologią, która na rynku jest od osiemnastu lat i nadal jest rozwijana, posiada wsparcie oraz uznawana jest za bezpieczną.

Następnie zostały wykonane szkice bazy danych, które determinują również budowę aplikacji serwerowej.

Według wstępnych założeń, w bazie danych powinny znaleźć się następujące tabele:

- **account_types** (rodzaje kont) w relacji jeden do wielu, z tabelą *users*,
- **answers** (odpowiedzi) – zawierająca możliwe odpowiedzi na konkretne pytania, w relacji jeden do wielu z tabelą *questions*,
- **ceos** (właściciele firm) – zawierająca informację o właścicielach firm, w relacji jeden do jednego z tabelą *users*, oraz wiele do jednego z tabelą *companies* – jako założyciele,
- **companies** (firmy) – zawierająca informacje o firmach, takie jak np. adres, czy nazwę,
- **company_pictures** (zdjęcia firmy) – przeznaczona do przechowywania informacji o multimedialach, dotyczących firmy, takich jak zdjęcia budynku, czy logo. W kolumnie o typie tekstowym, przechowywana będzie część odnośnika do serwera *FTP*, w ten sposób większość zawartości bazy danych nie będzie zapełniona przez dane typu *blob*, a w celu fizycznego przechowywania plików na dysku, zostanie wykorzystany serwer *FTP*,
- **hr_user** (użytkownicy działu zarządzania zasobami ludzkimi) – powiązania z tabelą *companies* w relacji wiele do jednego oraz z tabelą *users* w relacji jeden do jednego,
- **questions** (pytania) – zawierająca treści pytań do poszczególnych testów, w relacji wiele do jednego z tabelą *tests*,
- **test_participant** (przypisanie do testu) – zawierająca informacje o przypisanych kodach do testów dla poszczególnych użytkowników, aktualny numer pytania, na którym jest użytkownik, czas rozpoczęcia oraz zakończenia testu. W relacji wiele do jednego z tabelą *tests*, wiele do jednego z tabelą *users*,
- **tests** (testy) – zawierająca informacje o poszczególnych testach, utworzonych przez pracowników: czy test jest aktywny, czy jest możliwość cofania pytań, w relacji wiele do jeden z tabelą *companies*,

- **user_answers** (odpowiedzi użytkowników) – zapisują się w niej informacje o udzielonych odpowiedziach. W relacji wiele do jednego lub jeden do jednego z tabelą *questions* – zależne od dalszych ustaleń, wiele do jednego z tabelą *tests* oraz wiele do jednego z tabelą *users*,
- **users** (użytkownicy) – zawierająca informacje o zarejestrowanych użytkownikach, hasła użytkowników zaszyfrowane są przez *bcrypt*.

Natomiast podstawowe punkty końcowe powinny umożliwiać:

- logowanie oraz rejestrację użytkowników,
- dodawanie ogłoszeń przez pracowników firmy,
- dodawanie pracowników do firmy przez właściciela,
- tworzenie oraz rozwiązywanie testów,
- tworzenie oraz aplikowanie na ogłoszenia.

3.2 Wstępne założenia dotyczące aplikacji webowej

Przy omawianiu założeń dotyczących aplikacji, zespół brał pod uwagę takie wymagania, jak:

- dostępność źródeł i pomocy dla programisty,
- aktualność platformy programistycznej,
- modułowość,
- responsywność,
- możliwość obiektowego tworzenia aplikacji, zagnieżdżania komponentów,
- prosta obsługa nagłówków (w tym „doczepianie” informacji o tokenie).

W trakcie analizy podjęto decyzję o wyborze środowiska programistycznego *Angular*, który wykorzystuje język *TypeScript*, ponieważ doskonale spełnia on nakreślone wcześniej wymagania. Posiada bardzo przystępny sposób tworzenia komponentów, jest ciągle rozwijany, a także – dzięki *Node Package Module* – posiada dostęp

do wielu bezpłatnych bibliotek, pomagających rozwijać aplikację. Dodatkowo zespół posiada w nim największe doświadczenie, spośród wszystkich popularnych frameworków frontendowych.

Zespół w trakcie burzy mózgów zdecydował o tym, co powinna oferować aplikacja przeglądarkowa. Do tych wymagań należą:

- logowanie i rejestracja użytkowników,
- uwierzytelnienie na podstawie tokenu,
- rozróżnienie i dostosowanie interfejsu na podstawie rodzaju konta (administrator, zwykły użytkownik, pracownik działu zarządzania zasobami ludzkimi, przedstawiciel firmy),
- możliwość przeglądania listy ogłoszeń,
- aplikacja przygotowana w minimum dwóch językach (polskim i angielskim),
- możliwość tworzenia testów rekrutacyjnych przed pracowników firmą,
- możliwość dodania firmy,
- możliwość rozwiązywania testów rekrutacyjnych.

3.3 Wstępne założenia dotyczące aplikacji mobilnej

Zespół postawił na framework *Flutter*, korzystający z języka *Dart*. Czynniki, które o tym zadecydowały, to m.in.:

- tworzenie aplikacji na dwa systemy (*iOS* i *Android*) równolegle,
- prostota budowy aplikacji,
- technologia o dużym wsparciu społeczności.

Aplikacja mobilna jest mniej rozbudowaną wersją tej przygotowanej na przeglądarkę, powinna umożliwiać takie działania jak:

- logowanie i rejestracja zwykłego użytkownika,

- powiadomienia o przypisanych testach,
- możliwość rozwiązywania testu.

Wszystkie wymienione w tym rozdziale założenia nie są założeniami końcowymi, w trakcie rozwoju aplikacji mogły ulec zmianie – zostać zmodyfikowane, usunięte lub też mogły pojawić się nowe.

4 Metodyka

Rozdział ten przedstawia kolejne trzy fazy realizacji, które zostały wyszczególnione podczas tworzenia projektu inżynierskiego.

4.1 Faza planowania

Najważniejszym, a zarazem najbardziej oczywistym punktem, był wybór tematyki pracy. Po ustaleniu tej kwestii, nastąpiła faza planowania.

Dla łatwiejszego zobrazowania problemu, został sporządzony prosty szablon aplikacji webowej oraz mobilnej. Dzięki temu łatwiej było określić, jakie komponenty należy powołać do życia oraz jakie punkty końcowe trzeba stworzyć do ich obsługi. Po spisaniu wstępnych zadań, nastąpała pora na ich podział pomiędzy członków projektowych oraz śledzenie postępów. Pozostał jeszcze wybór narzędzia do monitorowania efektów. Rozważane były dwie opcje, należące do firmy *Atlassian*. Pierwszym z nich było *Trello* – aplikacja online do tworzenia list w stylu *Kanban*. Jest to propozycja sprawdzona przy mniejszych projektach i sprawuje się relatywnie dobrze, jednak przegrała ona z drugim kandydatem – *Jira*. Zarówno jedno, jak i drugie oprogramowanie pozwala na generowanie zadań, dzielenie ich na dowolne sekcje, przypisywanie wykonawców, ustalanie terminów oraz ustawianie statusów przebiegu realizacji. *Trello* nie zostało wybrane przez członków zespołu z kilku przyczyn. Pierwszą z nich było to, że darmowa wersja jego rywala posiadała wtyczkę, dzięki której można synchronizować system kontroli wersji *Git* wraz z tablicą. Wystarczy tylko, że nazwy zatwierdzeń zaczynają się od identyfikatora zadania. Wtedy w systemie zapisywane są informacje o kolejnych rozgałęzieniach i zatwierdzeniach zmian dla wybranego zadania. Dzięki temu można w prosty sposób sprawdzić z poziomu *Jiry* postęp prac, bez potrzeby manualnego doszukiwania się aktualizacji kodu. Kolejną przyczyną była chęć nauki czegoś, co jest realnie wykorzystywane przy dużych projektach w świecie biznesu. Co za tym idzie, można podnieść zarówno swoje kompetencje, jak i samą pracę inżynierską na wyższy poziom standardów. Ostatecznym punktem, który zadecydował o porażce *Trello*, był fakt, że nie pozwala ono na rejestrowanie czasu poświęconego na realizację zadania. Dzięki tej funkcjonalności można

z łatwością stwierdzić, które zagadnienia sprawiają najwięcej problemów i gdzie przydałby się pomoc innego członka projektu.

Ostatnim punktem, a zarazem najbardziej oczywistym, było zaadresowanie kwestii systemu kontroli wersji. Wybór dość szybko padł na platformę *GitHub*, głównie ze względu na jej znajomość i niezawodność. Pomimo tego, że chyba nie ma na świecie programisty, który by choć raz nie skorzystał z tego oprogramowania i większość uważa je za coś oczywistego, to nie można pomniejszać jego roli, którą odegrało w pracy. Rozwiążanie to stanowi swoisty szkielet całego projektu i bez niego nie byłoby szans na ukończenie kodu w zadanym terminie.

4.2 Wczesna faza implementacji

Pierwszym implementacyjnym krokiem projektu było powołanie do życia strony serwero–bazodanowej. Następną rzeczą, którą należało zrobić, było przetestowanie stworzonego zaplecza. W tym celu użyto oprogramowanie *Postman*. W skrócie, pozwala ono na generowanie i wysyłanie zapytań do *API* (ang. *application programming interface*) oraz wyświetlanie zwracanych odpowiedzi. Po upewnieniu się, że podstawowe punkty końcowe działają, nastąpiła faza implementacji aplikacji internetowej. Połączenie frontendu z backendem nie przysporzyło większych problemów i wszystko zadziałało tak, jak powinno. Zostały utworzone podstawowe panele i komponenty, które, choć puste, były bardzo ważne, bo to właśnie one uwidaczniały, jak wiele pracy jeszcze przed zespołem. Na tym etapie został utworzony szkielet aplikacji, wokół którego można było budować resztę ciała. Część mobilna dostała niższy priorytet i została przeznaczona na dalsze etapy rozwoju.

4.3 Zaawansowana faza implementacji

Mając gotową bazę, pozostało uzupełniać ją o funkcjonalności. W pierwszej kolejności należało obrać za cel konkretny komponent aplikacji. Następnie w pełnym składzie dokładnie przedyskutowano zasadę jego działania. Po omówieniu wszystkich za i przeciw, utworzono odpowiednie punkty końcowe w części serwerowej, a następnie obsłużono je po stronie aplikacji internetowej. Jeśli wszystko zadziałało zgodnie z planem, to cały ten proces powtarzany był od nowa, a jeśli nie, to należało dojść do tego, dlaczego założenia się nie sprawdziły i zmienić je na właściwe.

4.4 Faza stabilizacji

Faza stabilizacji nastąpiła po tym, gdy wszystkie planowane funkcjonalności zostały oprogramowane. To właśnie wtedy przyszedł czas na wyszukiwanie i zwalczanie wszelkich błędów w kodzie i niepożądanych zachowań aplikacji. Poza szlifowaniem działania serwisu, należało przyjąć wspólne założenia, co do finalnego stylu programu przeglądarkowego i mobilnego. Tak, aby były one w miarę możliwości jednolite, co do szaty graficznej oraz designu.

Poza pracami nad kodem, zostały także rozpoczęte intensywne prace nad dokumentacją, którą – według ustaleń – należało skończyć do połowy grudnia. Termin ten został wybrany tak, aby mieć odpowiedni bufor czasowy na wszelkie poprawki i niespodziewane problemy, o których warto byłoby wspomnieć.

Podczas prac nad projektem można było zauważać i wyróżnić kilka faz, w których projekt się znajdował. Były nimi:

- faza planowania – podczas której zespół stworzył pierwsze szkice projektu,
- wczesna faza implementacji – gdzie zespół wprowadził podstawowy model bazy danych; zaawansowana faza implementacji – skupiająca się przede wszystkim na funkcjonalnościach części klienckiej i punktach końcowych części serwerowej,
- faza stabilizacji – czyli ostatnie poprawki i testowanie projektu.

Podzielenie pracy na tego typu etapy sprawiło, że aplikacja nie była pisana chaotycznie, a każdy z członków zespołu łatwiej odnajdywał się w swoich zadaniach i wiedział, w jakim tempie posuwają się prace w perspektywie pozostałego czasu.

38:1012662495

5 Technologie

Rozdział ten został poświęcony opisom technologii, które zostały wykorzystane podczas tworzenia projektu inżynierskiego.

5.1 Serwer i baza danych

Głównymi technologiami, użytymi w tworzeniu zaplecza serwisu, były m.in. *PostgreSQL* oraz *SpringBoot*.

5.1.1 PostgreSQL

Jest to otwarty (ang. *open source*), obiektowo-relacyjny system zarządzania bazami danych. W okresie ponad trzydziestu lat zyskał na reputacji dzięki swojej wydajności, niezawodności, spójności danych i ich poprawności. Poza jego charakterystyką, ważne jest także to, że projekt ten jest rozwijany przez ochotników z całego świata, przez co jest bezpłatny.

Więcej informacji: <https://www.postgresql.org/>

5.1.2 SpringBoot

Wydajne i proste narzędzie, ułatwiające tworzenie aplikacji opartych o mikroserwisy w *Javie*. Dzięki niemu programiści nie muszą, między innym, zajmować się zachowaniem prawidłowych zależności pomiędzy dodawanymi bibliotekami, przez co unika się problemu niekompatybilności między nimi podczas konfigurowania *API*. Odchodzi również wymóg posiadania zewnętrznego serwera, na który trzeba by było wgrywać aplikację, a później weryfikować jej poprawność. Wszystko to przekłada się na bardziej efektywne pisanie kodu, oszczędność czasu i łatwość obsługi.

Głównymi zaletami *SpringBoota* są:

- budowanie samodzielnej aplikacji (ang. *stand-alone application* za pomocą tak zwanych starterów, czyli zbioru wszystkich niezbędnych konfiguracji oraz bibliotek,
- brak wymogu konfiguracji *XML* (ang. *Extensible Markup Language*),

- wbudowany serwer i inne niezbędne zależności i komponenty, potrzebne do uruchomienia aplikacji.

Więcej informacji: <https://spring.io/projects/spring-boot>

5.1.3 Spring Security

Potężna i wysoce konfigurowalna struktura (ang. *framework*), wykorzystywana do uwierzytelniania i kontroli dostępu. Jest uznawana jako standard przy zabezpieczeniu aplikacji, bazujących na *Springu*. Jej główną zaletą jest łatwość w użyciu oraz rozwoju, którego należy dokonać, aby sprostać wymaganiom projektowym.

5.1.4 Spring Data JPA

JPA (ang. *Java Persistence API*) to w skrócie specyfikacja, która określa mechanizmy, pozwalające na zarządzanie zawartością bazy danych poprzez obiekty w *Javie*. *Spring Data JPA* pozwala na używanie zapytań natywnych, które często możliwościami przewyższają zapytania wbudowane i własne. Dzięki niej można wprowadzić między innymi do adnotacji @Querry standardowe zapytanie *SQL* i operować nim na tabelach, zamiast na encjach.

Oczywiście dobrodziejstwa, płynące z korzystania z tej technologii, są o wiele większe, ale w tym projekcie była użyta głównie dla powyższych zastosowań.

Więcej informacji: <https://spring.io/projects/spring-data-jpa>

5.2 Aplikacja webowa

Podczas budowy i rozwoju aplikacji dostępnej z poziomu przeglądarki, zespół wykorzystał technologię *Angular*, która to wykorzystuje w swojej implementacji język *TypeScript*.

5.2.1 Angular

To otwarta platforma i framework, rozwijany przez *Google*, służący do tworzenia mobilnych i desktopowych aplikacji internetowych. Dzięki niemu oszczędza się dużo czasu, gdyż nie trzeba rozwijać oprogramowania na różne platformy. Dodatkowymi zaletami są na pewno prędkość działania, niezawodność i ogrom wtyczek do popularnych środowisk programistycznych, ułatwiających pracę z kodem.

Więcej informacji: <https://angular.io/>

5.2.2 MDBootstrap

Material Design for Bootstrap to jeden z najpopularniejszych frameworków, współpracujących z *Angularem*. Udostępnia ona zestaw elementów interfejsu użytkownika (ang. *user interface*), które są stworzone z myślą o responsywności, czyli wykorzystaniu ich na urządzeniach mobilnych. Największą zaletą tego rozwiązania jest to, że jest darmowe do użytku własnego i komercyjnego oraz to, że korzysta z *Google Material Design*.

Więcej informacji: <https://mdbootstrap.com/>

5.3 Aplikacja mobilna

Zespół, w ramach implementacji systemu na urządzeniach mobilnych, postawił na technologię *Flutter*, wykorzystującą język *Dart*.

5.3.1 Flutter

Otwarty (ang. *open source*) zestaw deweloperski oprogramowania interfejsu użytkownika (ang. *user interface*). Służy do budowania aplikacji internetowych, na takie systemy, jak: *Android*, *iOS*, *Linux*, *Mac*, *Windows* oraz *Google Fuchsia*. Aplikacje, na każdy z tych systemów, można stworzyć z jednego kodu źródłowego.

Więcej informacji: <https://flutter.dev/>

W powyższym rozdziale zostały omówione technologie użyte przez zespół w implementacji projektu. Są nimi głównie platformy programistyczne, takie jak *Flutter*, *Angular* czy *SpringBoot*. Pierwsza z nich to technologia, która pozwala użytkownikowi na pisanie aplikacji zarówno na mobilny system operacyjny *Android*, jak i *iOS*. To, jak i dynamiczny rozwój platformy oraz ciekawe podejście do samego budowania widoków, sprawiło, że grupa projektowa zdecydowała się na wybranie tej technologii. *Angular* został użyty w aplikacji webowej, bowiem doskonale sprawdza się on w budowaniu wielokomponentowych aplikacji. Dodatkowo dobra znajomość tej technologii przez jednego z członków zespołu sprawiła, że był to doskonały wybór, ponieważ potrafił on pokierować pracami nad częścią przeglądarkową aplikacji. Pewne doświadczenie w języku *Java* nakierowało zespół na zastosowanie platformy programistycznej *Spring*,

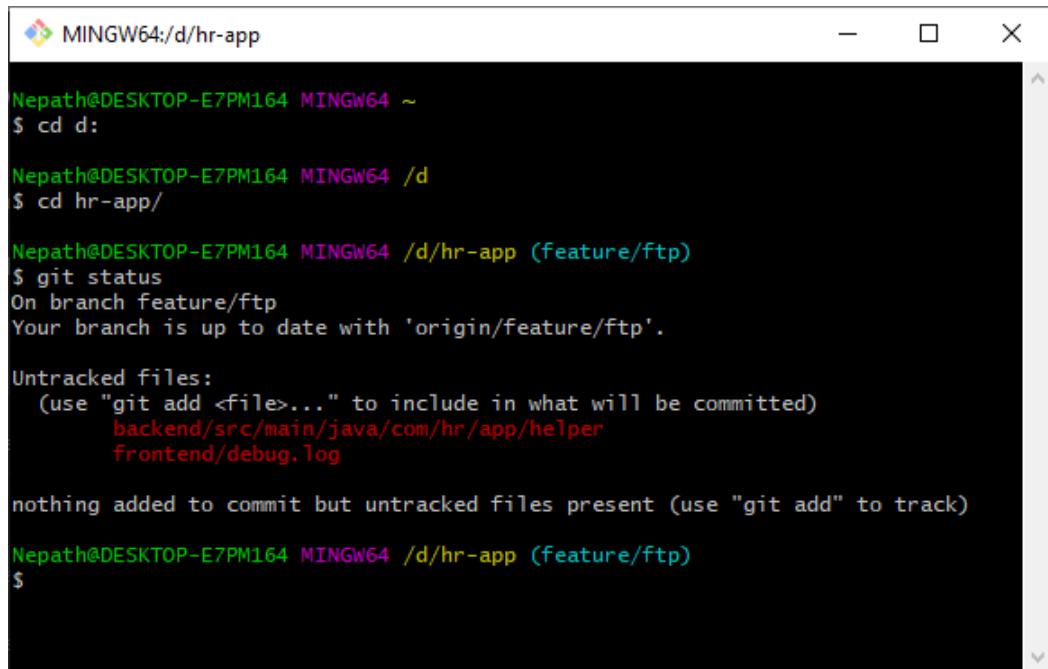
co okazało się dobrą decyzją, gdyż wszelkie wbudowane ułatwienia znaczco przyspieszyły prace nad projektem. Technologia ta również znakomicie współpracuje z bazą danych *PostgreSQL*.

6 Narzędzia

Projekt nie zostałby zrealizowany bez użycia niezbędnych narzędzi wspomagających, które to – w mniejszym bądź większym stopniu – przyczyniły się do komfortu pisania, testowania, wyglądu aplikacji, czy nawet komunikacji między poszczególnymi członkami zespołu.

6.1 System kontroli wersji

System kontroli wersji (ang. *VCS* – *version control system*), pozwala programistom w przyjemny i użyteczny sposób sprawdzać aktualny stan projektu, śledzić zmiany, które zostały wprowadzone na przestrzeni wielu miesięcy lub w przypadku błędów najnowszej wersji – powrotu do wcześniejszego wydania. Każda osoba, która jest członkiem zespołu prowadzącego prace nad danym zleceniem, ma dostęp do repozytorium, które to zawiera wszystkie ww. dane i opcje.



The screenshot shows a terminal window titled 'MINGW64:/d/hr-app'. The terminal output is as follows:

```
Nepath@DESKTOP-E7PM164 MINGW64 ~
$ cd d:
Nepath@DESKTOP-E7PM164 MINGW64 /d
$ cd hr-app/
Nepath@DESKTOP-E7PM164 MINGW64 /d/hr-app (feature/ftp)
$ git status
On branch feature/ftp
Your branch is up to date with 'origin/feature/ftp'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    backend/src/main/java/com/hr/app/helper
    frontend/debug.log

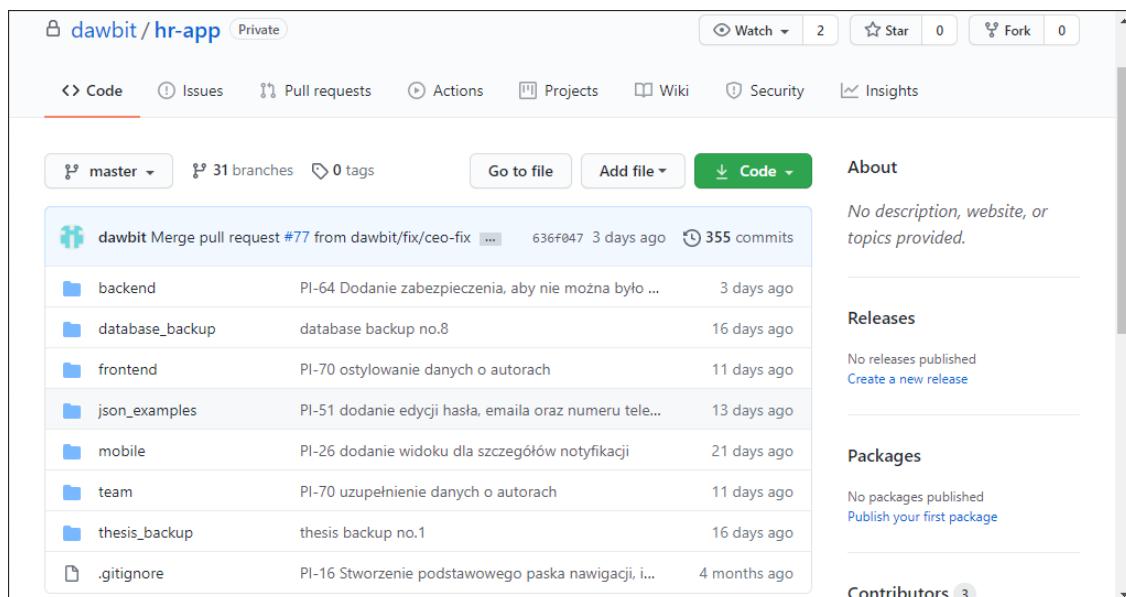
nothing added to commit but untracked files present (use "git add" to track)

Nepath@DESKTOP-E7PM164 MINGW64 /d/hr-app (feature/ftp)
$
```

Rysunek 6.1: Widok programu *GitBash*. Źródło: Opracowanie własne.

Nieodłączną częścią życia programisty, jest stałe pisanie nowego kodu. Jednak, biorąc pod uwagę to, że że nikt nie jest doskonały, może się zdarzyć, że ten ominie jakiś ważny aspekt, który miał zostać zaimplementowany. Problemy pojawiają się częściej, gdy programistą jest osoba na stanowisku młodszego dewelopera (ang. *Junior Developer*). Osoby niedoświadczane mogą produkować znacznie więcej błędów, które to muszą zostać sprawdzone przez bardziej doświadczonych programistów. Te wszystkie problemy sprawiają, że nie można od razu wprowadzić nowych zmian do wersji użytkowej. Rozwiązaniem są gałęzie (ang. *branch*). Za ich pomocą można skopiować główną wersję aplikacji i, nie naruszając głównej gałęzi, ulepszać kod. Następnie, gdy już dane zadanie zostało zakończone, należy wykonać tzw. *pull request*. Funkcja ta pozwala na wdrożenie zaaplikowanych zmian do głównej gałęzi projektu.

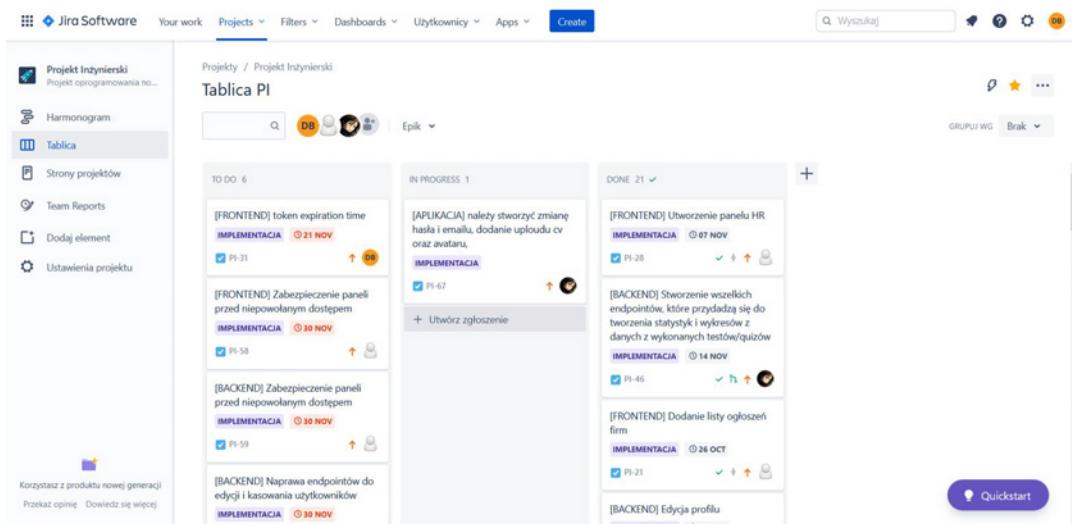
W projekcie został użyty *GitHub*, czyli hosting, wykorzystujący system kontroli wersji *Git*. Natomiast, jako program wspomagający wszelkiego rodzaju czynności, został użyty *GitBash*, jak i wbudowane opcje pomocnicze w środowiskach programistycznych, opisanych w dalszej części pracy.



Rysunek 6.2: Widok platformy *GitHub*. Źródło: Opracowanie własne.

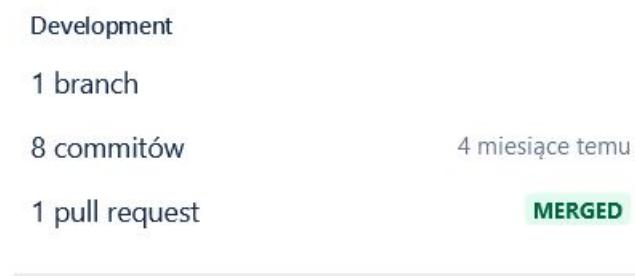
6.2 Oprogramowanie do zarządzania projektem

Podczas wykonywania danego projektu, niezbędną rzeczą jest przypisywanie poszczególnym członkom zespołu odpowiednich zadań. Wprowadza to jasny podział obowiązków w zespole.

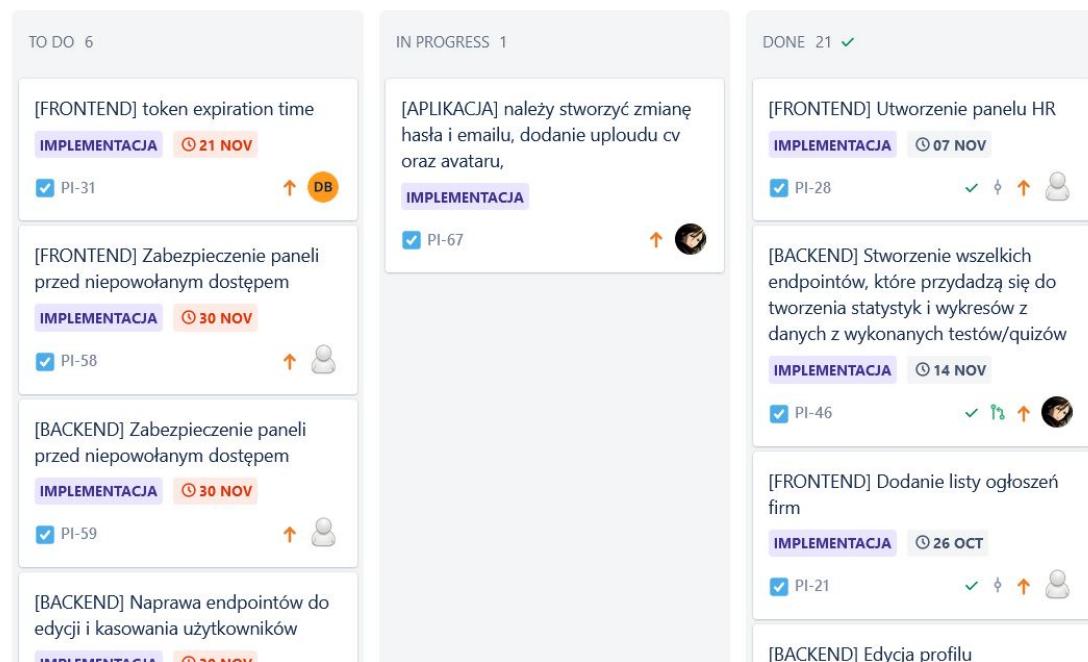


Rysunek 6.3: Główny widok *Jira*. Źródło: Opracowanie własne.

Dzięki systemowi *Jira*, każdy z członków zespołu wie, jakie zadania zostały mu przydzielone oraz ile czasu ma na ich wykonanie. System ten pozwala na kontrolę przepływu pracy w projekcie, członkowie mają wgląd do informacji o tym, jakie zadania są aktualnie wykonywane, jakie zostały zamknięte oraz jakie czekają w kolejce.



Rysunek 6.4: *Jira* – Szczegóły zadania pobranego z wtyczki *GitHub*. Źródło: Opracowanie własne.



Rysunek 6.5: *Jira* – Tablica Agile. Źródło: Opracowanie własne.

Jak widać *Jira* to zaawansowane narzędzie, dzięki któremu zyskujemy pełną kontrolę nad realizacją zadań projektowych.

The screenshot shows a Jira task card for "[BACKEND] Edycja profilu".

Opis: Należy umożliwić użytkownikowi edycję własnego profilu

- podstawowych danych
- hasła
- upload / wyświetlenie aktualnego CV

Aktynwość: Pokaż: Komentarze (aktualnie), Historia, Dziennik pracy

Komentarze:

- DB Dawid Bitner 14 listopada 2020, 11:34
Powyższe podpunkty najlepiej zrealizować na osobno wydzielonych (3) endpointach

Działalność:

- Dodaj komentarz... (input field)
- Fachowa porada: naciśnij M, aby skomentować (text)

Informacje o zadaniu:

Gotowe	Medium
Priorytet	Medium
Osoba przypisana	Mateusz Kowol
Osoba zgłoszająca	DB Dawid Bitner
Etykiety	Brak
Data zgłoszenia	07 lis 2020, 6:06 PM
Rejestrowanie czasu	Nie zarejestrowano czasu
Start date	Brak
Termin	2020/11/17
Development	2 commity 8 dni temu

Rysunek 6.6: *Jira* – Wygląd przykładowego zadania. Źródło: Opracowanie własne.

