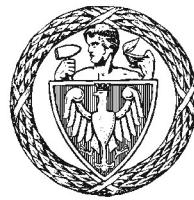


Politechnika Warszawska

W Y D Z I A Ł M A T E M A T Y K I
I N A U K I N F O R M A C Y J N Y C H



Praca dyplomowa inżynierska

na kierunku Informatyka

Wielowarstwowy system rekrutacji dla szkół z interfejsem
webowym i aplikacją mobilną

Andrzej Westfalewicz

Numer albumu 286423

Filip Zyskowski

Numer albumu 286437

promotor

dr inż. Paweł Kotowski

WARSZAWA 2020

.....
podpis promotora

.....
podpisy autorów

Streszczenie

Wielowarstwowy system rekrutacji dla szkół z interfejsem webowym i aplikacją mobilną

„RecruitMe” czyli „Wielowarstwowy system rekrutacji dla szkół z interfejsem webowym i aplikacją mobilną” to system upraszczający rekrutację dla szkoły jak i jej potencjalnych uczniów. Po analizie biznesowej projektu zidentyfikowane zostały funkcjonalności potrzebne w systemie, które następnie zostały zaimplementowane w sposób prosty i intuicyjny dla Użytkownika. Kandydat może wybrać, w jaki sposób chce używać systemu – czy poprzez stronę internetową, czy przez aplikację mobilną na telefony z systemem Android. Poza podstawowym proces rekrutacji obsłużonym w systemie – utworzenie konta, wprowadzenie danych i dokonanie opłaty, stworzone zostały dodatkowe funkcje takie jak bezpośredni kontakt pomiędzy Kandydatem a administracją szkoły czy możliwość drukowania i sczytywania kart egzaminacyjnych. System jest napisany według popularnych i sprawdzonych praktyk programistycznych w najnowocześniejszych technologiach, co ułatwia jego rozwój oraz utrzymanie. Aplikacje serwerowe są łatwe do wdrożenia i mogą być uruchamiane na dowolnym systemie operacyjnym.

Dla autorów pracy była to możliwość poznania nowych technologii, przetestowania wzorców projektowych oraz stworzenia złożonej aplikacji od podstaw. Projekt wymusił również wykształcenie umiejętności miękkich takich jak zarządzanie projektem, zdolności prezentacji czy pracy pod presją czasu.

Słowa kluczowe: aplikacja mobilna, aplikacja webowa, serwer API, uwierzytelnianie JWT, OMR, bramka płatnicza, ASP.NET Core, Vue, NativeScript, MySQL, Docker, Entity Framework Core

Abstract

N-tier system for managing school applications with mobile app and Web interface

"RecruitMe" also known as "N-tier system for managing school applications with mobile app and Web interface" is an integrated tool providing aid with recruitment process both for the school and its candidates. Candidates can interact with the system in two ways: using the website or the mobile application. Business analysis, that was carried out during initial phase of the project, provided required features that were later implemented in a clear and intuitive way for the end-user. The main functionality is the recruitment process composed of: creating an account, providing required data and issuing payment. Additional features include: direct communication between candidates and the school using chat feature, exam sheets that enable quick insertion of the scores to the system. System was implemented using popular and reliable design principles with state of the art technologies, which makes it easily expandable and cheap to maintain. Server applications are easy to deploy and can be hosted on any operating system.

Creating the project was an opportunity to learn new technologies, get experienced with design patterns and create a complicated application from scratch. Learning some soft skills such as project management, presentation skills and meeting deadline was also required.

Keywords: mobile application, web application, API server, JWT authentication, OMR, payment gateway, CQRS, Single Page Application, ASP.NET Core, Vue, NativeScript, MySQL, Docker, Entity Framework Core

Warszawa, dnia

Oświadczenie

Oświadczamy, że pracę inżynierską pod tytułem „Wielowarstwowy system rekrutacji dla szkół z interfejsem webowym i aplikacją mobilną”, której promotorem jest dr inż. Paweł Kotowski, wykonaliśmy samodzielnie, co poświadczamy własnoręcznymi podpisami.

.....

Spis treści

Wstęp	13
Podział pracy	14
1. Słownik pojęć	15
2. Funkcjonalności systemu	17
2.1. Funkcjonalności Użytkownika	17
2.2. Funkcjonalności Kandydata	17
2.3. Funkcjonalności Organizatora Rejestracji	19
2.4. Wymagania niefunkcjonalne	21
3. Analiza ryzyka	22
4. Instrukcja użytkowania	23
4.1. Aplikacja webowa	23
4.2. Aplikacja mobilna	23
4.3. Zarządzanie kontem	24
4.3.1. Rejestracja	24
4.3.2. Logowanie	25
4.3.3. Przypomnienie loginu	26
4.3.4. Resetowanie hasła	27
4.4. Profil Kandydata	28
4.4.1. Ustawianie zdjęcia profilowego	29
4.4.2. Dodawanie dodatkowych danych	30
4.4.3. Przesłanie dokumentów	31
4.4.4. Sprawdzenie statusu rekrutacji	31
4.5. Płatności	32
4.5.1. Dokonanie płatności	33
4.6. Egzaminy	34
4.7. Chat	35
4.8. Panel Administratora	36
4.8.1. Dodawanie danych do Systemu	37
4.8.2. Edycja istniejących danych	37
4.8.3. Usuwanie danych	38
4.8.4. Drukowanie identyfikatora i chat	38
4.8.5. Skanowanie ocen	39
5. Dokumentacja techniczna	41
5.1. Architektura	41
5.1.1. Baza danych	41
5.1.2. Serwer API	41
5.1.3. Aplikacja webowa	41
5.1.4. Aplikacja mobilna	42

5.2.	Wzorce projektowe modułów	42
5.2.1.	Baza danych	42
5.2.2.	Logika biznesowa	43
5.2.3.	Serwer API	44
5.2.4.	Aplikacja webowa	44
5.2.5.	Aplikacja mobilna	44
5.2.6.	Testy	44
5.3.	Warstwy systemu	45
5.4.	Opis modułów	46
5.4.1.	Logika biznesowa	46
5.4.2.	Testy	46
5.4.3.	API	46
5.4.4.	Aplikacja webowa	47
5.4.5.	Aplikacja mobilna	47
5.5.	Implementacja wybranych aspektów	48
5.5.1.	Konfiguracja	48
5.5.2.	Wstrzykiwanie zależności	49
5.5.3.	Abstrakcja operacji	49
5.5.4.	Integracja z Dotpay	50
5.5.5.	Generowanie identyfikatora Kandydata	52
5.5.6.	Generowanie i przetwarzanie kart egzaminacyjnych	52
5.5.7.	Wysyłanie wiadomości e-mail	52
5.5.8.	Docker	52
5.5.9.	Identity Server	53
5.6.	Opis UI/UX	54
5.6.1.	Aplikacja webowa	54
5.6.2.	Aplikacja mobilna	55
5.7.	Komunikacja	56
5.8.	Interfejsy zewnętrzne	56
5.8.1.	SMTP	56
5.8.2.	Integracja Dotpay	56
5.9.	Wybór technologii	56
5.9.1.	Języki programowania	56
5.9.2.	Użyte biblioteki	57
5.9.3.	System operacyjny	58
6.	Instrukcja instalacji	60
6.1.	Aplikacja mobilna	60
6.1.1.	Kompilacja Aplikacji do pliku APK	60
6.1.2.	Instalacja aplikacji na urządzeniu	61
6.2.	Aplikacja webowa	61
6.3.	Baza danych oraz Serwer API	62
6.3.1.	Konfiguracja	62
6.3.2.	Wdrożenie	64

6.4.	Elementy nie pokryte instrukcją	66
6.4.1.	Publikacja Aplikacji w sklepie Google Play	66
6.4.2.	Konfiguracja sieci Serwera	66
6.4.3.	Założenie konta produkcyjnego Dotpay	66
7.	Testy	67
7.1.	Testy automatyczne	67
7.1.1.	Testy jednostkowe	67
7.1.2.	Testy integracyjne	69
7.1.3.	Uruchomienie środowiska testowego	69
7.2.	Test manualne	70
8.	Podsumowanie	71

Wstęp

Celem pracy było stworzenie systemu rekrutacji umożliwiającego przeprowadzenie procesu rekrutacji do szkoły. Systemy takie są już używane w wielu instytucjach np. na Politechnice Warszawskiej (www.zapisy.pw.edu.pl). System „RecruitMe” nie jest skonfigurowany pod jedną konkretną szkołę, dzięki czemu może być wdrożony przez każdą instytucję.

Projekt został poddany analizie biznesowej od strony wymaganych funkcjonalności jak i mocnych stron oraz ryzyka. Dobrane funkcjonalności mają ułatwić przeprowadzenie procesu rekrutacji poprzez narzędzia dla administratora systemu takie jak drukowanie identyfikatorów, bezpośredni kontakt z zarejestrowanymi kandydatami czy drukowanie kart egzaminacyjnych. Natomiast dla kandydatów, zwłaszcza zagranicznych, najważniejsza jest możliwość opóźnienia wizyt w szkole oraz zminimalizowania ich liczby.

Stworzenie systemu miało w sobie kilka wyzwań, w tym niektóre nie były przewidziane. Największym wyzwaniem było utrzymanie kodu w wysokiej jakości według przyjętych praktyk – spowodowało to również, że realizacja projektu znacznie się wydłużała. Innym problemem był brak doświadczenia autorów w tworzeniu aplikacji na telefony komórkowe i brak znajomości paczek realizujących sczytywanie z dokumentów (OMR), ale dzięki temu zadanie było kształcące i pouczające. Nieprzewidzianymi problemami były natomiast trudności ze stworzeniem architektury i zintegrowanie ze sobą używanych paczek osób trzecich, zwłaszcza w aplikacjach klienckich. Problemy te są specyficzne dla projektów zaczynanych od zera, z którymi autorzy nie mieli doświadczenia.

Projekt był realizowany przez dwuosobowy zespół: Andrzej Westfalewicz i Filip Zyskowski. Zadania zostały podzielone według funkcjonalności: odpowiedzialnością Filipa było dostarczenie aplikacji mobilnej oraz zaimplementowanie chatu i zintegrowanie bramki płatniczej w całym systemie. Pozostałe funkcjonalności, czyli głównie stworzenie architektury oraz implementacja aplikacji webowej, były odpowiedzialnością Andrzeja.

Mimo rozłączności zadań programistów kod przechodził proces nadzoru kodu (ang. *Code Review*), aby drugi programista mógł sprawdzić, czy źródła są napisane poprawnie, ale również aby wiedział, co zostało dodane do systemu. Ewentualne uwagi były dodawane w postaci komentarzy, które twórca kodu musiał wyjaśnić przed dokonaniem scalenia z główną wersją kodu.

Podział pracy

Podczas wytwarzania aplikacji w pierwszej kolejności funkcjonalność była tworzona na Serwerze API wraz z Aplikacją webową, aby mogła ona być przetestowana. Dlatego też funkcjonalności na Aplikacji webowej i Serwerze API mają zawsze tego samego autora. Następnie, po wykonaniu testów i sprawdzeniu jakości kodu, funkcjonalności były implementowane na Aplikacji mobilnej. Wymienione w tabeli poniżej aspekty i komponenty zostały opisane w dalszej części pracy.

Szczegółowy opis podziału pracy względem komponentów projektu.

Aspekt projektu	Autor aspektu w Aplikacji webowej i w Serwerze API	Autor aspektu w Aplikacji mobilnej
Utworzenie projektu i podstawowej architektury	Andrzej Westfalewicz	Filip Zyskowski
Implementacja mechanizmu uwierzytelniania	Andrzej Westfalewicz	Filip Zyskowski
Umożliwienie tworzenia konta i innych powiązanych funkcjonalności	Andrzej Westfalewicz	Filip Zyskowski
Stworzenie strony profilowej Kandydata	Andrzej Westfalewicz	Filip Zyskowski
Implementacja panelu Administratora	Andrzej Westfalewicz	<i>Panel Administratora nie jest dostępny na aplikacji mobilnej</i>
Integracja bramki płatniczej	Filip Zyskowski	Filip Zyskowski
Stworzenie chatu	Filip Zyskowski	Filip Zyskowski
Przystosowanie projektu do wdrażania	Andrzej Westfalewicz	Filip Zyskowski

Praca dyplomowa była tworzona wspólnie i każdy członek zespołu tworzył opis (techniczny bądź biznesowy) wyprodukowanego przez siebie komponentu. Aby treść każdego z rozdziałów była w jednym stylu, praca została podzielona na poniższe sekcje, które wymieniona przy sekcji osoba, scalila i doprowadziła do końcowego stanu.

- Analiza biznesowa (rozdziały 1-3) – Andrzej Westfalewicz
- Instrukcja użytkowania – Filip Zyskowski
- Dokumentacja techniczna – Andrzej Westfalewicz
- Instrukcja instalacji – Filip Zyskowski

1. Słownik pojęć

- **System** – rozwiązanie mające na celu rekrutację uczniów do szkół.
- **Architektura** – podstawowa organizacja systemu wraz z jego komponentami, powiązaniemi i regułami ustanawiającymi sposób budowy.
- **Szkoła** – instytucja oświatowo-wychowawcza będąca odbiorcą systemu.
- **Użytkownik** – osoba korzystająca z systemu.
- **Organizator (Administrator)** – Użytkownik odpowiedzialny za proces rekrutacji uprawniony do oglądu w dane Kandydatów. Otrzymuje on dostęp do specjalnego konta umożliwiającego zarządzanie systemem.
- **Kandydat** – Użytkownik, który chce wziąć udział w rekrutacji do szkoły. Na dane Kandydata składają się:
 - ID Kandydata,
 - Imię i nazwisko,
 - Numer PESEL (lub stwierdzony jego brak),
 - Adres e-mail,
 - Data urodzenia,
 - Dane rodziców,
 - Nazwa szkoły podstawowej,
 - Adres zamieszkania,
 - Zdjęcie profilowe,
 - Przesłane pliki.
- **ID Kandydata** – unikalny identyfikator Kandydata składający się z: 3 pierwszych liter imienia, 3 pierwszych liter nazwiska oraz 3 cyfr gwarantujących unikalność.
- **Login** – specjalna fraza używana do zalogowania się do Systemu: dla Kandydatów jest to ID Kandydata, dla Administratora: „admin”.
- **Aplikacja webowa (internetowa)** – program komputerowy wykonujący się w środowisku przeglądarki internetowej na urządzeniu Użytkownika.
- **Aplikacja mobilna** – program działający na urządzeniach mobilnych (telefony komórkowe, smartfony, tablety).
- **Aplikacja kliencka** – aplikacja webowa lub aplikacja mobilna, która daje dostęp do systemu.
- **Serwer API** – program komputerowy, pracujący nieustannie na serwerze i komunikujący się poprzez sieć z Aplikacjami klienckimi.

- **Aplikacja** – aplikacja kliencka lub Serwer API.
- **Rejestracja** – proces przekazania danych Kandydata do Systemu oraz wpłacenia Kwoty rekrutacyjnej.
- **Okres rekrutacyjny** – okres ustalony przez szkołę, w którym można się rejestrować.
- **Egzamin** – forma sprawdzenia wiedzy Kandydatów po udanym procesie rejestracji, będąca podstawą do przyjęcia Kandydata do Szkoły lub odrzucenia wniosku. Na dane Egzaminu składają się:
 - Rodzaj Egzaminu (tematyka egzaminu np.: humanistyczny, ścisły),
 - Forma Egzaminu (indywidualny, pisemny),
 - Data i godzina rozpoczęcia,
 - Czas trwania,
 - Limit Kandydatów.
- **Kwota rekrutacyjna** – ustalona przez Szkołę na Okres rekrutacji kwota pieniężna, która musi być uiszczena przed ukończeniem procesu rejestracji.
- **OMR** – ang. *optical mark recognition*, technologia sczytywania danych ze specjalnie przygotowanych do tego celu, dokumentów bazując na znacznikach zawartych w dokumencie [1].

2. Funkcjonalności systemu

Projekt został poddany analizie biznesowej, aby sprecyzować wymagane w Systemie funkcjonalności. Sformułowane one zostały w postaci historyjek użytkownika (ang. *user stories*). Charakteryzują się one ustaloną strukturą:

Jako aktor chcę funkcjonalność, aby cel.

Aktorem w powyższym zdaniu w Systemie najczęściej jest Administrator lub Kandydat, jednak na ogół może to być dowolna osoba lub nawet zewnętrzny system. Funkcjonalność to czynność, którą wykonuje aktor lub narzędzie, z którego będzie on korzystał. Opis czynności powinien być możliwie jednoznaczny, jednak nie powinien wchodzić w szczegóły techniczne lub implementację. Cel jest najczęściej efektem czynności lub korzyścią dla aktora. Podczas tworzenia funkcjonalności należy zapewnić, że sposób w jaki ona jest implementowana, dąży do spełnienia celu. Cel jest również przydatny kiedy funkcjonalność okazuje się trudniejsza w implementacji niż było to przewidziane – programista ma wtedy opcję zmiany funkcjonalności tak, aby nadal go wypełnić. Dodatkowo do historyjki można dołączyć komentarze, które określają techniczne szczegóły lub sugerują sposób implementacji.

Główną funkcjonalnością systemu jest proces rejestracji Kandydatów, na który składa się założenie konta, wprowadzenia danych i plików do Systemu oraz uiszczenie opłaty egzaminacyjnej. Zaimplementowane zostały również funkcjonalności poboczne ułatwiające ten proces np. możliwość zresetowania hasła czy przypomnienie Loginu do Systemu. Dodatkowo używając funkcjonalności chatu, Kandydaci mogą szybko otrzymać odpowiedzi na ich pytania dzięki bezpośredniemu kontaktowi z Organizatorem. Użytkownicy mają dostęp do systemu zarówno z przeglądarki internetowej, jak i z Aplikacji mobilnej na telefonach z systemem Android.

Dla Organizatora natomiast, poza wspomnianym chatem, jest dostępny „Panel Administratora”, w którym może zarządzać wszystkimi głównymi obiektami w Systemie. Poza tym dostępne są funkcjonalności upraszczające proces rekrutacji takie jak drukowanie identyfikatorów Kandydatów lub wprowadzanie ocen z Egzaminów za pomocą kart egzaminacyjnych.

Wszystkie funkcjonalności systemu są poniżej dokładniej opisane w postaci historyjek.

2.1. Funkcjonalności Użytkownika

1. Jako Użytkownik mogę otworzyć stronę Systemu i przeglądać ogólnodostępną zawartość, aby zapoznać się ze Szkołą i procesem rekrutacji.
 - Aplikacja jest dostępna z internetu.
 - Aplikacja nie przestaje działać po wystąpieniu błędu.
 - Dostępne są strony: „Strona główna”, „Regulamin”, „O szkole”.

2.2. Funkcjonalności Kandydata

1. Jako Kandydat mogę się zarejestrować, aby wziąć udział w procesie rekrutacji.
 - Wymagane jest podanie następujących danych:
 - Imię,

- Nazwisko,
 - E-mail,
 - Hasło,
 - Data urodzenia,
 - Numer PESEL (lub potwierdzenie jego braku).
- Mail jest potwierdzony za pomocą przesłanego na niego linku.
 - Ustalony jest limit maksymalnej liczby wykorzystania danego adresu e-mail do utworzenia konta Kandydata.
 - Po potwierdzeniu adresu e-mail Użytkownik dostaje swoje ID Kandydata, za pomocą którego ma dostęp do reszty funkcjonalności.
2. Jako Kandydat mogę się zalogować, aby sprawdzić spersonalizowane informacje dotyczące rekrutacji.
- Logowanie odbywa się za pomocą ID Kandydata oraz hasła.
 - Aplikacja rozpoznaje, czy obecny Użytkownik jest zalogowany.
 - Część funkcjonalności jest dostępna tylko dla zalogowanych Użytkowników.
3. Jako Kandydat mogę zresetować hasło, aby nie utracić dostępu do Systemu.
- Na prośbę Systemu należy podać otrzymany Login i na podany w procesie rejestracji e-mail wysyłany jest link do zmiany hasła.
4. Jako Kandydat mogę sprawdzić swoje ID Kandydata, aby nie utracić dostępu do Systemu.
- Na prośbę Systemu należy podać adres e-mail i numer PESEL (w przypadku braku numeru PESEL – adres e-mail, imię i nazwisko).
 - Jeśli wskazany e-mail znajduje się w Systemie i jest potwierdzony, zostaje na niego wysłana informacja o ID Kandydata.
5. Jako zalogowany Kandydat mogę otworzyć stronę profilu, aby edytować dane.
- Strona wyświetla dane podane przy rejestracji, jednak są one zablokowane do edycji
 - Wyświetlane są pozostałe dane do edycji: imiona i nazwiska rodziców, nazwa szkoły podstawowej, adres zamieszkania.
 - Strona profilu umożliwia przesyłanie zdjęcia profilowego.
 - Strona profilu umożliwia przesyłanie dodatkowych dokumentów.
6. Jako zalogowany Kandydat mogę dodać zdjęcie profilowe, aby być lepiej rozpoznawany w Systemie.
- Zdjęcie może zostać wprowadzone na dwa sposoby: wybierając już istniejący plik z dysku lub robiąc zdjęcie kamerą internetową (jeżeli Użytkownik taką posiada).
 - Miniaturka zdjęcia pokazuje się na stronie profilu.
 - Zdjęcie może być podmieniane na nowe.
7. Jako zalogowany Kandydat mogę dokonać opłaty za rejestrację, aby dokonczyć rejestrację.

2.3. FUNKCJONALNOŚCI ORGANIZATORA REJESTRACJI

- Transakcja odbywa się przez internetową bramkę płatniczą Dotpay.
 - Po potwierdzeniu płatności w Systemie Kandydat jest automatycznie przypisany do Egzaminów.
8. Jako Kandydat, który dokonał płatności potwierdzonej w Systemie, mogę sprawdzić terminy swoich egzaminów, aby do nich przystąpić.
- Wyświetla się termin, rodzaj, forma oraz czas trwania egzaminu.
9. Jako Kandydat, mogę nawiązać konwersację z Organizatorem, aby ułatwić komunikację ze Szkołą.
- Konwersacja odbywa się w formie chatu.
 - Kiedy jest otwarta strona chatu, nowe wiadomości synchronizują się co 15 sekund.
10. Jako Kandydat, mogę sprawdzić wynik rekrutacji (przyjęty/nieprzyjęty), aby nie była wymagana bezpośrednia komunikacja ze Szkołą w tej sprawie.
- Wynik rekrutacji będzie widoczny na stronie profilu.

Aplikacja mobilna

Powysze funkcjonalności są również dostępne w aplikacji mobilnej, która ma na celu umożliwienie Kandydatom wygodniejszego dostępu do Systemu z urządzeń mobilnych. Dopuszczalne są zmiany w szczegółach implementacji w związku ze zmianą rozmiaru ekranu.

2.3. Funkcjonalności Organizatora Rejestracji

1. Jako Organizator mogę zalogować się do Aplikacji webowej, aby nadzorować proces rekrutacji.
 - Aplikacja rozpoznaje, że zalogowany Użytkownik jest Organizatorem.
 - Funkcjonalności Organizatora różnią się od funkcjonalności Kandydata.
2. Jako Organizator mogę zarządzać danymi w Systemie, aby obsłużyć nieprzewidziane przypadki oraz sprawdzić informacje w Systemie.
 - Stworzony zostaje Panel Administratora umożliwiający tworzenie, odczyt, edycję i usuwanie obiektów w Systemie.
3. Jako Organizator mogę komunikować się z Kandydatami przez chat, aby wymieniać z nimi informacje.
 - Dostępna jest lista konwersacji z Kandydatami.
 - Konwersacje z nieprzeczytanymi wiadomościami są wyróżnione.
 - Po kliknięciu w daną konwersację można odczytać wszystkie wymienione wiadomości oraz wysłać nową.
 - Gdy zostanie wysłana nowa wiadomość, będzie ona dostępna dla Kandydata po maksymalnie 30 sekundach. Jeśli Kandydat nie jest obecnie na stronie chatu, pojawi się powiadomienie o nieprzeczytanych wiadomościach.

4. Jako Organizator mam dostęp do strony umożliwiającej zarządzanie egzaminami, aby usprawnić proces rekrutacji.

- Aby utworzyć egzamin wymagane jest podanie danych:
 - Rodzaj Egzaminu,
 - Forma Egzaminu (pisemny, indywidualny),
 - Data i godzina rozpoczęcia,
 - Czas trwania,
 - Limit miejsc Kandydatów.
 - Po dokonaniu opłaty Kandydatowi jest przypisywany po jednym Egzaminie z każdego rodzaju. Jeśli Egzamin jest pisemny, godziną Egzaminu jest godzina rozpoczęcia, jeśli Egzamin jest ustny to godziną Egzaminu jest: (godzina rozpoczęcia + czas trwania * [numer na liście - 1]).
 - Można edytować listę Kandydatów przystępujących do danego Egzaminu.
5. Jako Organizator mogę pobrać identyfikator Kandydata, aby łatwiej przeprowadzać Egzaminy.
- Identyfikator jest generowany automatycznie i zawiera zdjęcie profilowe Kandydata oraz jego imię i nazwisko.
 - Na identyfikatorze są zakodowane dane Kandydata, np. w formie kodu QR, umożliwiające elektroniczne pobranie danych.
6. Jako Organizator mogę wprowadzać do systemu nauczycieli, którzy będą nadzorować Egzaminy, aby w Systemie przechowywać informację, kto egzaminował danego Kandydata.
- Nauczycielom nie jest tworzone konto w systemie.
 - Nauczyciela należy przypisać przy wystawianiu oceny Kandydatowi z danego egzaminu.
7. Jako Organizator mogę wygenerować w Systemie karty egzaminacyjne, aby nauczyciel mógł je wypełniać w trakcie egzaminu.
- Dokument jest przystosowany do przetwarzania OMR.
 - Zawiera informację o Kandydatach biorących udział w egzaminie. Zamalowanie odpowiedniego pola oznacza przyznanie Kandydatowi określonej oceny.
8. Jako Organizator mogę załadować do Systemu wypełnione karty egzaminacyjne, aby szybko wprowadzić dane do Systemu.
- Należy wybrać Nauczyciela, który wypełniał kartę egzaminacyjną.
 - Odczytane dane: dane Nauczyciela, Kandydata i Egzaminu, są zapisane automatycznie do Systemu.

2.4. Wymagania niefunkcjonalne

- Komunikacja wewnętrz Systemu odbywa się poprzez połączenie bezpieczne (protokół HTTPS).
- Wszystkie akcje zalogowanego Użytkownika są autoryzowane na podstawie kryptograficznie podpisanej tokenu.
- Wywołanie każdej funkcjonalności, w szczególności funkcjonalności Organizatora, jest poprzedzone sprawdzeniem, czy obecny Użytkownik jest do niej uprawniony.
- Aplikacja będzie dostępna przez co najmniej dwadzieścia dwie godziny w ciągu doby w Okresie rekrutacyjnym.
- Architektura zapewnia bezproblemowe równoległe korzystanie z systemu przez co najmniej stu Użytkowników.
- Przy połączeniu szerokopasmowym, wywołany przez Użytkownika interfejs otworzy się po czasie nie dłuższym niż trzy sekundy.
- Aplikacja będzie działała bezproblemowo na:
 - przeglądarkach internetowych Google Chrome, Opera, Mozilla Firefox, Safari, Microsoft Edge od wersji odpowiednio 60, 50, 60, 10.1 oraz 15
 - telefonach komórkowych z systemem Android od wersji 4.1.
- Aplikacja mobilna zadziała na telefonach, w których jest możliwość zainstalowania zewnętrznych aplikacji.
- Aplikacje klienckie do poprawnego działania wymagają połączenia z Internetem. Na urządzeniach bez możliwości połączenia z Internetem Aplikacje klienckie nie zadziałają.

Aplikacja mobilna do swojego działania wykorzystuje możliwości udostępniane przez obecne urządzenia mobilne. Aby zapewnić poprawne działanie aplikacji, przy instalacji lub przy pierwszym uruchomieniu funkcji korzystającej z danego modułu telefonu Użytkownik będzie poproszony o zgodę na:

- **odbieranie danych z internetu i pełny dostęp do sieci,**
- **wyświetlanie połączeń sieciowych,**
- **dostęp do aparatu** – na potrzeby zrobienia zdjęcia profilowego Kandydata,
- **dostęp do multimedialnych** – na potrzeby wybrania zdjęcia profilowego Kandydata z galerii telefonu.

3. Analiza ryzyka

W ramach analizy biznesowej przeprowadzona została analiza ryzyka w postaci analizy SWOT:

- **Mocne strony**

- Zespół ma doświadczenie w pisaniu aplikacji webowych, co znacznie przyspieszy proces tworzenia produktu.
- Projekt jest łatwo rozszerzalny o nowe funkcjonalności, które mogą być dodane w zależności od oczekiwaniń.
- System nie wymaga dużej serwerowni do uruchomienia – pozwala na zainstalowanie systemu na dostępnych maszynach danej Szkoły.

- **Słabe strony**

- Uzależnienie aplikacji od połączenia internetowego – awaria Internetu uniemożliwia dostęp i korzystanie z Systemu.
- Powolny proces dostosowywania aplikacji do potrzeb odbiorcy, jeżeli te są bardzo wymagające.

- **Szanse**

- Przejrzysta aplikacja mobilna jest szansą na zainteresowanie się projektem większej liczby instytucji.
- Unikalne dla tego typu produktów funkcjonalności w postaci sczytywania kart ocen lub chatu.
- Dobrze udokumentowane narzędzia do tworzenia aplikacji webowej i aplikacji mobilnej.

- **Zagrożenia**

- Istniejące systemy rekrutacji na rynku.
- Ustalony termin ukończenia prac nad systemem może spowodować brak realizacji założonych wcześniej funkcjonalności.
- Brak doświadczenia w tworzeniu aplikacji mobilnych w wymaganej technologii.

4. Instrukcja użytkowania

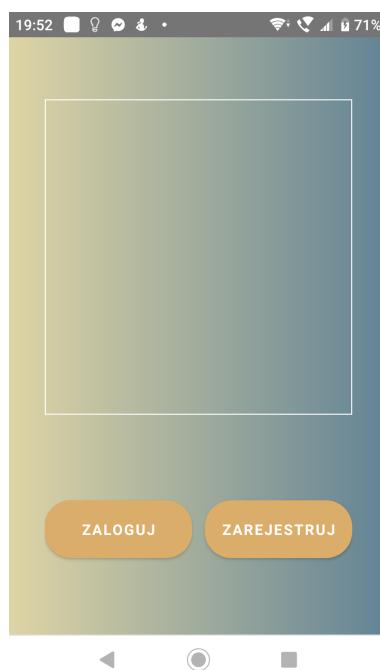
Ta część pracy ma na celu zaznajomienie Użytkownika z optymalnym użytkowaniem „Wielowarstwowego systemu rekrutacji dla szkół z interfejsem webowym i aplikacją mobilną”. Z nazwy projektu wynika naturalny podział tego dokumentu na dwie części – webową oraz mobilną. W każdej z nich zostaną opisane funkcjonalności aplikacji oraz gdzie w interfejsie Użytkownika można je znaleźć. Zastrzega się możliwość wystąpienia nieznacznych nieścisłości, które znalazłyby się na prezentowanych tutaj obrazkach względem ostatecznej wersji programu (np.: logo programu, kolory tła, przycisków), które zależą od ostatecznej konfiguracji zależnej od Szkoły. Niezmienione zostaną położenia przycisków w interfejsie graficznym aplikacji i ich napisy. Wszystkie zmiany będą miały na celu wyłącznie polepszenie jakości ostatecznego produktu.

4.1. Aplikacja webowa

Do Aplikacji webowej można się dostać poprzez użycie dowolnego urządzenia elektronicznego z dostępem do internetu z zainstalowaną przeglądarką internetową. Rezultatem zastosowania każdego kroku z instrukcji instalacji będzie otrzymanie publicznego adresu internetowego, który wpisany do paska adresu przeglądarki internetowej powinien otworzyć stronę główną aplikacji.

4.2. Aplikacja mobilna

Ukończony proces wdrażania (zgodny z procesem opisany w instrukcji instalacji aplikacji mobilnej) powinien skutkować otrzymaniem pliku z rozszerzeniem .apk, który może być zainstalowany na dowolnym urządzeniu mobilnym z systemem Android, a także być udostępniony w serwisie Google Play w celu otwarcia się na większą liczbę Użytkowników.



Rysunek 4.1: Pierwszy widok po otwarciu aplikacji mobilnej

4.3. Zarządzanie kontem

4.3.1. Rejestracja

Aby móc korzystać z jakichkolwiek funkcjonalności serwisu, należy posiadać w nim konto. Proces rejestracji umożliwia utworzenie konta Kandydata jak również dostęp do pozostałych udostępnionych Systemu.