Rozwój PI-7

Przekaż opinię

[Gałąź](#) [Zatwierdzenia](#) [Żądania ściągnięcia kodu](#) [Kompilacje](#) [Wdrożenia](#) [Flagi funkcji](#)

hr-app (GitHub) [Pokaż wszystkie pliki](#)

Autor	Zatwierdzenie	Wiadomość	Data	Pliki
e84b63		PI-7 Wyłączenie maksymalnej długości dla hasła (BCrypt)	23.07.2020	1 plik
1f7dd3		PI-7 Rejestracja użytkowników, zwracanie odpowiedzi w postaci obiektu	23.07.2020	4 pliki
0cb7ad		PI-7 Rejestracja użytkowników	22.07.2020	5 plików
4ac700		Merge pull request #4 from dawbit/feature/spring-security PI-7 Autoryzacja użytk...	21.07.2020	10 plików
216220		PI-7 Usprawnienie pobierania credentiali	21.07.2020	2 pliki
ca3769		PI-7 Usprawnienie pobierania credentiali	21.07.2020	2 pliki
c8cfa0		PI-7 Skonfigurowanie i umożliwienie pozyskiwania credentiali - po zalogowaniu, je...	21.07.2020	3 pliki
318570		PI-7 stworzenie podstawowej autoryzacji	20.07.2020	10 plików

Rysunek 6.7: Jira – Szczegóły przykładowego zadania. Źródło: Opracowanie własne.

Dzięki zakładce zatwierdzenia można w przyjazny dla użytkownika sposób, w formie listy, podejrzeć zatwierdzone postępy w realizacji danego zadania.

Issue	Total Issue	Aug, 2020				Sep, 2020				Oct, 2020				Nov, 2020		
		27-02 31	03-09 32	10-16 33	17-23 34	24-30 35	31-06 36	07-13 37	14-20 38	21-27 39	28-04 40	05-11 41	12-18 42	19-25 43	26-01 44	02-0 45
<input checked="" type="checkbox"/> PI-11	12h		12h													
<input checked="" type="checkbox"/> PI-12	8h					8h										
<input checked="" type="checkbox"/> PI-14	27h												16h	11h		
<input checked="" type="checkbox"/> PI-15	3h			3h												
<input checked="" type="checkbox"/> PI-16	69.5h		4.5h	4h				8h	13h		19h	18h		3h		
<input checked="" type="checkbox"/> PI-19	2h											2h				
<input checked="" type="checkbox"/> PI-21	6.08h											2h	4h	0.08		
<input checked="" type="checkbox"/> PI-22	6h												6h			
<input checked="" type="checkbox"/> PI-23	8.5h											8h		0.5h		
<input checked="" type="checkbox"/> PI-25	19h												17h	2h		
<input checked="" type="checkbox"/> PI-26	43h															

Rysunek 6.8: Jira – Wygląd raportu z użyciem wtyczki Team Reports, odnośnie przepracowanego czasu nad zadaniami. Źródło: Opracowanie własne.

Jira to łatwy sposób na kontrolę czasu poświęconego na zadania. Z tabeli na rysunku 6.8 można na przykład wywnioskować, że zadanie PI-16 było najbardziej skomplikowanym i wymagało dużo uwagi od zespołu projektowego.

Issue	Total Issue	Nov, 2020		Dec, 2020		
		30 Mo	01 Tu	02 We	03 Th	
<input checked="" type="checkbox"/> PI-46	6h			6h		
<input checked="" type="checkbox"/> PI-68	2h		2h			
Total	8h		2h	6h		

Rysunek 6.9: *Jira* – Wygląd raportu w skróconej wersji. Źródło: Opracowanie własne.

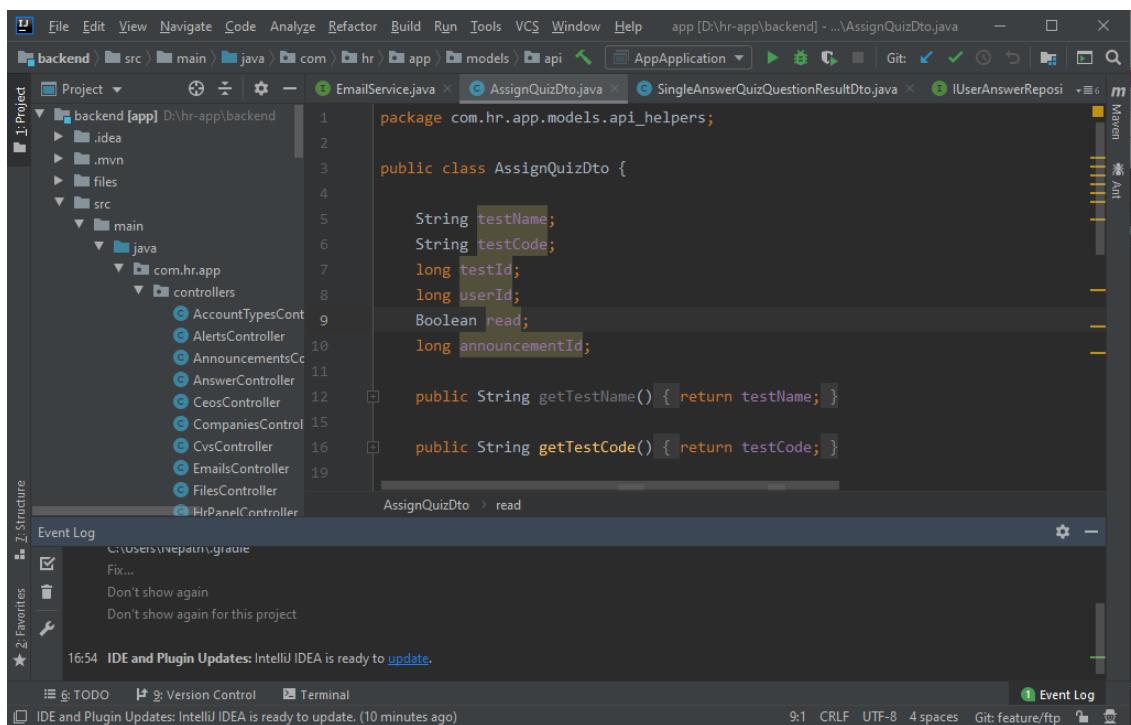
Work Log Author	Total Work Log Author	Aug 2020	Sep 2020	Oct 2020	Nov 2020	Dec 2020
 Daniel B	102.83h	10h	8h	51h	31.83h	2h
 Dawid Bitner	129.5h	21.5h	13h	53h	42h	
 Mateusz Kowol	96h		17h	18h	55h	6h
Total	328.33h	31.5h	38h	122h	128.83h	8h

Rysunek 6.10: *Jira* – raport przepracowanych godzin. Źródło: Opracowanie własne.

6.3 Środowiska programistyczne

6.3.1 IntelliJ

IntelliJ jest środowiskiem programistycznym, stworzonym przez producenta o nazwie *JetBrains* w 2001 roku. Jest doskonałym narzędziem, wspomagającym pisanie w języku *Java*, najczęściej wybieranym przez programistów, którzy są zorientowani właśnie na tą gałąź programowania. Licencje dzielą się na dwie wersje. Wersja pod nazwą *Community* jest bezpłatna i różni się od płatnej wersji *Ultimate* tym, że nie oferuje większego wsparcia dla takich struktur, jak *Spring*, *JavaEE* czy *Hibernate*²².



Rysunek 6.11: Środowisko programistyczne *IntelliJ*. Źródło: Opracowanie własne.

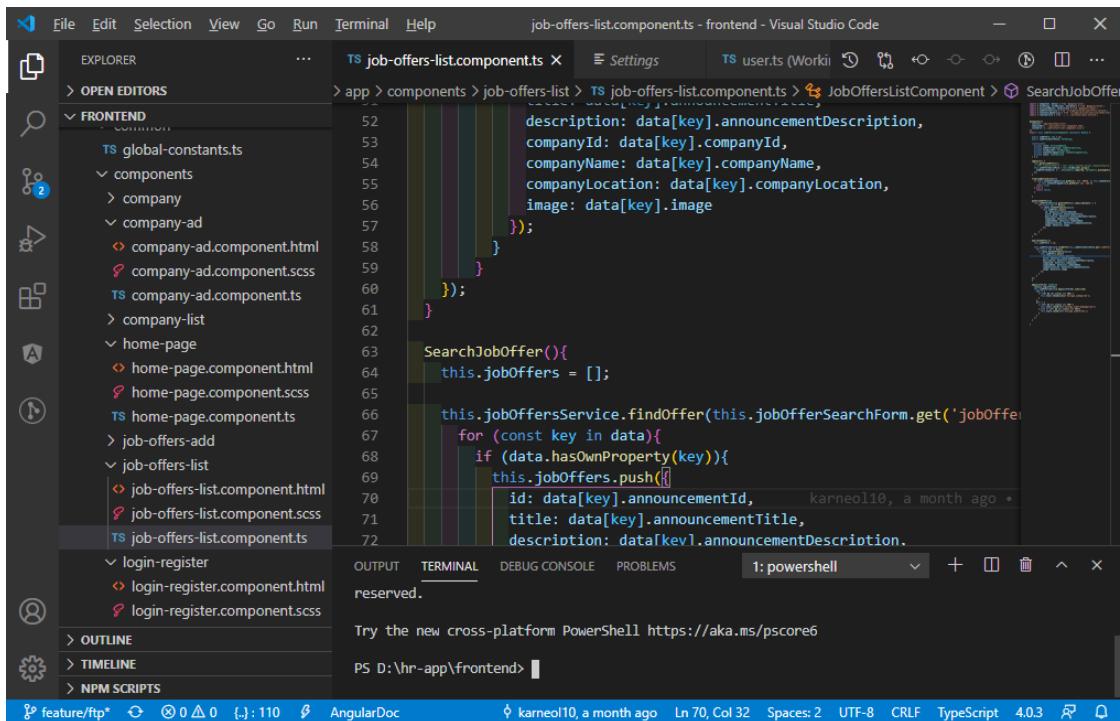
6.3.2 Visual Studio Code

Kolejnym narzędziem niezbędnym do stworzenia całego oprogramowania jest platforma, na której w nieskomplikowany sposób można stworzyć widok i logikę dla stron internetowych. Mając na uwadze doświadczenia członków zespołu z poszcze-

²²Informacje licencyjne firmy Jet Brains,

<https://www.jetbrains.com/idea/buy/#commercial?billing=yearly> (dostęp 11.12.2020)

głólnymi środowiskami, został wybrany *Visual Studio Code*. Środowisko programistyczne stworzyła rozpoznawalna firma *Microsoft* i jest w pełni darmowe²³.



Rysunek 6.12: Środowisko programistyczne *Visual Studio Code*. Źródło: Opracowanie własne.

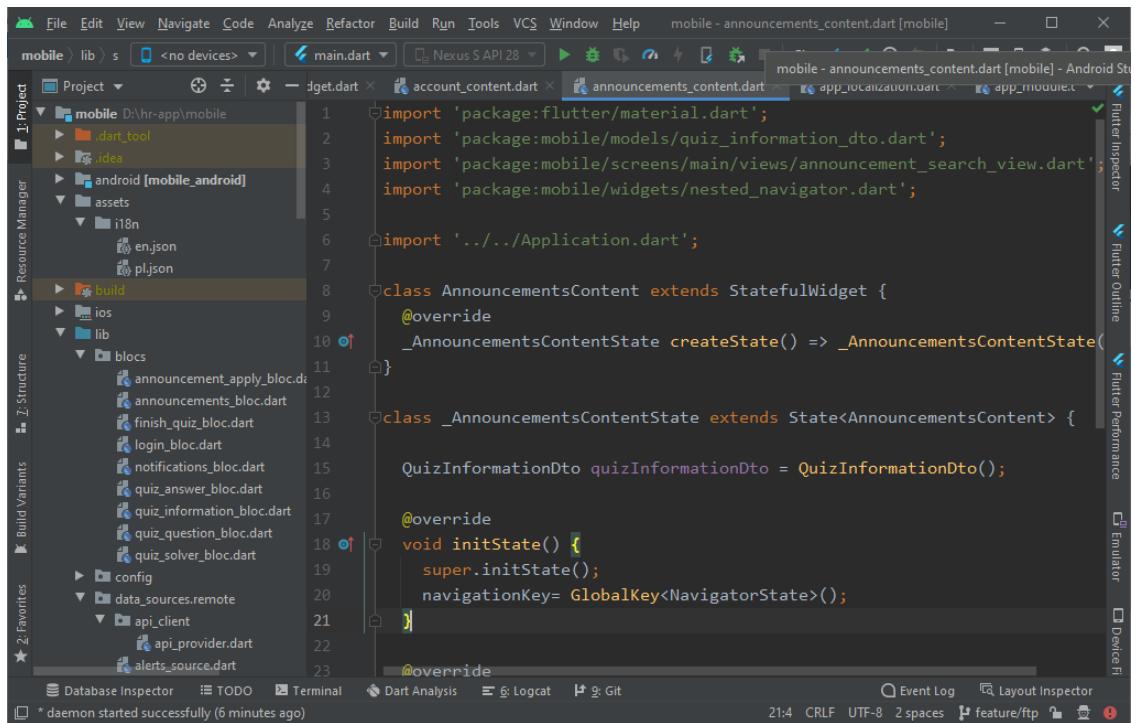
6.3.3 Android Studio

Na rynku można znaleźć trzy główne środowiska, wspierające pisanie w platformie programistycznej *Flutter*. Wspomniane już wcześniej *Visual Studio Code*, jest doskonałym środowiskiem dla osób, które mają problemy ze zbyt dużym użyciem pamięci *RAM* w systemie operacyjnym *Windows*. Drugą propozycją jest *Xcode* zaprojektowany przez przedsiębiorstwo *Apple Inc.*, jednak ta opcja jest jedynie dostępna dla osób posiadających sprzęt tejże firmy²⁴. Ostatnim środowiskiem, które ostatecznie zostało użyte w projekcie, jest *Android Studio*, korporacji *JetBrains*, oferujące głównie wsparcie dla natywnego pisania w języku *Kotlin*, bądź *Java*, jednak

²³Warunki licencyjne oprogramowania Microsoft – *Visual Studio Code*, <https://code.visualstudio.com/license> (dostęp 11.12.2020)

²⁴Xcode12.2 – umowa licencyjna, <https://apps.apple.com/pl/app/xcode/id497799835?l=pl> (dostęp 11.12.2020)

nie ma większych problemów ze wspieraniem języka *Dart*, po zaimplementowaniu odpowiednich wtyczek.



Rysunek 6.13: Środowisko programistyczne *Android Studio*. Źródło: Opracowanie własne.

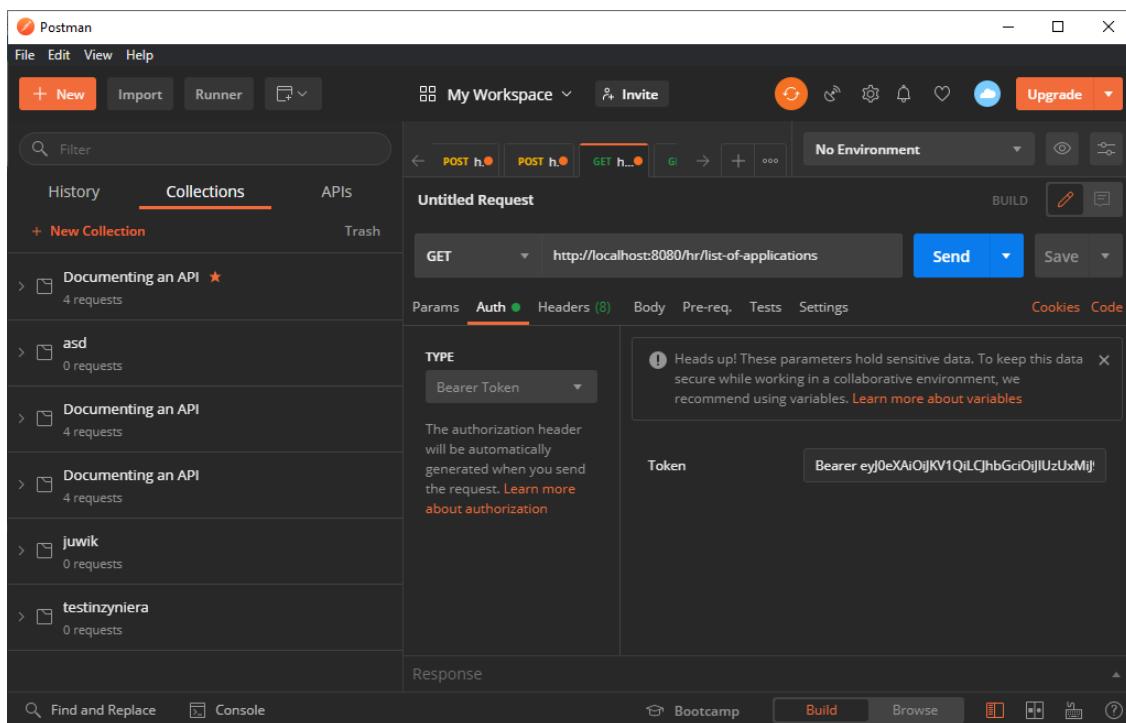
6.3.4 PG Admin

Koniecznym oprogramowaniem w projekcie, w którym została zaimplementowana baza danych typu *PostgreSQL*, jest kompatybilny z tą technologią program *PG Admin*²⁵. Poprzez odpowiednie zintegrowanie środowiska, jest się w stanie w szybki sposób zarządzać bazą danych, śledzić poprawność zmian dokonanych przy pomocy serwera bądź sztywno dodawać, usuwać i edytować elementy, co często przypisza pracę programisty. Pierwsza wersja *PG Admina* została wydana w 1998 roku, przez *pgAdmin Development Team*.

²⁵ Materiały System Administration Portal, „PostgreSQL – pgAdmin”, <https://pl.admininfo.info/postgresql-pgadmin> (dostęp 11.12.2020)

6.3.5 Postman

Drugorzędnym środowiskiem, które nie jest konieczne, jednak niezwykle usprawnia testowanie ostatnio wprowadzonych zmian w punktach końcowych, (ang. *end-point*) jest *Postman*. Narzędzie to gwarantuje nam w prosty sposób dostęp do takich zasobów jak: nagłówki, ciało wysyłanego żądania, czy również wspomaga wysyłanie całych plików na serwer. Nie można zapomnieć o zwrotnych informacjach, które oferuje platforma i są one wyświetlane w czytelny sposób. *Postman* został wydany w 2014 roku przez firmę *Postman inc.*



Rysunek 6.14: Program wspomagający *Postman*. Źródło: Opracowanie własne.

7 Specyfikacja

Sekcja przedstawia informacje odnośnie instalacji poszczególnych komponentów, potrzebnych do uruchomienia stworzonego systemu oraz instrukcję obsługi części serwerowej, mobilnej i przeglądarkowej.

7.1 Instalacja i konfiguracja środowisk

Przed przystąpieniem do tworzenia aplikacji, należy pobrać odpowiednie środowiska dla poszczególnych części projektu tj. zaplecza, części webowej oraz aplikacji mobilnej, zwracając przy tym uwagę na wybieraną licencję.

7.1.1 IntelliJ IDEA

Aby pobrać środowisko do pracy z kodem *Java (Spring Boot)*, należy udać się na stronę: <https://www.jetbrains.com/idea/download/#section=windows>, a następnie wybrać bezpłatną wersję społecznościową (ang. *Community*). Później należy przeprowadzić standardową instalację oprogramowania na aktualnym systemie operacyjnym. Po zakończeniu instalacji należy uruchomić środowisko, a następnie, w ustawieniach kompilatora, wskazać ścieżkę do folderu głównego *Javy*.

7.1.2 Visual Studio Code

Visual Studio Code można pobrać z oficjalnej strony producenta *Microsoft*: <https://code.visualstudio.com/>. Jako że edytor nie posiada wbudowanych wtyczek do pracy ze środowiskiem *Angular*, należy wyszukać odpowiednie wtyczki w zakładce rozszerzeń. Do pracy z *Angulariem* przydatne mogą być takie wtyczki jak np. zestaw *Angular Extension Pack*, czy rozszerzenia wspomagające debugowanie w przeglądarkach, np. *Debugger for Chrome*.

7.1.3 Android Studio

W celu zainstalowania środowiska *Android Studio*, należy udać się na stronę: <https://developer.android.com/studio>, gdzie znajduje się odnośnik do najnowszej wersji programu. Po standardowej instalacji, opcjonalną rzeczą jest dodanie

emulatora, czyli wirtualnego telefonu z systemem Android, symulującego zachowanie prawdziwego telefonu. W tym celu, z listy rozwijanej, w górnej części aplikacji, gdzie znajduje się informacja o braku dostępnych urządzeń, należy wybrać opcję *AVD Manager*. Następnie z dostępnych urządzeń należy wybrać jedno, przejść dalej, zdecydować która wersja *Androida* jest pożądana, ilość pamięci *RAM*, którą chcemy przeznaczyć dla emulatora i go zainstalować.

7.1.4 Postman

Narzędzie *Postman* można pobrać ze strony internetowej, znajdującej się pod adresem: <https://www.postman.com/downloads/>. Instalację należy przeprowadzić w sposób standardowy.

7.1.5 GitBash

Można pobrać oprogramowanie do systemu kontroli wersji. Jest to jednak opcjonalne, ponieważ jest ono wbudowane zarówno w *Visual Studio Code*, jak i *Android Studio*, czy *IntelliJ IDEA*. Aby zainstalować program, należy pobrać plik instalacyjny ze strony producenta: <https://git-scm.com/downloads>. Po instalacji użytkownik dostaje dostęp do *Git GUI* oraz *Git Bash*, które służą do wykonywania poleceń *git*.

7.1.6 PostgreSQL

Do zainstalowania silnika baz danych *PosgreSQL* oraz narzędzia do zarządzania bazą wykonanych zapytań – *pgAdmin4*, należy odwiedzić stronę internetową: <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>. Następnie należy wybrać plik instalacyjny najnowszej wersji, zgodny z systemem operacyjnym, na którym program ma zostać zainstalowany. Podczas instalacji użytkownik powinien wybrać komponent opcjonalny *pgAdmin4*.

Po udanym pobraniu i zainstalowaniu należy przejść do folderu w którym serwer został umieszczony, a następnie wybieramy folder *bin*, z którego można utworzyć skrót do aplikacji *pgAdmin4.exe* na pulpicie.

7.1.7 Java

W celu zainstalowania *Javy* należy pobrać ją ze strony producenta: <https://www.java.com/pl/download/>. Po przejściu przez standardową instalację, podczas której użytkownik powinien zdecydować, w którym miejscu chcemy zainsta-

lować oprogramowanie, należy dodać je do zmiennych środowiskowych. W tym celu należy wybrać w *Menu Start* komponent *Edytuj zmienne środowiskowe*. Gdy pojawi się okienko, kolejnym krokiem jest przejście do zakładki *Zaawansowane*, po czym wybranie *Zmienne środowiskowe....* Należy dodać nową zmienną o nazwie *JAVA_HOME*, a jako wartość wpisać ścieżkę do folderu w którym *Java* została zainstalowana.

7.1.8 Node.js

Aby korzystać z platformy programistycznej *Angular*, niezbędne jest posiadanie środowiska *node.js*. W tym celu należy odwiedzić oficjalną stronę oprogramowania: <https://nodejs.org/en/>. Po pobraniu najnowszej wersji (zalecana jest wersja o długim wsparciu (ang. *Long Time Support (LTS)*)) należy wywołać instalator i przejść standardową instalację. W celu weryfikacji prawidłowego przebiegu instalacji należy wpisać w wierszu poleceń następujące polecenia: **node -v** oraz **npm -v**. Jeżeli wszystko zostało poprawnie wykonane, zostanie zwrócony numer wersji.

7.1.9 Angular

W celu instalacji platformy programistycznej *Angular* należy uruchomić konsolę systemową i wykonać następujące polecenie: **npm install -g @angular/cli**.

7.1.10 Flutter

Aby móc korzystać z platformy programistycznej *Flutter*, podstawowym krokiem jest pobranie pliku *.zip* ze strony: <https://flutter.dev/docs/get-started/install>, a następnie wypakować go do rekomendowanej przez *Google* ścieżki *C:\src\flutter*.

Kolejnym krokiem jest przejście do zmiennych środowiskowych. *Dokładna instrukcja, jak tego dokonać znajduje się w punkcie 7.1.7.* W tym miejscu, do zmiennej o nazwie *Path* należy dodać nową ścieżkę, która odpowiada miejscu zapisanego oprogramowania. Jeżeli wszystko przebiegło pomyślnie, po wpisaniu komendy **flutter --version** w wierszu poleceń, ukaże się informacja o wersji *Fluttera*.

7.2 Instrukcja obsługi

W tej części podrozdziału zawarte są podstawowe informacje, które dotyczą instrukcji obsługi poszczególnych części projektowych.

7.2.1 Część serwerowa

Aby uruchomić serwer, należy posiadać na docelowym sprzęcie zainstalowany serwer bazodanowy *PostgreSQL*, a następnie uruchomić aplikację *Hr-App.jar*, która jest dostępna u twórców projektu. Gdy serwer się uruchomi, można bezproblemowo się z nim połączyć, o ile użytkownik używa tej samej sieci, w której się znajduje lub zmodyfikować ją, by była ogólnodostępna. Istnieje również możliwość produkcyjnego wystawienia serwera do sieci, przy pomocy np.: kontenera aplikacyjnego *Apache Tomcat*. W takim przypadku, wystarczy, że pliki aplikacji znajdują się w odpowiednim podfolderze *Tomcata* oraz że serwer jest uruchomiony.

7.2.2 Część internetowa

Aby uruchomić serwer, odpowiedzialny za dostęp użytkownika do strony internetowej, należy otworzyć folder *frontend* w programie *Microsoft Visual Studio Code* i w konsoli wykonać komendę **ng serve --open**, która spowoduje komplikację kodu i włączenie domyślnej przeglądarki internetowej, z otwartą zakładką aplikacji. Istnieje również możliwość wdrożenia części internetowej aplikacji na serwer. Aby tego dokonać, należy analogicznie, lecz zamiast komendy **ng serve --open**, użyć **ng build**, która spowoduje komplikację kodu i wygeneruje pliki, niezbędne do jej uruchomienia na serwerze własnego wyboru. Po udostępnieniu strony za pomocą własnego hostingu, można się z nią połączyć, wpisując w przeglądarce internetowej ustalone przez hosting adres.

7.2.3 Część mobilna

Aby korzystać z oprogramowania na swoim telefonie, należy na nim uruchomić otrzymany plik *Hr-App.apk* i zainstalować zawartość. Gdy wszystko przebiegnie pomyślnie, wyświetli się okno aplikacji. Po pojawienniu się okna logowania, należy się zarejestrować bądź zalogować. Następnie, po udanej autoryzacji, użytkownik zostanie przeniesiony do widoku głównego kontentu aplikacji.

8 Implementacja

Rozdział przedstawia najważniejsze informacje, które dotyczą implementacji projektu, z podziałem na część serwerową i kliencką, w postaci aplikacji mobilnej i przeglądarkowej.

8.1 Serwer

Część serwerowa odpowiada za obsługę bazy danych oraz działań użytkownika z nią związanych, takich jak np.: obsługa rozwiązywania quizów, zwracanie tabel, zabezpieczenia i inne. Najważniejsze cechy zaplecza, jak i pozostałych części aplikacji, zostały omówione w tym podrozdziale.

8.1.1 Specyfikacja bazy danych

Pomysł oraz budowa bazy danych zostały omówione w rozdziale **3.1**. Samo tworzenie bazy danych obyło się bez tworzenia podzapytań *SQL*. Zapytania pisane ręcznie są wykorzystywane tylko w kilku specyficznych przypadkach, w których zapytanie *JPASQL*, z wykorzystaniem zapytań wbudowanych, było zbyt skomplikowane lub zbyt długie, co negatywnie wpływało na czytelność kodu. Zapytania *Spring Data JPA* umożliwiają tworzenie zapytań, które składają się z nazwy akcji, jak np.: *find...By*, *count...By* lub *delete...By* i nazwy pól encji. Przykładowo:

```
long countByReadAndFkhrAlertAnnouncementFkannouncemet  
    CompanyId(boolean read, long companyId);
```

Listing 1: Kod źródłowy wysyłający zapytanie do bazy o ilość nieodczytanych powiadomień dla działu HR danej firmy. Źródło: Opracowanie własne.

zwróci liczbę powiadomień użytkownikowi działu HR, na podstawie tego, czy chce on zwrócić liczbę odczytanych czy nieodczytanych oraz na podstawie identyfikatora firmy. Wszystkie zapytania znajdują się w repozytoriach, które zostały rozszerzone o klasę *JpaRepository*, umożliwiającą wykonywanie tego typu działań. Natomiast, własne, natywne zapytanie wygląda w następujący sposób:

```

@Query( value = "SELECT * FROM Announcements announcement
JOIN Companies company " +
"ON company.id = announcement.company_id " +
"WHERE LOWER(announcement.title)
LIKE LOWER(CONCAT( '%', ?1, '%' )) " +
"OR LOWER(announcement.description)
LIKE LOWER(CONCAT( '%', ?1, '%' )) " +
"OR LOWER(company.name)
LIKE LOWER(CONCAT( '%', ?1, '%' )) " +
"OR LOWER(company.about)
LIKE LOWER(CONCAT( '%', ?1, '%' )) " +
"OR LOWER(company.location)
LIKE LOWER(CONCAT( '%', ?1, '%' ))",
nativeQuery = true
)
List<AnnouncementsModel> findAnnouncementByAnything(
    String value);

```

Listing 2: Natywne zapytanie do bazy danych w celu pobrania konkretnych informacji. Źródło: Opracowanie własne.

W tym przypadku zostanie zwrócona lista ogłoszeń, na podstawie wpisanego ciągu znaków, który może zawierać się w: nazwie ogłoszenia, opisie ogłoszenia, nazwie firmy, opisie firmy lub lokalizacji firmy. Rozwiążanie to zostało wykorzystane w szybkiej wyszukiwarce ogłoszeń, na podstawie powyższych kryteriów.

8.1.2 Projekt bazy danych

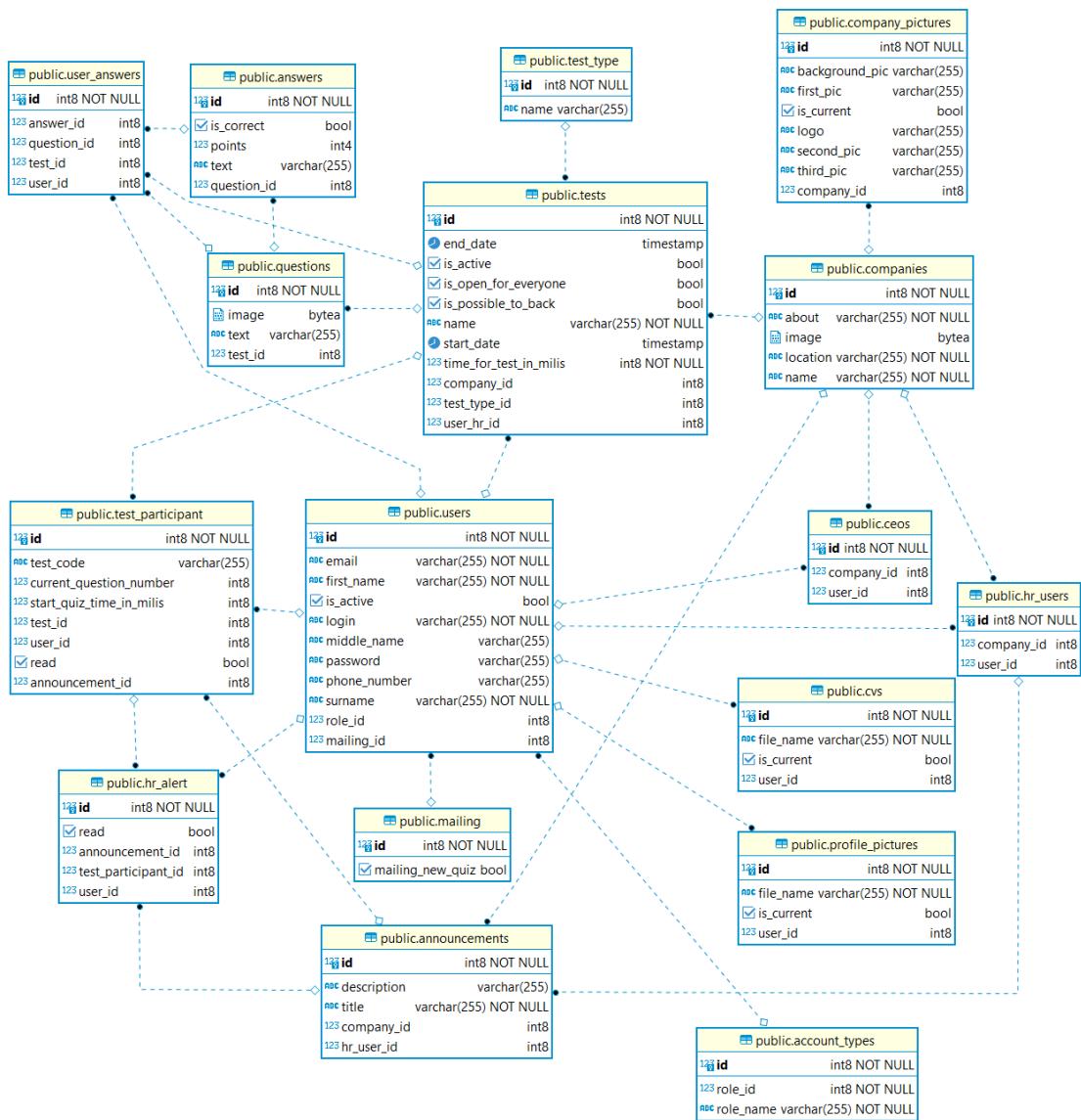
W pierwszej kolejności, baza danych została zaprojektowana na kartce, następnie zespół przeszedł do implementacji struktury bazy danych poprzez tworzenie klas w projekcie oraz zależności pomiędzy tymi klasami. W dalszej kolejności, zostały wykryte błędy z początkowej fazy projektu oraz naniesiono odpowiednie poprawki. Stworzenie struktur danych było pierwszym etapem początkowej fazy projektu.