Aplikacja webowa

Należy znaleźć w prawym górnym rogu przycisk *Zarejestruj się* i go nacisnąć. Następnie trzeba uzupełnić formularz wymaganymi do utworzenia konta danymi. Kliknięcie na przycisk *Zarejestruj się* spowoduje, że na adres mailowy podany w formularzu wysłany zostanie mail z linkiem potwierdzającym. Po kliknięciu na link zostanie wyświetlona informacja o specjalnie wygenerowanym loginie, za pomocą którego umówione jest logowanie.

Rysunek 4.2: Formularz rejestracji

Aplikacja mobilna

Po uruchomieniu aplikacji na głównym ekranie wyświetla się dwa przyciski – należy wybrać ten z napisem *Zarejestruj*. Do rejestracji potrzebne są dane, o wpisaniu których Użytkownik zostanie poproszony w tym ekranie. Po kliknięciu przycisku *Zarejestruj się* na górze ekranu pojawi się potwierdzenie wysłania danych do serwera, jak również prośba o potwierdzenie adresu e-mail.

4.3. ZARZĄDZANIE KONTEM



Rysunek 4.3: Formularz rejestracji aplikacji mobilnej

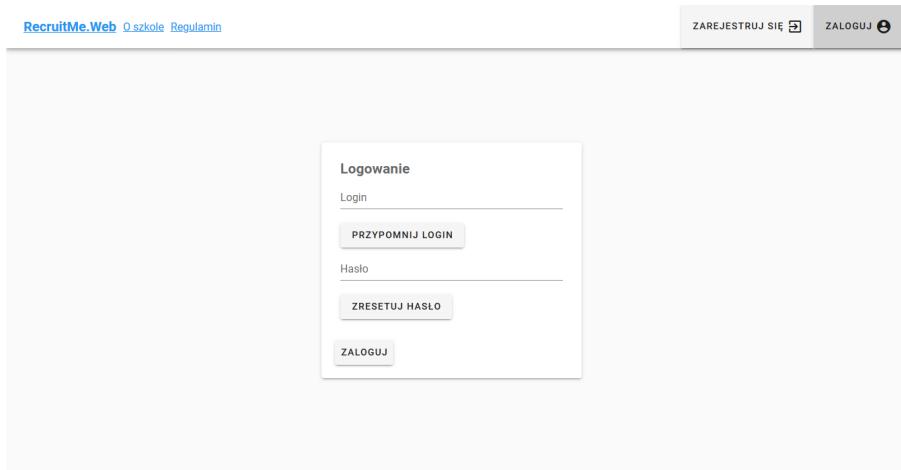
4.3.2. Logowanie

Proces logowania jest kolejnym krokiem po procesie rejestracji Kandydata do uzyskania dostępu do funkcjonalności systemu. Zalogowanie wymaga pamiętania danych uzupełnianych w poprzednim kroku. Warto zaznaczyć, że w przypadku zgubienia lub zapomnienia tych danych istnieje możliwość skorzystania z funkcji przypomnienia loginu lub zresetowania hasła do konta Kandydata.

Aplikacja webowa

Do ekranu logowania można przejść, klikając w prawym górnym rogu na przycisk *Zaloguj się*. Pojawi się wówczas formularz z polami login oraz hasło, które powinny być uzupełnione danymi odpowiednio otrzymanymi i podanymi w procesie rejestracji. Po kliknięciu w przycisk *Zaloguj* Użytkownik zostanie przeniesiony na stronę główną. W przypadku błędnych danych zostanie on poinformowany odpowiednim komunikatem.

4. INSTRUKCJA UŻYTKOWANIA



Rysunek 4.4: Ekran logowania

Aplikacja mobilna

Aby się zalogować, należy nacisnąć na głównym ekranie przycisk z napisem *Zaloguj*. W dostępne pola Użytkownik musi wpisać swój login i hasło oraz kliknąć *Zaloguj się*. Jeżeli zostały wpisane poprawne dane, aplikacja przekieruje go do widoku głównego.



Rysunek 4.5: Ekran logowania w aplikacji mobilnej

4.3.3. Przypomnienie loginu

Przypomnienie loginu odbywa się w tym samym ekranie co proces logowania. Z funkcjonalności powinno się korzystać w przypadku, gdy zapomniane zostanie unikalne ID Kandydata, które umożliwia logowanie.

Aplikacja webowa

Należy kliknąć w prawym górnym rogu przeglądarki w przycisk *Zaloguj się*, a następnie w przy-

4.3. ZARZĄDZANIE KONTEM

cisk *Przypomnienie loginu*. Użytkownik zostanie poproszony o wpisanie swoich danych podanych w procesie rejestracji. Po kliknięciu w przycisk *Sprawdź mój login* na adres e-mail podany podczas tworzenia konta zostanie wysłany login.

The screenshot shows a web page with a header containing 'RecruitMe.Web', 'O szkole', and 'Regulamin' links, along with 'ZAREJESTRUJ SIĘ' and 'ZALOGUJ' buttons. The main content area is titled 'Przypomnij login'. It contains two input fields: 'Adres email' and 'Nr PESEL', and a checkbox labeled 'Nie mam numeru PESEL'. A 'SPRAWDZ MOJ LOGIN' button is at the bottom.

Rysunek 4.6: Formularz przypomnienia loginu

Aplikacja mobilna

W ekranie logowania należy kliknąć przycisk *Przypomnij login*. Następnie Użytkownik musi wy- pełnić potrzebne dane. Po kliknięciu w przycisk *Przypomnij login* na adres e-mail podany w re- jestracji zostanie wysłany login.



Rysunek 4.7: Formularz przypomnienia loginu w aplikacji mobilnej

4.3.4. Resetowanie hasła

Resetowanie hasła odbywa się w tym samym ekranie, co proces logowania. Z funkcjonalności powinno się korzystać w przypadku, gdy zapomni się hasła, bez którego nie będzie możliwe zalogowanie się.

Aplikacja webowa

Należy kliknąć w prawym górnym rogu przeglądarki w przycisk *Zaloguj się*, a następnie w przycisk

Zresetuj hasło. Użytkownik zostanie poproszony o wpisanie swoich danych podanych w procesie rejestracji. Po kliknięciu w przycisk *Resetuj hasło* na adres e-mail podany w rejestracji zostanie wysłany link umożliwiający zmianę hasła. Link przenosi do formularza zmiany hasła, gdzie należy wpisać dwa razy to samo nowe hasło. Po kliknięciu w przycisk *Resetuj hasło*, zostanie wyświetlony ekran potwierdzający i od tej pory można się logować nowym hasłem.

Rysunek 4.8: Formularz resetowania hasła

Aplikacja mobilna

W ekranie logowania należy kliknąć w tekst *Zresetuj hasło*. Po wpisaniu potrzebnych danych i następnie kliknięciu w przycisk *Zresetuj hasło*, na adres e-mail podany w rejestracji zostanie wysłany link umożliwiający zmianę hasła. Dalsze resetowanie hasła odbywa się poprzez aplikację webową.



Rysunek 4.9: Formularz resetowania hasła w aplikacji mobilnej

4.4. Profil Kandydata

Profil Kandydata jest miejscem, do którego zalecane jest się udać tuż po swoim pierwszym zalogowaniu – tam można uzupełnić swoje dodatkowe dane czy ustawić zdjęcie profilowe. W Profilu Kandydata można również sprawdzić stan swojej rekrutacji oraz wysłać dodatkowe pliki takie jak świadectwo ukończenia szkoły podstawowej.

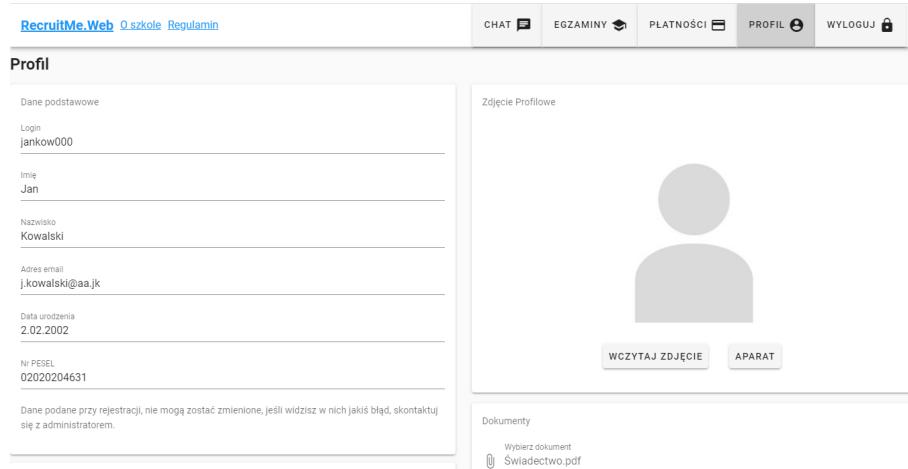
4.4. PROFIL KANDYDATA

4.4.1. Ustawianie zdjęcia profilowego

Kandydat, by ułatwić weryfikację przez nauczycieli na egzaminie, może wysłać do Systemu swoje zdjęcie profilowe.

Aplikacja webowa

Po zalogowaniu Kandydat kliką w przycisk *Profil* w prawym górnym rogu. Po prawej stronie widoczny jest kwadrat z domyślnym zdjęciem Kandydata oraz dwoma przyciskami pod nim. Możliwy jest wybór zdjęcia z komputera i wgranie go do Systemu, klikając w zdjęcie lub w przycisk *Wczytaj zdjęcie*. Drugą opcję jest zrobienie nowego zdjęcia za pomocą kamery internetowej, dostępnej po naciśnięciu przycisku *Aparat*. Użycie tej opcji będzie wymagało zezwolenia na dostęp aplikacji do kamery urządzenia w dialogu wyświetlonym przy pasku adresu strony.



Rysunek 4.10: Ekran profilu Kandydata

Aplikacja mobilna

By uzyskać dostęp do menu aplikacji należy – po zalogowaniu – kliknąć na ikonę trzech białych pasków lub przeciągnąć palcem z lewej do prawej strony ekranu telefonu. Tam należy kliknąć *Ustawienia*. Następnie należy kliknąć jeden z dwóch przycisków: *Z aparatu* lub *Z galerii*. Naciśnięcie pierwszego przycisku po raz pierwszy sprawi, że zostanie wyświetlona prośba o zgodę na użycie aparatu komórki oraz zgodę na użycie multimedialnych. Nieudzielenie zgody uniemożliwi wysłanie zdjęcia na serwer aplikacji. Aplikacja po zrobieniu zdjęcia przez Użytkownika automatycznie wysyła je na serwer. Podobnie się staje po kliknięciu drugiego przycisku – wyświetla się prośba o udzielenie zgody na dostęp do plików, po udzieleniu której Kandydat wybiera zdjęcie, które zostaje zapisane jako zdjęcie profilowe Użytkownika.



Rysunek 4.11: Ekran profilu Kandydata w aplikacji mobilnej

4.4.2. Dodawanie dodatkowych danych

Przed zapisaniem na egzaminy Kandydat zostanie poproszony o wypełnienie dodatkowych danych takich jak imiona i nazwiska opiekunów Kandydata czy adres zamieszkania. Zarówno w Aplikacji webowej, jak i mobilnej, wpisanie dodatkowych danych jest umieszczone w tym samym ekranie co ustawianie zdjęcia profilowego.

Aplikacja webowa

Poniżej danych podstawowych Użytkownika w profilu Kandydata znajduje się formularz danych dodatkowych. Użytkownik po wypełnieniu danych powinien kliknąć przycisk *Zapisz dane*, by dane prawidłowo zapisały się w Systemie.

4.4. PROFIL KANDYDATA

Data urodzenia
31.12.2000

Nr PESEL
90000000000

Dane podane przy rejestracji, nie mogą zostać zmienione, jeśli widzisz w nich jakiś błąd, skontaktuj się z administratorem.

Adres

Imię i nazwisko rodzica 1

Imię i nazwisko rodzica 2

Szkoła podstawowa

ZAPISZ DANE

Rysunek 4.12: Uzupełnianie dodatkowych danych

Aplikacja mobilna

Poniżej zdjęcia profilowego Użytkownika w profilu Kandydata znajduje się formularz danych dodatkowych. Użytkownik po wypełnieniu danych powinien kliknąć przycisk *Zapisz*, by dane prawidłowo zapisały się w Systemie. O sukcesie bądź niepowodzeniu Kandydat zostanie poinformowany odpowiednim komunikatem, który się pojawi u góry ekranu. Funkcjonalność jest widoczna na Rysunku 4.11.

4.4.3. Przesłanie dokumentów

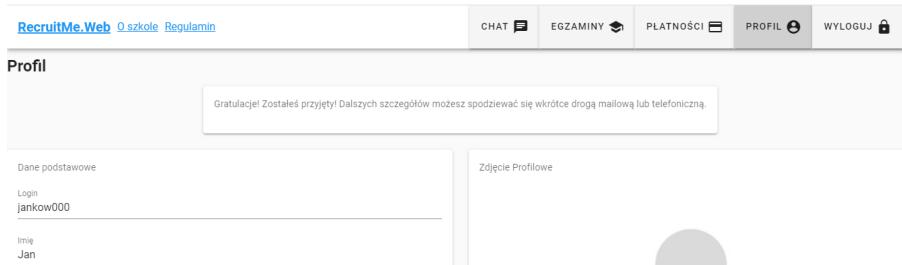
Kandydat może za pomocą systemu przesyłać dodatkowe dokumenty np. świadectwo ukończenia poprzedniej szkoły. Funkcjonalność ta jest dostępna na stronie profilu Kandydata w Aplikacji webowej. Aby przesyłać plik Użytkownik kliką najpierw w pole *Wybierz dokument*. Po wybraniu tymczasowa wartość „Świadectwo.pdf” zamieni się w nazwę pliku. Zapisanie pliku na serwerze odbędzie się dopiero po kliknięciu przycisku *Prześlij dokument*. Przesłane dokumenty są wymienione poniżej pola do ich wprowadzania. Każdy dokument można pobrać lub usunąć z listy. Funkcjonalność jest widoczna na Rysunku 4.10.

4.4.4. Sprawdzenie statusu rekrutacji

W dowolnym momencie po zalogowaniu się do Systemu Kandydat może sprawdzić swój status rekrutacji do Szkoły. W aplikacji zostanie przedstawiona informacja, czy dany Kandydat został przyjęty do Szkoły.

Aplikacja webowa

Jeżeli zostanie ustalony status rekrutacji Kandydata na przyjęty lub odrzucony, na samej górze nad podstawowymi danymi Kandydata i wyborem zdjęcia profilowego, będzie widoczny napis, który jednoznacznie stwierdza status rekrutacji. W przypadku gdy status Kandydata nie zostanie ustalony, Profil Kandydata nie będzie posiadał żadnej dodatkowej informacji.



Rysunek 4.13: Profil Kandydata z pozytywnym statusem rekrutacji

Aplikacja mobilna

W pierwszym ekranie po zalogowaniu Kandydat może sprawdzić swój stan rekrutacji. Na środku ekranu komórki Użytkownik zobaczy tekst wraz z ikoną jednoznacznie stwierdzającą stan jego rekrutacji. Gdy Kandydat nie będzie miał przypisanego statusu rekrutacji, otrzyma informację, że proces rekrutacji jest obecnie w trakcie.



Rysunek 4.14: Profil Kandydata ze statusem rekrutacji w aplikacji mobilnej

4.5. Płatności

Przed zapisaniem się na egzaminy wymagane jest uiszczenie Kwoty rekrutacyjnej. By to ułatwić, System wystawia dogodny interfejs Użytkownika umożliwiający łatwy transfer środków.

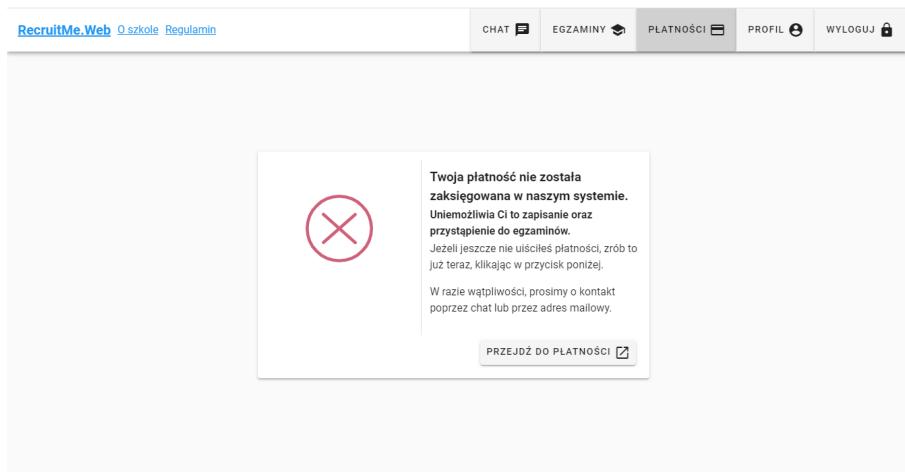
4.5. PŁATNOŚCI

Po realizacji płatności przez bramkę płatniczą System automatycznie rejestruje Kandydata na odpowiednie dostępne w procesie rekrutacji Egzaminy.

4.5.1. Dokonanie płatności

Aplikacja webowa

Aby dokonać płatności, należy przejść do ekranu statusu płatności znajdującego się w prawym górnym rogu ekranu w zakładce *Płatności*. Jeżeli nie opłacono Kwoty rekrutacyjnej, wyświetlona zostaje informacja o braku zaksięgowanej płatności i przycisk w prawym dolnym rogu z możliwością dokonania płatności.



Rysunek 4.15: Ekran statusu płatności

Po kliknięciu Użytkownik zostaje przekierowany na stronę bramki płatniczej Dotpay. Następnie należy wybrać metodę płatności i podać swoje dane. By zatwierdzić płatność, należy wybrać przycisk *Zapłać*. W zależności od wybranej metody płatności, Użytkownik może zostać przekierowany na stronę banku, by uzupełnić dodatkowe dane potrzebne do procesu. Następnie, jeśli proces przejdzie pomyślnie, bramka płatnicza zwróci Użytkownikowi informację o udanej transakcji. Użytkownik ma możliwość skorzystania z przycisku powrotu do Systemu. W zależności od wybranej metody płatności, proces weryfikacji może potrwać od kilku sekund do nawet kilku dni roboczych.

Aplikacja mobilna

Należy kliknąć w trzy białe paski w lewym górnym rogu albo wyciągnąć ekran menu z lewej strony ekranu, a następnie wybrać *Płatności*. Jeżeli nie opłacono Kwoty rekrutacyjnej, wyświetlona zostanie informacja o braku zaksięgowanej płatności i przycisk na dole ekranu z możliwością dokonania płatności. Potem analogicznie do aplikacji webowej następuje przejście przez proces bramki płatniczej. Przy powrocie do naszego Systemu Użytkownik zostanie skierowany z powrotem do aplikacji mobilnej.



Rysunek 4.16: Ekran statusu płatności w aplikacji mobilnej

4.6. Egzaminy

Po wykonaniu procesu płatności system przypisze automatycznie Kandydata do odpowiednich zdefiniowanych przez Administratora rekrutacji egzaminów. Kandydat może zobaczyć wszystkie jego Egzaminy w specjalnie stworzonym do tego widoku. Są one opatrzone informacjami: nazwą kategorii (np. język polski), formą (np. indywidualny), datą rozpoczęcia i czasem trwania egzaminu w minutach.

Aplikacja webowa

Należy wybrać z paska na górze po prawej stronie przycisk *Egzaminy*. Na ekranie wyświetcone zostaną przypisane do Użytkownika Egzaminy, o ile wcześniej została zapłacona Opłata rekrutacyjna i Egzaminy zdefiniowane zostały.

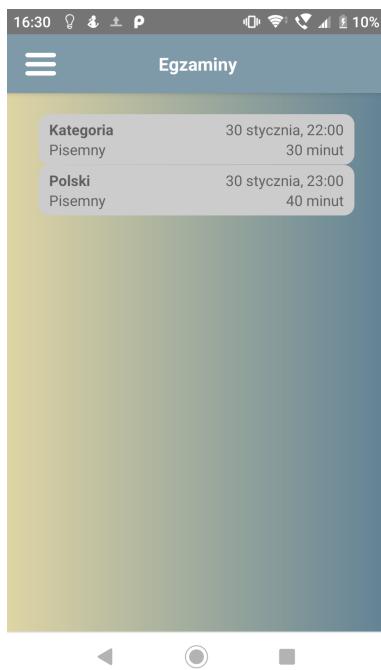
Rysunek 4.17: Ekran egzaminów

Aplikacja mobilna

Należy kliknąć trzy białe paski w lewym górnym rogu albo wyciągnąć z lewej strony ekran menu i wybrać *Egzaminy*. W wypadku, gdy nie zapłacono jeszcze opłaty rekrutacyjnej lub nie

4.7. CHAT

ma jeszcze zdefiniowanych egzaminów dla tej rekrutacji, po przejściu do widoku egzaminów wyświetlna zostanie odpowiednia informacja. W przeciwnym razie wyświetlona zostanie lista egzaminów, na które Kandydat jest zapisany.



Rysunek 4.18: Ekran egzaminów w aplikacji mobilnej

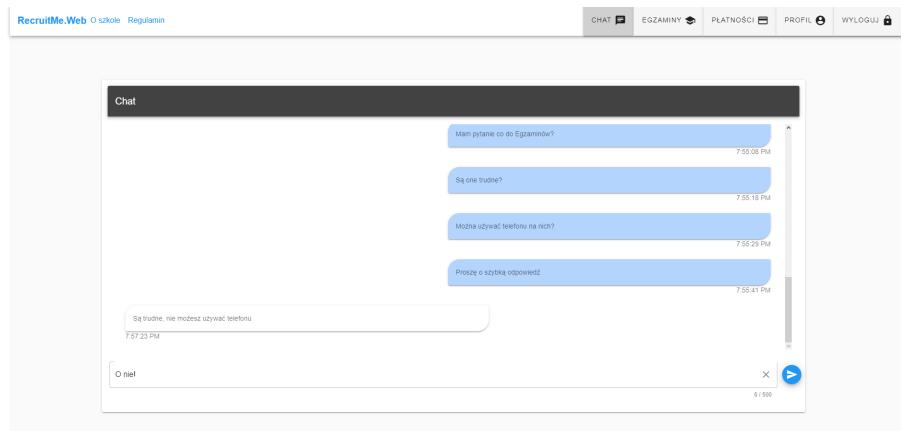
4.7. Chat

Funkcjonalność chatu umożliwia wygodną komunikację pomiędzy Administratorem i Kandydatem. Dany Użytkownik (albo Administrator) pisze wiadomość do Administratora (w przypadku, gdy konwersację rozpoczyna Administrator – do Kandydata), którą odbiorca może szybko odczytać i na nią odpowiedzieć.

Aplikacja webowa

Należy wybrać z paska na górze po prawej stronie przycisk *Chat*. Na ekranie wyświetlona zostanie konwersacja z Administratorem (lub z wybranym Kandydatem). Jeżeli wymienionych było wiele wiadomości, pobierze się jedynie ich część, a resztę należy pobrać, klikając w przycisk *Ściagnij więcej wiadomości*. Aby wysłać wiadomość, należy wpisać jej treść w pole tekstowe na dole strony oraz wcisnąć enter (lub kliknąć w przycisk z ikoną wyślij). Jeśli Użytkownik ma nieprzeczytane wiadomości, przy przycisku *Chat* w pasku na górze strony wyświetli się notyfikacja z ich liczbą.

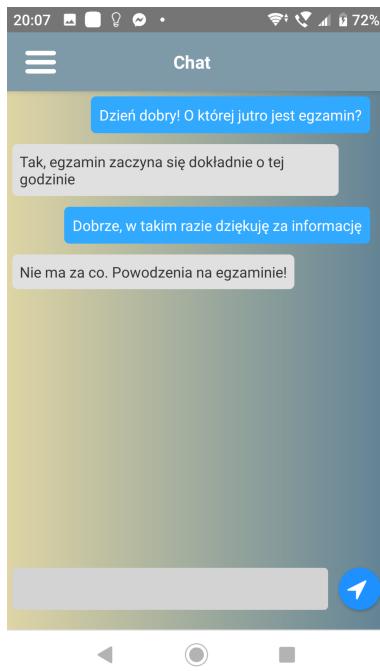
4. INSTRUKCJA UŻYTKOWANIA



Rysunek 4.19: Ekran chatu

Aplikacja mobilna

Należy kliknąć trzy białe paski w lewym górnym rogu albo wyciągnąć z lewej strony ekran menu i wybrać *Chat*. Po wybraniu tej opcji wyświetli się lista ostatnich piętnastu wiadomości wymienionych pomiędzy użytkownikami. By przeczytać więcej wiadomości, należy kliknąć przycisk *Pobierz więcej wiadomości* u góry listy. Aby wysłać wiadomość, należy najpierw wpisać jej treść w pole tekstowe na dole ekranu i wcisnąć niebieski przycisk po prawej stronie pola tekstowego. Jeśli Kandydat ma nieprzeczytaną wiadomość od Administratora, dostanie powiadomienie o nieprzeczytanych wiadomościach w pasku powiadomień telefonu.



Rysunek 4.20: Ekran chatu w aplikacji mobilnej

4.8. Panel Administratora

W celu sprawnego zarządzania Użytkownikami i Egzaminami został zbudowany panel administratora. Administrator, by skorzystać z tego panelu, musi się zalogować do systemu

4.8. PANEL ADMINISTRATORA

specjalnym loginem otrzymanym przy wdrożeniu Systemu.

W kolejnych podpunktach opisane są operacje, jakie Administrator może wykonać na Kandydatach, Egzaminach, kategoriach Egzaminów oraz nauczycielach w Systemie. Należy założyć, że jeżeli przy danej operacji nie jest napisane inaczej, to operację można wykonać dla każdej grupy wymienionej w poprzednim zdaniu.

The screenshot shows the main dashboard of the RecruitMe.Web application. At the top, there are links for 'RecruitMe.Web', 'O szkole', and 'Regulamin'. On the right, it says 'Zalogowany jako Admin' and has buttons for 'CHAT', 'ZARZĄDZAJ', and 'WYLOGUJ'. Below the header, there's a navigation bar with tabs: 'KANDYDACI', 'EGZAMINY', 'KATEGORIE EGZAMINÓW', and 'NAUCZYCIELE'. The main content area is titled 'Kandydaci' and displays a table with three rows of candidate data:

ID kandydata	Imię	Nazwisko	Email
abcabc000	Abc	Abc	abcabc@abc.abc
pionow000	Piotr	Nowak	p.nowak@gmail.com
jankow000	Jan	Kowalski	j.kowalski@aa.jk

At the bottom of the table, there are buttons for 'Rows per page' (set to 10), '1-3 of 3', and navigation arrows. The background of the main content area is light grey.

Rysunek 4.21: Główny ekran panelu administratora

4.8.1. Dodawanie danych do Systemu

Administrator może przez panel Administratora dodać nauczycieli, kategorie Egzaminów jak i same Egzaminy. Należy przejść do wybranej zakładki wyświetlonej na szarym pasku, a następnie należy kliknąć przycisk *Dodaj* znajdujący się przy nazwie danej grupy. Na koniec trzeba uzupełnić potrzebne dane i kliknąć *Zapisz*.

The screenshot shows a form for adding an exam term. At the top, there are links for 'RecruitMe.Web', 'O szkole', and 'Regulamin'. On the right, it says 'Zalogowany jako Admin' and has buttons for 'CHAT', 'ZARZĄDZAJ', and 'WYLOGUJ'. The main title is 'Dodaj termin egzaminu' with a 'WRÓĆ' button. The form fields are:

- Start Egzaminu
- Liczba osób
- Czas trwania
- Kategoria egzaminu (dropdown menu)

At the bottom is a large 'ZAPISZ' button.

Rysunek 4.22: Przykładowe dodawanie Egzaminu do Systemu

4.8.2. Edycja istniejących danych

Aby zmienić istniejące dane w Systemie należy najpierw przejść do wybranej zakładki wyświetlonej na szarym pasku. Potem trzeba kliknąć na dany rekord tabeli. Wyświetli się wtedy co najmniej jeden formularz z danymi, które można edytować. Na koniec edycji należy kliknąć przycisk *Zapisz*.

Administrator musi uważać, by przy zmianie danych **nie modyfikować** informacji, które uniemożliwiłyby zalogowanie się Użytkownikowi.

4.8.3. Usuwanie danych

Usuwanie danych przeprowadza się w tym samym ekranie co ich edycję. Gdy znajdziemy Użytkownika, kategorię, Egzamin lub nauczyciela, który ma być usunięty z Systemu, wystarczy kliknąć w ekranie edycji przycisk *Usuń*. Po udanym procesie następuje przekierowanie na stronę danej grupy.

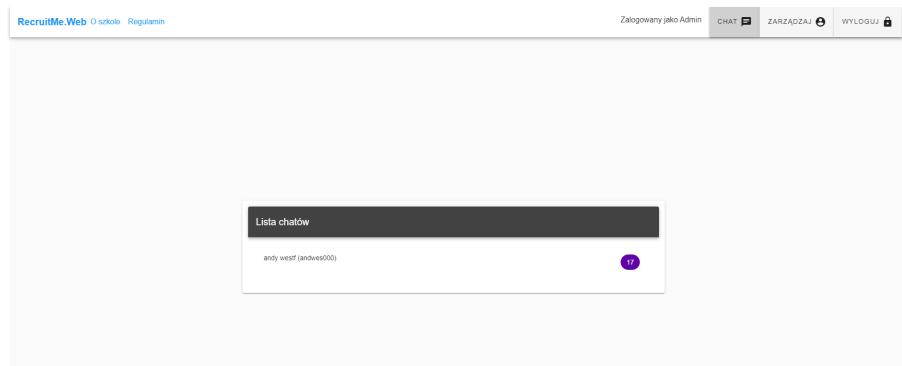
Rysunek 4.23: Ekran edycji Użytkownika

4.8.4. Drukowanie identyfikatora i chat

Wydrukowanie identyfikatora Kandydata i otworzenie okienka chatu z Użytkownikiem wykonuje się w ekranie edycji Użytkownika. Po kliknięciu *Kandydaci* z szarego paska następuje przejście do listy Użytkowników. Należy wybrać Użytkownika, a następnie wybrać odpowiedni z dwóch przycisków po prawej stronie: *Drukuj identyfikator* lub *Otwórz chat*.

Do chatu z danym Kandydatem można również przejść poprzez zakładkę Chat otwieraną przez przycisk *Chat* na górze strony. Wyświetlona jest tam lista konwersacji z Kandydatami, którzy kiedykolwiek napisali do Administratora, wraz z liczbą nieprzeczytanych przez Administratora wiadomości z danym Kandydatem (po prawej stronie). W przypadku, gdy żaden z Kandydatów nie napisze do Administratora, lista konwersacji będzie pusta. Kliknięcie na dany element listy konwersacji jest równoważne kliknięciu przycisku *Otwórz chat* w ekranie edycji Kandydata w panelu administratora.

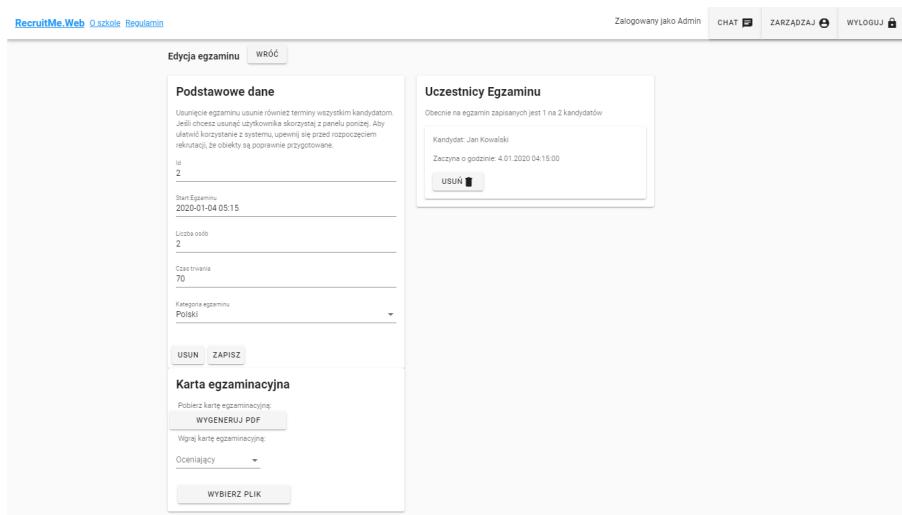
4.8. PANEL ADMINISTRATORA



Rysunek 4.24: Ekran listy konwersacji

4.8.5. Skanowanie ocen

Na stronie edycji danego Egzaminu istnieje możliwość wydrukowania karty egzaminacyjnej, za pomocą której można w prosty sposób wprowadzić jego wyniki do Systemu. Z uwagi na ograniczenie możliwości skanowania tylko jednej strony, ta funkcjonalność jest dostępna tylko dla Egzaminów, na które zapisanych jest nie więcej niż 15 uczestników, co powinno wystarczyć dla egzaminów ustnych (indywidualnych). Nauczyciel oceniający w trakcie Egzaminu zaznacza wynik Kandydata na karcie, zamalowując odpowiednie pole ciemnym długopisem (najlepiej czarnym). Zaznaczając wynik, nie można wyjść poza granice pola z punktami oraz na karcie nie może być żadnych obiektów (np.: przebarwień, notatek, rysunków) potencjalnie utrudniających automatyczne sczytanie karty. Wprowadzając kartę do systemu należy zadbać, aby czarna ramka była najbardziej zewnętrznym konturem na skanie karty. Podczas wybierania pliku zeskanowanej karty trzeba również podać oceniającego nauczyciela. Po czynności zaleca się sprawdzenie, czy wprowadzone oceny są zgodne z zaznaczonymi na karcie.