Oprócz błędów związanych z logiką łączeń, zespół napotkał problem z tworzeniem zapytań *JPA* za pomocą metod wbudowanych. Metody te nie radzą sobie z tworzeniem zapytań w momencie, kiedy w nazwie kolumny znajduje się myślnik.

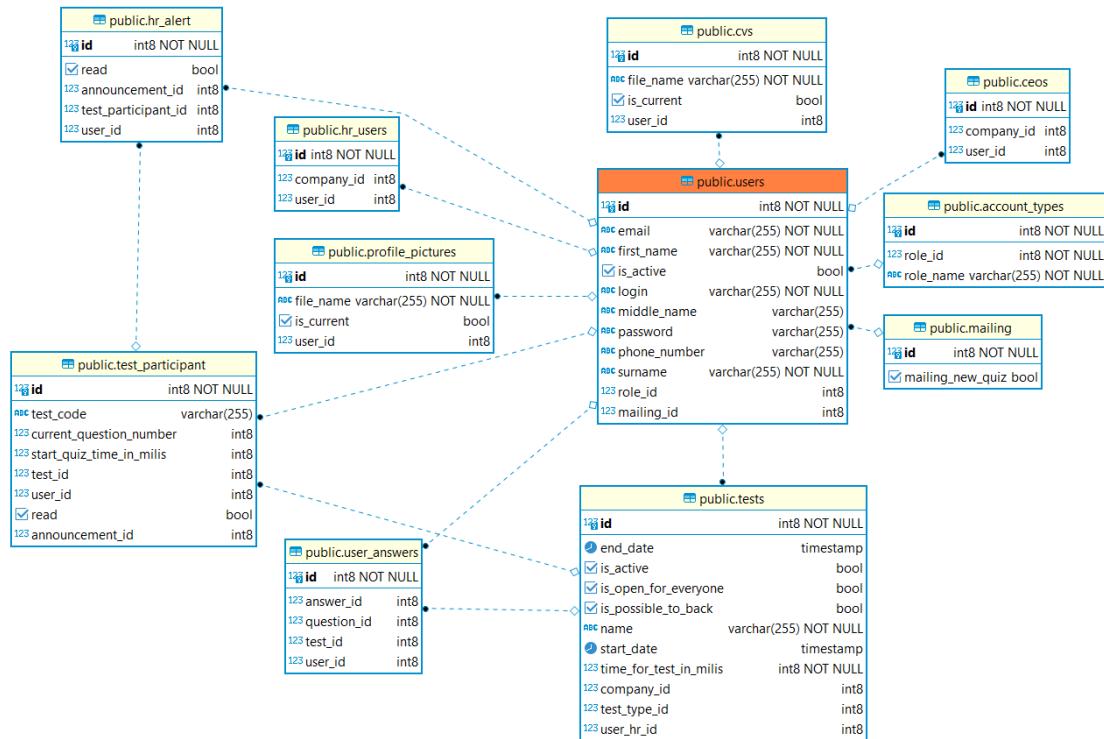
Myślniki zostały zamienione na znaki podkreślenia.

8.1.2.1 Diagram związków encji

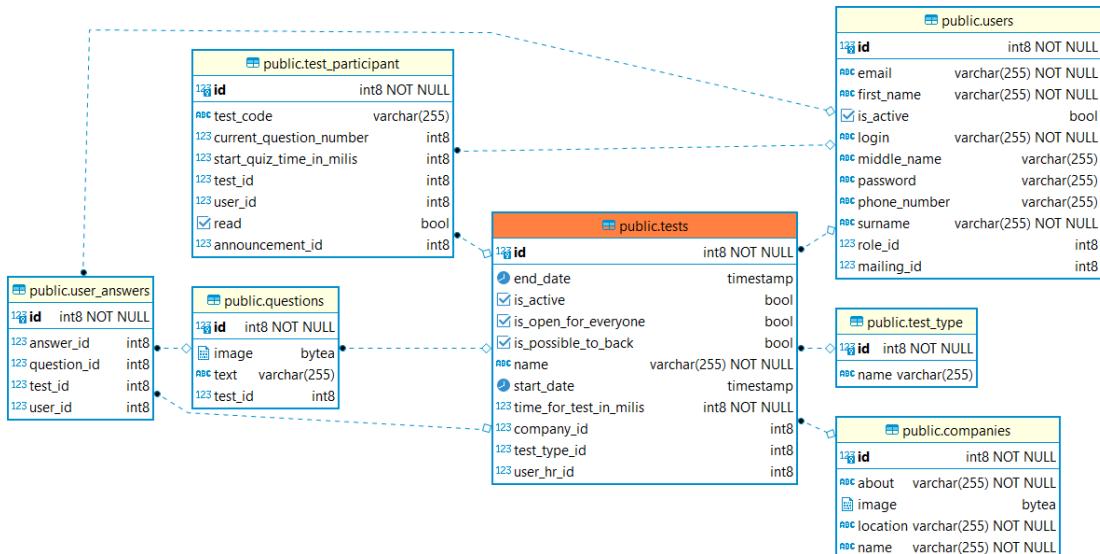
Na kolejnych stronach zostały przedstawione diagramy związków encji. Główny – zawierający całą strukturę bazy danych oraz dwa mniejsze – zawierające wyszczególnione informacje, odnośnie najważniejszych tabel w bazie: kolejno testy i użytkownicy.



Rysunek 8.1: Diagram związków encji. Źródło: Opracowanie własne.



Rysunek 8.2: Diagram związków encji – tabela użytkowników. Źródło: Opracowanie własne.



Rysunek 8.3: Diagram związków encji – tabela testów. Źródło: Opracowanie własne.

8.1.3 Spring Security i bcrypt

W projekcie zostały wykorzystane algorytmy szyfrujące, służące m.in. do zabezpieczenia haseł użytkowników. Został również zaimplementowany system autoryzacji użytkowników za pomocą *JSON Web Token*.

Spring Security i JSON Web Token – *Spring Security* dostarcza zbiór narzędzi, który pozwala na zabezpieczenie tworzonej aplikacji z wykorzystaniem gotowych mechanizmów. Dlatego, w prosty sposób i przy niskim nakładzie pracy programisty, można wykorzystać rozwiązania zaimplementowane w *Spring Security* do autoryzacji, w tym autoryzacji dwuetapowej, poprzez hashowanie danych, *OAuth2*, *JWT*, czy *OpenID*.

W projekcie, w celu autoryzacji użytkowników, został wykorzystany *JSON Web Token*.

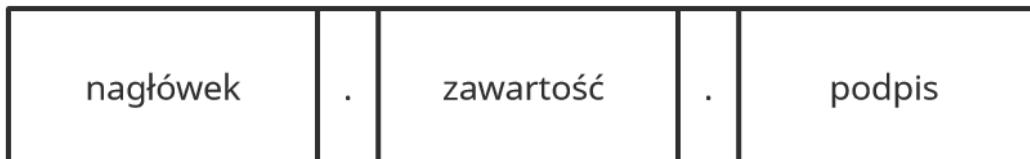
```
{  
    "login": "x",  
    "password": "y"  
}
```

Listing 3: Przykład danych w formacie JSON wysyłane przy logowaniu. Źródło: Opracowanie własne.

Gdzie *x* i *y* to kolejno nazwa użytkownika i hasło, na punkt końcowy: */login*, użytkownik w odpowiedzi dostaje nagłówek *Authorization* z zawartością w postaci *Bearer <token>*, gdzie *<token>* to token *JWT*. Zawiera on dane m.in.: o czasie jego przydatności, jak i o uprawnieniach użytkownika do wybranych zasobów. Następnie jest on doczepiany do nagłówków pozostałyzych zapytań *REST*²⁶. Bez tego, serwer nie jest w stanie stwierdzić, czy użytkownik posiada poświadczenia do wybranych zasobów. Jeżeli token okaza się nieprawidłowy, wygrasie lub użytkownik spróbuje dostać się do zasobów, do których nie ma uprawnień, i zostanie zwrocony błąd *http 403* – zabroniona wartość (ang. *forbidden*) – oznacza to, że serwer zrozumiał żądanie (czyli jest skonstruowane poprawnie), ale odmówił użytkownikowi dostępu

²⁶A. Rahmatulloh, „Performance comparison of signed algorithms on JSON Web Token”, 2019, <https://iopscience.iop.org/article/10.1088/1757-899X/550/1/012023/pdf>, (dostęp 12.12.2020)

do zasobów. Sama zawartość tokenu składa się z trzech części: nagłówka, właściwej zawartości (ang. *payload*) i podpisu. Każda z części jest łączona kropkami, a następnie szyfrowana.



Rysunek 8.4: Graficzne przedstawienie budowy *JSON Web Tokenu*. Źródło: Opracowanie własne.

Sekcja nagłówkowa składa się z dwóch części: zastosowanego algorytmu i typu tokenu. Zawartość jest kodowana za pomocą *Base64*. Druga część to właściwa zawartość, zawierająca roszczenie o dostęp, które zawiera dane użytkownika, w tym nazwę użytkownika i hasło. Ta część również szyfrowana jest w *Base64*. Trzecia część to sygnatura, zawierająca zakodowany nagłówek, zakodowany ładunek, sekretny klucz (który definiowany jest po stronie serwera), a także algorytm, który został zdefiniowany w nagłówku. W tym projekcie zastosowano algorytm kryptograficzny *HMAC*, używający *RSA256*²⁷.

$$\text{Token} = f(\text{KodowanieBase64}) \sum_{n=\alpha, \beta}^{\infty} (\text{naglowek.wlasciwa_zawartosc.sygnatura})$$

Wzór 1: Matematyczne przedstawienie tworzenia tokenu.

```
{ "alg": "HS256", "typ": "JWT" }
```

Listing 4: Budowa nagłówka JWT. Źródło: Opracowanie własne.

Gdzie *alg* to wykorzystany algorytm do szyfrowania.

²⁷A. Michalczyk – Sekurak, „Kompendium bezpieczeństwa haseł – atak i obrona (część 1.)”, <https://sekurak.pl/kompendium-bezpieczenstwa-hasel-atak-i-obrona/>, (dostęp 12.12.2020)

```
{
    "login": "user1",
    "iat": 1422779222,
    "password": 12345
}
```

Listing 5: Właściwa zawartość tokenu. Źródło: Opracowanie własne.

W części właściwej administrator systemu definiuje, jakie dane są przekazywane. Następnie dwie powyższe zawartości są szyfrowane w sygnaturze.

bcrypt – w projekcie został wykorzystany, w celu bezpiecznego przechowywania haseł użytkowników w bazie danych, w postaci zaszyfrowanej.

```
[identyfikator algorytmu][koszt][22 znaki soli]
[31 znaków hashu]
```

Listing 6: Sposób w jaki jest zapisywany ciąg bcrypt. Źródło: Opracowanie własne.

Na przykład:

```
$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxadL17lhWy
\--/\-\-----/\-----/
alg koszt           sól           hasz
```

Listing 7: Przykładowy ciąg bcrypt. Źródło: Opracowanie własne.

Gdzie:

- **alg** – wyznacza rodzaj algorytmu, gdzie `$2a$` oznacza zmodyfikowany algorytm *Blowfish*.
- **koszt** – to współczynnik kosztu. Czyli informacja o tym, ile razy należy uruchomić podany *hash*. Im wyższa wartość, tym odczytywanie hasła jest powolniejsze, ale zwiększeniu zostaje odporność na ataki brutalne.
- **sól** – to ciąg zaburzający. Są to losowe dane dodawane do hasła, które mają zwiększać ochronę przed atakami słownikowymi. Zapisana jest w danych 128 bitowych, zakodowana do 22 znaków *Base64*.
- **hasz** – powstaje w wyniku użycia funkcji skrótu. Jest to funkcja przyporządkowująca dowolnie dużej liczbie, zawsze posiadającą stały rozmiar, krótką, quasi-losową wartość, tzw. skrót nieodwracalny. Hasz przetrzymywany jest w danych 192 bitowych, zakodowany do 31 znaków *Base64*.

8.1.4 Architektura

Zespół projektowy zdecydował się na użycie popularnego wzorca projektowego *MVC* (ang. *Model View Controller*), który jest używany w tworzeniu m.in: aplikacji serwerowych. Wzorzec ten składa się z trzech głównych części, jakimi są²⁸:

- Model – używany do reprezentowania i utworzenia tabel bazodanowych i informacji w nich zawartych.
- View – który przechowuje odpowiednie dane, używane później w aplikacji do przetwarzania lub reprezentowania informacji przesyłanych między serwerem a klientem.
- Controller – odpowiedzialny za obsługę żądań, wysyłanych przez klienta, a następnie zwrócenie mu oczekiwanych informacji.

8.1.5 Metody protokołu HTTP

W ramach projektu, zostały wykorzystane cztery główne metody protokołu *http*, tj.: *POST* – służący do wysyłania danych na serwer, *GET* – pobierający dane z serwera, *DELETE* – usuwający dane i *UPDATE* – aktualizujący dane.

8.1.5.1 PostMapping

Metoda ta została wykorzystana np.: przy logowaniu czy rejestracji użytkowników.

```
@PostMapping(serviceUrlParam + "/register")
@ResponseBody
public ResponseTransfer saveUser(
    @RequestBody RegisterCommandDto registerCommandDto,
    HttpServletRequest response) {
    try {
        if (userLoginAlreadyExists(registerCommandDto
            .getLogin()))
    }
}
```

²⁸Materiały serwisu 1024kb.pl, <https://1024kb.pl/spring/jak-dziala-spring-web-mvc/>, (dostęp 17.12.2020)

```
response.setStatus(httpServletResponse
    .SC_CONFLICT);
return new ResponseTransfer(
    "LOGIN_EXISTS"
);
}
UsersModel checkEmailUser =
usersRepository.findByEmail(registerCommandDto
    .getEmail());
if(checkEmailUser!=null) {
    response.setStatus(httpServletResponse
        .SC_CONFLICT);
    return new ResponseTransfer(
        "EMAIL_EXISTS"
);
}
if(registerCommandDto.getPassword()
    .length() <6) {
    response.setStatus(httpServletResponse.
        SC_BAD_REQUEST);
    return new ResponseTransfer(
        "WEAKPASSWORD"
);
}
else {
    registerCommandDto.setPassword(
        passwordEncoder
            .encode(registerCommandDto
                .getPassword())));
    UsersModel usersModel =
new UsersModel(registerCommandDto);
    usersModel.setFKuserMailing(
        new MailingModel()
```

```

    );
usersModel.setFKUserAccountTypes(
    getDefaultAccountType());
usersRepository.saveAndFlush(usersModel);
sendMail.sendRegistrationMessage(
    registerCommandDto.getEmail(),
    registerCommandDto.getLogin());
return new ResponseTransfer(
    "User_saved"
);
}
}
catch (Exception exc) {
    response.setStatus(
        httpServletResponse
            .SC_INTERNAL_SERVER_ERROR); //ERROR 500
    return new ResponseTransfer("Not_saved",
        exc.toString());
}
}

```

Listing 8: Implementacja projektowa rejestracji nowych użytkowników. Źródło: Opracowanie własne.

Aby skontaktować się z punktem końcowym, który dotyczy rejestracji, należy wysłać zapytanie na adres serwera, z dodatkową ścieżką: `/user/register`, zawierającą obiekt `JSON`, który znajduje się w ciele zapytania (ang. *body*).

```
{
    "firstName": "Jan",
    "middleName": "Mariusz",
    "surname": "Kowalski",
    "email": "jan@kowalski.pl",
    "phoneNumber": "0000000000",
    "login": "jKowalski",
    "password": "jKowalskipssswd",
```

```
    "isActive": true  
}
```

Listing 9: Przykładowy obiekt JSON wysyłany w celu rejestracji użytkownika. Źródło: Opracowanie własne.

Wartości kluczów są wartościami przykładowymi. W przypadku odpowiedzi przez serwer, dostaniemy informację, również w postaci *JSON* o tym, czy rejestracja przebiegła poprawnie. Odpowiedzi są widoczne w przykładzie implementacji powyższego punktu końcowego. Należą do nich obiekty *Response Transfer*, tj. w tym przypadku możliwymi odpowiedziami są:

LOGIN_EXISTS – w przypadku, gdy nazwa użytkownika jest już zajęta.

WEAK_PASSWORD – gdy hasło wybrane podczas rejestracji ma mniej niż 6 znaków.

User saved – w przypadku rejestracji zakończonej sukcesem.

User not saved – w przypadku wystąpienia wewnętrznego problemu na serwerze, komunikat zwracany jest wraz z błędem *500*.

8.1.5.2 GetMapping

W projekcie metoda *GET* została wykorzystana w wielu miejscach. Zastosowano ją m.in.: w przypadku pobierania szczegółowych informacji o użytkowniku przez administratora systemu.

```
@GetMapping(serviceUrlParam + "/getUser/{userid}")  
public Object getUser(@PathVariable long userid,  
httpServletResponse response){  
try {  
    UsersModel user = usersRepository  
        .findById(userid);  
    CvsModel cv = cvsRepository  
        .findByFKcvUserId(  
            user.getId());  
    return new UserDataWithCvDto(user,  
        cv.getFileName());  
}
```

```

    } catch (Exception e) {
        response.setStatus(
            httpServletResponse
                .SC_INTERNAL_SERVER_ERROR);
        return new ResponseTransfer(
            "Internal-server-error"
        );
    }
}

```

Listing 10: Wykorzystanie metody GET na przykładzie funkcji zwracającej danego użytkownika. Źródło: Opracowanie własne.

W przypadku chęci pobrania informacji o wybranym użytkowniku, należy odpytać punkt końcowy: *getUser/{userid}*, gdzie *{userid}* to identyfikator użytkownika z bazy danych. W ciele zwrócone zostaną informacje o użytkowniku, które przechowywane są w tabeli.

8.1.5.3 DeleteMapping

Metoda *http*, pozwalająca na usuwanie, została zaimplementowana dla usuwania użytkowników z działów *HR*. Punkt końcowy został odpowiednio zabezpieczony przed niepowołanym dostępem, czy przed usuwaniem użytkowników z innych firm, do których usuwający nie ma dostępu.

```

@Transactional
@DeleteMapping(serviceUrlParam + "/hrusers/delete")
public ResponseTransfer deleteHrUser(@RequestBody
AddNewHrCommandDto addNewHrCommandDto,
httpServletResponse response){
    UsersModel usersModel;
    CeosModel ceosModel;
    UsersModel userToDelete;
    try {
        usersModel = getUsersModel();
        ceosModel = getCeosModelByOwnerId(

```

```
        usersModel.getId()
    );
    userToDelete = getUserById(addNewHrCommandDto
        .getHrUserId());
    HrUsersModel hrUsersModel = hrUsersRepository
        .findByFKhrUserUserId(userToDelete.getId());

    String[] ceoFlags = new String[] { "admin" ,
        "ceo" };
    String[] canBecomeDeletedFromHr
        = new String[] { "hr" };
    List<String> ceoFlagsList
        = Arrays.asList(ceoFlags);
    List<String> canBecomeDeletedList =
        Arrays.asList(canBecomeDeletedFromHr);

    if (!ceoFlagsList.contains(
            usersModel.getFKuserAccountTypes()
                .getRoleName()) ||
        !canBecomeDeletedList.contains(
            userToDelete
                .getFKuserAccountTypes()
                .getRoleName()) ||
        ceosModel.getFKceoCompany().getId() !=
            hrUsersModel
                .getFKhrUserCompany().getId()) {
        response.setStatus(httpServletResponse
            .SC_FORBIDDEN); // 403
        return new ResponseTransfer(
            "Forbidden_command"
        );
    } else {
        AccountTypesModel userRoleModel
```

```
        = getUserModel();
        userToDelete.setFKUserAccountTypes(
            userRoleModel
        );
        usersRepository.save(userToDelete);
        hrUsersRepository
            .delete(hrUsersRepository
                .findByFKhrUserUserId(userToDelete
                    .getId())));
        response.setStatus(httpServletResponse
            .SC_OK
        );
        return new ResponseTransfer(
            "Hr-user-deleted"
        );
    }
} catch (Exception e) {
    response.setStatus(
        httpServletResponse
            .SC_INTERNAL_SERVER_ERROR); // 500
    return new ResponseTransfer(
        "Internal-server-error"
    );
}
}
```

Listing 11: Implementacja metody DELETE na przykładzie usunięcia użytkownika z działu HR. Źródło: Opracowanie własne.

8.1.5.4 PutMapping

Metoda aktualizująca dane została wykorzystana m.in. w przypadku modyfikowania informacji o subskrypcjach e-mail.

```
@PutMapping(serviceUrlParam + "/mailing/edit")
public Object saveListOfMailings(
    @RequestBody MailingModel mailingModel,
    HttpServletRequest response) {
    long mailingId;
    try {
        mailingId = getUserModel().getFKuserMailing()
            .getId();
        if (mailingId == mailingModel.getId()) {
            mailingModel
                .setMailingNewQuiz(mailingModel
                    .getMailingNewQuiz());
            mailingRepository.save(mailingModel);
            return new ResponseTransfer(
                "E-mail_preferences_have_changed");
        } else {
            response.setStatus(
                HttpServletResponse
                    .SC_FORBIDDEN); // 403
            return new ResponseTransfer(
                "You_cannot_perform_this_operation");
        }
    } catch (Exception e) {
        response.setStatus(
            HttpServletResponse
                .SC_INTERNAL_SERVER_ERROR); // 500
        return new ResponseTransfer(
            "Internal_server_error", e.toString());
    }
}
```

Listing 12: Przykład użycia metody PUT. Źródło: Opracowanie własne.

Wartości kluczy w *RequestBody*, które należy wysłać na punkt końcowy */mailing/edit*.

```
{  
    "id": 83,  
    "mailingNewQuiz": false  
}
```

Listing 13: Przykładowa wartość klucza. Źródło: Opracowanie własne.

Gdzie *id* to identyfikator wpisu w tabeli subskrypcji e-mail, natomiast wartość booleanska dla klucza *mailingNewQuiz* określa, czy użytkownik chce otrzymywać informację o nowo przypisanych testach do jego konta.

8.1.6 Opis działania najważniejszych punktów końcowych

Najważniejszą funkcjonalnością projektu, wokół której została stworzona cała aplikacja, jest rozwiązywanie testów rekrutacyjnych. Mając na uwadze to oraz to że, program został stworzony dla firm i osób ubiegających się o pracę, niezbędną jest możliwość dodania swojego życiorysu. W tym celu członkowie zespołu stworzyli odpowiednie punkty końcowe.

8.1.6.1 Pobieranie informacji o quizie

Aby pobrać informacje ogólne o quizie, jakimi są: liczba pytań, czas na jego wykonanie; informacja, czy można powracać do poprzednich pytań itp., należy użyć metody *GET*.

```
@GetMapping("quiz/getQuizInformations/{quizcode}")
```

Listing 14: Deklaracja metody żądania wraz z punktem końcowym. Źródło: Opracowanie własne.

Każdy użytkownik, który dostał dostęp do wykonania danego testu, posiada swój unikalny kod. Jest on wpisywany do żądania podanego powyżej w miejsce *quizcode*. Kolejnym krokiem, który jest wykonywany po stronie serwera, jest sprawdzenie czy kod jest poprawny i istnieje, a jeżeli nie – zwrócenie odpowiedniej informacji użytkownikowi i zakończenie wykonywania dalszych instrukcji.

```
testParticipantModel =
    getCodeModelByTestCode( quizcode );
if( testParticipantModel == null ) {
    response.setStatus(
        httpServletResponse.SC_NOT_FOUND
    );
    return new QuizCodeDto(
        ResponseEnum.TEST_NOT_FOUND
    );
}
usersModel = getUserModel();
testsModel = testParticipantModel
    .getFKtestCodetest();
listOfQuestions =
    getAllQuestionFromQuizId( testsModel.getId() );
```

Listing 15: Sprawdzenie poprawności zapytania. Źródło: Opracowanie własne.

Następnie zostają pobrane główne informacje o teście i użytkowniku, który chce go rozwiązać oraz lista pytań wraz z informacjami:

```
usersModel
    = getUserModel();
testsModel
    = testParticipantModel
    .getFKtestCodetest();
listOfQuestions
    = getAllQuestionFromQuizId( testsModel
    .getId() );
```

Listing 16: Pobranie informacji z bazy danych. Źródło: Opracowanie własne.

Aby poprawnie przeprowadzić test, należy sprawdzić, czy jest możliwe jego wykonanie. Dlatego też, w kolejnej części, zostały sprawdzone takie informacje jak:

- czy wpisany kod należy do użytkownika, który wysłał zapytanie,
- czy test jest wciąż aktywny,

- czy test został już zakończony bądź czy czas na jego wykonanie nie upłynął.

Jeżeli na którymś z tych punktów serwer nie pozwoli użytkownikowi przejść dalej, zostanie zwrócona odpowiednia wiadomość. Natomiast, gdy wszystko przebiegnie pomyślnie, wartość pola informującego o tym, czy użytkownik rozpoczął test, zostanie ustawione na wartość *true*.

```
testParticipantModel
    .setRead(true);
testParticipantRepository
    .save(testParticipantModel);
```

Listing 17: Zmiana i zaktualizowanie informacji w bazie danych. Źródło: Opracowanie własne.

Zespół użył odpowiedniej funkcji, która zwraca aktualny czas w milisekundach, a której wartość 0 przypada na datę 1.01.1970. Następnie przypisał użytkownikowi tę wartość, jako moment, w którym rozpoczął test.

```
ZonedDateTime.now().toInstant().toEpochMilli();
```

Listing 18: Metoda zwracająca aktualny czas w milisekundach. Źródło: Opracowanie własne.

Na koniec zostaje zwrócony użytkownikowi odpowiedni model, który to zawiera wszystkie niezbędne dla niego informacje:

```
return new QuizInformationsResultDto(testsModel.getId(),
    listOfQuestions.size(),
    testsModel.isPossibleToBack(),
    timeLeft,
    ResponseEnum.SUCCESS,
    testParticipantModel.getQuestionNumber());
```

Listing 19: Utworzenie modelu oraz odesłanie informacji klientowi na jego podstawie. Źródło: Opracowanie własne.

8.1.6.2 Pobieranie pytania quzu

Kiedy użytkownik pobierze informacje o swoim teście, należy wysłać zapytanie o pierwsze pytanie testu. Ten punkt końcowy jest również wykorzystywany za każdym razem, gdy użytkownik chce przejść do kolejnego pytania lub gdy serwer udostępnia tę opcję w stworzonym teście – do poprzedniego pytania. Podobnie, jak w poprzednim przypadku, została tu wykorzystana metoda *GET*. W zapytaniu tym, należy dodać w miejsca *quizId*, *testCode*, *questionNumber* odpowiednio: id testu, kod testu oraz numer żądanego pytania.

```
@GetMapping(  
    "quiz/quizquestion/{quizId}/{testCode}/{questionNumber}"  
)
```

Listing 20: Deklaracja metody GET wraz z deklaracją punktu końcowego. Źródło: Opracowanie własne.

Gdy zapytanie zostanie wysłane, podobnie jak w przypadku pobierania informacji o teście, serwer sprawdza uprawnienia użytkownika, czy test istnieje i nie został zakończony. Jeżeli test został stworzony w ten sposób, by nie można było cofnąć się do poprzednich pytań, numer pytania zostaje pobrany z bazy i na tej podstawie zostaje wysłane kolejne pytanie użytkownikowi, a następnie następuje nadpisanie tej wartości, inkrementując ją.

```
long questionNumberToReturn  
    = testParticipantModel.getQuestionNumber();  
testParticipantModel  
    .setQuestionNumber(questionNumberToReturn + 1);  
testParticipantRepository  
    .save(testParticipantModel);
```

Listing 21: Zapisanie informacji pytaniu na którym użytkownik aktualnie się znajduje. Źródło: Opracowanie własne.

Następnie należy skompletować odpowiedź dla użytkownika. W tym celu, za pomocą identyfikatora testu, zostaje pobrana lista z pytaniami, które do niego należą. Wszystkie pytania są poszgregowane, a zawężenie pola poszukiwań do numeru wcześniej wspomnianego pytania sprawi, że dostaniemy konkretne, jedno pytanie. Za pomocą identyfikatora pytania można pobrać wszystkie możliwe odpowiedzi.

```

listOfQuestions =
    getAllQuestionFromQuizId(
        quizQuestionCommandDto . getQuizid ()
    );

private QuestionsModel
    getExpectedQuestionModel(
        List<QuestionsModel> listofQuestions ,
        long questionNumber) {
    return listofQuestions
        . get((int) questionNumber -1);
}

private List<AnswersModel>
    getAnswersByQuestionId(long questionId) {
    return answersRepository
        . findAllByFKanswerQuestionId( questionId );
}

```

Listing 22: Pobranie pytania oraz możliwych odpowiedzi z bazy danych. Źródło: Opracowanie własne.

W kolejnym, ostatnim kroku, całość zostaje skompletowana do odpowiedniego modelu i zwrócona użytkownikowi.

8.1.6.3 Wysyłanie odpowiedzi na pytanie z quizu

Gdy już użytkownik wybierze satysfakcjonującą go odpowiedź, musi o tej czynności powiadomić serwer, by ten zapisał ją w bazie danych. Do tego celu służy punkt końcowy z metodą *POST*.

```
@PostMapping(”question/setanswer”)
```

Listing 23: Deklaracja metody POST oraz punktu końcowego. Źródło: Opracowanie własne.

Ciało tego żądania musi zawierać informację o identyfikatorze testu i pytania, kiedy dostępu do testu oraz numerze pytania. Gdy wszystkie niezbędne modele ba-

zadanowe zostaną pobrane, a informacje dotyczące poprawności zapytania (opisane dokładniej w rozdziałach 8.1.6.1 i 8.1.6.2) są odpowiednie, odpowiedź zostaje zapisana bądź, o ile test pozwala na wracanie do poprzednich pytań, nadpisana.

```
userAnswersModel = new UserAnswersModel( usersModel ,  
    questionsModel , answersModel , testsModel );  
userAnswerRepository . save( userAnswersModel );
```

Listing 24: Zapisanie odpowiedzi na dane pytanie w bazie danych. Źródło: Opracowanie własne.

8.1.6.4 Protokół FTP na przykładzie wysyłania CV

W celu dodania bądź zmiany swojego aktualnego życiorysu, stosowany jest punkt końcowy z metodą typu *POST*. Ciało żądania musi zawierać plik w formacie *Form data*, który zapisany jest pod zmienną *file*. Po wysłaniu odpowiedniego pliku, serwer go weryfikuje, sprawdzając po kolej, czy plik nie jest pusty, jest w formacie *.pdf* lub czy przypadkiem nie będzie zajmował zbyt dużo miejsca.

```
if ( file == null ) {  
    response . setStatus ( httpServletResponse  
        . SC_BAD_REQUEST );  
    return new ResponseTransfer (  
        " File _ is _ required "  
    );  
}  
if ( ! FileCorrectness  
    . fileExtensionIsCorrect ( file , " pdf" ) ) {  
    response . setStatus ( httpServletResponse  
        . SC_BAD_REQUEST );  
    return new ResponseTransfer (  
        " File _ is _ not _ pdf "  
    );  
}  
if ( ! FileCorrectness  
    . fileSizeIsOk ( file ) ) {  
    response
```

```

        .setStatus( httpServletResponse
        .SC_BAD_REQUEST);

    return new ResponseTransfer(
        "File is too big"
    );
}

```

Listing 25: Sprawdzenie poprawności wysyłanego pliku. Źródło: Opracowanie własne.

Jeśli wszystko przebiegnie prawidłowo, program zapisze plik z życiorysem użytkownika do odpowiedniego folderu, znajdującego się na serwerze pod postacią:

```

uzytkownik123_dfuhsdsg2sdv23rfds_1.01.1970.pdf
\-----/\-----/\----/\-
      login     unikalny identyfikator      data
      użytkownika                      rozszerzenie

```

Listing 26: Struktura nazwy pliku życiorysu. Źródło: Opracowanie własne.