Rysunek 4.25: Ekran edycji egzaminu

4. INSTRUKCJA UŻYTKOWANIA

Recruit.Me Exam Sheet
Exam: Matematyka pisemny
Start: 14:10 01-01-2020
1. Jan Kowalski 1
<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>
2. Jan Kowalski 2
<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>
3. Jan Kowalski 4
<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
4. Jan Kowalski 3
<input checked="" type="radio"/> <input type="radio"/>
5. Jan Kowalski 5
<input type="radio"/> <input checked="" type="radio"/>
6. Jan Kowalski 7
<input checked="" type="radio"/> <input type="radio"/>
7. Jan Kowalski 6
<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
8. Andrzej Testowy 1
<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
9. Andrzej Testowy 2
<input type="radio"/> <input checked="" type="radio"/>
10. Andrzej Testowy 3
<input checked="" type="radio"/> <input type="radio"/>
11. Andrzej Testowy 4
<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>

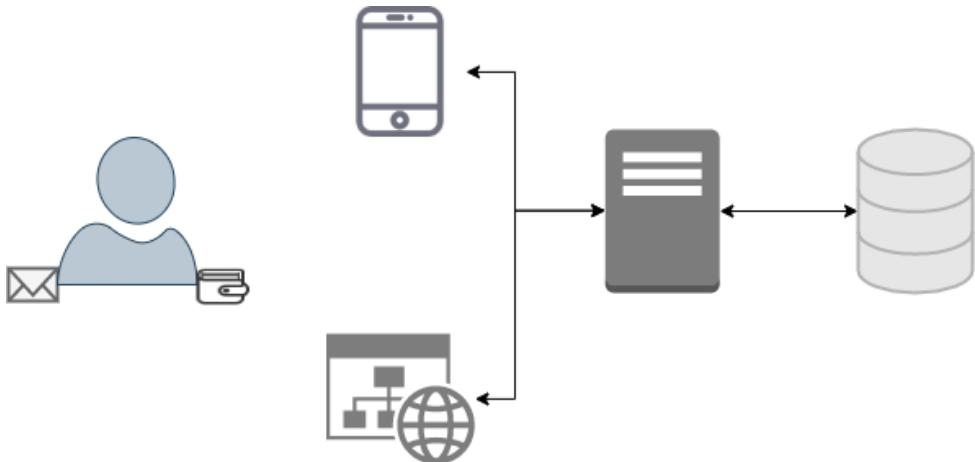


Rysunek 4.26: Przykładowa karta egzaminacyjna

5. Dokumentacja techniczna

5.1. Architektura

Architektura systemu jest klasycznym, 3-warstwowym układem dla prostych systemów informatycznych. Składa się z Aplikacji klienckich komunikujących się z Serwerem API, który z kolei dalej komunikuje się z bazą danych. Użytkownicy wchodzą w interakcję z Systemem również poprzez komunikację mailową oraz poprzez bramkę płatniczą.



Rysunek 5.1: Schemat architektury systemu

5.1.1. Baza danych

Jest odpowiedzialna za przechowywanie danych systemu. Znajdują się w niej główne obiekty systemu takie jak Użytkownicy, Kandydaci czy Egzaminy. Z dostępnych systemów relacyjnych baz danych wybrany został MySQL Community Server, ponieważ jest popularnym i bezpłatnym rozwiązaniem.

5.1.2. Serwer API

Jest głównym komponentem Systemu. Jest odpowiedzialny za logikę biznesową oraz nadzorowanie dostępu do danych w bazie oraz zdjęć lub dokumentów przesyłanych przez Użytkowników. Umożliwia on pobranie przez przeglądarkę internetową Aplikacji webowej. Za jego pośrednictwem dokonywane są płatności przez bramkę płatniczą oraz wysyłane wiadomości e-mail. Do implementacji Serwera API została wykorzystana technologia ASP.NET Core z uwagi na jej znajomość przez twórców Systemu oraz popularność.

5.1.3. Aplikacja webowa

Umożliwia dostęp do Systemu przez przeglądarkę internetową. Akcje Użytkownika są interpretowane przez aplikację i wywołują odpowiednie akcje w serwerze API. Aby dane prezentowane na stronie były takie same na wszystkich urządzeniach, aplikacja nie przechowuje żadnych danych – są one pobierane z Serwera API. Została zaimplementowana w popularnym framework'u Vue.js.

5.1.4. Aplikacja mobilna

Umożliwia dostęp do Systemu przez telefon z systemem Android. Na urządzeniach mobilnych można również uzyskać dostęp do Systemu poprzez przeglądarkę internetową. Ze względu na większy komfort użytkowania zaleca się jednak korzystanie Aplikacji dedykowanej dla urządzeń mobilnych. Akcje Użytkownika są interpretowane przez aplikację i wywołują odpowiednie akcje w Serwerze API. Została zaimplementowana w framework'u NativeScript z nakładką umożliwiającą tworzenie aplikacji tak jak we framework'u Vue.js.

5.2. Wzorce projektowe modułów

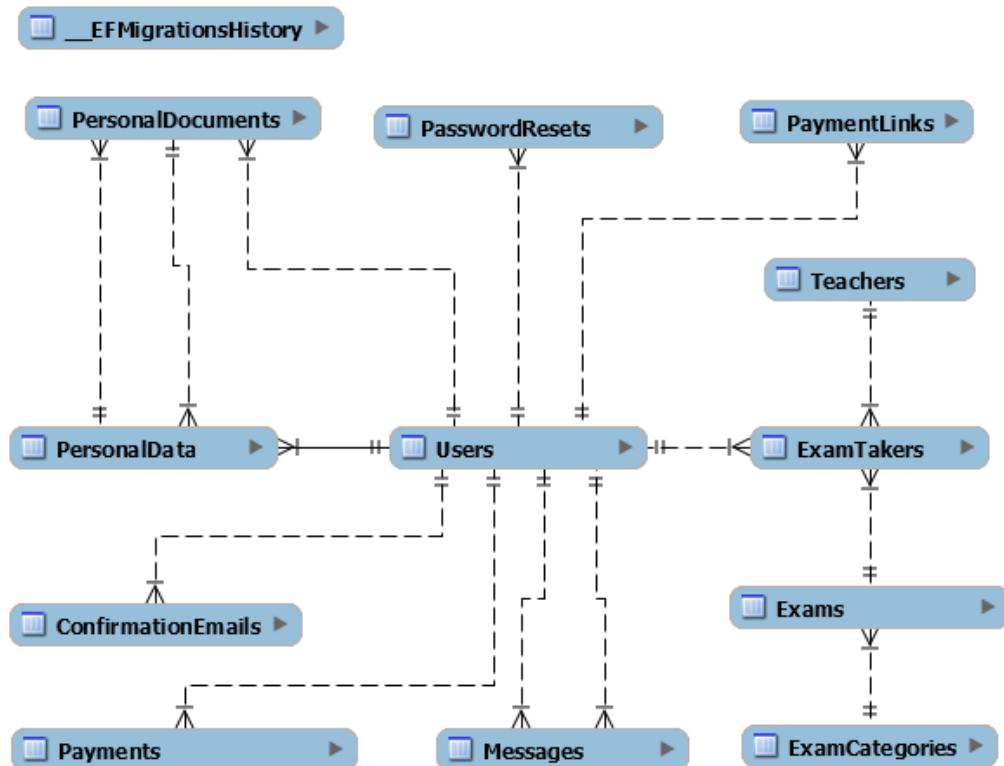
5.2.1. Baza danych

Baza została zaprojektowana zgodnie z podejściem *code-first* z wykorzystaniem narzędzia Entity Framework Core, połączonym z serwerem MySQL za pomocą dostawcy (ang. provider) *Pomelo.EntityFrameworkCore.MySql*.

Podejście *code-first* polega na projektowaniu bazy danych poprzez implementowanie obiektów bazodanowych najpierw w kodzie aplikacji, a następnie uaktualnianie bazy do postaci zgodnej z zaimplementowanymi klasami. Uaktualnienie dzieje się za pomocą mechanizmu *migracji* – wykrywane są dokonane zmiany w modelu zawartym w kodzie względem poprzedniej migracji, które następnie są tłumaczone na zmiany w bazie danych (np.: dodanie pola w klasie jest wykrywane i zapisywane w pliku migracyjnym jako dodanie kolumny do odpowiedniej tabeli). Następnie dzięki poleceniu *dotnet ef database update* generowany jest kod SQL, który uaktualnia bazę danych do postaci opisanej w ostatniej migracji. Pozwala to na wygodniejsze, szybsze i śledzone przez kontrolę wersji zmiany na bazie.

Pliki migracyjne systemu znajdują się w katalogu *Migrations* wewnątrz projektu RecruitMe.Web.

5.2. WZORCE PROJEKTOWE MODUŁÓW



Rysunek 5.2: Diagram schematu bazy

5.2.2. Logika biznesowa

CQRS

Logika biznesowa została zorganizowana według reguły rozdzielenia komend i zapytań (ang. *Command Query Responsibility Segregation*, CQRS). Zgodnie z nią akcje w systemie są podzielone na komendy – modyfikujące stan systemu i zwracające minimalną ilość danych (np. identyfikatory albo informację o powodzeniu lub błędzie) oraz zapytania, które zwracają dane. Reguła ta powoduje konieczność pisania większej ilości kodu, jednak stworzony kod jest łatwiejszy do utrzymania oraz testowania, ponieważ każda funkcjonalność jest zamknięta we własnej klasie. Dodatkowo w większych systemach umożliwia to optymalizację działania systemu poprzez powielenie bazy na repliki do odczytu, z których może korzystać część *Query* kodu.

Wstrzykiwanie zależności (ang. Dependency Injection)

Jeśli pomiędzy akcjami są jakieś zależności (jedna agreguje wiele innych lub działa na wyniku poprzedniej), zgodnie z wzorcem projektowym wstrzykiwania zależności (ang. *dependency injection*, DI) akcje nie są tworzone wewnątrz klasy, lecz przekazywane do jej wnętrza przez konstruktor. Zastosowanie takiego wzorca umożliwia jednostkowe testowanie kodu, ponieważ testując jedną metodę, możemy „zamockować” (stworzyć atrapę zwracającą z góry znany wynik) wszystkie zależności tej metody.

Programowanie do interfejsu (ang. Programming to interface)

Część klas potrzebnych w warstwie logiki, ze względu na „biblioteczny” charakter projektu (tzn. warstwa logiki nie jest pisana jako wykonywalna aplikacja), powinna być zaimplementowana poza projektem RecruitMe.Logic. W odpowiedzi na to zostały stworzone abstrakcyjne klasy i interfejsy, które są zaimplementowane w projekcie RecruitMe.Web i wstrzyknięte w miejsca użycia w projekcie logiki. Taką klasą jest na przykład *LocalFileStorage* implementujący m.in. interfejs *IFileRepository*. Projekt logiki powinien działać na abstrakcji zbioru plików i jedynie pobierać z niego pliki, natomiast projekt RecruitMe.Web powinien wiedzieć, czy jest uruchomiony na maszynie lokalnej, czy w chmurze i odpowiednio dostarczyć do logiki sposób pobierania plików. Taki schemat nazywany jest „programowaniem do interfejsu” (ang. *programming to interface*), który jest implementacją jednego z filarów programowania obiektowego – abstrakcji.

5.2.3. Serwer API

Zadaniem projektu Serwera API jest pośredniczenie pomiędzy Aplikacjami klienckimi a warstwą logiki oraz dostarczenie wyżej omówionych klas zależnych od środowiska wykonawczego do projektu logiki. Dodatkowo Serwer jest odpowiedzialny za uwierzytelnienie Użytkowników wykonyujących żądania do Systemu, czyli za tworzenie i validację tokenu JWT (ang. JSON Web Token). Jeśli wykonywana akcja jest dozwolona jedynie dla Administratora, Serwer również sprawdza, czy Użytkownik inicjujący akcję jest Administratorem.

Z tego powodu większość metod w projekcie RecruitMe.Web upraszcza się do wywołania pojedynczej metody z projektu RecruitMe.Logic, jednakże w niektórych przypadkach, aby przyśpieszyć aplikacje klienckie oraz zmniejszyć zużycie transferu danych, wołane są dwie metody: *Command* modyfikujący stan oraz, jeśli ten się powiedzie, *Query* zwracające nowy stan.

5.2.4. Aplikacja webowa

Aplikacja webowa napisana została we frameworku JavaScriptowym Vue.js, który wymusza wzorzec MVVM. Warstwy odpowiadają za:

- Model – plik api.gateway.ts wykonujący jedynie żądania do Serwera API, jednocześnie chowający kwestie połączenia sieciowego jak np. uwierzytelnianie,
- View – kod html/css,
- View-Model – kod TypeScript.

5.2.5. Aplikacja mobilna

Z racji, że Aplikacja mobilna jest zaimplementowana we frameworku NativeScript z nakładką umożliwiającą pisanie aplikacji w Vue.js, jest ona koncepcyjnie podobna do aplikacji webowej i również używa wzorca MVVM.

5.2.6. Testy

Projekt testowy został napisany przy użyciu technologii nUnit i jest podzielony na 3 części:

- UnitTests – testy jednostkowe sprawdzające działanie pojedynczych metod lub klas,

5.3. WARSTWY SYSTEMU

- Integration Tests – testy integracyjne sprawdzające bardziej globalne aspekty systemu np.: sprawdzenie konfiguracji kontenera wstrzykiwania zależności,
- Helpers – części wspólne dla testów lub metody pomocnicze pozwalające na czytelniejsze testy.

Projekt testowy jest bardziej szczegółowo opisany w rozdziale „Testy”.

5.3. Warstwy systemu

Zawarte w tytule pracy inżynierskiej warstwy systemu były pierwszym pomysłem implementacji systemu. Poza podziałem aplikacji na funkcjonalności, podzielona jest ona również ze względu na poziomy abstrakcji i dostęp do danych. Pomimo że model ewoluował w trakcie rozwoju projektu, jego końcowa wersja w dużym stopniu spełnia pierwotne założenia. Model warstw nie pokrywa się w pełni z modelem architektury – dzieje się tak, ponieważ połączenie bazy danych i Serwera API jest obsłużone przez narzędzie Entity Framework Core.

1. warstwa widoku – Vuetify, html, css:

Odpowiada za wygląd aplikacji i jest prezentowana bezpośrednio Użytkownikowi. Jej logika upraszcza się do minimum (np.: podstawowe pętle lub formatowanie dat).

2. warstwa widoku-modelu – komponenty i serwisy w kodzie Typescript:

Odpowiada za tłumaczenie akcji Użytkownika (np.: kliknięcie w przycisk, wypełnienie formularza na stronie) na odpowiednie wywołanie funkcji wyższej warstwy oraz aktualnienie stanu aplikacji po skończeniu się akcji (w szczególności ukazanie powodu niepowodzenie akcji).

3. warstwa modelu – plik api.gateway.ts:

Jest bramką pomiędzy Aplikacją kliencką a Serwerem API – wszystkie żądania do serwera przechodzą przez ten plik. Umożliwia to konfigurowanie czynności przed lub po wykonaniu żądania oraz łatwe konfigurowanie wykonywanych żądań np.: ustawianie niestandardowych nagłówków wiadomości. Zadaniem tej klasy jest również ukrycie połączenia HTTP i szczegółów technicznych z nim związanych np. uwierzytelnianie.

4. warstwa kontrolerów – kontrolery ASP.NET Core:

Odpowiada za uwierzytelnienie wykonującego żądanie, sprawdzenie jego upoważnień, przygotowanie parametrów oraz wywołanie odpowiedniej metody z projektu logiki. Jeśli aplikacja kliencka oczekuje w odpowiedzi nowego stanu danych, po wykonaniu operacji typu *Command* wykonywana jest druga typu *Query*.

5. warstwa validacji – walidatory uruchamiane z abstrakcyjnej klasy *BaseOperation*:

Aby możliwie uprościć implementację kolejnej warstwy, całą możliwą walidację danych należy dokonać poprzez wskazanie klasy implementującej *BaseValidator*. Walidacja odbywa się za pomocą paczki „FluentValidation” [3]. Jeśli validator stwierdzi, że parametry są niewłaściwe, rzucony zostanie odpowiedni wyjątek, który następnie jest przekształcony przez mechanizm filtrów [4] do błędu HTTP o kodzie 400.