Gdy plik zostanie zapisany, zostaje zbudowany odpowiedni odnośnik, który pozwoli na dotarcie do tego pliku. Odnośnik ten jest następnie zapisany w modelu bazodanowym użytkownika, w polu, które dotyczy jego życiorysu.

```

fileDownloadUri
    = ServletUriComponentsBuilder
        .fromCurrentContextPath()
        .path("/downloadFile/")
        .path(fileName)
        .toUriString();
saveNewCvModel(currentUser, fileDownloadUri);

```

Listing 27: Utworzenie odnośnika do pliku. Źródło: Opracowanie własne.

Na samym końcu użytkownik dostaje odpowiedź zwrotną o pomyślności wykonania zadania, wraz z informacjami jakimi są: nazwa dodanego pliku, odnośnik do niego, typ pliku oraz wielkość, jaką zajmuje na serwerze.

```
return new UploadFileResponse(  
    fileName, fileDownloadUri,  
    file.getContentType(),  
    file.getSize()  
) ;
```

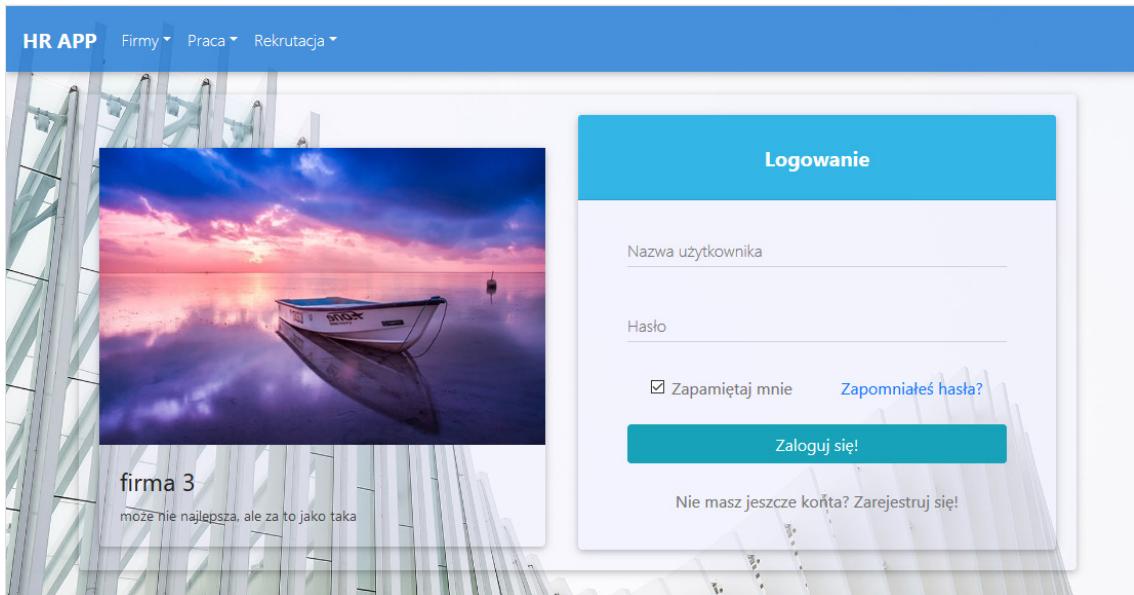
Listing 28: Utworzenie i zwrot modelu informującego o poprawnym dodaniu pliku.
Źródło: Opracowanie własne.

8.2 Aplikacja webowa

Aplikacja webowa posiada najbardziej zaawansowaną funkcjonalność. Zespół podjął taką decyzję, ponieważ o ile rozwiązywanie testów może być przyjemne zarówno na aplikacji mobilnej, jak i przeglądarkowej, to ich tworzenie – gdzie często wykonawca musi posługiwać się dokumentacją, czy też innymi źródłami – na urządzeniu mobilnym może być mocno uciążliwe. Z tego też powodu zdecydowano, że takie funkcjonalności jak: tworzenie testów, zarządzanie zespołem pracowników działu HR i przypisywanie testów, zostaną zaimplementowane w tej fazie produktu tylko w aplikacji przeglądarkowej, z pominięciem programu na urządzenia mobilne.

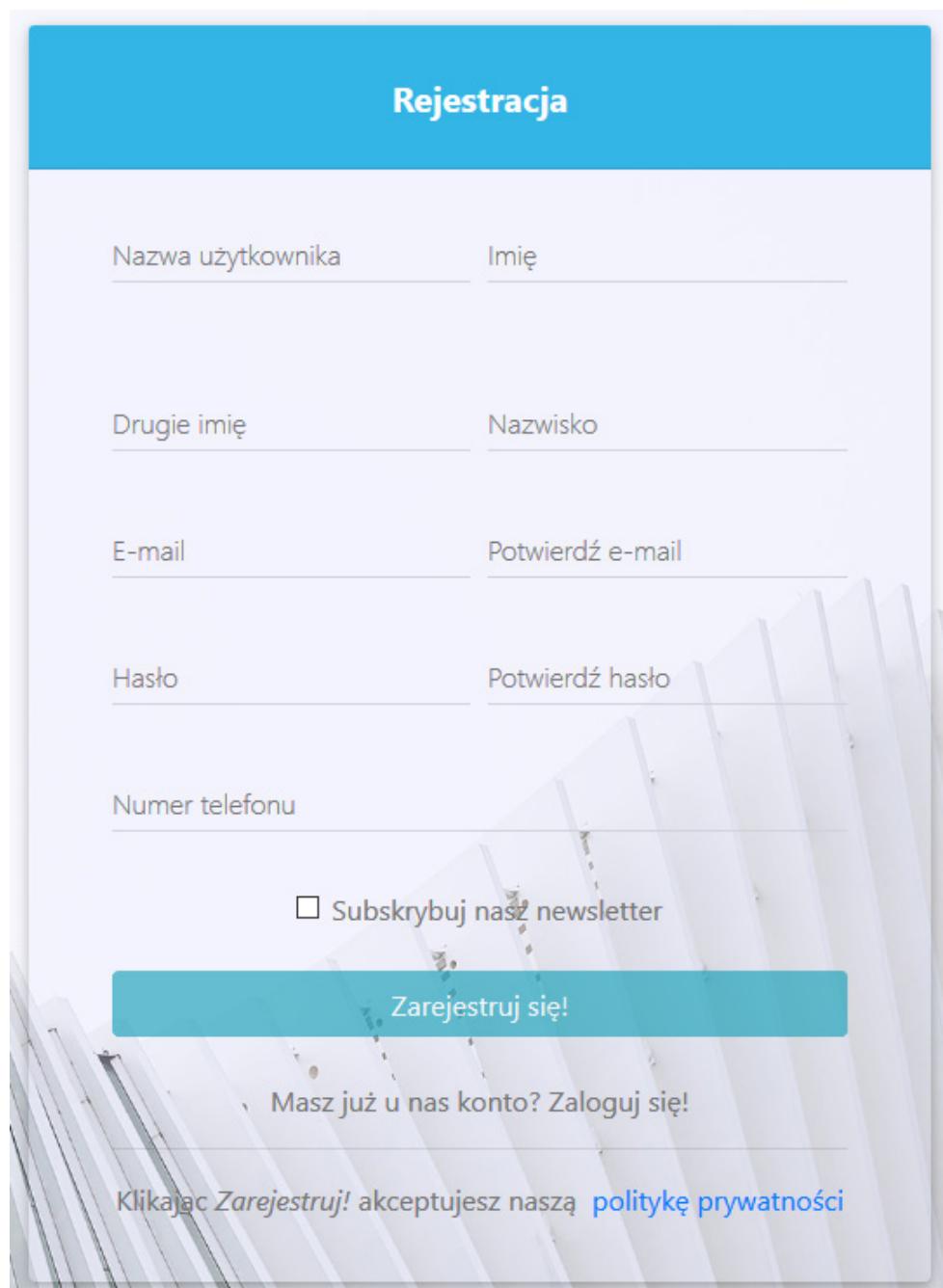
8.2.1 Interfejs użytkownika

Przy pierwszym odwiedzeniu strony, ukazuje się panel logowania, wraz z komponentem, prezentującym losowo wybraną firmę z bazy danych.



Rysunek 8.5: Strona główna. Źródło: Opracowanie własne.

Jako że jest to pierwsza wizyta, należy kliknąć w napis *Nie masz jeszcze konta? Zarejestruj się!*, co zmieni widok na okno rejestracji.

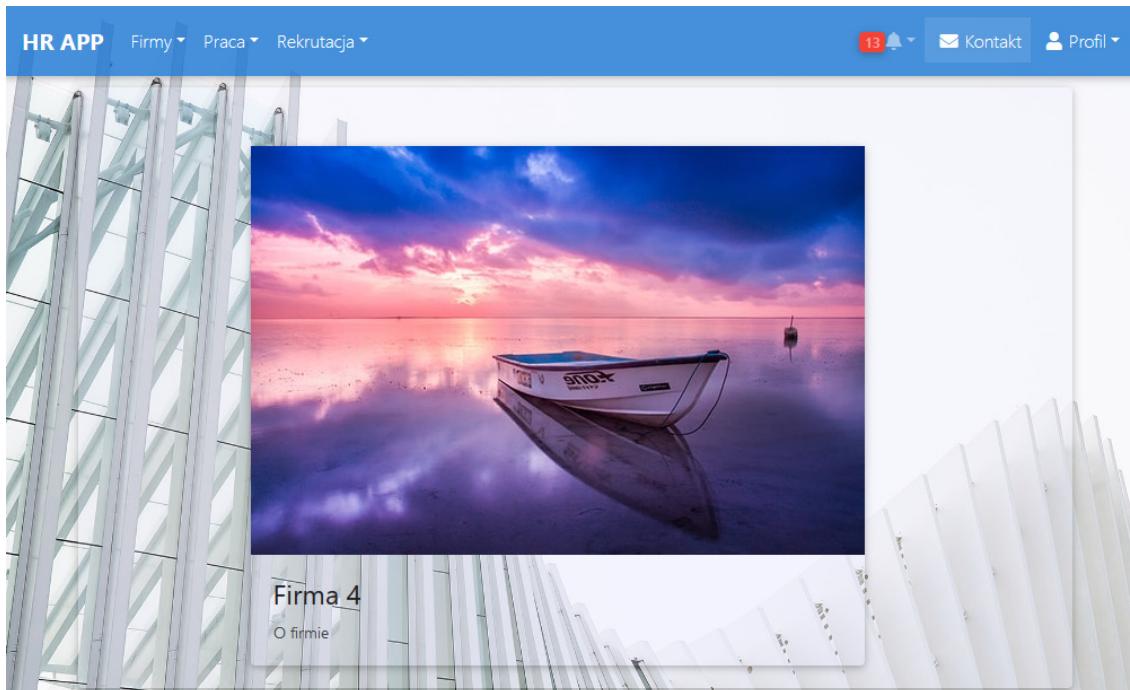


Rysunek 8.6: Okno rejestracji. Źródło: Opracowanie własne.

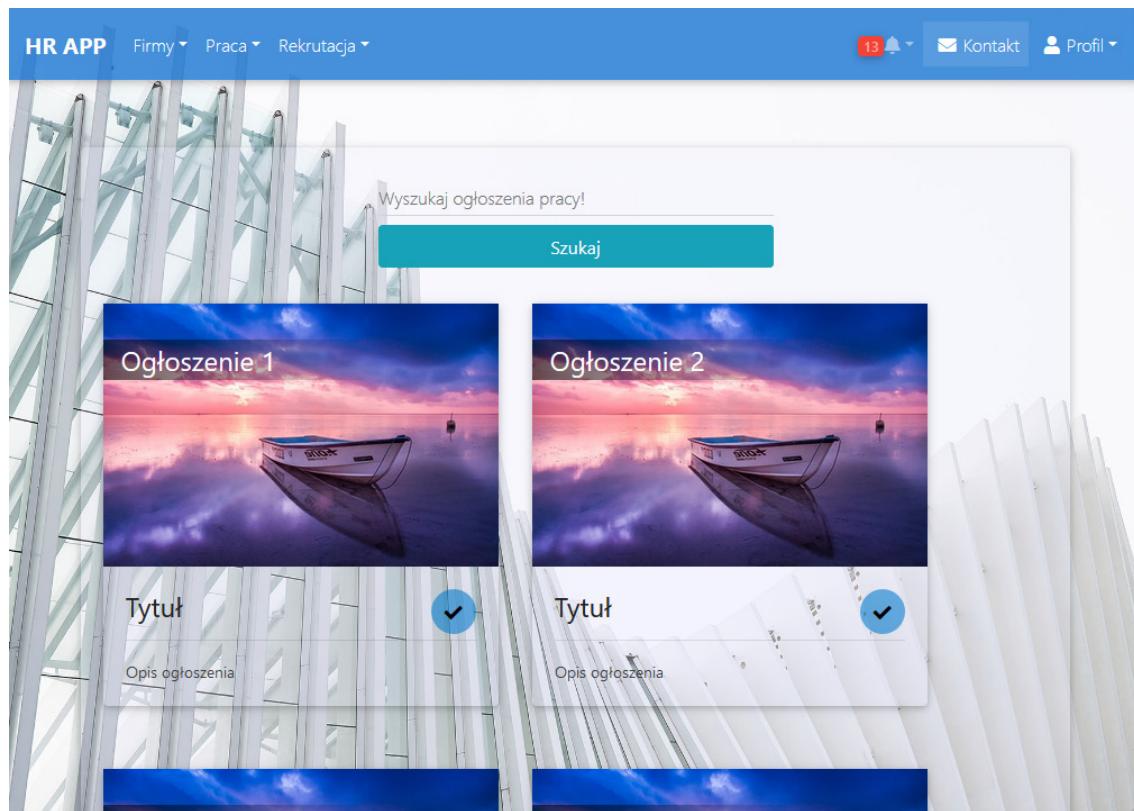
Oczywiście wszystkie pola przed wysłaniem są walidowane po stronie aplikacji internetowej. Jeśli któryś z nich nie spełnia jakiejś normy, to użytkownik zostanie poinformowany o tym, poprzez komunikat koloru czerwonego, pojawiający się pod nieprawidłowym polem.

Po zarejestrowaniu i zalogowaniu, użytkownik zostaje przeniesiony na stronę główną, zawierającą komponent z prezentacją losowej firmy. Komponent ten, jest jedyną częścią wspólną dla wszystkich typów kont. Teraz w zależności, czy zalogowana jest osoba z kontem *USER*, *HR*, *CEO* czy *ADMIN*, taki widok i panele ma dostępne.

Interfejs zwykłego użytkownika:

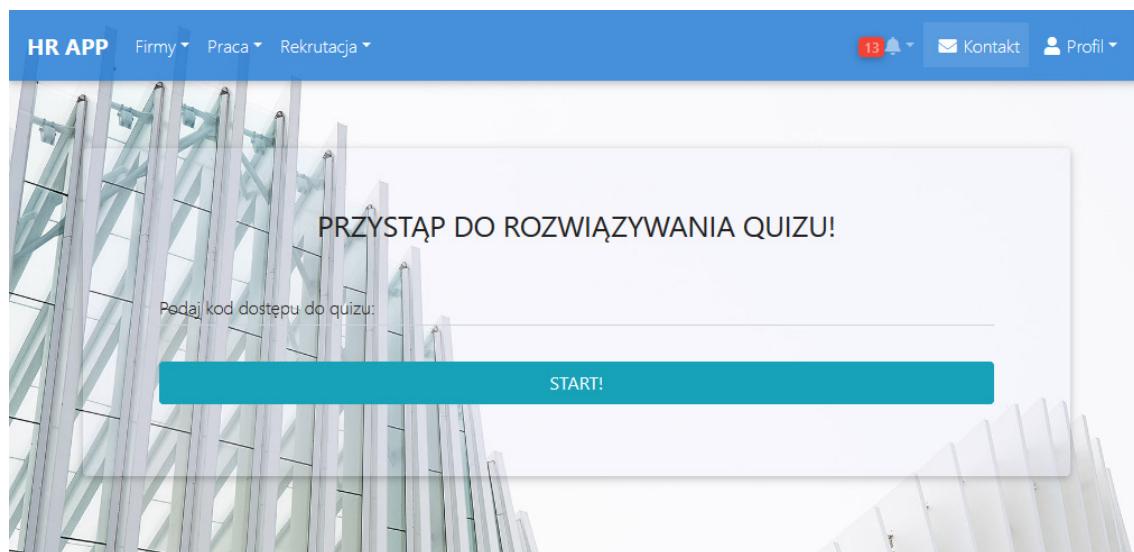


Rysunek 8.7: Okno strony głównej użytkownika. Źródło: Opracowanie własne.

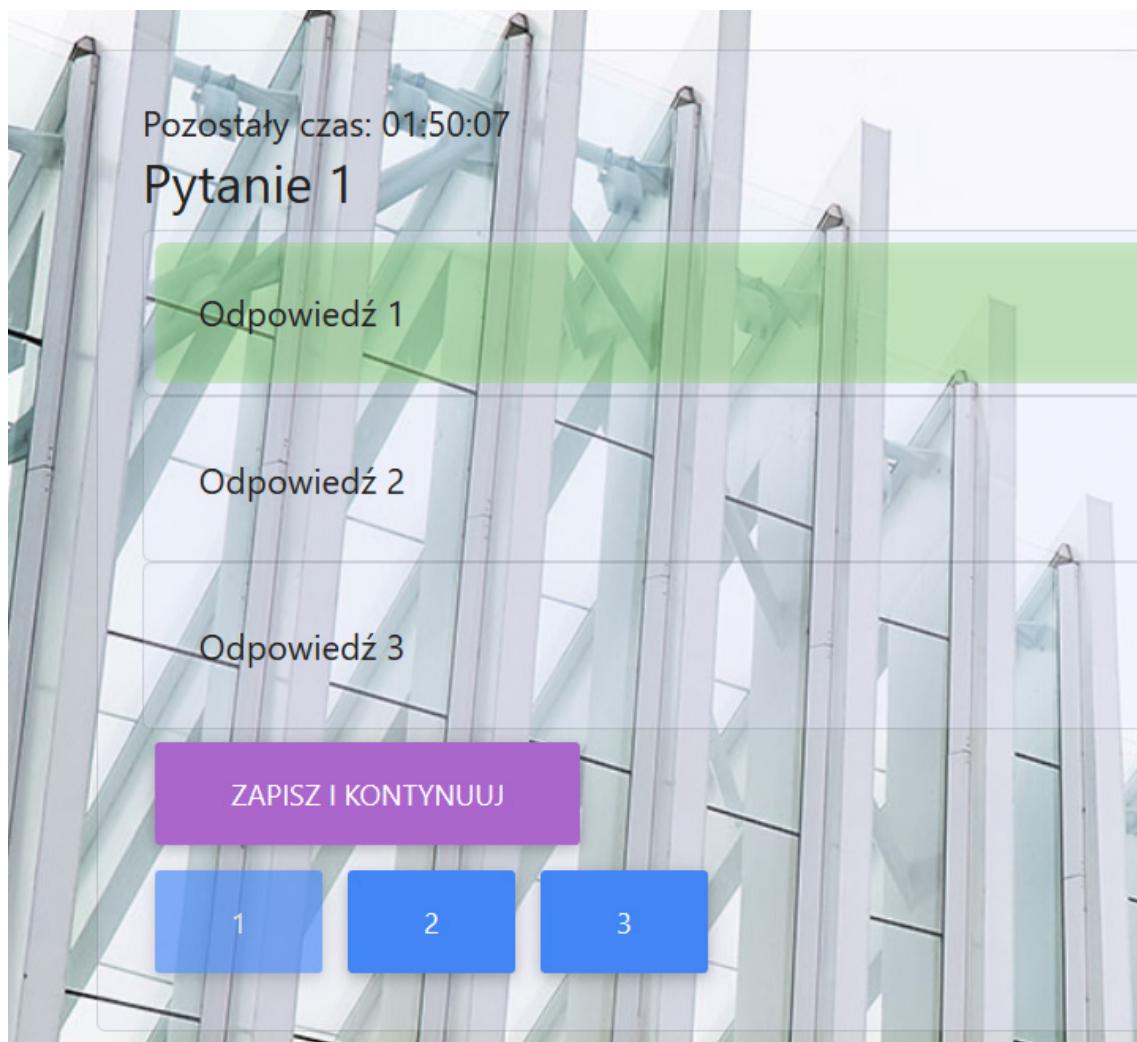


Rysunek 8.8: Okno listy ogłoszeń. Źródło: Opracowanie własne.

Duże karty stworzone w myśl zasad *Material Design* czynią stronę przejrzystą i przyjazną dla urządzeń z mniejszymi ekranami.

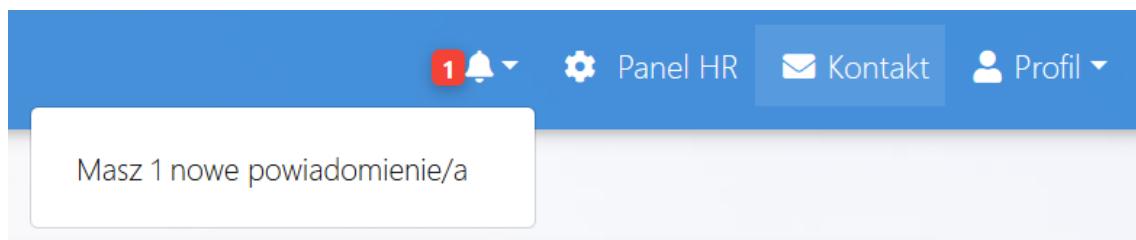


Rysunek 8.9: Okno wyboru testu. Źródło: Opracowanie własne.

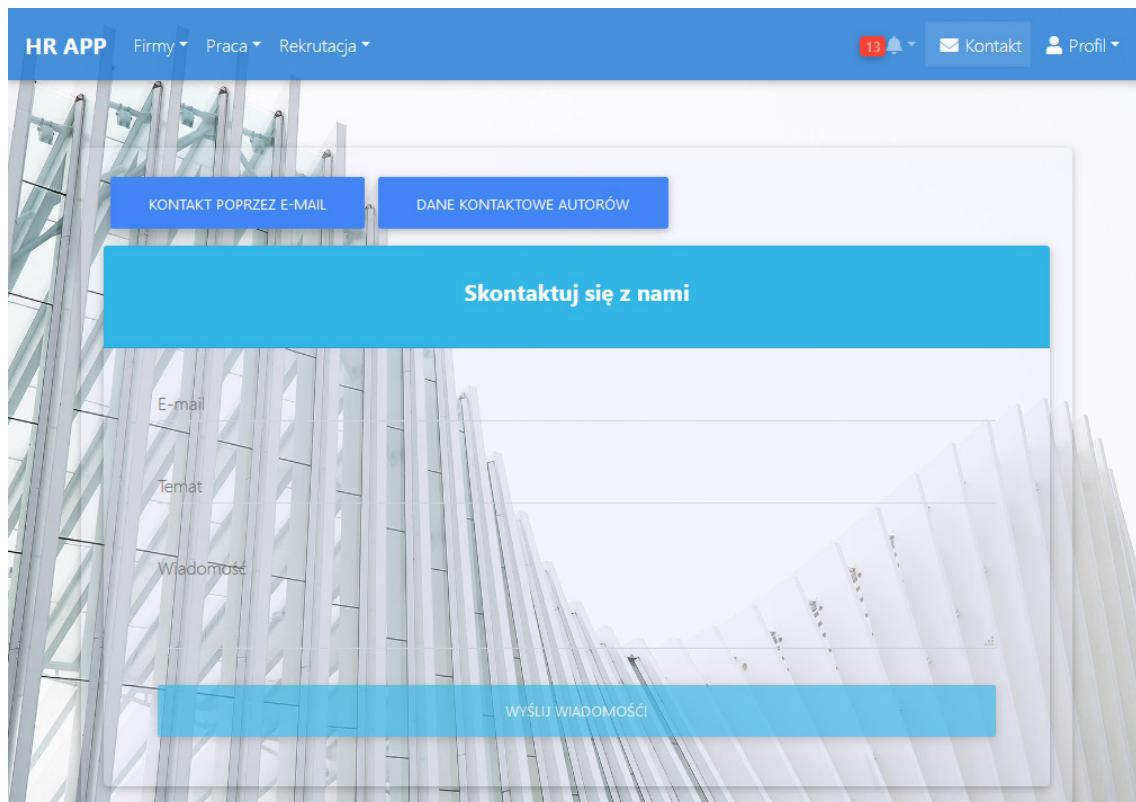


Rysunek 8.10: Okno rozwiązywania testu. Źródło: Opracowanie własne.

Panel rozwiązywania testu (rysunek 8.10) posiada prosty, duży i przejrzysty interfejs, dzięki czemu osoba odpowiadająca na pytania może się skupić na nich samych, zamiast na obsłudze aplikacji.

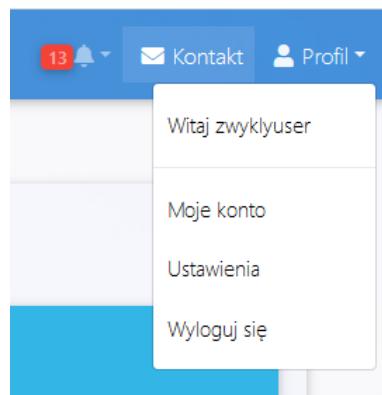


Rysunek 8.11: Rozwinięcie ikonki powiadomień. Kliknięcie w tekst spowoduje przeniesienie do listy aplikacji użytkownika. Źródło: Opracowanie własne.

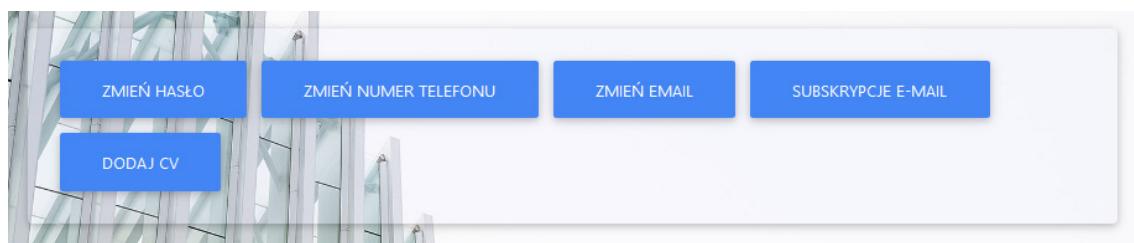


Rysunek 8.12: Okno kontaktu e-mail. Źródło: Opracowanie własne.

Dzięki zastosowaniu obsługi poczty na części serwerowej, został udostępniony formularz kontaktowy dla wszystkich tych, którzy chcieliby podzielić się czymś z administratorami strony.

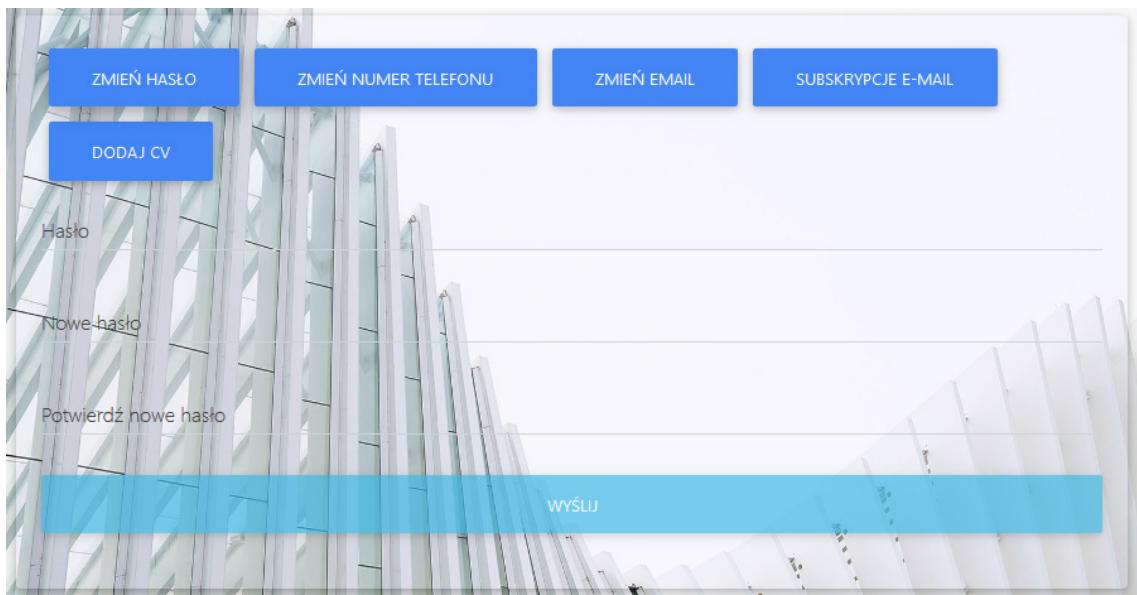


Rysunek 8.13: Rozwinięcie menu profilu. Źródło: Opracowanie własne.



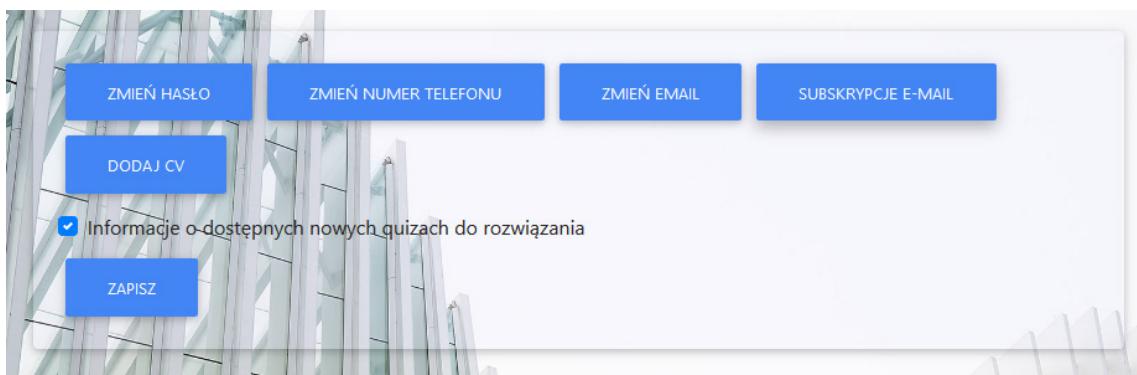
Rysunek 8.14: Okno moje konto. Źródło: Opracowanie własne.

Pod zakładką moje konto znajdziemy szereg przycisków zajmujących się obsługą profilu (rysunek 8.14). Rysunki 8.15, 8.16 oraz 8.17 przedstawiają rozwinięcie wybranych opcji.

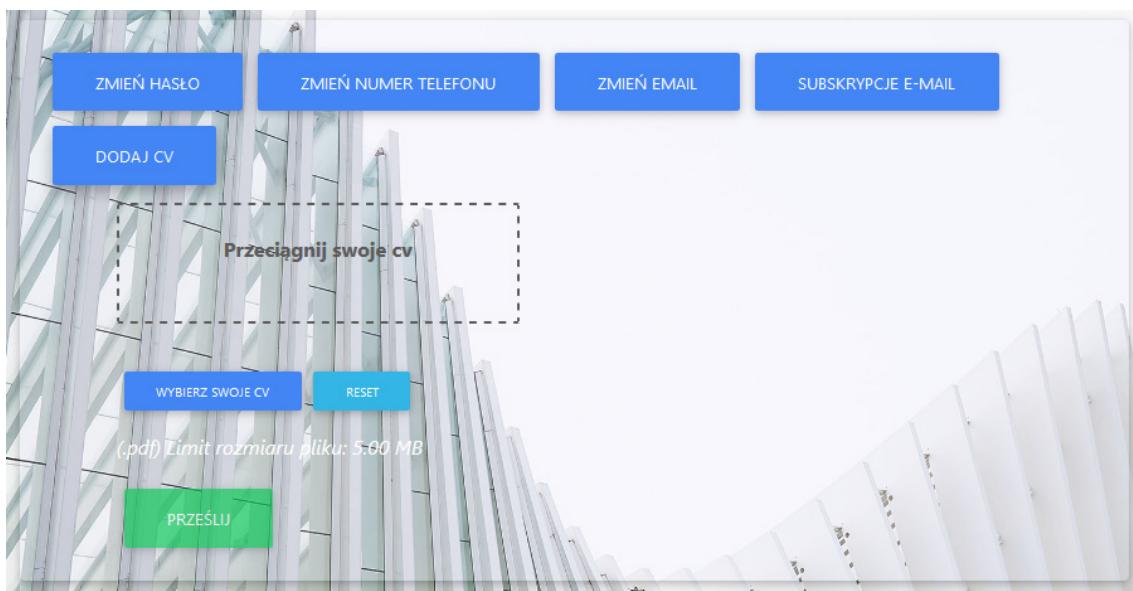


Rysunek 8.15: Okno zmiany hasła. Źródło: Opracowanie własne.

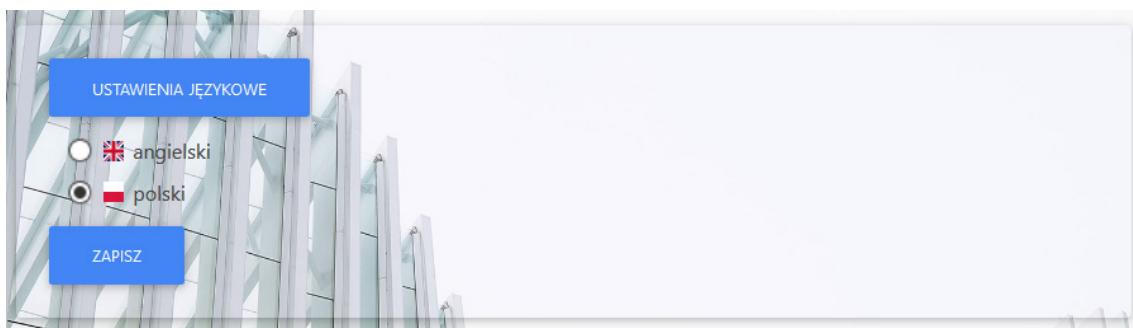
Na rysunku 8.15 widać rozwiniętą opcję zmiany hasła. Każda zmiana jakichkolwiek danych konta, musi być potwierdzona poprzez podanie aktualnego hasła. Przycisk wyslij jest nieaktywny do momentu poprawnego uzupełnienia formularza (jest on także walidowany). Komponenty do zmiany numeru telefonu i adresu e-mail wyglądają identycznie i różnią się tylko odpowiednimi polami.



Rysunek 8.16: Okno subskrypcji e-mail. Źródło: Opracowanie własne.

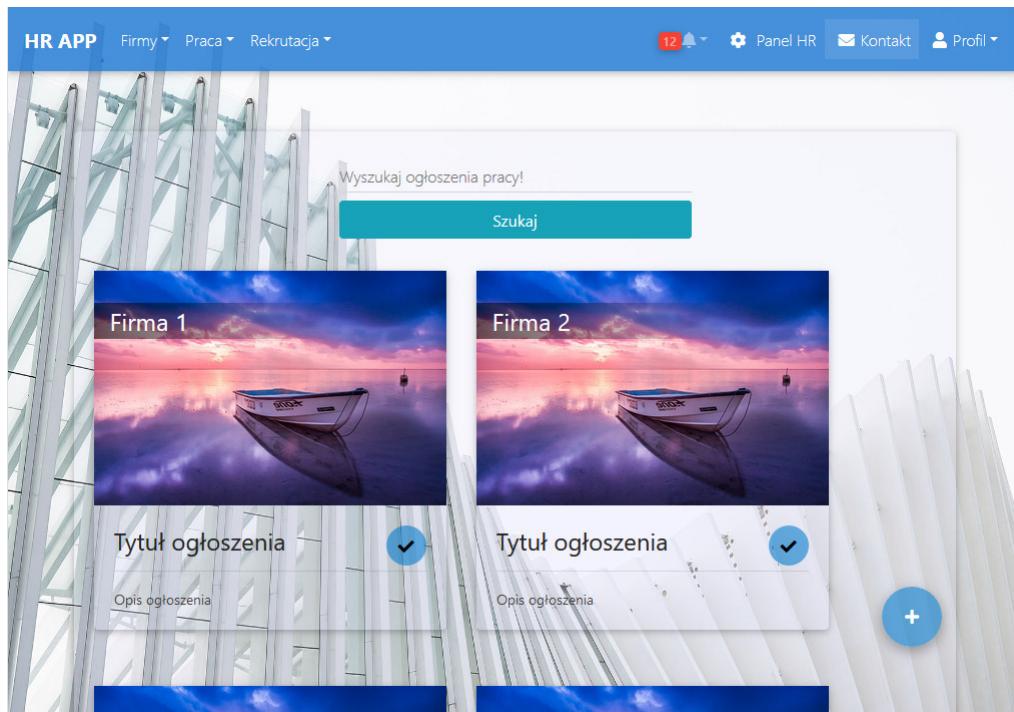


Rysunek 8.17: Okno przesyłania CV. Źródło: Opracowanie własne.

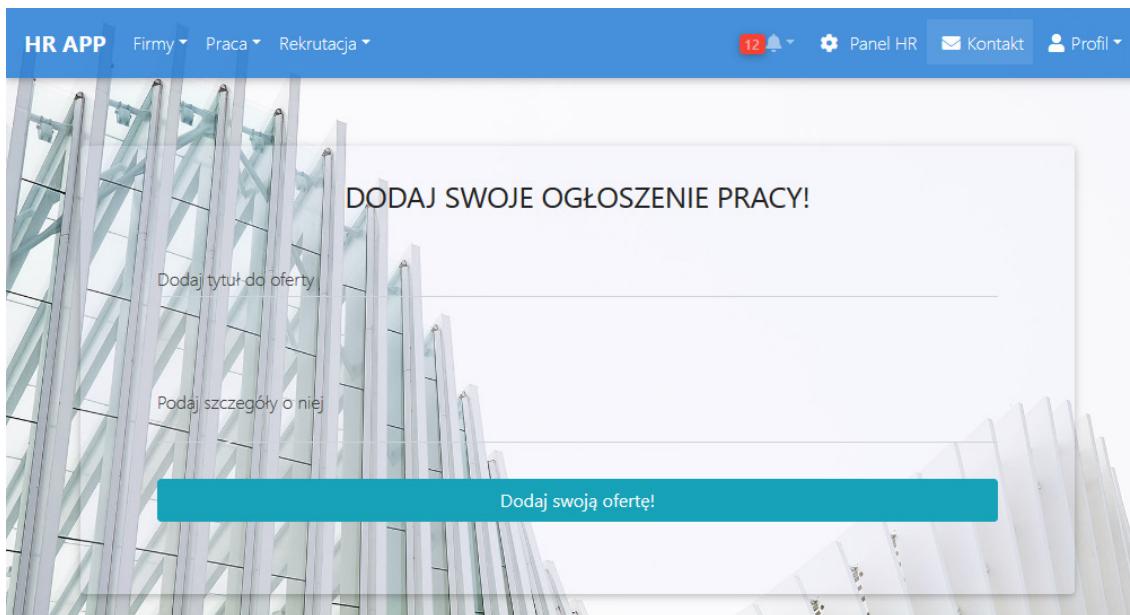


Rysunek 8.18: Okno ustawień. Źródło: Opracowanie własne.

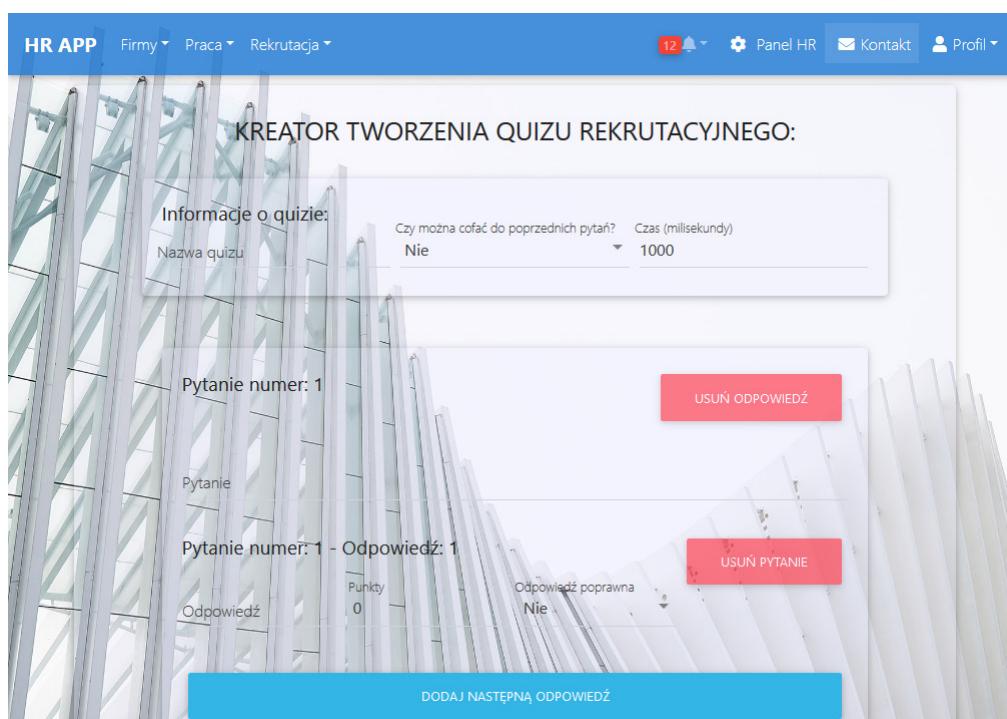
Jako, że aplikacja jest przewidziana dla wielu narodowości, tak też zostały udostępnione dwie opcje językowe do wyboru. Dzięki niezaszywaniu tekstów w znacznikach HTML, a korzystaniu z usługi tłumaczącej, która wstawia w odpowiednie miejsca odpowiednią wersję językową.

Interfejs użytkownika HR:

Rysunek 8.19: Okno listy ogłoszeń, u użytkownika *HR* różni się lewitującym przyciskiem w prawej dolnej części ekranu, który przenosi do widoku tworzenia nowego ogłoszenia. Źródło: Opracowanie własne.



Rysunek 8.20: Widok tworzenia nowego ogłoszenia. Źródło: Opracowanie własne.



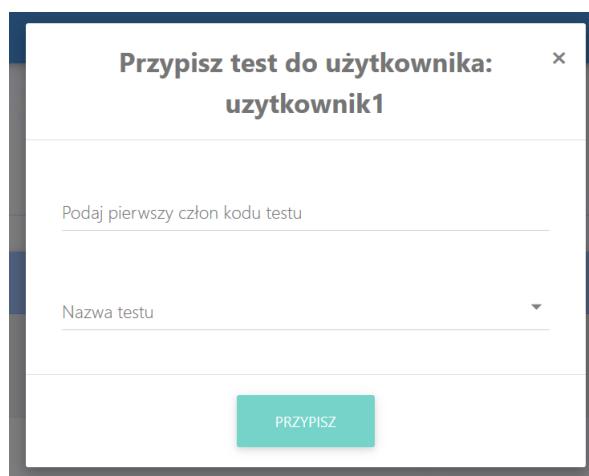
Rysunek 8.21: Tworzenie nowego testu rekrutacyjnego. Źródło: Opracowanie własne.

Jeśli nastąłaby sytuacja, że podczas tworzenia nowego testu rekrutacyjnego, zostanie przerwane połączenie z internetem, bądź wystąpi jakakolwiek inna sytuacja,

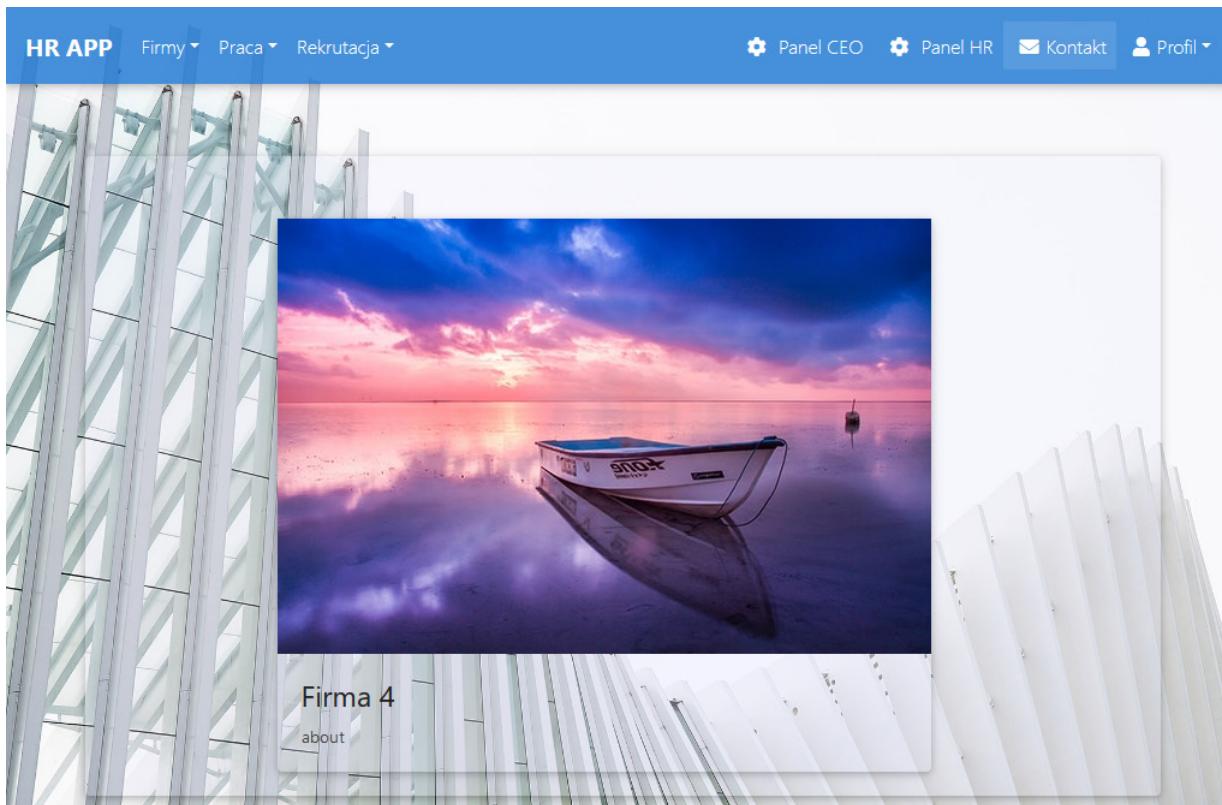
która spowoduje konieczność przerwania prac nad kreatorem, tym samym nie ukańczając całego procesu, to nic straconego. Wersja robocza testu zostaje zapisywana do pamięci podręcznej przeglądarki i podczas powrotu do okna kreacji, zostaniemy poinformowani o tym fakcie i zapytani czy chcemy wczytać zapisany stan. Rysunek z tym komunikatem jest przedstawiony w rozdziale 8.2.2.

Id Zgłoszenia	Tytuł Ogłoszenia	Login Użytkownika	Nazwa Testu	Cv	Kod Testu
130	Tytuł 3	zwyklyuser			
354	titleZ	uzytkownik1			

Rysunek 8.22: Widok listy aplikacji złożonych przez użytkowników. Źródło: Opracowanie własne.

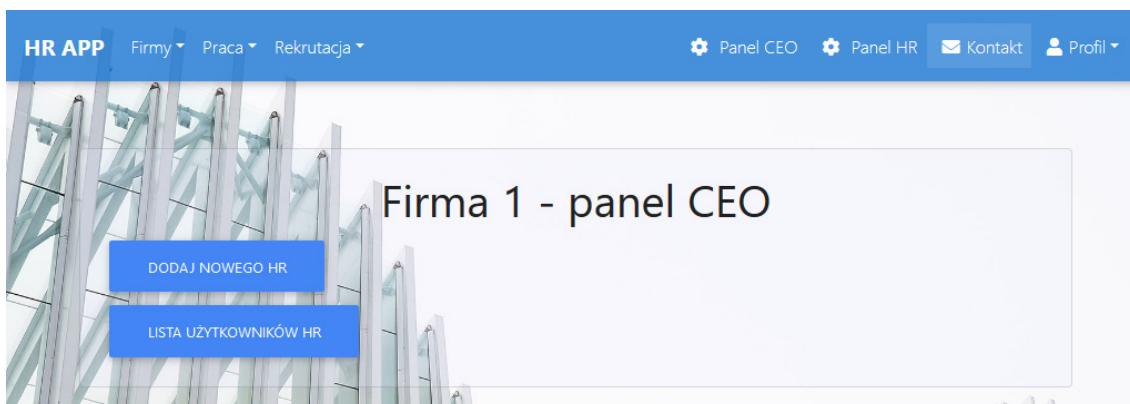


Rysunek 8.23: Okno przypisywania testu do użytkownika. Źródło: Opracowanie własne.

Interfejs użytkownika CEO:

Rysunek 8.24: Okno strony głównej użytkownika *CEO*. Źródło: Opracowanie własne.

Jednymi z funkcjonalności dostępnych tylko dla użytkownika *CEO* jest przypisywanie nowych i kasowanie obecnych użytkowników *HR*. Okno ich wyboru zostało przedstawione na rysunku 8.25.



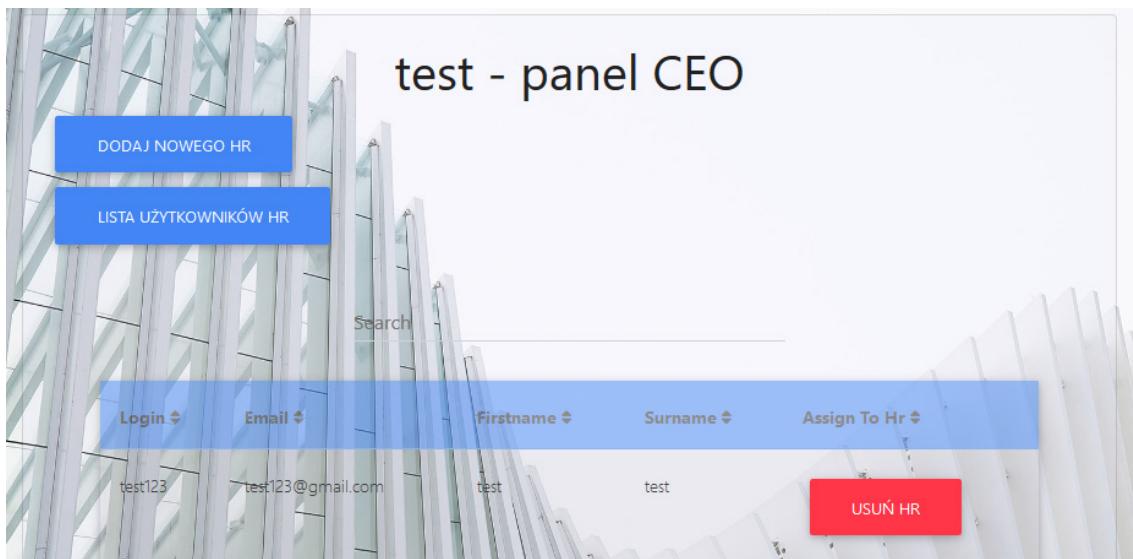
Rysunek 8.25: Panel użytkownika *CEO*. Źródło: Opracowanie własne.

The screenshot shows a registration form titled "FORMULARZ REJESTRACJI FIRMY W BAZIE". The form includes fields for "Nazwa" (Name) and "Miejsce: Kraj/Miasto" (Place: Country/City). A note below the fields says: "Powiedz nam coś o swojej firmie.. Pomoże to wyszukać twoją firmę innym użytkownikom w naszej wyszukiwarce!". A blue button at the bottom right says "Dodaj swoją firmę!" (Add your company!). The background features a stylized image of a modern building's glass facade.

Rysunek 8.26: Okno rejestracji firmy. Źródło: Opracowanie własne.

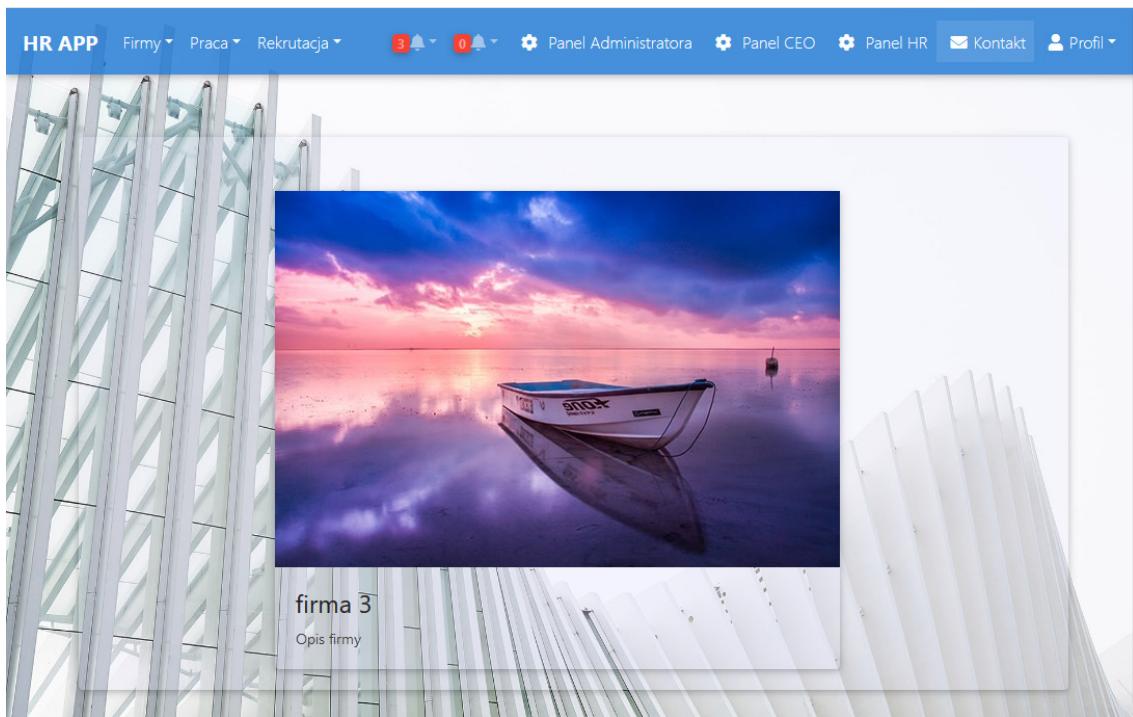
The screenshot shows a table titled "Firma 1 - panel CEO" with a header "DODAJ NOWEGO HR" (Add new HR) and a button "LISTA UŻYTKOWNIKÓW HR" (List of HR users). The table has columns: Login, Email, Firstname, Surname, and Assign To Hr. It lists four rows: log1, log@log-log, log, and log. A blue button at the bottom right says "PRZYPISZ DO HR" (Assign to HR).

Rysunek 8.27: Fragment tabeli dodawania użytkownika *HR* do firmy. Źródło: Opracowanie własne.

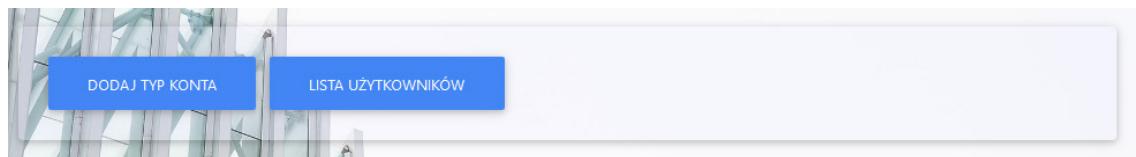


Rysunek 8.28: Fragment tabeli wykluczania użytkownika *HR* z firmy. Źródło: Opracowanie własne.

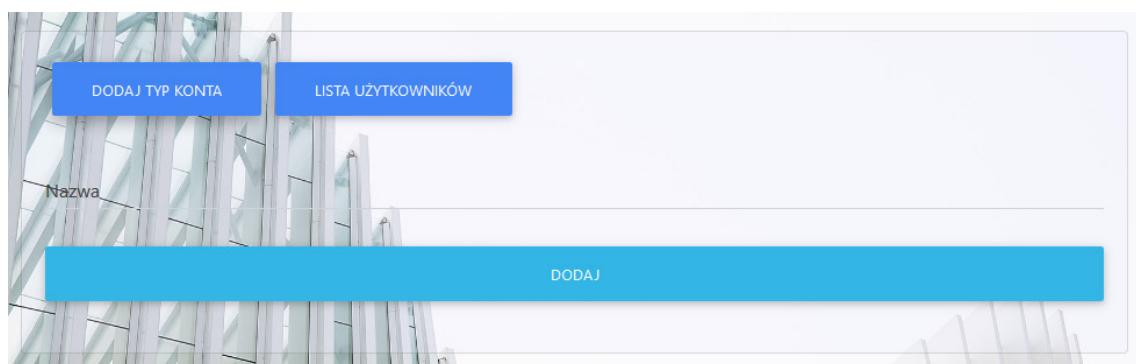
Interfejs użytkownika ADMIN.



Rysunek 8.29: Okno strony głównej administratora. Źródło: Opracowanie własne.

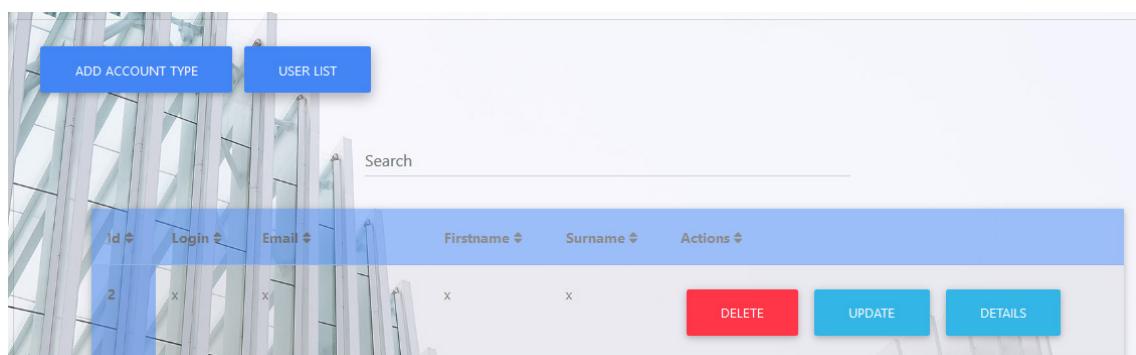


Rysunek 8.30: Panel administratora. Źródło: Opracowanie własne.

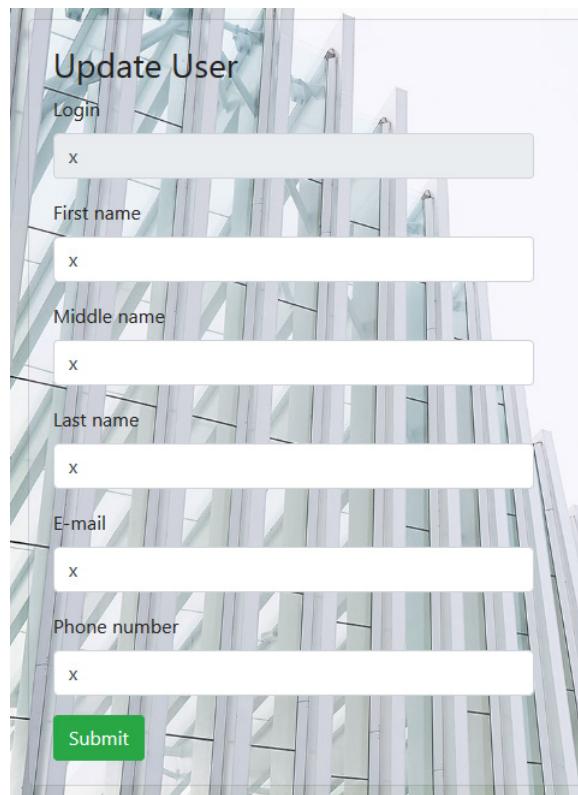


Rysunek 8.31: Panel dodawania typu konta. Źródło: Opracowanie własne.

Jako, że projekt jest wysoce przygotowany na zmiany i rozbudowę, zespół projektowy przygotował pole, gdzie administrator może dodać kolejne typy kont, jeśli tylko znajdzie taka potrzeba.

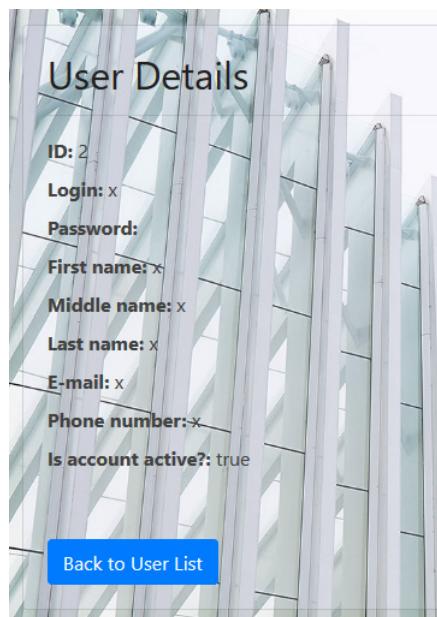


Rysunek 8.32: Fragment listy użytkowników. Źródło: Opracowanie własne.



Rysunek 8.33: Okno edycji użytkownika. Źródło: Opracowanie własne.

Administrator jako jedyny ma możliwość pełnej edycji danych użytkownika.



Rysunek 8.34: Okno szczegółów użytkownika. Źródło: Opracowanie własne.

Oczywiście strona jest w pełni responsywna i jej wygląd zmienia się wraz z rozdzielczością ekranu, na którym ją przeglądamy.

8.2.2 Biblioteki

- **chart.js**

Biblioteka została zainportowana na wypadek, gdyby w aplikacji była konieczność wdrożenia wykresów np. dla statystyk użytkowników, czy testów.

- **crypto-js**

Pozwala na szyfrowanie danych. Została wykorzystana podczas zapisywania danych o tworzonym teście do *localStorage*. Zapewnia to bezpieczne przechowywanie danych o rozpoczętym tworzonym teście, ale jeszcze nie zapisanym do bazy. Takie rozwiązanie zostało zaimplementowane, by twórca testu nie musiał od razu go zapisywać, tylko – by do momentu zatwierdzenia – przechowywany był lokalnie. Zaszyfrowanie zawartości *JSON* zabezpiecza przed podejrzeniem *localStorage* przed osobami niepowołanymi.

- **flag-icon-css**

Mała biblioteka, pozwalająca na insercję do pliku *HTML* ikonki flagi. Wykorzystana przy wyborze języka aplikacji.

```
<span  
    class="flag-icon_flag-icon-{{language | transformFlag}}">  
</span>
```

Listing 29: Implementacja ikonki flagi. Źródło: Opracowanie własne.

Gdzie $\{{\textit{language} \mid \textit{transformFlag}}\}$ to skrócona nazwa języka, np. *pl*, *en*.

- **jwt-decode**

Pozwala na dekodowanie tokenów *JWT*. Dzięki tej bibliotece, można skorzystać z przesyłania przycisków, miejsc w aplikacji, do których użytkownik nie ma dostępu, ponieważ nie ma przydzielonych uprawnień do tych zasobów.

```
public isAuthenticated(): boolean {
    const jwtHelper = new JwtHelperService();
    const token = localStorage.getItem(TOKEN_KEY);

    return !this.jwtHelper.isTokenExpired(token);
}
```

Listing 30: Implementacja sprawdzenia poprawności tokenu oraz jego ważności. Źródło: Opracowanie własne.

```
public saveUserInLocalStorage(token) {
    if (token) {
        const decodedToken = decode(token);
        this.saveToken(token);
        this.saveUser(decodedToken.sub);
        this.saveRole(decodedToken.role);
    }
}
```

Listing 31: Pobranie nazwy użytkownika i przypisanej roli do localStorage. Źródło: Opracowanie własne.

Poza zabezpieczeniem od strony klienckiej, również serwer jest zabezpieczony przed niepowołanym dostępem do zasobów. Jeżeli użytkownik, pomimo braku uprawnień, spróbuje odpytać punkt końcowy, do którego nie ma dostępu, nie uzyska odpowiedzi.

- **mdbootstrap**

Biblioteka, która wspiera szybkie tworzenie stron internetowych, za pomocą wbudowanych stylów. W projekcie została wykorzystana darmowa, ograniczona wersja.

- **ngx-toastr**

Paczka, która pozwala na implementację małych powiadomień po wykonaniu akcji. W projekcie powiadomienia pokazują się np. podczas wystąpienia błędu po stronie serwera, po poprawnym dodaniu testu itp.



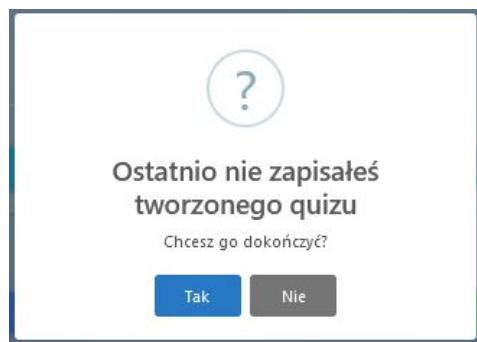
Rysunek 8.35: Komunikat błędu. Źródło: Opracowanie własne.