6. warstwa logiki – implementacje klasy *BaseOperation*:

Implementacja logiki biznesowej zmieniającej stan danych systemu albo zwracającej odpowiednio sformatowane informacje.

7. warstwa danych – klasy *ApplicationContext* oraz *LocalFileStorage*:

Klasy zaimplementowane w folderze Services/Data w projekcie RecruitMe.Web są wstrzykiwane do warstwy logiki. Warstwa ta ma bezpośredni dostęp do bazy (*ApplicationContext* dziedziczy po klasach bazowych z paczki Entity Framework Core) lub do plików na dysku.

5.4. Opis modułów

5.4.1. Logika biznesowa

Moduł logiki biznesowej znajduje się w projekcie *RecruitMe.Logic*. Został podzielony na części (domeny) według wykonywanych zadań. Implementacja każdej domeny jest umieszczona w osobnej przestrzeni nazw:

- **RecruitMe.Logic.Data** – zawiera abstrakcyjne klasy i interfejsy służące do przechowywania danych np. pośredniczący z bazą danych *DbContext* lub do przechowywania plików. W tej domenie znajdują się również klasy encji przechowywanych w bazie danych, które definiują jej schemat dołączony wyżej.
- **RecruitMe.Logic.Logging** – zawiera klasy służące do logowania. Wybrana klasa jest wstrzykiwana do klas, które zapisują logi systemu, aby ujednolicić proces zbierania logów i móc go łatwo konfigurować.
- **RecruitMe.Logic.Operations** – zawiera operacje definiujące logikę systemu.
- **RecruitMe.Logic.Utilities** – zawiera ogólne obiekty takie jak parametry stronicowania. W tej przestrzeni nazw znajdują się również metody rozszerzające kontener wstrzykiwania zależności.
- **RecruitMe.Logic.Configuration** – zawiera klasy konfiguracji sczytywanej z plików oraz wartości stałe w kodzie.

5.4.2. Testy

Znajdują się w projekcie *RecruitMe.Logic.Tests* i sprawdzają poprawność podstawowych funkcji systemu. Opisane są dokładniej w rozdziale „Testy”.

5.4.3. API

Moduł API odpowiada za udostępnienie komunikacji pomiędzy modułem Aplikacji klienckiej a modułem logiki biznesowej. Znajduje się on w folderze *RecruitMe.Web/*. Struktura aplikacji jest typowa dla projektów w technologii ASP.NET Core używających Entity Framework Core [10]. Nietypowymi folderami są natomiast:

- Scripts – zawierający skrypt napisany w języku Python przetwarzający karty egzaminacyjne,
- ssl – folder zawierający certyfikat SSL. Opisany dokładniej w rozdziale „Instrukcja Instalacji”,

5.4. OPIS MODUŁÓW

- Configuration – zawiera konfigurację paczki Identity Server, ścieżek do plików i pomocnicze metody używane w klasie StartUp,
- Services – zawiera klasy *ApplicationContext*, *LocalFileStorage* oraz klasy używane przez Identity Server, które np. sprawdzają, czy dany Użytkownik istnieje.

5.4.4. Aplikacja webowa

Kod aplikacji webowej można znaleźć w folderze *RecruitMe.WebApp/ClientApp/*. Ten folder jest podzielony na logiczne części ułatwiające nawigację po module. Poza folderami znajdziemy plik *boot.ts*, w którym zostało opisane mapowanie ścieżek adresu do odpowiedniego komponentu aplikacji oraz ogólna konfiguracja startowa aplikacji.

Komponenty

Folder *components/* zawiera komponenty renderowane do widoków w aplikacji. Są one podzielone na foldery względem ścieżki, po jakiej można się do nich dostać (przykładowo: komponent do logowania znajdziemy w folderze *components/account/login*).

Każdy komponent jest podzielony na dwa lub trzy pliki:

- *.ts – model i logika komponentu,
- *.vue.html – widok komponentu,
- *.css – stylowanie widoku (w większości zastąpione przez użycie gotowych komponentów z paczki Vuetify).

Modele

W folderze *models/* znajdują się definicje modeli danych używanych w komunikacji z Serwerem API. Każdy plik o końcówce *.model.ts* zawiera co najmniej jeden interfejs, który specyfikuje format, w jakim powinniśmy wysłać zapytanie lub otrzymać odpowiedź.

Serwisy

Folder *services/* zawiera pliki o końcówce *.service.ts*, które zawierają logikę aplikacji, upraszczając w ten sposób pliki komponentów. W części aplikacji poświęconej Kandydatowi większość wykonanych żądań odbywa się poprzez odpowiedni serwis, dzięki czemu komponent nie odwołuje się bezpośrednio do pliku *apiGateway.ts*.

5.4.5. Aplikacja mobilna

Moduł aplikacji mobilnej swoją strukturą odwzorowuje aplikację webową. W głównym folderze projektu aplikacji mobilnej (*RecruitMe.MobileApp/app/*) mamy uporządkowaną strukturę plików i folderów odpowiadającą poszczególnym modułom zaimplementowanych w aplikacji mobilnej.

Widok-model

W folderze *components/* znajdują się wszystkie pliki typu widok-model używane w aplikacji. Znajdziemy tam pliki z rozszerzeniem *.vue*, które są rozmieszczone w odpowiednich podfolderach oznaczających, w jakich operacjach pojawiają się poszczególne widoki (np.: plik *Login.vue* znajduje się w podfolderze *account/*, gdyż akcja logowania jest związana z operacjami na koncie Użytkownika).

Pliki zawierają trzy główne sekcje rozdzielone znacznikami:

- *<template>...</template>* – rozmieszczenie komponentów na ekranie telefonu,
- *<script>...</script>* – dane i logika widoku-modelu,
- *<style>...</style>* – stylowanie komponentów.

Modele

W folderze *models/* znajdują się definicje modeli danych używane w komunikacji z Serwerem API. Każdy plik z rozszerzeniem *.ts* zawiera co najmniej jeden interfejs, który specyfikuje format, w jakim powinniśmy wysłać zapytanie i otrzymać odpowiedź.

Serwisy

Folder *services/* zawiera pliki o rozszerzeniu *.ts*, które zawierają logikę łączenia się do API i zarządzania otrzymanymi z API danymi. Pliki są pogrupowane według kategorii, np.: serwis Użytkownika i serwis przechowywania danych logowania znajdują się w jednym folderze.

Magazyn danych

W folderze *RecruitMe.MobileApp/* znajduje się plik o nazwie *store.ts* będący wynikiem zastosowania biblioteki Vuex, który przechowuje dane pomiędzy widokami Aplikacji. W Aplikacji wykorzystywany jest m.in.: do utrzymywania tokenu zalogowanego Użytkownika oraz niektórych danych profilu Kandydata.

5.5. Implementacja wybranych aspektów

5.5.1. Konfiguracja

Istnieje wiele sposobów na implementację konfiguracji w projektach informatycznych i w zależności od konkretnego przypadku różne sposoby są poprawne. Minimalistycznym podejściem jest wyjęcie stałych używanych w systemie do pojedynczej klasy dla danego typu konfiguracji. Ta metoda została zastosowana do m.in. treści wiadomości e-mail, ścieżek do plików na dysku i ustawień paczki Identity Server. Natomiast w przypadku, kiedy wartość jest zależna od środowiska uruchamiania Aplikacji lub będzie z niej korzystać klient, zostało użyte automatyczne wstrzykiwanie konfiguracji z pliku *appsettings.json* [5]. Zdefiniowane zostały klasy *BusinessConfiguration* oraz *PaymentConfiguration*, których nazwy odpowiadają odpowiednim sekcjom w pliku konfiguracyjnym, a nazwy właściwości odpowiadają kluczom w tych sekcjach. Zaimplementowana została generyczna funkcja *AddSingletonConfiguration*, która mechanizmem refleksji sczytuje nazwy właściwości klasy, następnie pobiera dla nich wartości z konfiguracji i tworzy silnie typowany

5.5. IMPLEMENTACJA WYBRANYCH ASPEKTÓW

obiekt danej klasy wypełniony danymi, który jest dodawany do kontenera z okresem życia „Singleton” [6]. W ten sposób w dowolnym miejscu w systemie można otrzymać ten obiekt i pobrać z niego wartości.

5.5.2. Wstrzykiwanie zależności

Z racji wybrania reguły CQRS do implementacji Systemu wstrzykiwanie każdej klasy ręcznie byłoby męczące i trudne w utrzymaniu. Problem ten został rozwiązany poprzez automatyczne wstrzykiwanie wszystkich klas, które implementują pusty interfejs *IAutoComponent*. Wstrzykiwanie odbywa się poprzez wywołanie metody *RegisterAutoComponents*, która iteruje po wszystkich typach w bibliotece RecruitMe.Logic, wybierając te typy, które implementują wspomniany interfejs i dodając je do kontenera z okresem życia „Transient” [6].

5.5.3. Abstrakcja operacji

Działanie całego Systemu opiera się głównie o „operacje” – implementacje klasy *BaseOperation*. Ich implementacja definiuje prawie całą logikę aplikacji. Przykładową operacją jest *LoginUserQuery* przedstawione na Listingu 5.1.

```
namespace RecruitMe.Logic.Operations.Account.Login
{
    public class LoginUserQuery
        : BaseAsyncOperation<User, LoginDto, LoginRequestValidator>
    {
        private readonly PasswordHasher _passwordHasher;

        public LoginUserQuery(ILogger logger,
            LoginRequestValidator validator,
            BaseDbContext dbContext,
            PasswordHasher passwordHasher)
            : base(logger, validator, dbContext)
        {
            _passwordHasher = passwordHasher;
        }
        protected override async Task<User> DoExecute(LoginDto request)
        {
            User user = await _dbContext.Users
                .SingleAsync(u => u.CandidateId == request.CandidateId);
            var result = _passwordHasher
                .VerifyPassword(user, request.Password);
            if (result)
            {
                return user;
            }
            throw new UnauthorizedAccessException();
        }
    }
}
```

Listing 5.1: Przykładowa operacja: LoginUserQuery

Nazwa klasy informuje o tym, gdzie klasa jest używana, czyli przy zalogowaniu Użytkownika do Systemu. Dalej w definicji klasy widać również, że dziedziczy po bazowej operacji pośrednio poprzez generyczną *BaseAsyncOperation*, gdzie typy w definicji oznaczają następująco: typ zwracany, parametr operacji oraz validator. Poniżej w konstruktorze widać proces wstrzykiwania zależności, w szczególności pomocniczą klasę haszującą hasło Użytkownika. Zdefiniowana metoda jest metodą o zakresie dostępu *protected* i jest wywoływana z klasy bazowej dopiero po udaniu się validacji. Ostatecznie zwracany jest obiekt klasy *User* lub, jeśli hasło się nie zgadza, rzucany jest wyjątek.

Przy użyciu takiego schematu nawet skomplikowane operacje są stosunkowo małe i czytelne.

5.5.4. Integracja z Dotpay

Bramka płatnicza Dotpay wystawia interfejs sieciowy, dzięki któremu można w prosty sposób zintegrować cały proces płatności. Wymagane jest stworzenie konta w serwisie, aby otrzymać klucze dostępu do API.

Cały proces płatności w systemie rekrutacji przebiega w następujący sposób:

1. System tworzy unikalny link płatności z danymi w systemie Dotpay.
2. System przekierowuje Użytkownika na wygenerowany link.
3. Użytkownik dokonuje płatności w systemie Dotpay.
4. Po dokonaniu płatności system Dotpay wykonuje zapytanie do Systemu rekrutacji o powodzeniu płatności.
5. System po validacji otrzymanych danych zapisuje, że płatność została uiszczena.

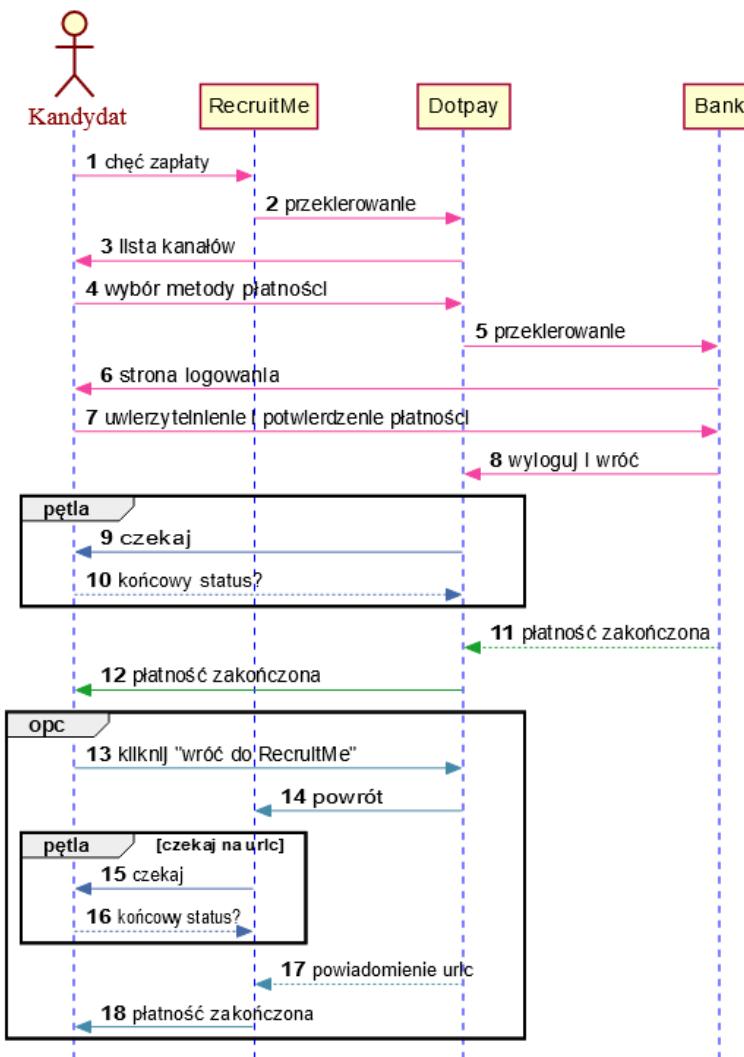
Każde zapytanie do i z systemu Dotpay przebiega za pomocą REST API. Przy tworzeniu linku płatniczego wysyłany jest token z kluczami dostępu oraz dane płatności (np.: Kwota rekrutacyjna, waluta). W ostatnim kroku wykonywane jest jeszcze żądanie do systemu Dotpay, by usunąć wygenerowany poprzednio link płatności.

Do całego procesu płatności napisane zostały testy, które zapewniają, że cały proces będzie przechodził bez wad.

Szczegółowy diagram procesu dokonania płatności znajduje się na Rysunku 5.3, gdzie kolejne akcje oznaczają:

1. Kandydat wyraża chęć uiszczenia zapłaty poprzez kliknięcie przycisku *Przejdz do płatności*.
2. System rekrutacji wysyła żądanie do systemu Dotpay o wygenerowanie linku płatniczego oraz przekierowuje Kandydata na stronę Dotpay.
3. Dotpay wysyła listę dostępnych kanałów.
4. Kandydat wybiera metodę płatności.
5. Dotpay przekierowuje płatność na stronę banku.
6. Bank wyświetla stronę logowania.
7. Kandydat loguje się do banku i potwierdza płatność.

5.5. IMPLEMENTACJA WYBRANYCH ASPEKTÓW



Rysunek 5.3: Diagram integracji z serwisem Dotpay [7]

8. Bank wylogowuje Kandydata ze strony i wraca na stronę systemu Dotpay.
9. Czekanie na odpowiedź z Banku.
10. Dotpay wysyła okresowe zapytania o status płatności.
11. Bank wysyła informację o skończonej płatności.
12. Dotpay powiadamia Kandydata o zakończonej płatności.
13. Kandydat kliką przycisk *Wróć do RecruitMe*.
14. Dotpay przekierowuje Kandydata do RecruitMe.
15. Czekanie na potwierdzenie z systemu Dotpay.
16. System wysyła okresowe żądania do serwera, by odpytać, czy płatność jest zakończona.

17. Dotpay wysyła potwierdzenie do Systemu rekrutacji na adres z linku płatniczego.

18. System rekrutacji powiadamia Kandydata o zakończonej płatności.

5.5.5. Generowanie identyfikatora Kandydata

Do wygenerowania identyfikatora Kandydata używana jest klasa *GetCandidateIdCardQuery*, która jako parametr przyjmuje ID Kandydata, dla którego Administrator chce wydrukować kartę. Do generowania dokumentu PDF używana jest paczka *PdfSharpCore*. Na stronie rysujemy kolejno: zewnętrzną ramkę, zdjęcie profilowe, imię i nazwisko Kandydata oraz kod QR z linkiem prowadzącym do profilu Kandydata w panelu Administratora. Kod QR jest generowany jako zdjęcie za pomocą paczki *QRCode*.