- **rxjs**

Biblioteka została wykorzystana w celu obsługi zapytań serwera. Szerzej opisana została w rozdziale **8.2.11**.

- **sweetalert2**

Dodatek, który pozwala na tworzenie wyskakujących okienek, z którym użytkownik może wchodzić w interakcję. Został wykorzystany np. w pytaniu o potwierdzenie, czy użytkownik chce wczytać poprzednio tworzony test, czy rozpocząć tworzenie nowego.



Rysunek 8.36: Pytanie stworzone za pomocą *SweetAlert2*. Źródło: Opracowanie własne.

8.2.3 Budowa projektu

Budowa projektu zostanie omówiona z pominięciem podstawowych plików, generowanych przy każdej aplikacji, wytworzonej w środowisku programistycznym *Angular*. Uproszczone drzewo plików:

```
>src
  >app
    >classes
    >common
    >components
      >navbar
        -navbar.component.html
        -navbar.component.scss
        -navbar.component.ts
      >panels
        >user-panel
          >user-list-of-applications
            -user-list-of-applications
              .component.html
            -user-list-of-applications
              .component.scss
            -user-list-of-applications
              .component.ts
            -user-panel.component.html
            -user-panel.component.scss
            -user-panel.component.ts
          ...
        >factories
        >helpers
        >modules
        >pipes
        >services
          >security
            -auth-guard.service.ts
          ...
          -mailing.service.ts
          ...
        -app.component.html
```

```
—pp . component . scss  
—app . component . ts  
>assets  
>environments
```

Listing 32: Uproszczone drzewo plików aplikacji internetowej. Źródło: Opracowanie własne.

W głównym folderze *src*, znajduje się kod źródłowy, rozmieszczony po różnych podfolderach. Znajdują się w nim również: plik odpowiedzialny za trasowanie stron oraz najważniejsze cztery pliki, czyli: *app.component.html*, *app.component.scss*, *app.component.ts* oraz *app.module.ts*, czyli składowe głównej strony.

W katalogu *classes* znajdują się pliki *TypeScript*, określające klasy danych, później używanych w kodzie. Jest tam między innymi definicja klasy użytkownika w pliku *user.ts*.

W lokacji *common* został umieszczony plik *globalconstants.ts*, w którym są stałe globalne, takie jak adres serwera, czy opcje *http* potrzebne do generowania zapytań.

Największym z podfolderów jest *components*. Jak sama nazwa wskazuje, znajdują się tam wszystkie komponenty (które także są umieszczone we własnych podfolderach) użyte do budowy aplikacji. Można tam znaleźć takie elementy, jak: wszystkie panele użytkowników, pasek nawigacji, czy stronę edycji danych konta.

Kolejną, mniejszą lokacją, jest *factories*, która przewidziana jest na fabryki. Znajduje się tam tylko jeden plik *appInitializerFactory.ts*, którego zadaniem jest implementacja działania usługi językowej aplikacji.

Folder *helpers* z założenia ma zawierać pliki, ułatwiające walidację formularzy. Znajduje się tam plik *must-match.ts*, który porównuje dwa przekazane mu argumenty, czy są identyczne. Jest używany między innymi do walidacji adresu e-mail przy zakładaniu konta.

Katalog *modules* przewidziany jest na przechowywanie modułów. W projekcie znalazłem tam miejsce modułu *pipes.module.ts*, który jest używany między innymi do tłumaczeń tekstu w plikach *HTML*.

W lokacji *pipes* jest miejsce na funkcje bezpośredniego przekształcania. Znajduje się tam plik *transform-flag.pipe.ts*, który jest odpowiedzialny za dostosowanie flagi przypisanej do języka wybranej nacji.

Services to druga większa lokacja, skupiająca serwisy. Znajduje się w niej tyl-

ko jeden podfolder *security*, w którym skatalogowane są pliki odpowiedzialne za kontrolę dostępu, autoryzację, szyfrowanie i obsługę tokenów.

8.2.4 HTTP interceptors

Interceptory służą do przechwytywania zapytań *http* i ich modyfikacji. W *Angularze* zostały zaimplementowane narzędzia do ich obsługi, które zapewniają ich prawidłowe działanie. Poprzez interceptory możemy przechwycić każde wywołanie *http* od strony klienckiej oraz zmodyfikować jego zawartość, po czym przekazać zapytanie dalej, do obsługi wewnętrz aplikacji. Można również przy pomocy interceptorów wykonywać operacje w drugą stronę, tzn. przechwytywać odpowiedzi z serwera, a następnie wykonywać na nich dowolne operacje. Aby poinformować aplikację o chęci korzystania z interceptorów, należy w module (najlepiej głównym) zaimportować interceptory.

```
import { http_INTERCEPTORS } from '@angular/common/http';
```

Listing 33: Przykład importowania interceptorów. Źródło: Opracowanie własne.

Następnie należy poinformować moduł aplikacji o tym, która klasa będzie miała możliwość edycji zapytań.

```
providers [  
  ...  
  {  
    provide: http_INTERCEPTORS,  
    useClass: TokenInterceptor ,  
    multi: true ,  
  }  
  ...  
]
```

Listing 34: Poinformowanie aplikacji o możliwości edycji zapytań. Źródło: Opracowanie własne.

W przypadku tej aplikacji, głównym zadaniem interceptorów jest doczepianie *Bearer tokenu* do nagłówka zapytania. Token otrzymywany jest po udanej próbie zalogowania. Pobierany jest z lokalnych zasobów przeglądarki predefiniowanych dla

wybranego adresu (ang. *local storage*). Oprócz tego, dodatkowo doczepiana jest informacja o typie *MIME* (ang. *Multipurpose Internet Mail Extensions*), czyli rodzaju wysyłanego pliku poprzez zapytanie. W przypadku tej aplikacji, w większości konstruowane są zapytania w formacie *JSON*.

8.2.5 Przechowywanie informacji

W aplikacji internetowej wykorzystywana jest pamięć lokalna (ang. *local storage*). Są w niej przechowywane takie dane, jak:

- auth-role – typ zalogowanego konta,
- auth-token – token uwierzytelniający użytkownika,
- auth-user – login użytkownika,
- lang – język interfejsu aplikacji,
- quiz-create-panel-data – dane zapisane z nieukończonego procesu tworzenia nowego testu.

Key	Value
auth-role	HR
auth-tok...	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJocjEiLCJyb2xlIjoSF
auth-user	hr1
lang	pl
quiz-crea...	U2FsdGVkX1/G9BCdzSLXQzBgUtVBQyx1Hn6hjqflzfbaMaoN2BseMcBy9ETCon

Rysunek 8.37: Przykładowe dane przechowywane w pamięci lokalnej. Źródło: Opracowanie własne.

8.2.6 Wykorzystane szyfrowanie

W części webowej aplikacji, szyfrowanie zostało użyte do przetrzymywania w *local storage* informacji o tworzonym teście – tak, by móc wrócić do jego tworzenia w przyszłości. W tym cel została wykorzystana biblioteka *crypto-js*. W projekcie wykorzystano szyfrowanie za pomocą *AES* (*Advanced Encryption Standard*), czyli symetrycznego szyfru blokowego. *AES* wykorzystuje sieć substytucji i permutacji

macierzową. Do szyfrowania i deszyfrowania danych, wykorzystywany jest zdefiniowany, sekretny klucz o długości 128, 192 lub 256 bitów. Im dłuższy klucz, tym większa jest liczba rund szyfrujących²⁹.

W procesie szyfrowania tekst jawny dzielony jest na bloki 128-bitowe. Bloki tworzone są jako macierze 4×4 :

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

Dla każdej z macierzy wykonywane są następujące operacje:

- Przygotowanie podkluczy

Następuje wygenerowanie jednego podklucza początkowego, a następnie po jednym kluczku dla każdej z rund szyfrujących.

- Runda inicjująca

Na każdym bajcie w bloku danych wykonywana jest operacja sumowania XOR , z odpowiadającym mu pierwszym podkluczem.

- n rund szyfrujących

Gdzie n dla klucza 128-bitowego wynosi 9, dla klucza 192-bitowego – 11, natomiast dla 256-bitowego – 13.

Każda runda szyfrująca składa się z następujących operacji:

- Zastępowanie każdego bajtu innym bajtem — wykorzystuje się do tego tzw. *S-Box'y Rijndael'a*.
- Przesunięcie bajtów w trzech ostatnich macierzach stanu w lewo.

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix} \rightarrow \begin{bmatrix} b_4 & b_8 & b_{12} & b_0 \\ b_5 & b_9 & b_{13} & b_1 \\ b_6 & b_{10} & b_{14} & b_2 \\ b_7 & b_{11} & b_{15} & b_3 \end{bmatrix}$$

- Mnożenie kolumn — każda z kolumn w macierzach stanu jest przemnażana przez macierz 4×4 bajtów.

²⁹Materiały serwisu Crypto-IT, <http://www.crypto-it.net/pl/symetryczne/aes.html>, (dostęp 14.12.2020)

- Działanie *XOR* na wszystkich bajtach bloku danych i bajtach podklucza danej rundy.
- Runda kończąca

Wykonywane są wszystkie operacje z rundy szyfrującej, poza operacją mnożenia kolumn.

Przykładowy wynik szyfrowania ciągu znaków zbudowanego z obiektu *JSON*, który przechowywał strukturę tworzonego testu:

```
U2FsdGVkX1+AtQ8Tg2ru29Lfw7Ub0qK4jNwQw1B+RNFdKAO3nF/9K/ge
8yyv0pleYsqVyDkJjKaWQQ72huq7eYy7dKaES+JhjrfsDxhyyxHEltAT
ad8vJHBcarNckEXhXk4ySohvnoQZRRDO2WM5TQ0a9+esoHVrOXNcsyhm
oyGX/cEFOt78cBpKkpKCX7+GZQphyQPSZ3owusSlOb950XslpNnDyOA
RS+Wx/V3T92RLEdP+nrVnqFdSxwafjreiih4dqxyvY/BCWzQT9tLpV
3W4la6ZLmixjlh95Kww7jUpS1L8i4NV7cE1Mhd1NsTx4l4Tr4P2iV2Ue
KR3a3aZ7YsMYhwBNQ0nOv7Y2Lxj+6Vood5OWWOtKXdLXiniIJ4l3iQ
d3x8jGyijrhTSAQFXGOlFU2qDsRPWsHNwCb9hMjjHETQsQvR9I8VE1S
```

Listing 35: Zaszyfrowany obiekt *JSON* przechowujący strukturę tworzonego testu.
Źródło: Opracowanie własne.

```
import { Injectable } from '@angular/core';
import * as CryptoJS from 'crypto-js';

@Injectable({
  providedIn: 'root',
})
export class CryptoService {

  encryptSecretKey = 'ca17828f861a0a6c7cc70586af2e0ee1';

  constructor() { }

  encryptData(data) {
    try {
      return CryptoJS.AES.encrypt(JSON.stringify(data)
```

```
    , this.encryptSecretKey).toString();  
} catch (e) {  
    return null;  
}  
}  
  
decryptData(data) {  
    try {  
        const bytes = CryptoJS.AES.decrypt(data,  
            this.encryptSecretKey);  
        if (bytes.toString()) {  
            return JSON.parse(bytes.toString(CryptoJS  
                .enc.Utf8));  
        }  
        return data;  
    } catch (e) {  
        return null;  
    }  
}
```

Listing 36: Implementacja szyfrowania i deszyfrowania poprzez AES wykorzystując wygenerowany sekretny klucz. Źródło: Opracowanie własne.

8.2.7 Angular Hooks

Każdy komponent i dyrektywa (ang. *directive*) w *Angularze* posiada swój cykl życia. Aby mieć jak największą kontrolę nad aplikacją, należy się w jakiś sposób podpiąć pod te cykle. Można to zrobić właśnie za pomocą haków cyklu życia komponentu (ang. *lifecycle hooks*). Wywołują się one w takiej kolejności:

- constructor,
- ngOnChanges,
- ngOnInit,

- ngDoCheck,
- ngAfterContentInit,
- ngAfterContentChecked,
- ngAfterViewInit,
- ngAfterViewChecked,
- ngOnDestroy.

Fazy te można podzielić na te połączone z samym komponentem albo z dzieckiem komponentu. Hakami komponentów są:

- Konstruktor (ang. *constructor*), który wywoływany jest tylko raz, wtedy gdy *Angular* tworzy nowy komponent albo dyrektywę, poprzez powołanie ich do życia za pomocą słowa kluczowego *new*.
- Metoda **ngOnChanges** wywoływana jest każdego razu, kiedy następuje jakaś zmiana w argumencie wejściowym. Wywołuje się ona wtedy wraz z uwzględnieniem tego parametru.
- Metoda **ngOnInit** wywołuje się, kiedy komponent zostanie zainicjalizowany. Hak ten wykonuje się tylko raz, zaraz po pierwszym wywołaniu *ngOnChanges*.
- Metoda **ngDoCheck** jest wywoływana zaraz po *ngOnInit* oraz wtedy, kiedy zostaje wywołany detektor zmian (ang. *change detector*). Rozszerza ona działanie *ngOnChange*, ponieważ reaguje na zmianę właściwości obserwowanego obiektu, a *ngOnChange* reaguje tylko na zmianę jego referencji.
- Metoda **ngOnDestroy** zostaje wywołana zaraz przed momentem, gdy *Angular* niszczy komponent. Tego haka używa się między innymi do odwołania subskrypcji obiektów typu *observable* oraz odpięcia obsługi zdarzeń (ang. *event handler*). Zapobiega to wyciekom pamięci.

Hakami *dzieci komponentów* są zaś:

- Metoda **ngAfterContentInit**, która wywoływana jest po tym, kiedy *Angular* dokonuje osadzenia komponentu wewnątrz innego komponentu.

- Metoda **ngAfterContentChecked**, która wywoływana jest zaraz po haku *ngAfterContentInit* oraz za każdym razem, gdy zawartość danego komponentu ulegnie zmianie.
- Metoda **ngAfterViewInit** wykonywana jest zaraz po wyrenderowaniu widoku strony. Dzięki niej można zaoszczędzić sobie wielu kłopotów, wiążących się z brakiem synchronizacji widoku z połączonymi do niego zmiennymi.
- Metoda **ngAfterViewChecked** wywoływana jest po wygenerowaniu widoku komponentu oraz po wystąpieniu zmian w nim.

8.2.8 Przykłady haków cyklu życia aplikacji

NgOnInit użyty w tym kontekście do wywołania metody *getAllJobOffers* oraz do walidacji pola *jobOfferSearchForm* formularza *jobOfferToSearch*.

```
ngOnInit (){  
    this . getAllCompanies ();  
    this . companySearchForm = this . formBuilder . group ({  
        companyToSearch: [ ' ', [ Validators . required ,  
            Validators . minLength (1)]]  
    });  
}
```

Listing 37: Przykład użycia OnInit. Źródło: Opracowanie własne.

Konstruktor użyty do wstrzykiwania zależności, w tym przypadku do serwisu, odpowiedzialnego za obsługę punktów końcowych, związanych z ogłoszeniami ofert pracy oraz serwisu, który generuje powiadomienia typu *toast*.

```
constructor (  
    private jobOffersService: JobOffersService ,  
    private toast: ToastService  
) { }
```

Listing 38: Użycie konstruktora do generowania powiadomień. Źródło: Opracowanie własne.

NgAfterViewInit używany do aktualizacji składników interaktywnej tabeli. Określone w nim są:

- maksymalna ilość wyświetlanych elementów na stronie tabeli,
- indeks pierwszego elementu, wyświetlonego na aktualnej stronie tabeli,
- indeks ostatniego elementu, wyświetlonego na aktualnej stronie tabeli,
- wywołanie metody *detectChanges*, która wykrywa wszelkie zmiany w obrębie tabeli i podejmuje według nich określone akcje.

```
ngAfterViewInit (){  
    this . mdbTablePagination  
        . setMaxVisibleItemsNumberTo (  
    this . maxVisibleItems );  
    this . mdbTablePagination  
        . calculateFirstItemIndex ();  
    this . mdbTablePagination  
        . calculateLastItemIndex ();  
    this . cdRef . detectChanges ();  
}
```

Listing 39: Przykład użycia NgAfterViewInit. Źródło: Opracowanie własne.

8.2.9 Powiadomienia w czasie rzeczywistym

W aplikacji zaimplementowano powiadomienia w czasie rzeczywistym. Działają one na dwóch tabelach:

- test_participant (kolumny *read* oraz *announcement_id*),
- hr_alert.

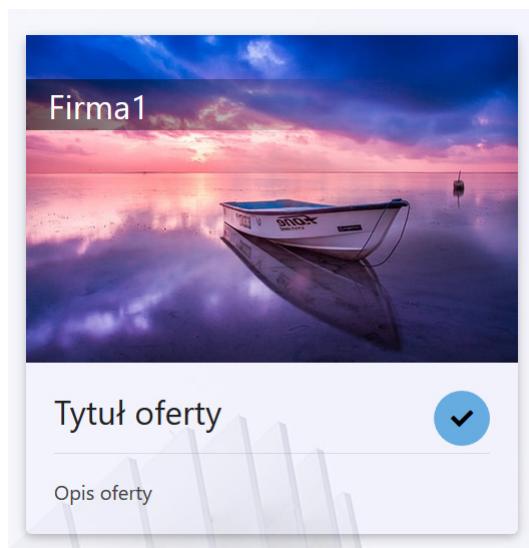
Pierwsza tabela zawiera dane do obsługi powiadomień *użytkowników*, druga zaś dla użytkowników *HR* oraz *CEO* (zakładamy, że administrator również może należeć do tych dwóch grup). Zasada działania powiadomień zostanie opisana na przykładzie scenariusza z użyciem aplikacji internetowej.

Użytkownik, zwany dalej *U1* po utworzeniu konta i zalogowaniu się na portal, zastaje widok wyszarzałej ikonki powiadomień, znajdującej się na pasku nawigacji.



Rysunek 8.38: Nieaktywna ikonka powiadomień. Źródło: Opracowanie własne.

Jest tak dlatego, że aplikacja wysłała zapytanie o nowe powiadomienia do serwera, a ten zwrócił liczbę 0. Aby zmienić stan rzeczy, *U1* musi zaaplikować na ofertę pracy, wybraną z listy ogłoszeń.

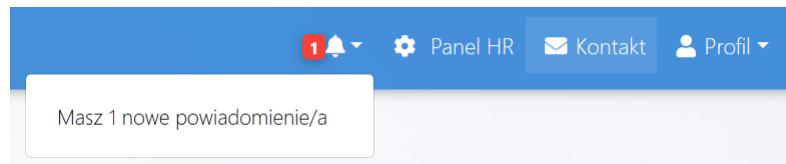


Rysunek 8.39: Przykładowa oferta pracy. Źródło: Opracowanie własne.

U1 zgłosił swoją kandydaturę do firmy, zwanej dalej *F1*, do której należą dwóch użytkowników działu HR – zwanych dalej *H1* oraz *HR2*. *HR1* zalogował się do aplikacji internetowej. Pierwsze, na co zwrócił uwagę, to ikonka powiadomień z jedynką na czerwonym tle. Oznacza to, że ktoś zaaplikował na ofertę pracy z jego firmy.

Po kliknięciu w informację, że ma jedno nowe powiadomienie, zostaje przeniesiony do HR panelu. Na tej stronie widnieje przycisk listy aplikacji na ogłoszenia firmowe. Jeśli nie ma żadnych niezaadresowanych zgłoszeń, to jest on niebieski, jeśli zaś są jakieś zgłoszenia, którym nie zostały przypisane kody testów, to wtedy przycisk ten zmienia kolor na zielony i pojawia się obok tekstu liczba aplikantów, oczekujących na przypisanie im kodu. Po jego kliknięciu przez *HR1*, rozwija się sortowalna lista zgłoszeń, zawierająca niezbędne dane do podjęcia decyzji. Po kliknięciu w przycisk „*przypisz test*”, wyskakuje okienko, gdzie *HR1* wybiera test oraz

ustawia unikatowy klucz dostępu do niego dla $U1$.



Rysunek 8.40: Powiadomienie o nowej aplikacji na stanowisko. Źródło: Opracowanie własne.



Rysunek 8.41: Interaktywny przycisk rozwijający listę aplikacji. Źródło: Opracowanie własne.



Przypisz test do użytkownika:
uzytkownik1

Podaj pierwszy człon kodu testu

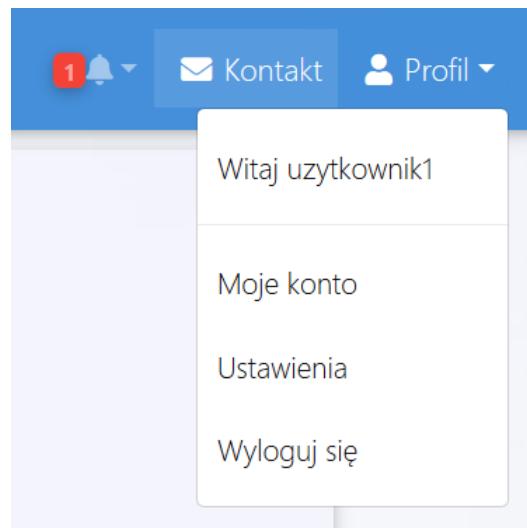
Nazwa testu

PRZYPISZ

Rysunek 8.42: Okno przypisywania testu. Źródło: Opracowanie własne.

Gdyby jednak $HR1$ kliknął przycisk *przypisz test*, ale ostatecznie nie sfinalizował tej akcji, to wtedy i tak powiadomienie dla działu HR się skasuje, jednak liczba

niezaadresowanych zgłoszeń, w przycisku listy aplikacji, nie zmieni się. Dzięki temu, gdy *HR2* zaloguje się do portalu, to nie będzie miał żadnych nowych powiadomień, ale gdy wejdzie do panelu *HR*, to już będzie widział, ile osób czeka na decyzję. Zakładając pomyślny dla *U1* przebieg wydarzeń, po jego ponownym zalogowaniu (albo odświeżeniu widoku strony), jego oczom ukaże się ikonka powiadomień z identyfikatorem jednego, nowo przypisanego testu.



Rysunek 8.43: Powiadomienie dla użytkownika o nowym teście. Źródło: Opracowanie własne.

Po jego kliknięciu *U1* zostanie przekierowany do panelu użytkownika, gdzie znajduje się przycisk *lista aplikacji*, który zachowuje się podobnie, jak ten w panelu pracownika działu *HR*.



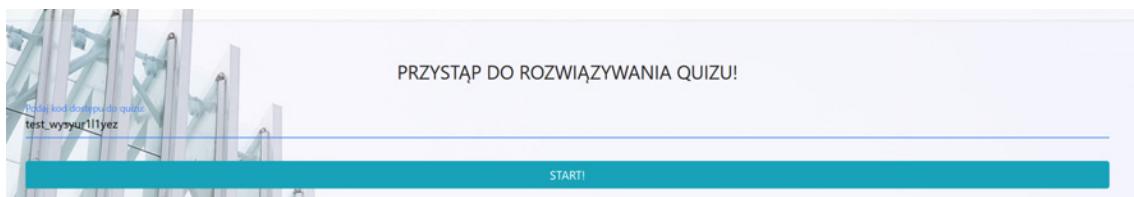
Rysunek 8.44: Interaktywny przycisk, który rozwija listę aplikacji użytkownika. Źródło: Opracowanie własne.

Po jego kliknięciu, rozwija się lista ofert, na które zaaplikował *U1*, wraz z ich stanem.

Firma	Tytuł Ogłoszenia	Kod Do Testu	Status Testu
Firma1	Tytuł1	kod_tyk9y129uzlw	Ukończony
Firma1	Tytuł2	kod_wysyurlllyez	IDŹ DO QUIZU

Rysunek 8.45: Widok listy aplikacji użytkownika z podświetlonym nieukończonym testem. Źródło: Opracowanie własne.

Jako że ma już przypisany jeden kod do testu, to wystarczy, że kliknie przycisk przenoszący go do okna, gdzie może rozpocząć rozwiązywanie wybranego testu.



Rysunek 8.46: Widok okna wyboru quizu. Źródło: Opracowanie własne.

Po tej akcji, jego liczba powiadomień jest odpowiednio aktualizowana, ale liczba niezaadresowanych testów przy przycisku *Listy zgłoszeń* zmieni się dopiero wtedy, kiedy *U1* ukończy dany test.

8.2.10 Budowa stylu aplikacji na przykładzie MDBootstrap

Jednym z założeń projektowych jest trzymanie się, jak to tylko najbardziej możliwe, koncepcji *Material Design*. Aby w prosty sposób zapanować nad stylem całek aplikacji internetowej, została użyta platforma programistyczna (ang. *framework*) *Material Design for Bootstrap*. Za jej pomocą została utworzona zdecydowana większość komponentów strony.

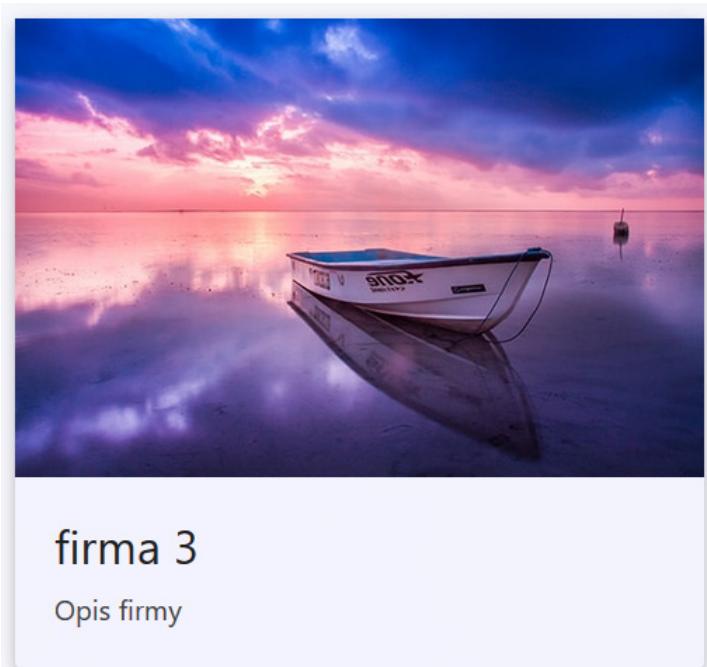
```
<div class="card-deck">
<div *ngFor="let comp of companies">
  <mdb-card>
    <!--Card image-->
    <mdb-card-img src="{{comp.image}}"
      alt="Card image">
    </mdb-card-img>
    <!--Card content-->
    <mdb-card-body>

      <!--Title-->
      <mdb-card-title>
        <h4>{{comp.name}}</h4>
      </mdb-card-title>

      <!-- Text -->
      <app-company-list-single-element
        [text]="comp.about" [maxLength]="" 100">
      </app-company-list-single-element>

    </mdb-card-body>
  </mdb-card>
</div>
</div>
```

Listing 40: Przykładowe elementy utworzone na kanwie *MDBootstrap*. Źródło: Opracowanie własne.



Rysunek 8.47: Karta firmy, utworzona za pomocą *MDBootstrap*. Źródło: Opracowanie własne.

Praktycznie całość karty, prezentującej daną firmę. Została utworzona za pomocą komponentów, udostępnianych przez *MDBootstrap*, co finalnie prowadzi do elementu zgodnego ze standardami *Material Design*.



Rysunek 8.48: Przycisk zmiany hasła, utworzony za pomocą *MDBootstrap*. Źródło: Opracowanie własne.

Poza gotowymi elementami, *MDBootstrap* udostępnia również predefiniowany zestaw kolorów do przycisków, który nie dość, że jest zgodny z *Material Design*, to jeszcze koresponduje do funkcji pełnionej przez przycisk. W tym przypadku została użyta klasa *info*, jako przycisk informacyjny (nie zmienia on hasła, a jedynie otwiera panel zmiany hasła). Jeśli potrzeba dodać przycisk, którego akcją byłoby skasowanie jakiegoś rekordu, to wtedy należałoby użyć klasy *danger*, ponieważ kliknięcie go może nieść za sobą niebezpieczne konsekwencje.



Rysunek 8.49: Przycisk kasowania konta, utworzony za pomocą *MDBBootstrap*. Źródło: Opracowanie własne.

```
<button (click)="deleteUser(user.id)"  
        class="btn btn-danger">  
    {{ 'delete' | translate }}  
</button>
```

Listing 41: Kreowanie przycisku opartego o style MDB. Źródło: Opracowanie własne.

8.2.11 RxJS

Reactive Extensions for JavaScript (RxJS) to biblioteka, która ułatwia programowanie reaktywne w języku *JavaScript*. Udostępnia ona ogrom komponentów, usprawniających pracę z asynchronicznością, przez co pisanie kodu staje się prostsze i bardziej intuicyjne.

Najbardziej podstawowym elementem, jaki udostępnia *RxJS*, jest obiekt typu obserwowlanego (ang. *observable*). Implementuje on wzorzec projektowy *Observer*, który w uproszczeniu zakłada, że istnieje grupa obiektów, obserwujących nasz obiekt i reaguje ona na jego zmiany.

Z tego typu obiektami można pracować za pomocą metod *subscribe* i *unsubscribe*. Subskrypcja pozwala na dokładne zaadresowanie zmian, następujących w nasłuchiwanym obiekcie. Posiada ona trzy przeciążenia. Pierwsze zachodzi w momencie zwrócenia przez obserwowany obiekt poprawnej wartości. Drugie zachodzi w przypadku niepowodzenia przy próbie odbioru tej wartości. Trzecie jest wykonywane po zakończeniu strumienia. W momencie użycia funkcji *subscribe*, nasza subskrypcja jest dopisywana do listy subskrypcji. Aby usunąć utworzoną subskrypcję, należy użyć metody *unsubscribe*. Te elementy biblioteki *RxJS* były wykorzystywane najczęściej. Cały projekt opiera się na połączeniach z serwerem, których wykonanie zajmuje określony wymiar czasu, zależny od złożoności zapytania i ilości wyników w odpowiedzi. Bez możliwości subskrypcji zapytań do *API*, większość funkcjonalności aplikacji działałaby w trybie zbliżonym do losowego. Raz odpowiedź nadążyłaby

za generacją widoku, a raz nie. W większości przypadków nastąpiłaby jednak ta druga opcja. Z tego też powodu *RxJS* jest jedną z kluczowych bibliotek użytych w projekcie.

8.3 Aplikacja mobilna

Aplikacja mobilna została skierowana jedynie dla użytkowników, którzy ubiegają się o posadę w danym przedsiębiorstwie, a więc posiada jedynie część funkcjonalności, dostępnych na stronie internetowej. Jednakże ze względu na łatwość w dostępie do telefonu, jest doskonałym wyborem dla osób, które mają jakikolwiek problem z dostępem do komputera. Oczywiście aplikacji webowej można również używać na telefonie, jednak komfort jej użytkowania jest zdecydowanie mniejszy.

8.3.1 Wymagania urządzenia

Aby aplikacja zadziałała, urządzenie na którym zostanie zainstalowana powinno posiadać wersję systemu:

- API poziomu 16 bądź wersję późniejszą dla systemu Android.
- iOS w wersji 8 bądź późniejszą.

Nie są to jednak wymagania wystarczające. Z powodu ograniczonego dostępu, aplikacja została w pełni przetestowana jedynie na trzech urządzeniach, dlatego zespół nie gwarantuje bezproblemowego funkcjonowania aplikacji na innych telefonach.

Tymi urządzeniami są:

- Xiaomi Mi A2,
- LG V30,
- Xiaomi Redmi Note 5 Pro.

8.3.2 Architektura MVVM

Architektura *MVVM* oznacza *Model*, *View*, *View Model*. Została ona użyta w opisywanym projekcie. *MVVM* składa się z trzech części:

- *View*, czyli warstwa, która bezpośrednio towarzyszy użytkownikowi aplikacji. Ukazuje ona interfejs i wszelakie zmiany w nim, jak: przejścia do nowego okna, wpisywanie wartości w odpowiednie pola, animacje, podmienianie widoków, czy obsługę przycisków.
- *Model*, jest to warstwa, która odpowiada za przechowywanie różnego rodzaju danych, najczęściej takich, które otrzymujemy w odpowiedzi z serwera.
- *ViewModel*, jest warstwą komunikacyjną, odpowiadającą za wywoływanie odpowiednich funkcjonalności, które zostały nakazane z poziomu *View* lub wysyłanie żądanego przez użytkownika odpowiedzi. Aby warstwa *View* otrzymała te odpowiedzi, należy je dodać do zmiennej typu *LiveData*, która powinna być obserwowana przez warstwę widokową.

8.3.3 Biblioteki

Aplikacja mobilna korzysta z kilku różnych bibliotek, których zadaniem było przyspieszenie tworzenia oprogramowania. Szczegółowe dokumentacje bibliotek zostały, przedstawione przez ich autorów na witrynie <https://pub.dev/>. Wykorzystane biblioteki zostały przedstawione poniżej.