5.5.6. Generowanie i przetwarzanie kart egzaminacyjnych

Za generowanie kart egzaminacyjnych odpowiada klasa *GetExaminationSheetQuery*, przyjmująca jako parametr ID egzaminu. W pierwszej kolejności sprawdzane jest, czy egzamin ma poniżej 15 uczestników, ponieważ wygenerowany tekst musi zmieścić się na pojedynczej stronie. Następnie generowany jest dokument PDF z nagłówkiem „kółkami” z punktami dla każdego Kandydata oraz kodem QR z dodatkowymi danymi na potrzeby przetwarzania.

Odpowiadającą za to klasą jest *LoadExaminationSheetCommand*, jednak większość kodu jest zawarta w zewnętrznym skrypcie napisanym w Pythonie, który znajduje się w projekcie *RecruitMe.Web* w folderze *Scripts*. Otrzymane w parametrze zdjęcie jest zapisywane w folderze skryptu na czas przetwarzania. Z tego powodu ważne jest, aby tylko jedno zdjęcie było w danym momencie przetwarzane, więc został użyty blok *lock*. Skrypt jest uruchamiany jako nowy proces, któremu dajemy 4 sekundy na przetworzenie zdjęcia. Używana jest biblioteka OpenCV do przetwarzania bazującego na konturach w zdjęciu. Na początku zdjęcie jest poddawane wstępnej obróbce m.in.: zmieniane jest do formatu czarno-białego i lekko rozmywane. Następnie dokument jest normalizowany do zewnętrznej czarnej ramki na kartce. Odbywa się to poprzez znalezienie najbardziej zewnętrznego, prostokątnego konturu, przetransformowanie oraz przycięcie zdjęcia do przyjętych rozmiarów. Następnie wykryte są kontury na zdjęciu, które są filtrowane po ich kształcie, pozycji i wielkości, aby znaleźć „kółka” z punktami. Otrzymana lista jest iterowana, aby znaleźć najciemniejsze „kółko” w danym rzędzie. Wynik jest wypisywany na konsolę, która jest przekierowana do aplikacji serwera. Odczytany wynik jest następnie dopasowany do identyfikatorów scztanych z kodu QR na karcie egzaminacyjnej. Mając i ID Kandydata, i punkty z egzaminu, można te dane zapisać w bazie danych.

Algorytm bazuje na prostszej wersji opisanej w artykule Adriana Rosebrock'a [11].

5.5.7. Wysyłanie wiadomości e-mail

Do wysyłania wiadomości e-mail służą obiekty znajdujące się w przestrzeni nazw *RecruitMe.Logic.Operations.Email*. Do wysyłania wiadomości został użyty protokół SMTP z powodu jego prostoty oraz wsparcia u większości dostawców poczty elektronicznej.

5.5.8. Docker

Problemy z instalacją serwera MySQL były motywacją do przeniesienia Systemu na środowisko Dockera. Użycie Dockera zapewnia Systemowi poprawne środowisko wykonawcze, ponieważ

5.5. IMPLEMENTACJA WYBRANYCH ASPEKTÓW

w procesie tworzenia obrazu instalowane są na nim potrzebne narzędzia (np. pakiety języka Python). Przeniesienie Systemu nie było problemem, ponieważ platforma ASP.NET Core i użyte w niej paczki mają bardzo mało zależności od systemu operacyjnego. Proces komplikacji jest opisany w pliku *Dockerfile*:

```
FROM microsoft/dotnet:2.2-sdk AS builder
COPY . /src
WORKDIR /src
RUN apt-get update && apt-get install -y curl \
    && curl -sL https://deb.nodesource.com/setup_9.x | bash - \
    && apt-get install -y nodejs \
    && curl -L https://www.npmjs.com/install.sh | sh \
    && apt-get install -y libpng-dev \
    && apt-get install -y build-essential
RUN npm install -g node-gyp
RUN dotnet publish -c Release -o /recruitme RecruitMe.Web

FROM microsoft/dotnet:2.2-sdk AS runtime
COPY --from=builder /recruitme .
RUN apt-get update \
    && apt-get install -y libsm6 libxext6 libxrender-dev libgdiplus \
    && apt-get install -y python3-pip python3-dev \
    && cd /usr/local/bin \
    && ln -s /usr/bin/python3 python \
    && pip3 install --upgrade pip \
    && pip install imutils \
    && pip install numpy \
    && pip install opencv-python \
    && pip install scipy
EXPOSE 443
EXPOSE 80
ENV ASPNETCORE_ENVIRONMENT Release
ENTRYPOINT [ "dotnet", "RecruitMe.Web.dll" ]
```

Listing 5.2: Zawartość pliku Dockerfile

Proces komplikacji składa się z dwóch etapów: budowania oraz konfiguracji środowiska wykonańczego. W pierwszym instalowane są narzędzia potrzebne jedynie w procesie komplikacji jak np. npm. W drugim natomiast pobierane są np.: biblioteki potrzebne w skrypcie Pythonowym oraz zależności bibliotek przetwarzających obraz.

Użycie Dockera w Systemie zostało dokładniej opisane w „Instrukcji Instalacji”.

5.5.9. Identity Server

Do uwierzytelniania Użytkowników w Systemie został użyty model tokenów JWT, który jest obsługiwany przez paczkę *Identity Server 4* (IS). Dodanie jej do systemu odbywa się w klasie *Startup* poprzez wywołanie metody *services.AddIdentityServer* oraz następujących potem metod

konfiguracyjnych, które m.in.: wskazują jakim certyfikatem podpisywać token, jakie zasoby są w Systemie oraz jakich serwisów używać do sprawdzania Użytkowników. Na potrzeby IS zostały dodane w folderze Services: *CustomResourceOwnerPasswordValidator*, który sprawdza czy login i hasło są poprawne oraz *CustomProfileService*, który definiuje jakie dane mają być zapisane w tokenie. Szczegóły konfiguracji paczki zostały zapisane w klasie *ISConfig*.

5.6. Opis UI/UX

5.6.1. Aplikacja webowa

Aplikacja webowa korzysta z gotowych komponentów z biblioteki Vuetify, które gwarantują przejrzysty i intuicyjny layout strony. Dodatkowo dzięki użyciu biblioteki Bootstrap strona będzie responsywna, aby wyglądała dobrze na dużych i na małych ekranach. W razie potrzeby niektóre elementy będą dopasowywane do stylu systemu za pomocą css.

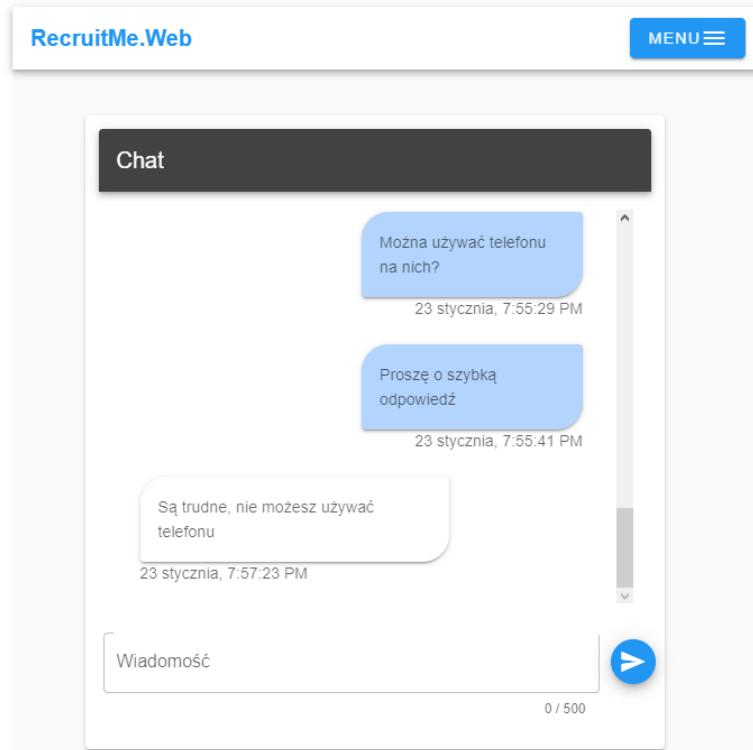
The screenshot shows a registration form titled "Rejestracja". The fields include:

- Imię: Kaja
- Nazwisko: Kot
- Nie mam numeru PESEL
- Data urodzenia: 2005-05-11
- Adres email: awestfa@gmail.com
- Haseł:
- Powtórz hasło:

The "ZAREJESTRUJ SIĘ" button is at the bottom left. Navigation links "RecruitMe.Web", "O szkole", and "Regulamin" are at the top left, and "ZAREJESTRUJ SIĘ" and "ZALOGUJ" are at the top right.

Rysunek 5.4: Przykładowy wygląd strony rejestracji

5.6. OPIS UI/UX



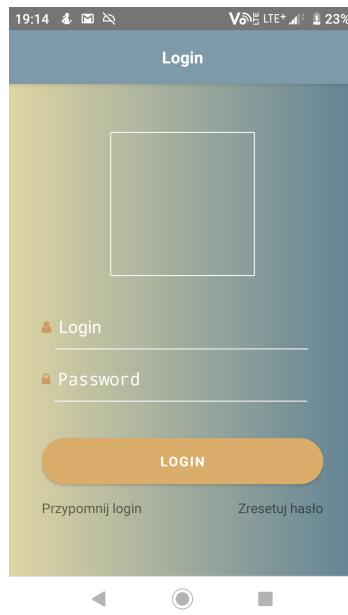
Rysunek 5.5: Przykładowy wygląd strony na mniejszym ekranie

5.6.2. Aplikacja mobilna

Aplikacja mobilna z racji implementacji wybranego frameworka nie jest w stanie korzystać z tak szerokiej gamy komponentów i bibliotek co aplikacja webowa. Istnieje duże ograniczenie w dostosowywaniu kontrolek do własnej koncepcji spowodowane niezaimplementowaniem możliwości zmiany danego parametru zarówno przez twórców systemu Android jak i frameworka NativeScript. Z tego powodu Aplikacja mobilna posługuje się dużą ilością własnych stylów css oraz znaczącą liczbą małych bibliotek rozszerzających wygląd i funkcjonalność brakujących części widoku.

Aplikację mobilną można podzielić na dwie części – widoki przed zalogowaniem i po zalogowaniu. Na starcie, przed zalogowaniem Użytkownik widzi, duże logo aplikacji i pod spodem dwa przyciski prowadzące do ekranu logowania oraz ekranu rejestracji. W ekranie logowania Użytkownik oprócz podania danych do autoryzacji może przypomnieć sobie login i/lub hasło. W ekranie rejestracji widoczny jest formularz z polami do wypełnienia oraz przycisk do ukończenia procesu rejestracji.

Po zalogowaniu Użytkownik jest przeniesiony na ekran Kandydata, na którym znajdzie informację o statusie swojej rekrutacji. Z tego widoku może łatwo przenieść się do innych funkcjonalności aplikacji za pomocą paska nawigacji, wyciąganego za pomocą przesunięcia palca z lewej do prawej lub poprzez kliknięcie ikony w lewym górnym rogu ekranu. Jeżeli Kandydat nie wpisze swoich danych dodatkowych, na większości ekranów aplikacji mobilnej po zalogowaniu będzie widoczny czerwony pasek o ich braku.



Rysunek 5.6: Przykładowy wygląd strony logowania w aplikacji mobilnej

5.7. Komunikacja

Komunikacja pomiędzy aplikacjami klienckimi a Serwerem API odbywa się poprzez protokół HTTPS. Połączenie musi być bezpieczne (zaszyfrowane), ponieważ przesyłane są dane osobowe Użytkowników, ale również ponieważ protokół JWT w pełni polega na ukryciu tokenów, tzn. jeśli ktoś otrzyma dostęp do tokena innej osoby, może bez możliwości wykrycia wykonywać akcje w jej imieniu.

Połączenie pomiędzy serwerem API a bazą danych obsługiwane jest przez Entity Framework Core. Używa on połączenia sprecyzowanego w „parametrach połączenia” (ang. connection string), gdzie możliwe jest skonfigurowanie m.in. adresu bazy czy używanego użytkownika.

5.8. Interfejsy zewnętrzne

5.8.1. SMTP

Protokół SMTP jest używany do wysyłania wiadomości e-mail z Systemu. Jest to stary, prosty i sprawdzony protokół oparty na komunikacji TCP, implementowany przez większość (o ile nie wszystkich) dostawców poczty internetowej.

5.8.2. Integracja Dotpay

Do przetwarzania płatności w Systemie została użyta bramka płatnicza Dotpay. Wyżej opisano techniczną implementację tego rozwiązania po stronie Systemu. Sama integracja nie implementuje żadnego ogólnodostępnego standardu, jednak interfejs bramki jest dobrze udokumentowany wraz z przykładami [7].

5.9. Wybór technologii

5.9.1. Języki programowania

Do implementacji systemu zostały wykorzystane następujące języki programowania:

5.9. WYBÓR TECHNOLOGII

- C# (Serwer API),
- TypeScript/JavaScript (Aplikacje klienckie),
- SQL (język do komunikowania się z bazą danych, głównie automatycznie generowany przez wykorzystanie Entity Framework Core),
- Python (skrypt przetwarzający karty egzaminacyjne).

5.9.2. Użyte biblioteki

Aplikacja mobilna

Głównym frameworkiem pozwalającym na budowę systemu na telefony z systemem Android jest NativeScript. W Aplikacji mobilnej jest wykorzystywany jego plugin nazwany NativeScript-Vue, pozwalający na pisanie aplikacji mobilnej w sposób bardzo podobny do pisania aplikacji webowej za pomocą framework'a Vue.js.

Innymi bibliotekami użytymi w aplikacji mobilnej są:

- axios – wykonywanie zapytań HTTP,
- nativescript-background-http – pobieranie zdjęć z serwera,
- nativescript-camera – robienie zdjęć,
- nativescript-feedback – wyświetlanie powiadomień w aplikacji,
- nativescript-imagepicker – wybieranie zdjęcia z galerii,
- nativescript-local-notifications – pokazywanie powiadomień o przychodzących wiadomościach,
- nativescript-ui-sidedrawer – panel boczny aplikacji,
- nativescript-urlhandler – wykrywanie powrotu do aplikacji mobilnej z płatności,
- rxjs – śledzenie zmian komponentów po zalogowaniu,
- vue-class-component, vue-property-decorator – poprawna obsługa TypeScriptu dla NativeScripta,
- vuex – przechowywanie informacji zalogowanego Użytkownika w trakcie działania aplikacji.

Aplikacja webowa

Analogicznie do Aplikacji mobilnej głównym frameworkiem użyтыm w Aplikacji webowej jest Vue.js. Dodatkowe biblioteki to:

- axios – komunikacja sieciowa,
- file-saver – zapisanie pliku,
- material-design-icons-iconfont – ikony,
- vuetify – wspomniana wcześniej paczka z komponentami widoku,

- vuetimepicker – formatka wybierania daty,
- inne zainportowane poprzez zależności lub udział w procesie ładowania aplikacji: inversify, node-gyp, ajv, popper.js, ts-loader.

Biblioteki wymagane podczas komplikacji i rozwoju Aplikacji to: aspnet-webpack, awesome-typescript-loader, bootstrap, copy-webpack-plugin, css-loader, deepmerge, eslint, eslint-config-standard, eslint-friendly-formatter, eslint-loader, eslint-plugin-html, eslint-plugin-promise, eslint-plugin-standard, extract-text-webpack-plugin, fibers, file-loader, jquery, sass, sass-loader, style-loader, typescript, url-loader, vue-loader, vue-property-decorator, vue-style-loader, vue-template-compiler, webpack, webpack-bundle-analyzer, webpack-cli, webpack-dev-middleware, webpack-hot-middleware, webpack-merge.

Serwer API i logika

Do zaimplementowania logiki Systemu oraz API wykorzystane zostały następujące paczki:

- ASP.NET Core – biblioteka odpowiadająca za działanie serwera,
- Pomelo.EntityFrameworkCore.MySQL – do mapowania obiektów z bazy danych na klasy w logice i na odwrót,
- FluentValidation – silnie typowane walidacje dla modeli,
- IdentityServer4 – OpenID Connect i OAuth 2.0 framework dla ASP.NET Core,
- PdfSharpCore, QRCode, ZXing – biblioteki służące do generowania i czytania dokumentów,
- liczne pomocnicze biblioteki ze środowiska .NET Core.