- bloc_pattern

Przy użyciu platformy programistycznej *Flutter*, warstwa *ViewModel* reprezentowana jest przez tzw. *Bloc*, który jest jedną z opcji na zaimplementowanie warstwy komunikacyjnej w aplikacji.

```
import 'package:bloc_pattern/bloc_pattern.dart';

class NazwaKlasyKtoraChcemyRozszerzyc
extends BlocBase {
}
```

Listing 42: Rozszerzenie klasy o BlocBase. Źródło: Opracowanie własne.

Nie jest to jednak wbudowana część tego środowiska, dlatego należy ją wcześniej zimportować. Biblioteka ta pozwala nam również rozwiązać problem ze wstrzykiwaniem zależności za pomocą modułów. Moduły te można nazwać danymi segmentami aplikacji. Przykładowo, większość aplikacji możemy podzielić na jeden moduł, odpowiedzialny za logowanie i rejestrację, oraz drugi, który zawiera w sobie kontent widoczny po zalogowaniu. Powoduje to, że nie trzeba wstrzykiwać wszystkich zależności od razu po uruchomieniu aplikacji. Czym jednak jest to wstrzykiwanie zależności? Jest to jednorazowe stworzenie obiektu danej klasy, dzięki czemu nie trzeba za każdym razem, gdy jakąś funkcjonalność jest potrzebna, tworzyć nowego obiektu. Należy także dodać, że minimalizuje to ryzyko wycieków pamięci, które to może być spowodowane utworzeniem wielu obiektów tej samej klasy.

Podczas implementowania modułu, należy rozszerzyć klasę o *ModuleWidget* oraz wpisać w nią listę zależności i bloków, które chcemy wstrzyknąć, a następnie zadeklarować widget, który będzie miał dostęp do tych informacji.

```
class NaszaKlasa extends ModuleWidget {  
    @override  
    List<Bloc> get blocs => [  
        Bloc(( i ) =>  
            JakisBlocZDostepemDoZaleznosciZaleznosc(  
                i .get())),  
    ];  
    @override  
    List<Dependency> get dependencies => [  
        Dependency(( _ ) => Zaleznosc()),  
    ];  
    @override  
    Widget get view => JakisWidget();  
    static Inject get injector =>  
        Inject<AppModule>.of();  
}
```

Listing 43: Rozszerzenie klasy o ModuleWidget. Źródło: Opracowanie własne.

- retrofit

Retrofit umożliwia w łatwy sposób wymianę informacji z *API*. Jest to interfejs, który znaczco ułatwia oraz przyśpiesza pracę programistów, przez swoją prostotę oraz wbudowane funkcje. Aby połączyć się z serwerem, należy utworzyć klasę abstrakcyjną, wraz z odpowiednią adnotacją `@RestApi`, wraz z bazowym *URL* (ang. *Uniform Resource Locator*), czyli ujednoliconym formatem adresowania. W ciele zbudowanej klasy powinien się znaleźć konstruktor fabryki. Następnym niezbędnym krokiem jest zaimplementowanie odpowiednich metod, wraz z adnotacją, końcową częścią *URL*, a także – o ile to konieczne – parametrem, zapytaniem bądź ciałem. Ostatnim krokiem jest utworzenie logiki działania połączenia. Mimo pozornie trudnego w implementacji warunku, jest to najłatwiejszy krok, bowiem należy jedynie dodać informację o nazwie pliku, z rozszerzeniem `.g.dart`, która będzie częścią tej klasy. Po poprawnej implementacji oraz użyciu komendy `flutter pub run build_runner build` wygeneruje się plik z wcześniej wspomnianą logiką.

```
part 'authorization_source.g.dart';

@RestApi(baseUrl: "http://192.168.43.228:8080")
abstract class AuthorizationSource{

    factory AuthorizationSource(
        Dio dio,
        {String baseUrl}) = _AuthorizationSource;

    @POST("/login")
    Future<\HttpResponse> attemptToLogin(
        @Body() LoginCommandDto loginCommandDto
    );

}
```

Listing 44: Użycie biblioteki retrofit – na przykładzie modułu logowania. Źródło: Opracowanie własne.

- json_serializable

Większość zapytań bądź odpowiedzi z serwera jest wysyłanych w formacie *JSON*. Jednak, aby w łatwy sposób zarządzać danymi z *API* w języku *Dart*, należy przekonwertować argumenty obiektu na *JSON* – w przypadku wysyłania zapytania bądź w przeciwny sposób, w przypadku odbierania informacji. W tym celu biblioteka dla klasy, która będzie używana w celu tworzenia lub odczytywania formatu *JSON*, generuje logikę działania tejże transformacji. Aby to osiągnąć, klasa powinna posiadać odpowiednią anotację zadeklarowaną nazwą pliku, który ma być jej częścią, a następnie dodaniem klucza *JSON* dla każdego z argumentów i dwoma metodami, które umożliwią konwersję. Przykład implementacji *json_serializable*:

```
part 'answer_result_dto.g.dart';

@JsonSerializable()
class AnswerResultDto {
    @JsonKey(name: "id")
    int id;

    @JsonKey(name: "text")
    String text;

    AnswerResultDto({this.id, this.text});

    factory AnswerResultDto.fromJson(
        Map<String, dynamic> json
    ) => _$AnswerResultDtoFromJson(json);

    Map<String, dynamic> toJson() =>
        _$AnswerResultDtoToJson(this);
}
```

Listing 45: Przykład implementacji serializacji danych typu *JSON*. Źródło: Opracowanie własne.

- rx dart

W projekcie biblioteka *rx dart* została użyta w celu zarządzania strumieniem informacji poprzez odpowiednie kontrolery. Pierwszym z nich jest *BehaviorSubject*, który przetrzymuje ostatnią dodaną do niego wartość i w momencie podłączenia się nowego obserwującego, wysyła ją jako wartość początkową. Kolejnym jest *PublishSubject*, który nie przechowuje ostatnich informacji, a jedynie wysyła otrzymane do aktualnie obserwujących go słuchaczy. Ostatnim z nich jest *ReplaySubject*. Kontroler różni się od pozostałych tym, że magazynuje on wszystkie otrzymane dane.

- shared_preferences

W celu przechowywania informacji, które są na tyle istotne, że nie mogą zostać usunięte po zamknięciu aplikacji, stosuje się narzędzia zwane *Shared Preferences*, które umożliwiają przechowywanie danych w plikach na urządzeniu. Taką informacją jest przykładowo token, gdyż powinien on się stać niedostępny jedynie w momencie wylogowania bądź jego wygaśnięcia. Jednakże jest to narzędzie, które należy zaimplementować natywnie, ponieważ inaczej zapisuje się pliki na systemie *Android*, a inaczej na *iOS*. Dlatego też, w projekcie została użyta biblioteka *shared_preferences*, która pozwala na szybki zapis danych w obu systemach. Aby skorzystać z biblioteki, należy – najlepiej za pomocą funkcji statycznej – wywołać instancję klasy *SharedPreferences*, a następnie zapisać lub odczytać żądaną wartość, przy użyciu odpowiedniej metody oraz wpisaniu klucza, do którego jest ona przypisana.

```
static Future<String> getToken() async {
    SharedPreferences sharedpreferences =
        await SharedPreferences.getInstance();
    return sharedpreferences.getString('token');
}
```

Listing 46: Implementacja sposobu odczytywania tokenu. Źródło: Opracowanie własne.

- fluttertoast

Aby użytkownik po wykonanej czynności dostawał informacje, które nie są widoczne w interfejsie, została użyta biblioteka *fluttertoast*, która w dolnej części aplikacji, przez parę sekund, ukazuje okienko z odpowiednim komunikatem.

- `jwt_decoder`

Biblioteka użyta w celu dekodowania informacji zawartych w tokenie, dzięki czemu aplikacja jest w stanie rozpoznać, do jakich zasobów użytkownik ma dostęp oraz czy na przykład token stracił swoją ważność.

8.3.4 Cykle życia aktywności w Androidzie

Aplikacje na urządzeniu z systemem Android mogą przybierać jeden z czterech stanów:

- stan aktywny, kiedy to aplikacja jest w danym momencie otwarta i widoczna dla użytkownika,
- stan zapauzowany, gdy to aplikacja zostanie częściowo zasłonięta przez inną aplikację,
- stan zatrzymany, charakteryzujący się tym, że aplikacja nie została zamknięta, a jedynie działa w tle i użytkownik nie widzi jej bezpośrednio,
- stan zniszczony, który to może zostać wywołany na polecenie użytkownika, który chce zamknąć aplikację bądź sam system Android ją zatrzyma w celu uzyskania dodatkowej pamięci.

Z tego powodu, implementacja różnych funkcji musi być dokonana w odpowiednim momencie. Takimi momentami są cykle życia, czyli metody, które są nadpisywane, gdy aplikacja zmienia swój stan³⁰. Są nimi:

- **onCreate()**, czyli moment w którym aktywność zostaje utworzona,
- **onStart()**, funkcja wywołująca się zaraz po **onCreate()** lub gdy aplikacja została zatrzymana podczas metody **onStop()**, a następnie uruchomiona ponownie,

³⁰M. Załączny, „Tutorial Android – Cykl życia aktywności”,
<https://progmar.net.pl/tutorials/android/index.php?topic=007-activities-life-cycle>
(dostęp 15.12.2020)

- **onResume()**, metoda wywołująca się po **onStart()**, a także gdy aplikacja wróci na pierwszy plan po pobycie w stanie zapauzowania,
- **onPause()**, jest to moment, w którym aplikacja przechodzi w stan zapauzowania,
- **onStop()**, wywoływana, gdy aplikacja przestaje być widoczna dla użytkownika,
- **onRestart()**, funkcja inicjowana, gdy aplikacja, po byciu w stanie zatrzymania, powraca do stanu aktywnej, zaraz po niej wywoływana zostaje metoda **onStart()**.
- **onDestroy()**, czyli moment, w którym aplikacja zostaje zniszczona.

8.3.5 Rodzaje widgetów

Podczas projektowania wyglądu aplikacji na platformie programistycznej *Flutter*, należy zadecydować, czy widgetem, który będzie reprezentował daną część wyglądu użytkowego, zostanie widget ze stałym stanem (ang. *stateless*) czy widget ze zmiennym stanem (ang. *stateful*). Różnica między nimi jest prosta, bowiem w środku widgetu ze zmiennym stanem można na bieżąco przeładowywać i wyświetlać nowe dane, niezbędne do ukazania użytkownikowi. Widget ze stałym stanem nie posiada tego mechanizmu i raz zadeklarowane wartości, pozostają w nim do końca jego życia. Przeładowanie każdego elementu typu zmienno-stanowego zużywa bardzo dużo zasobów. Dlatego też należy używać ich roztropnie³¹.

8.3.6 HTTP interceptors

Zasada działania interceptorów została już przedstawiona w rozdziale 8.2.4. W aplikacji użyto ich w celu dodawania tokenu autoryzacyjnego do każdego pytania, a także w celu wysyłania zadania, by użytkownik został wylogowany, gdy jego token utraci ważność.

³¹S. Biswas, „Flutter: Stateful vs Stateless Widget”,
<https://medium.com/flutter-community/flutter-stateful-vs-stateless-db325309deae>
(dostęp 15.12.2020)

```
Future<RequestOptions>
    _onRequest(RequestOptions options) async {
        _apiClient.lock();
        logger.i('Request: ${options.path}');
        String token = await TokenSharedPref.getToken();
        if (token == null) {
            _apiClient.clear();
            _apiClient.unlock();
            _logoutSubject.add(null);
            return options;
        }
        _apiClient.unlock();
        return options..headers['Authorization'] = '$token';
    }

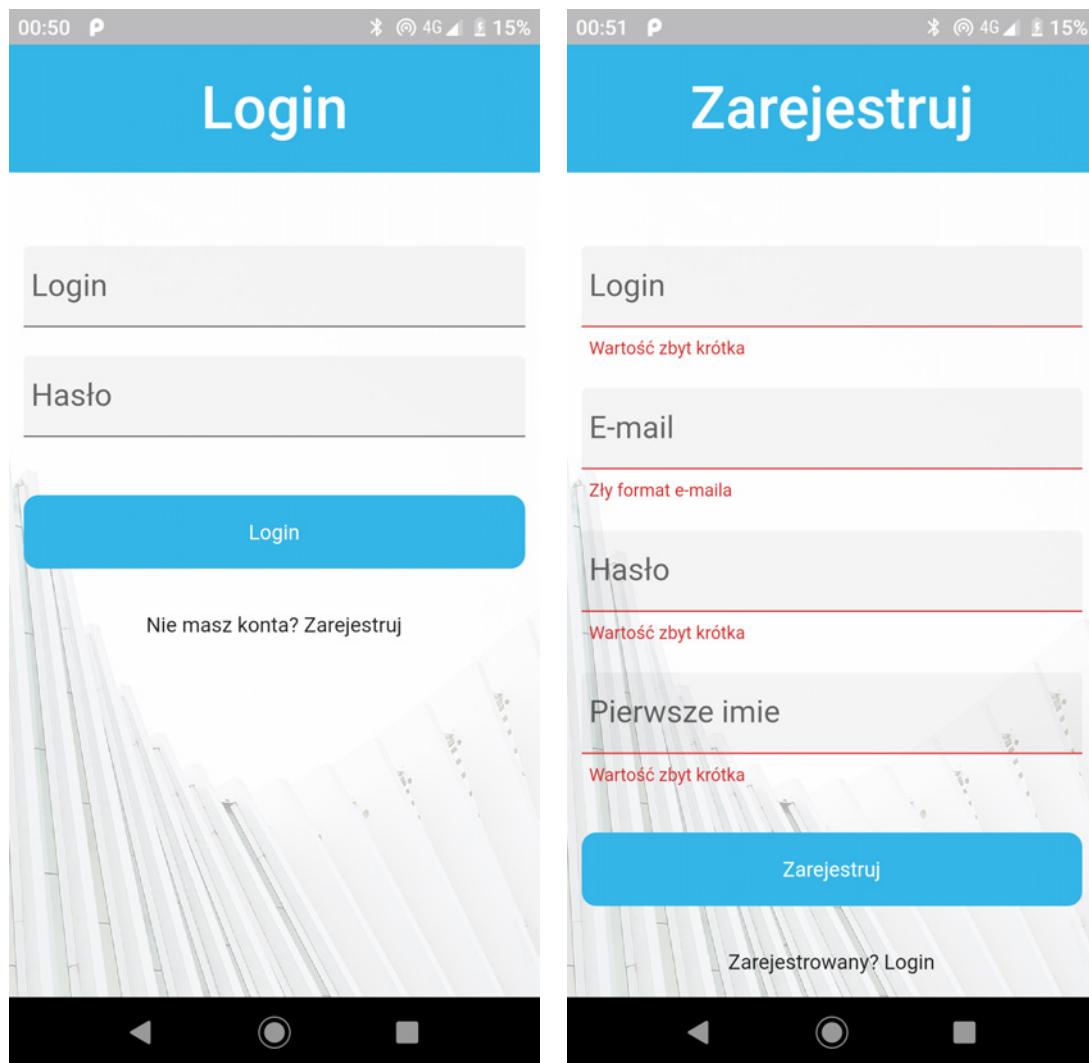
Future<Response> _onResponse(Response response) async {
    logger.i('Response: (${response.statusCode}) —
        ${response.request.path}');
    return response;
}

Future< DioError> _onError(DioError error) async {
    logger.w('Error: (${error.request.path})', error.error);
    if (await TokenSharedPref.tokenExpired()) {
        logoutCurrentUser();
    }
    return error;
}
```

Listing 47: Implementacja interceptorów. Źródło: Opracowanie własne.

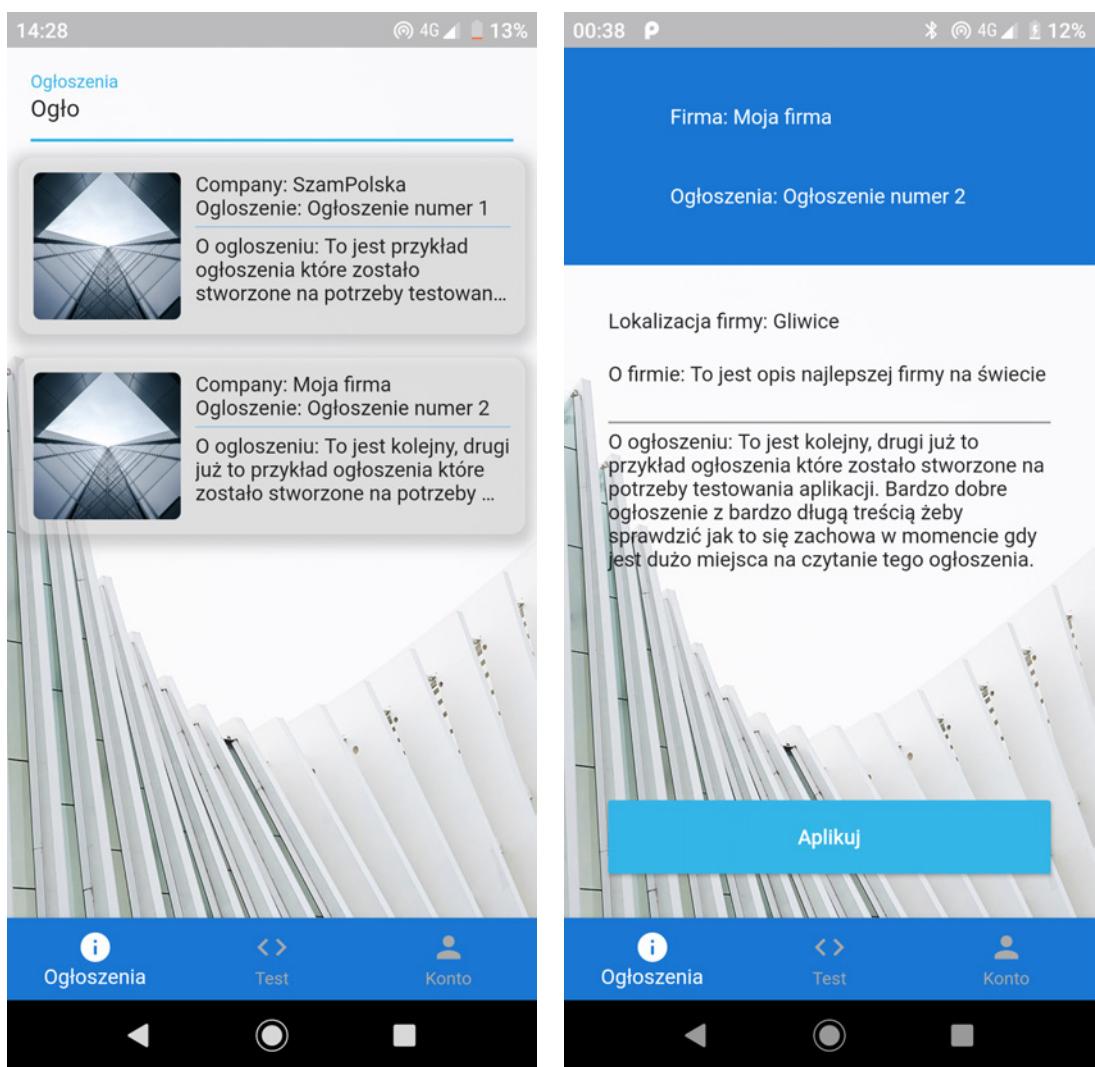
8.3.7 Interfejs użytkownika

Po pierwszym uruchomieniu aplikacji oczom użytkownika ukaże się ekran logowania i rejestracji. Został on zaprojektowany w taki sposób, aby użytkownik był informowany na bieżąco o błędach występujących w formularzu. Przykładowo, jeżeli przy rejestracji wpisze e-mail, który nie jest prawidłowy, aplikacja wyświetli odpowiedni komunikat, informujący go o tym.



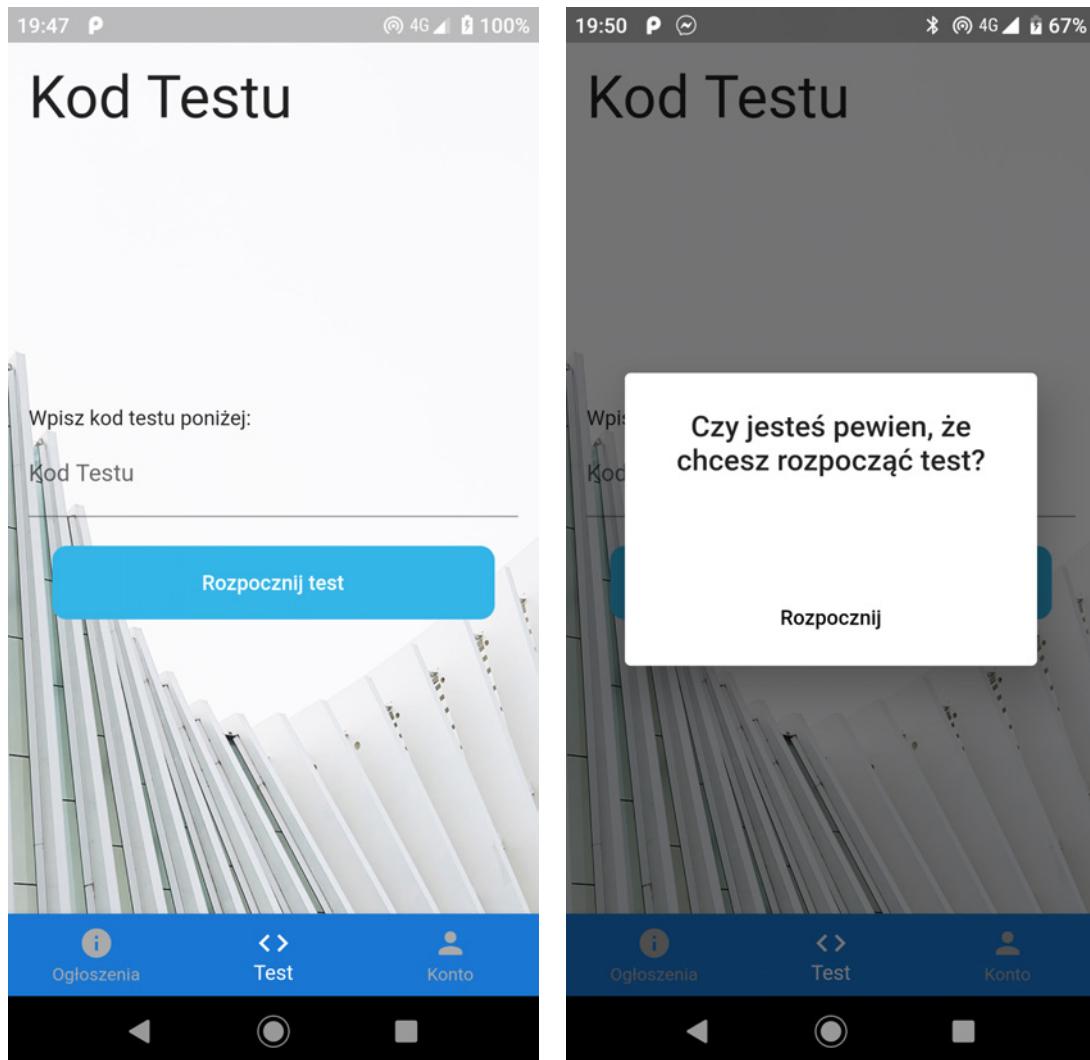
Rysunek 8.50: Ekran logowania i rejestracji. Źródło: Opracowanie własne.

Po zalogowaniu, aplikacja pozwala na wyszukiwanie dostępnych ogłoszeń pracy. Wybranie jednego z ogłoszeń przeniesie do widoku jego szczegółów, gdzie można wysłać swoją kandydaturę na ogłaszone stanowiska. Prośba ta, wraz z danymi użytkownika, zostanie skierowana do pracownika działu zarządzania zasobami ludzkimi i zweryfikowana.



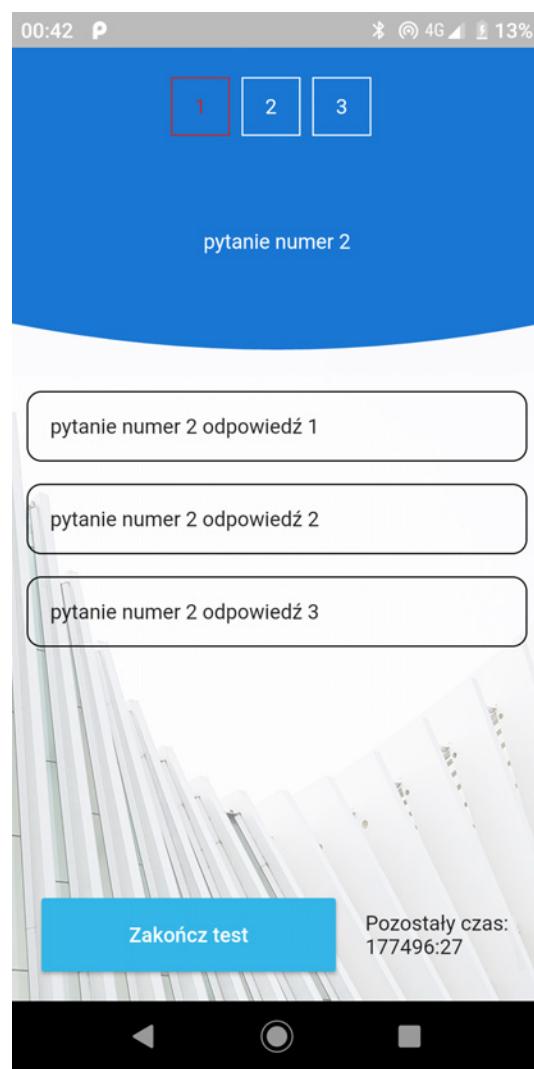
Rysunek 8.51: Widok ogłoszeń wraz ze szczegółami danego ogłoszenia. Źródło: Opracowanie własne.

Zakładka rozwiązywania testów ukazuje okno, w którym można wpisać kod do testu, który użytkownik planuje wykonać, a który wcześniej został mu przypisany.



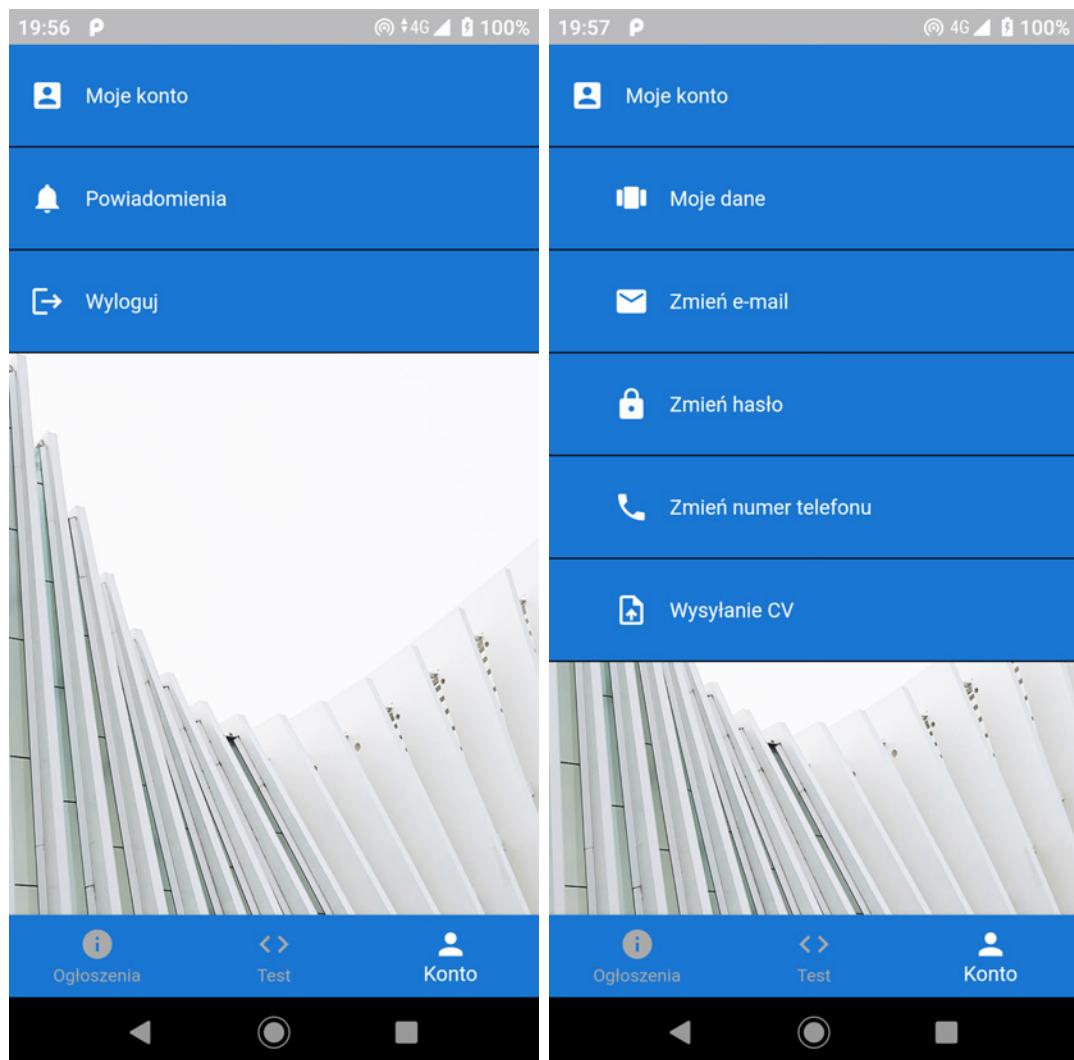
Rysunek 8.52: Widok ogłoszeń wraz ze szczegółami danego ogłoszenia. Źródło: Opracowanie własne.

Gdy użytkownik zdecyduje się, by podjąć wyzwanie, jego oczom ukaże się widok testu. Po wybraniu odpowiedzi, aplikacja wyśle ją na serwer, a następnie pobierze i ukaże następne pytanie. Każdy numer pytania, na które użytkownik udzielił odpowiedzi, zostanie w górnej części zaznaczony na czerwono. Osoba, rozwiązująca test, widzi w dolnej części aplikacji pozostały czas na rozwiązywanie go oraz ma możliwość jego zakończenia przed czasem. Jeżeli wystąpi błąd podczas rozwiązywania testów w aplikacji, widok zostanie zamknięty, a użytkownik o tym fakcie poinformowany. Zawsze istnieje możliwość wrócenia do rozwiązywania, o ile czas na rozwiązywanie się nie zakończył.



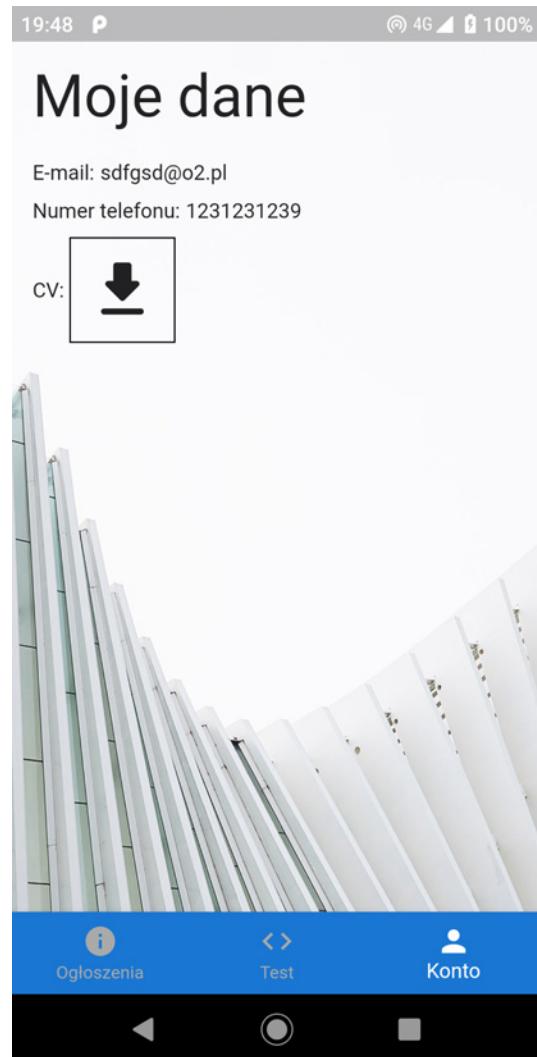
Rysunek 8.53: Widok rozwiązywania testu. Źródło: Opracowanie własne.

Ostatnia zakładka dotyczy obsługi konta użytkownika. Są tutaj dostępne opcje, takie jak: wylogowanie, zmiana hasła i maila, dodanie życiorysu, czy dostęp do powiadomień o dostępie do nowego testu.



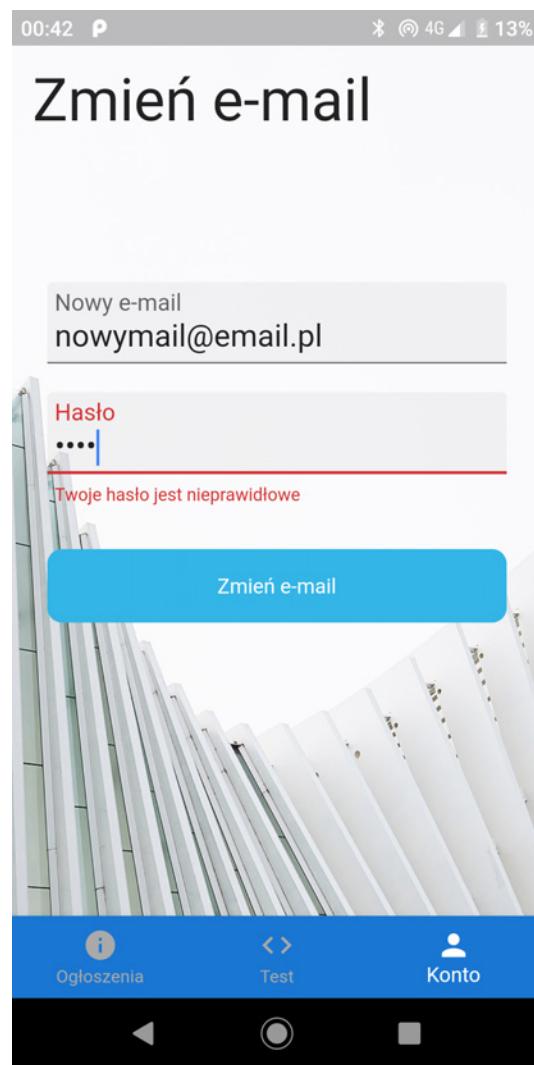
Rysunek 8.54: Wszystkie dostępne opcje w menu użytkownika. Źródło: Opracowanie własne.

W zakładce *Moje konto* użytkownik jest w stanie sprawdzić swoje podstawowe dane, jakimi są: e-mail, numer telefonu, a także – po kliknięciu w odpowiednią ikonę – pobrać dodany wcześniej życiorys w formacie *.pdf*.



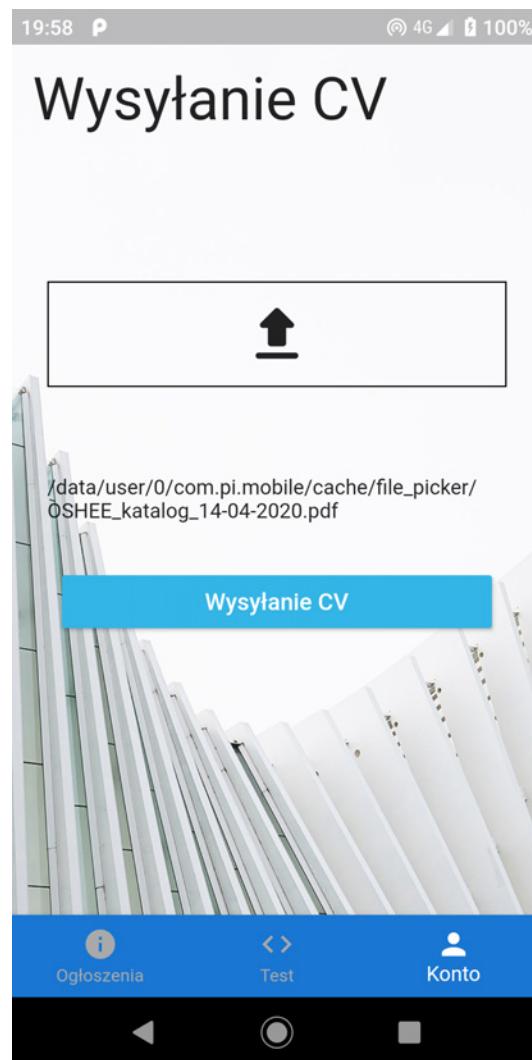
Rysunek 8.55: Zakładka zawierająca dane użytkownika. Źródło: Opracowanie własne.

Rysunek 8.56 przedstawia widok dla zmiany jednej z informacji o użytkowniku. Na zrzucie ekranu można zauważyc komunikat z serwera, który informuje o podaniu błędnego hasła, przez co e-mail użytkownika nie został zmieniony. Dla przykładu, widok dla zmiany hasła jest zbudowany analogicznie do widoku przedstawionego na rysunku 8.56.



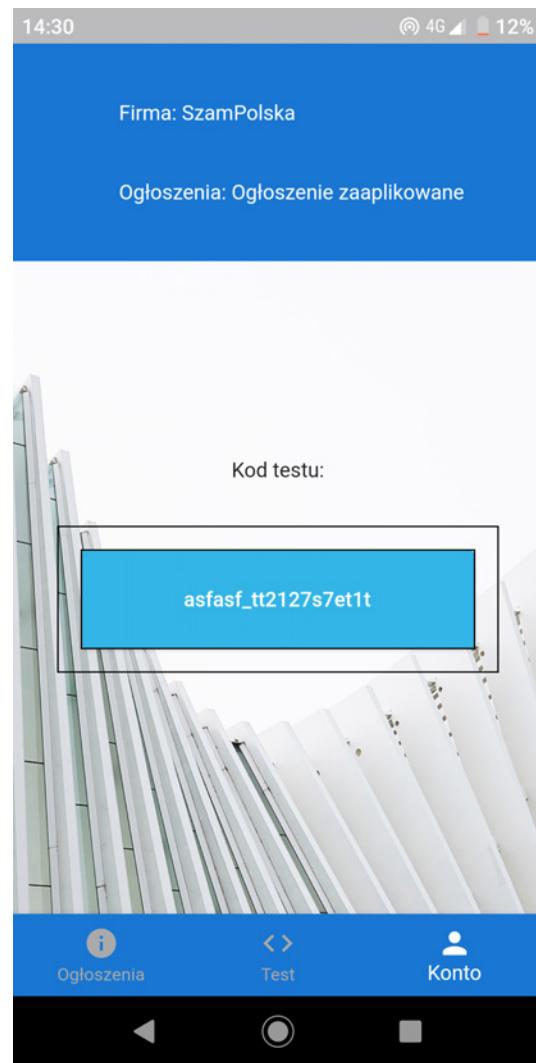
Rysunek 8.56: Zakładka do obsługi zmiany e-maila. Źródło: Opracowanie własne.

Ostatni kontener w tym podmenu zawiera opcję do wysłania nowego lub podmiany obecnego życiorysu. Wystarczy wybrać z urządzenia odpowiedni plik, a następnie wybrać opcję: *Wysyłanie CV*.



Rysunek 8.57: Widok zakładki, w której można wysłać życiorys. Źródło: Opracowanie własne.

W menu powiadomień została zaprezentowana lista wszystkich powiadomień użytkownika. Po wybraniu któregoś z nich, użytkownik widzi szczegóły ogłoszenia, do którego dostał dostęp, jak również kod testu. Nie musi on jednak przepisywać go we właściwe pole. Wystarczy wcisnąć przycisk, na którym znajduje się ów kod, a aplikacja sama przeniesie go do rozwiązywania quizu.



Rysunek 8.58: Widok szczegółów powiadomienia wraz z kodem testu. Źródło: Opracowanie własne.

W powyższym rozdziale skupiono uwagę na poszczególnych komponentach, zastosowanych w różnych gałęziach projektu. Każda z części, czy to serwerowa, czy kliencka, została napisana z myślą o prostocie użytkowania, jak również przygotowano ją na ewentualne zmiany, bądź samą rozbudowę. Wszystko to zostało wsparcie odpowiednimi zabezpieczeniami, takimi jak *Spring Security*, czy *JWT*. W celu usprawnienia całego działania projektu i szybszej implementacji, zespół korzystał z odpowiednich darmowych bibliotek, jak na przykład *retrofit*, która wspomaga programistę przy wysyłaniu zapytań do serwera, a także sprawia, że sam kod jest dużo bardziej czytelny. Po stronie klienta poszczególne elementy widoku są wspomagane przez rozszerzenia zgodne z *Material Design*, co powoduje, że strona jest wizualnie przyjazna użytkownikowi. W każdym fragmencie omawiane zagadnienia zostały poparte przykładami wizualnymi, którymi były na przykład zrzuty ekranu aplikacji webowej lub mobilnej, jak również wycinki z fragmentami kodu źródłowego wraz z wyjaśnieniem, na czym polega jego działanie.

136:1136998229

9 Testy

Testowanie aplikacji w fazie jej produkcji i rozwoju, należy do bardzo ważnego punktu procesu wytwarzania programowania. Dobrze prowadzony projekt wymaga testów w każdej z faz produkcji. W aplikacji projektowej, poza standardowymi manualnymi testami np.: części przeglądarkowej – w których sprawdzane było, czy wywołując odpowiednie zdarzenia, otrzymujemy oczekiwany wynik; czy też części bazodanowo-serwerowej, gdzie podczas procesu tworzenia punktów końcowych, osoby odpowiedzialne za kod programu sprawdzały, czy działa on poprawnie. Jednak poprzestanie na tego typu testach nie należy do dobrych praktyk, ponieważ osoba programująca daną funkcjonalność może nie zauważyc błędów, które popełniła w trakcie tworzenia logiki działania aplikacji. W tym celu, w każdej z części produktu, zespół postanowił stworzyć testy jednostkowe, automatycznie sprawdzające poprawność działania ważniejszych funkcjonalności.

9.1 Testy części serwerowej

Część serwerowa powinna sprawdzać na przykład, czy wybrane punkty końcowe zwracają dane w odpowiednim formacie. Poniższy test sprawdza, czy lista wszystkich użytkowników jest tablicą oraz czy wybrane pola w niej są odpowiedniego typu.

```
@EnableWebMvc  
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class UsersListTest {  
  
    @Autowired  
    private WebApplicationContext context;  
    private MockMvc mvc;  
  
    @Before  
    public void setup() {  
        mvc = MockMvcBuilders
```

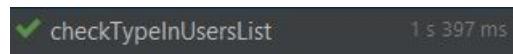
```

        . webAppContextSetup( context )
        . apply( SecurityMockMvcConfigurers
        . springSecurity() ). build();
    }

    @WithMockUser( value = "test123" , password = "test" )
    @Test
    public void checkTypeInUsersList() throws Exception {
        this.mvc.perform( get( "/user/getall" ) )
            . andExpect( status().isOk() )
            . andExpect( MockMvcResultMatchers
            . jsonPath( "$" ). isArray() )
            . andExpect( MockMvcResultMatchers.jsonPath(
                "$[0].firstName" , Is.isA( String.class ) ) )
            . andExpect( MockMvcResultMatchers.jsonPath(
                "$[0].isActive" , Is.isA( Boolean.class ) ) );
    }
}

```

Listing 48: Testy jednostkowe aplikacji przeglądarkowej. Źródło: Opracowanie własne.



Rysunek 9.1: Wynik testu, sprawdzającego poprawność danych w liście użytkowników. Źródło: Opracowanie własne.

9.2 Testy części przeglądarkowej

W części przeglądarkowej zespół skupił się na testach jednostkowych komponentów, tworzonych przy pomocy platformy programistycznej *Jasmine*. Poniższy test dotyczy paska nawigacji. Test uwzględnia sprawdzenie takich składowych, jak: inicjacja poprawnymi wartościami parametrów; test funkcji, składających się na budowę cyklu życia komponentu, jak również test poprawności danych, zwracanych przez utworzone metody.

```
describe( 'NavbarComponent' , () => {
  let component: NavbarComponent;
  let fixture: ComponentFixture<NavbarComponent>;

  beforeEach(() => {
    const companyServiceStub = () => ({
      getCurrentCompany: () => ({
        subscribe: f => f({})
      })
    });
    const alertsServiceStub = () => ({
      getUserAlerts: () => ({ subscribe: f => f({}) }),
      getHrAlerts: () => ({ subscribe: f => f({}) })
    });
    const routerStub = () => ({
      navigate: array => ({})
    });
    const tokenStorageServiceStub = () => ({
      isAuthenticated: () => ({}) ,
      getRole: () => ({}) ,
      getUser: () => ({}) ,
      deleteUserFromLocalStorage: () => ({})
    });
    TestBed.configureTestingModule({
      imports: [RouterTestingModule ,
        TranslateModule.forRoot()
      ] ,
      schemas: [NO_ERRORS_SCHEMA] ,
      declarations: [NavbarComponent] ,
      providers: [
        { provide: CompanyService ,
          useFactory: companyServiceStub } ,
        { provide: AlertsService ,
```

```
useFactory: alertsServiceStub },
{ provide: Router,
useFactory: routerStub },
{ provide: TokenStorageService,
useFactory: tokenStorageServiceStub }

]);
});

fixture = TestBed.createComponent(NavbarComponent);
component = fixture.componentInstance;
});

it('can_load_instance', () => {
expect(component).toBeTruthy();
});

it('userAlerts has default value', () => {
expect(component.userAlerts).toEqual(0);
});

it('hrAlerts has default value', () => {
expect(component.hrAlerts).toEqual(0);
});

it('isLogged has default value', () => {
expect(component.isLogged).toEqual(false);
});

it('userRole has default value', () => {
expect(component.userRole).toEqual('USER');
});

describe('ngOnInit', () => {
it('makes_expected_calls', () => {
```

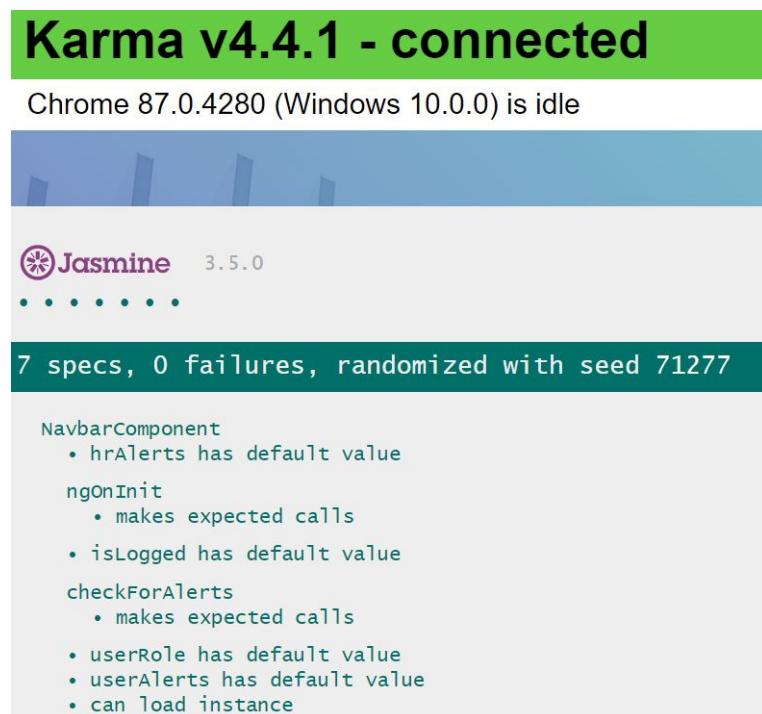
```
const tokenStorageServiceStub: TokenStorageService =
  fixture.debugElement.injector
    .get(TokenStorageService);
spyOn(component, 'checkForAlerts')
  .and.callThrough();
spyOn(tokenStorageServiceStub, 'isAuthenticated')
  .and.callThrough();
spyOn(tokenStorageServiceStub, 'getRole')
  .and.callThrough();
spyOn(tokenStorageServiceStub, 'getUser')
  .and.callThrough();
component.ngOnInit();
expect(component.checkForAlerts).toHaveBeenCalled();
expect(tokenStorageServiceStub.isAuthenticated)
  .toHaveBeenCalled();
expect(tokenStorageServiceStub.getRole)
  .toHaveBeenCalled();
expect(tokenStorageServiceStub.getUser)
  .toHaveBeenCalled();
});

});

describe('checkForAlerts', () => {
  it('makes_expected_calls', () => {
    const companyServiceStub: CompanyService =
      fixture.debugElement.injector.get(
        CompanyService
      );
    const alertsServiceStub: AlertsService =
      fixture.debugElement.injector.get(
        AlertsService
      );
    spyOn(companyServiceStub, 'getCurrentCompany')
```

```
. and . callThrough ();
spyOn( alertsServiceStub , 'getUserAlerts' )
.and . callThrough ();
component . checkForAlerts ();
expect ( companyServiceStub . getCurrentCompany )
.toHaveBeenCalled ();
expect ( alertsServiceStub . getUserAlerts )
.toHaveBeenCalled ();
});
});
});
```

Listing 49: Testy jednostkowe aplikacji przeglądarkowej. Źródło: Opracowanie własne.



Rysunek 9.2: Wynik przebiegu testów paska nawigacji aplikacji internetowej. Źródło: Opracowanie własne.

9.3 Testy aplikacji mobilnej

W aplikacji mobilnej został przedstawiony prosty test jednostkowy, który ma na celu ukazanie sposobu tworzenia takich funkcjonalności, jak: sprawdza on logikę przemieszczania się pomiędzy pytaniami w czasie rozwiązywania testu. Przykładowo, jeżeli użytkownik z trzech pytań rozwiązał ostatnie, aktualnie znajduje się na drugim, a pierwsze zostawił na koniec, aplikacja po podaniu odpowiedzi na dru- gie pytanie powinna przenieść go do najbliższego nierożwiązanego pytania. W tym przypadku byłoby to pytanie pierwsze.

Test składa się z trzech etapów:

- faza wprowadzenia danych,
- faza wywołania odpowiedniej metody,
- faza sprawdzenia zgodności wyników.

```
import 'package:flutter_test/flutter_test.dart';
import 'package:mobile/models/
    current_question_controller.dart';

void main() {

    test('Given next question call When user is
        at question 2, solved question 3 and didnt
        solve question 1 And when is on question 2, solved
        question 1 and 3', () async {

        // Version one
        CurrentQuestionController currentQuestionController
            = CurrentQuestionController(3, true, 2);
        currentQuestionController.listOfAnsweredQuestions =
            [3];

        // ACT
        currentQuestionController.onAnswerSuccess();
    });
}
```

```
// ASSERT
expect( currentQuestionController .currentQuestion ,
1);

// Version two
currentQuestionController
= CurrentQuestionController(3, true, 2);
currentQuestionController
.listOfAnsweredQuestions = [1, 3];

// ACT
currentQuestionController
.onAnswerSuccess();

// ASSERT
expect( currentQuestionController
.currentQuestion , 1);
});

}
```

Listing 50: Zaimplementowany test jednostkowy w aplikacji mobilnej. Źródło: Opracowanie własne.

Zespół w głównej mierze skupił się na testach jednostkowych każdej z części projektu (tj. aplikacji mobilnej, przeglądarkowej, a także części serwerowej, wraz z bazą danych). Sama liczba testów, która została zaimplementowana w projekcie, nie jest duża. Zostało to podyktowane tym, że zespół podjął się napisania bardzo rozbudowanej aplikacji w relatywnie krótkim czasie, biorąc pod uwagę liczbę osób i ich doświadczenie w wybranych technologiach. Niemniej jednak, wszystkie zaimplementowane testy przeszły pozytywną weryfikację podczas ich uruchamiania. Oczywiście poza testami jednostkowymi, przeprowadzonych zostało mnóstwo testów

manualnych aplikacji internetowej, mobilnej czy części serwerowej. Bez nich członkowie zespołu nigdy nie byliby pewni, czy zaimplementowane przez nich funkcjonalności działają poprawnie. Tego typu testy przeprowadzane są na każdym etapie pisania kodu i dają w pewnym stopniu zapewnienie, że dany moduł będzie prawidłowo funkcjonował.

146:1678917082

10 Podsumowanie

W wyniku przeprowadzonych prac, udało się spełnić główne założenia, postawione podczas tworzenia mapy projektu. System został napisany w taki sposób, aby w przyszłości mógł być łatwo rozszerzany. Spełnione zostały też główne wymogi projektu, tj.: zarządzanie systemem rekrutacji i możliwość rozwiązywania testów rekrutacyjnych.

10.1 Możliwości wykorzystania systemu

Projekt w obecnym stanie może być wykorzystany jako platforma internetowa, która zrzeszałaby pracodawców i przyszłych pracowników. Rozwiązanie to przyniosłoby ogromne korzyści dla obu stron. Z punktu widzenia petenta:

- ujednolicenie systemu, co redukuje stres przed kolejnym, nieznanym procesem rekrutacyjnym,
- wszystkie zgłoszenia, wraz z ich stanem, zebrane w jednym miejscu,
- minimalizacja ryzyka przeoczenia odpowiedzi od rekrutera – powiadomienia na stronie, w aplikacji i na adres e-mail.

Korzyści płynące dla prezesa firmy:

- redukcja kosztów związanych z rekrutacją,
- możliwość podglądu przebiegu rekrutacji w dowolnym momencie,
- możliwość tworzenia bazy skategoryzowanych testów, których można używać wielokrotnie,
- dzięki możliwości obsługi rekrutacji przez cały dział HR, łatwość w zastępowaniu pracowników przez innych, na przykład w czasie urlopu.

Co prawda, funkcjonalność testów rekrutacyjnych może nie być postrzegana przez wszystkie firmy jako atut, ponieważ nie na każde stanowisko są takowe wymagane, jednak myśląc nieszablonowo, można je wykorzystać chociażby jako ankiety, które ułatwiają proces pozyskiwania pracowników.

10.2 Wnioski

Tworzenie od podstaw systemu, który ma być łatwo rozszerzalny, jest czasochłonne. Jednak pomimo tego, że podczas początkowych faz tworzenia oprogramowania, jego implementacja zajmuje więcej czasu, to przez to, że jest to rozwiązanie przemyślane – zwraca się z nawiązką czasową w następnych fazach wytwarzania produktu. Zastosowanie się do wszystkich zasad programowania obiektowego, trzymanie się od początku wzorców projektowych, czy pisanie czystego i przejrzystego kodu, jak również rozdzielanie wszystkiego na klasy i obiekty, owocuje w przyszłości. Dzięki zastosowaniu powyższych zasad deweloper, który rozwija produkt, może korzystać z już zdefiniowanych wcześniej klas, czy też rozszerzać je o własne pola. Dzięki obiektowości, nie ma potrzeby tworzenia kopii kodu w wielu miejscach. Raz zdefiniowana metoda, jeżeli tylko jest taka możliwość, powinna być stosowana w różnych miejscach, z możliwością jej przeciążania. Jasna definicja struktur plików zwiększa czytelność programu, co na przykład pomaga we wdrażaniu do projektu nowych programistów.

Zdaniem zespołu, projekt – biorąc pod uwagę powyższe zalety – miałby szansę zainstnieć na rynku komercyjnym. Po wdrożeniu dodatkowych funkcjonalności i rozszerzeniu obecnych, system mógłby odgrywać znaczącą rolę w procesach rekrutacyjnych, prowadzonych przed średnie i duże przedsiębiorstwa. Baza kandydatów, możliwość ich oceny na podstawie rozwiązań testów oraz zbiór życiorysów w jednym miejscu, w opinii zespołu jest bardzo przydatnym narzędziem, które pozwala ustandaryzować proces rekrutacji. Łącząc powyższe z faktem, że system został stworzony w taki sposób, by implementacja nowych rozwiązań i wsparcie obecnych, było możliwe jak najłatwiejsze do wprowadzenia – otrzymuje się kod z niskim kosztem niezbędnym na rozwój produktu.

10.3 Dalsze możliwości rozwoju projektu

Jednym z założeń tego projektu było to, aby był on skalowalny. Założenie to udało się osiągnąć i nic nie stoi na przeszkodzie dalszemu doskonaleniu pracy. Jedną z wielu możliwości rozwoju, może być stworzenie live chatu na linii pracownik działu HR – kandydat. Na pewno funkcjonalność ta w pewnej mierze ułatwiłaby cały proces rekrutacji. Wszelkie niedopowiedziane kwestie, czy to z jednej, czy z drugiej

strony, byłyby prostsze do wyjaśnienia, co przekładałoby się również na zaoszczędzony czas na rozmowach telefonicznych. Łatwiej i szybciej jest bowiem odpisać na kilka wiadomości, niż przeprowadzić jedną rozmowę głosową. Taki chat można zaimplementować na przykład z wykorzystaniem *Google Firebase* – platformą, która jest przewidziana właśnie do tworzenia aplikacji mobilnych i internetowych. Kolejnym polem rozwoju może być raportowanie statystyk rekrutacji – automatyzowane bądź też manualnie wywoływanie, np. generowanie wykresów, przedstawiających efektywność pracy działu HR firmy. Raporty takie można, za pomocą istniejącego już modułu do mailingu, wysyłać cyklicznie do odpowiednich osób w firmie, na przykład do prezesa. Innym, a zarazem powiązanym do poprzedniego aspektem, może być generowanie wykresów dla samych pracowników działu HR. Statystyki takie bardzo łatwo uwidoczniałyby mocne i słabe pytania w testach rekrutacyjnych. Jeśli blisko sto procent petentów odpowiadałoby na dane pytanie, to może warto wykreślić je z listy i zastąpić innym. Jeśli większość odpowiada źle, to może samo pytanie zostało źle sformułowane i również należaałoby wtedy pochylić się nad tym problemem. Serwer ma nawet zaimplementowane punkty końcowe, które zwracają dane, na podstawie których można takie statystyki wygenerować. Kolejnym krokiem w rozwoju projektu jest zaimplementowanie powiadomień w czasie rzeczywistym w aplikacji mobilnej, przykładowo w momencie, gdy użytkownikowi zostanie przypisany kod testu lub gdy z uwzględnieniem wcześniej wspomnianego pomysłu przyjdzie wiadomość od pracownika działu *HR*. Dobrym kierunkiem działań jest również zaimplementowanie możliwości tworzenia własnego życiorysu na podstawie podanych danych, dzięki czemu użytkownik nie musiałby korzystać z innych programów, wspierających jego tworzenie bądź sam nie musiałby marnować na to większej ilości czasu. Podsumowując, aplikacja daje możliwości wielowektorowego rozwoju. Jest to zasługa głównie dwóch składowych: samego pomysłu na projekt oraz projektowania go z uwzględnieniem dalszych rozbudowań.

150:6423631036

Literatura

- [1] P. Zdziech, „Nie kupuj Pan cegły», czyli dla kogo platforma do rekrutacji”,
<https://erecruiter.pl/blog/dla-kogo-plataforma-do-rekrutacji/>,
(dostęp 21.11.2020).
- [2] W. Earp, „Why sharing passwords is a bad idea”,
<https://swgfl.org.uk/magazine/why-sharing-passwords-is-a-bad-idea/>, (dostęp 21.11.2020).
- [3] Materiały firmy SolarWinds, „Top Seven Reasons Why You Should Not Share Your Passwords”,
<https://logicalread.com/top-seven-reasons-why-you-should-not-share-your-passwords>, (dostęp 21.11.2020).
- [4] Materiały firmy SUNY Broome, „Top reasons why you shouldn't share your username and password”,
<https://news.sunybroome.edu/focus/top-reasons-why-you-shouldnt-share-your-username-and-password/>, (dostęp 21.11.2020).
- [5] Materiały Massachusetts Institute of Technology, „Strong Passwords”,
<http://kb.mit.edu/confluence/display/istcontrib/Strong+Passwords>,
(dostęp 21.11.2020).
- [6] Materiały spółki CIS®, „Do Not Share Your Password”,
<https://www.cisecurity.org/daily-tip/do-not-share-your-password/>,
(dostęp 21.11.2020).
- [7] K. Hristozov, „MySQL vs PostgreSQL – Choose the Right Database for Your Project”,
<https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres>,
(dostęp 21.11.2020).

- [8] Materiały spółki VMware, „Spring Framework Documentation”
<https://docs.spring.io/spring-framework/docs/5.1.0.RELEASE/spring-framework-reference/>, (dostęp 21.11.2020).
- [9] M. Wojciechowski, „Aplikacje SPA, Angular, TypeScript”,
<http://www.cs.put.poznan.pl/mwojciechowski/cdv/ria/Angular.pdf>,
(dostęp 21.11.2020).
- [10] Materiały firmy Microsoft, „TypeScript Documentation”,
<https://www.typescriptlang.org/docs/>, (dostęp 21.11.2020).
- [11] Materiały firmy Google, „Flutter documentation”,
<https://flutter.dev/docs>, (dostęp 21.11.2020).
- [12] A. Sulich, „Modele i techniki rekrutacji i selekcji realizowane przez przedsiębiorstwa w województwie dolnośląskim”, [w:] Nurty badawcze w zarządzaniu, Wydawnictwo GSP, Zgorzelec 2015.
- [13] Materiały firmy Grant Thornton, „Rekrutacja i selekcja nie tylko w czasach SARS-CoV-2 – Purporowy informator”,
https://grantthornton.pl/wp-content/uploads/2020/06/Covid-rekrutacja-i-selekcja_GrantThornton_16062020.pdf,
(dostęp 22.11.2020)
- [14] S. Adam, „Modele i techniki rekrutacji i selekcji realizowane przez przedsiębiorstwa w województwie dolnośląskim” [w:] Nurty badawcze w zarządzaniu., Wydawnictwo GSP, Zgorzelec 2015.
- [15] L. Kulczycka, „Jak najlepiej zaprezentować się podczas rozmowy kwalifikacyjnej”, Wydawnictwo Wolters Kluwer Polska SA, Warszawa 2013.
- [16] J. Conway, R. A. Jako, D. F. DeLong, „A meta-analysis of interrater and internal consistency reliability of selection interviews”, Artykuł z „Journal of Applied Psychology”, 1995.
- [17] McEwen, Bruce S. „Stressful experience, brain, and emotions: Developmental, genetic, and hormonal influences”, 1995.

- [18] K. Rojewska, „Filmy dla rekrutera”,
<https://nofluffjobs.com/blog/filmy-dla-rekrutera/>,
(dostęp 23.11.2020).
- [19] Rozporządzenie Parlamentu Europejskiego i Rady (UE) 2016/679 z dnia 27 kwietnia 2016 r. w sprawie ochrony osób fizycznych w związku z przetwarzaniem danych osobowych i w sprawie swobodnego przepływu takich danych oraz uchylenia dyrektywy 95/46/WE (ogólne rozporządzenie o ochronie danych).
- [20] Rozporządzenie Ministra Rodziny, Pracy i Polityki Społecznej z dnia 10 grudnia 2018 r. w sprawie dokumentacji pracowniczej (Dz. U. poz. 2369) 21 Ustawa z dnia 26 czerwca 1974 r.- Kodeks Pracy (Dz. U. z 2019 r. poz. 1040, 1043 i 1495).
- [21] Ustawa z dnia 26 czerwca 1974 r.- Kodeks Pracy (Dz. U. z 2019 r. poz. 1040, 1043 i 1495).
- [22] Informacje licencyjne firmy Jet Brains,
<https://www.jetbrains.com/idea/buy/#commercial?billing=yearly>,
(dostęp 11.12.2020).
- [23] Warunki licencyjne oprogramowania Microsoft – Visual Studio Code,
<https://code.visualstudio.com/license> (dostęp 11.12.2020).
- [24] Xcode12.2 – umowa licencyjna,
<https://apps.apple.com/pl/app/xcode/id497799835?l=pl>,
(dostęp 11.12.2020).
- [25] Materiały System Administration Portal, „PostgreSQL – pgAdmin”,
<https://pl.admininfo.info/postgresql-pgadmin>, (dostęp 11.12.2020).
- [26] A. Rahmatulloh, „Performance comparison of signed algorithms on JSON Web Token”, 2019,
<https://iopscience.iop.org/article/10.1088/1757-899X/550/1/012023/pdf>, (dostęp 12.12.2020).

- [27] A. Michalczyk „Kompendium bezpieczeństwa haseł – atak i obrona (część 1.)”,
<https://sekurak.pl/kompendium-bezpieczenstwa-hasel-atak-i-obrona/>, (dostęp 12.12.2020).
- [28] Materiały serwisu 1024kb.pl,
<https://1024kb.pl/spring/jak-dziala-spring-web-mvc/>,
(dostęp 17.12.2020).
- [29] Materiały serwisu Crypto-IT,
<http://www.crypto-it.net/pl/symetryczne/aes.html>, (dostęp 14.12.2020).
- [30] M. Załęczny, „Tutorial Android – Cykl życia aktywności”,
<https://progmar.net.pl/tutorials/android/index.php?topic=007-activities-life-cycle>, (dostęp 15.12.2020).
- [31] S. Biswas, „Flutter: Stateful vs Stateless Widget”,
<https://medium.com/flutter-community/flutter-stateful-vs-stateless-db325309deae>, (dostęp 15.12.2020).