Testy

W testach użyto bibliotek:

- NUnit – framework do testów jednostkowych,
- Moq – framework do tworzenia atrap obiektów do testów.

5.9.3. System operacyjny

Aplikacja webowa

Aplikacji webowej jest dostępna na każdym systemie operacyjnym, który posiada przeglądarkę zgodną z wymaganiami niefunkcjonalnymi aplikacji.

Aplikacja mobilna

Aplikacja mobilna działa na telefonach z systemem Android w wersji wyższej bądź równej wersji 4.1 (Jelly Bean).

5.9. WYBÓR TECHNOLOGII

Serwer API oraz baza danych

Do hostowania tych komponentów zalecane jest użycie Dockera (opisane w „Instrukcji instalacji”), który jest wspierany przez większość systemów operacyjnych. Należy jednak pamiętać, że maszyna musi mieć dużą moc obliczeniową oraz wysoką prędkość łącza internetowego, najlepiej łącze symetryczne.

Jeżeli nie może być użyte środowisko Dockera, Serwer API jest napisany na platformie .NET Core, która jest dostępna na najpopularniejszych systemach operacyjnych: Windows, Linux i macOS. Analogicznie jest w przypadku serwera MySQL. Należy pamiętać wtedy, aby te dwa komponenty były w tej samej sieci LAN lub VPN ze względów bezpieczeństwa i wydajności.

6. Instrukcja instalacji

Poniżej znajdują się opisy instalacji poszczególnych modułów potrzebnych do wdrożenia Systemu. Konfiguracja i instalacja kolejnych elementów Systemu jest przedstawiona w poszczególnych punktach. Wszystkie procesy zostały opisane w prosty, przystępny sposób, a wykonanie każdego podpunktu z tego rozdziału pozwoli na uruchomienie całego Systemu. W instrukcji wskazane są również elementy, które nie zostały tutaj opisane, ponieważ jest wiele sposobów na ich implementację bądź nie zależą od kodu systemu – te punkty również powinny zostać wykonane, by cieszyć się pełną funkcjonalnością i wszystkimi narzędziami Systemu.

6.1. Aplikacja mobilna

Zanim Użytkownicy będą mogli skorzystać z Aplikacji mobilnej, trzeba ją najpierw skompilować, tzn. zamienić kod źródłowy z folderu `/src/RecruitMe.MobileApp` w jeden plik, a następnie zainstalować na urządzeniu. Cały proces składa się z paru kroków, które zostaną przedstawione poniżej.

6.1.1. Kompilacja Aplikacji do pliku APK

Do procesu komplikacji wymagane jest zainstalowanie co najmniej kilku narzędzi programistycznych, w tym *Node.js* oraz *NativeScript CLI*. Pełna instrukcja instalacji wymaganych aplikacji znajduje się na stronie NativeScript Docs w artykule *CLI Setup* [8].

Po przejściu całego poradnika instalacji potrzebnych narzędzi można przejść do procesu komplikacji. Aby tylko skompilować Aplikację lokalnie (czyli sprawdzić na własnym telefonie, jak działa Aplikacja), należy uruchomić okno konsoli w folderze `src/RecruitMe.MobileApp` i wykonać polecenie:

```
tns run android
```

Proces komplikacji może potrwać nawet kilka minut i zakończy się automatyczną instalacją oraz uruchomieniem Aplikacji na podłączonym do komputera urządzeniu z systemem Android lub do wirtualnej maszyny Androida zainstalowanej wcześniej. Warto zaznaczyć, że próba zainstalowania Aplikacji lokalnie z podłączonym do komputera po kablu USB telefonem powiedzie się tylko wtedy, gdy na urządzeniu zostanie włączona opcja „debugowania USB”.

Aby otrzymać jedynie plik APK, należy wykonać polecenie:

```
tns build android
```

W obu przypadkach po wykonaniu procesu w folderze:

`src/RecruitMe.MobileApp/platforms/android/app/build/outputs/apk/debug/` znajdzie się plik APK aplikacji mobilnej w formacie debug.

W przypadku, gdy jednak Aplikacja mobilna ma być skompilowana do pliku, który będzie można potem rozpowszechniać w sklepach z aplikacjami (np. Google Play), należy przeprowadzić proces komplikacji w trybie *release*. Aby to zrobić, potrzebny jest certyfikat – klucz prywatny potwierdzający bycie właścicielem Aplikacji mobilnej. Szczegółowe instrukcje dotyczące tego, jak taki certyfikat stworzyć, znajdują się w artykule *Publishing for Android* na stronie NativeScript Docs [9].

6.2. APLIKACJA WEBOWA

Po stworzeniu certyfikatu można przystąpić do procesu komplikacji w trybie *release*. W oknie konsoli otwartej w folderze src/RecruitMe.MobileApp należy wykonać:

```
tns build android --release  
--key-store-path <ścieżka-do-certyfikatu>  
--key-store-password <hasło-certyfikatu>  
--key-store-alias <alternatywna-nazwa-certyfikatu>  
--key-store-alias-password <hasło-altern.-nazwy-certyfikatu>
```

Stworzony plik znajduje się w folderze

src/RecruitMe.MobileApp/platforms/android/app/build/outputs/apk/release.

Uwaga: Przed procesem komplikacji aplikacji należy zainstalować najpierw serwer i otrzymać jego adres. Następnie należy otworzyć plik src/RecruitMe.MobileApp/app/services/common/apiGateway.ts w dowolnym edytorze tekstu i zmienić w linii 9. pomiędzy cudzysłowami adres, na jaki aplikacja będzie wysyłała żądania. W przeciwnym wypadku proces komplikacji będzie trzeba powtórzyć już ze zmienionymi danymi. Niezmienienie danych skutkuje wadliwym działaniem całej Aplikacji mobilnej.

6.1.2. Instalacja aplikacji na urządzeniu

Aby móc korzystać z Aplikacji mobilnej, należy ją wcześniej zainstalować na urządzeniu z systemem operacyjnym Android. W zależności od wybranej przez Organizatorów rekrutacji metody dystrybucji Aplikacji mobilnej istnieją różne sposoby instalacji tego programu.

Przekazywanie pliku APK

Organizatorzy mogą się zdecydować udostępniać we własnym zakresie Aplikację mobilną na systemy Android poprzez plik APK. Zainteresowani programem Użytkownicy muszą ściągnąć go na swoje telefony i kliknąć na pobrany plik, by go zainstalować. W takim wypadku Użytkownicy zmuszeni są zezwolić na **instalowanie aplikacji z nieznanych źródeł**.

Google Play lub inny sklep z aplikacjami

By potencjalnie zwiększyć zakres odbiorców Systemu, istnieje opcja upublicznienia Aplikacji w sklepach z aplikacjami. Gdy Aplikacja znajdzie się w takim sklepie, Użytkownik może ją wyszukać i zainstalować jak każdą inną z wybranego sklepu.

6.2. Aplikacja webowa

Aplikacja webowa nie wymaga instalacji – jedynym wymaganiem jest posiadanie zainstalowanej przeglądarki internetowej. Chociaż Aplikacja powinna działać na dowolnej nowoczesnej przeglądarce internetowej, zalecane jest używanie przeglądarek Google Chrome lub Mozilla Firefox, za pomocą których aplikacja była tworzona i została na nich przetestowana. Podczas testów znalezione zostały niepożądane zachowania w przypadku niektórych przeglądarek:

- **Mozilla Firefox** – jeśli w procesie konfiguracji (opisanym poniżej) podjęta zostanie decyzja o rezygnacji z przekierowania Użytkowników na połączenie bezpieczne (HTTPS), przeglądarka (ze względów bezpieczeństwa) odmówi aplikacji dostępu do kamerki internetowej urządzenia,

- **Microsoft Edge** – ze względu na ograniczone funkcjonalności przeglądarki, niektóre funkcjonalności mogą nie działać poprawnie np.: przeglądarka nie wspiera konstruktora `new File()`, który jest używany w zapisie zdjęcia zrobionego kamerką – tym samym ta funkcjonalność nie działa poprawnie.

6.3. Baza danych oraz Serwer API

Instalacja tych dwóch komponentów jest ze sobą ściśle powiązana oraz dość złożona, więc została ona podzielona na dwie części: konfigurację, czyli dostosowanie Systemu do potrzeb Szkoły, oraz wdrożenie, czyli upublicznienie Systemu.

6.3.1. Konfiguracja

Potrzebne narzędzia

Absolutnym minimum do konfiguracji Aplikacji jest dowolny edytor tekstowy, jednak zaleca się użycie zintegrowanego środowiska programistycznego – IDE (np.: Visual Studio, Visual Studio Code), które ułatwi zadanie i wskaże ewentualne błędy. Dodatkowo, jeśli Organizator będzie chciał przetestować stworzoną konfigurację, musi ściągnąć narzędzia programistyczne wymienione poniżej:

- .NET Core SDK (<https://dotnet.microsoft.com/download>),
- Node.js oraz npm (<https://nodejs.org/en/download/>),
- MySQL Server (<https://www.mysql.com/downloads/>).

Strony dla Użytkowników niezalogowanych

W Systemie zostały przygotowane strony ogólnodostępne: strona główna, „O szkole” i „Regulamin”. Ich zawartość można zmienić, edytując odpowiadające im pliki w folderze `src/RecruitMe.Web/ClientApp/components/staticpages`. Domyślnie są one wypełnione przykładowym tekstem „lorem ipsum”, który powinien być zmieniony przed instalacją Systemu.

Treść wysyłanych maili

Aby zmienić treść wiadomości e-mail wysyłanych do Użytkowników, należy zmodyfikować pola w klasie `EmailContentConfiguration` znajdującej się w folderze `src/RecruitMe.Logic/Configuration`. Możliwe jest zmienienie tytułów i treści wszystkich maili wysyłanych przez System.

- RegisteredTitle, RegisteredBody odpowiadają tytułowi i treści wiadomości wysyłanej po zarejestrowaniu się do Systemu.
- EmailVerifiedTitle, EmailVerifiedBody odpowiadają tytułowi i treści wiadomości wysyłanej po potwierdzeniu adresu e-mail.
- LoginRemindedTitle, LoginRemindedBody odpowiadają tytułowi i treści wiadomości wysyłanej, kiedy Użytkownik używa funkcjonalności przypomnienia loginu.
- ResetPasswordTitle, ResetPasswordBody odpowiadają tytułowi i treści wiadomości, kiedy Użytkownik chce zresetować hasło.

6.3. BAZA DANYCH ORAZ SERWER API

Certyfikat SSL

Z uwagi na dane osobowe przesyłane pomiędzy Aplikacjami, połączenie musi być zabezpieczone a Użytkownicy powinni używać protokołu HTTPS przy łączaniu się z Serwerem API. Do Aplikacji dołączony jest testowy certyfikat SSL wygenerowany na potrzeby rozwoju i testów Aplikacji. Nie należy jednak go używać w środowisku produkcyjnym. Wykupiony certyfikat należy umieścić w formacie .pfx w folderze *src/RecruitMe.Web/ssl/* z nazwą aspnetapp.pfx. Hasło, którym zaszyfrowany jest plik, należy umieścić w pliku appsettings.json w folderze *src/RecruitMe.Web/* pod kluczem SslCertificatePassword.

Pliki konfiguracyjne

W pliku appsettings.json znajdują się dane konfigurujące zachowanie Aplikacji. Poza wspomnianym powyżej hasłem do certyfikatu należy zmienić:

- ConnectionString – wartość *password* należy zmienić z domyślnej („Tester!123”) na wybrane przez siebie hasło.
- UseHttpsRedirection – jeśli dostarczony certyfikat jest zaufany (tzn. jest wystawiony przez Urząd Certyfikacji, angl: CA), należy zmienić wartość na „True” – wówczas Użytkownicy będą przekierowywani na połączenie bezpieczne, nawet jeśli pierwsze połączenie nastąpiło po kanale niezabezpieczonym.
- BusinessConfiguration:Email – konto poczty elektronicznej, z której chcemy, aby były wysyłane wiadomości do Użytkowników. Potrzebny jest dostęp SMTP do wskazanego konta.
- BusinessConfiguration:EmailPassword – hasło do powyższego konta.
- BusinessConfiguration:LowestRegistrationDate – najniższa możliwa data urodzenia, jaką może mieć Użytkownik, aby mógł się zarejestrować w Systemie.
- BusinessConfiguration:HighestRegistrationDate – najwyższa możliwa data urodzenia, jaką może mieć Użytkownik, aby mógł się zarejestrować w Systemie.
- BusinessConfiguration:BaseAddress – adres publiczny Serwera API, po którym będą się łączyć użytkownicy. Ważne jest, aby adres kończył się bez ukośnika („https://example.com” – prawidłowy adres, „https://example.com/” – nieprawidłowy adres).
- PaymentConfiguration:Id – unikatowy numer sklepu w serwisie Dotpay.
- PaymentConfiguration:RegistrationFee – kwota, jaką Kandydaci muszą uiścić w serwisie Dotpay.
- PaymentConfiguration:Currency – waluta określająca *RegistrationFee*, dostępne wartości: *PLN, EUR, USD, GBP, JPY, CZK, SEK, UAH, RON, NOK, BGN, CHF, HRK, HUF, RUB*.
- PaymentConfiguration:PIN – kod PIN sklepu w serwisie Dotpay, który jest dostępny w panelu Dotpay.

- PaymentConfiguration:AuthToken – token autoryzacji do konta Dotpay, to ciąg znaków w kodowaniu *Base64*, który po odkodowaniu sprowadza się do następującego napisu: <*nazwa Użytkownika*>:<*hasło do systemu Dotpay*> (znaki < oraz > muszą być pominięte, zostały one użyte tylko dla czytelności).
- PaymentConfiguration:UseProductionServer – parametr określający, czy System ma korzystać z serwera produkcyjnego Dotpay. Wartość *false* będzie oznaczała, że wszystkie zapytania do serwisu Dotpay będą przechodziły przez serwer testowy - należy wtedy w pola **PaymentConfiguration:PIN** oraz **PaymentConfiguration:AuthToken** przekazać dane testowe.

6.3.2. Wdrożenie

Potrzebne narzędzia

Do wdrażania i hostowania Systemu użyty został Docker. Jest to narzędzie umożliwiające zdefiniowanie środowiska wykonawczego aplikacji, usuwając jednocześnie zależność od systemu operacyjnego i zainstalowanych bibliotek.

Wymagania sprzętowe

Wymagania sprzętowe są w pełni zależne od wykorzystania Systemu w trakcie rejestracji – Aplikacja, z której będzie korzystać 100 Użytkowników, ma o wiele mniejsze wymagania od Aplikacji, z której korzysta 3000 osób. Dlatego też zaleca się monitorowanie wolnych zasobów Systemu w czasie jego pracy – należy również pamiętać, że przed kluczowymi datami (np.: koniec okresu rejestracji, ogłoszenie wyników) ruch w Systemie znacznie wzrośnie, a z nim zapotrzebowanie na zasoby. Dla Aplikacji, których liczba Użytkowników nie przekroczy 1000, sugerujemy serwer o co najmniej 32GB wolnego miejsca na dysku twardym, 8GB pamięci RAM oraz dowolny nowoczesny procesor o co najmniej 2 rdzeniach.

Utworzenie sieci

Pierwszym krokiem procesu wdrożenia jest utworzenie wirtualnej sieci dla kontenerów Dockera. Sprowadza się to do wykonania polecenia:

```
docker network create --driver=bridge recruit-me-network
```

Utworzenie bazy

Następnie wewnątrz sieci należy uruchomić narzędzie MySQL Server. Poniższy skrypt kolejno: pobiera obraz serwera bazy danych, startuje go wewnątrz sieci i przegląda logi w poszukiwaniu wygenerowanego hasła.

```
docker pull mysql/mysql-server
docker run -p 3306:3306 --net=recruit-me-network
--name=rm-mysql -d mysql/mysql-server
docker logs rm-mysql 2>&1 | grep GENERATED
```

6.3. BAZA DANYCH ORAZ SERWER API

Oczekiwanym wynikiem ostatniej komendy jest wiersz podobny do poniższego zawierającego hasło. Jeśli skrypt nie zwróci takiego wyniku, należy chwilę odczekać, aż baza danych się uruchomi. W przypadku niepowodzenia informacje o błędach możemy sprawdzić polecienniem `docker logs rm-mysql`.

```
[Entrypoint] GENERATED ROOT PASSWORD: 6IBLUwAaKXEG[aBc0MaONz
```

Następnie należy zalogować się do serwera za pomocą komendy:

```
docker exec -it rm-mysql mysql -uroot -p
```

Polecenie zapyta o hasło, które można znaleźć w logach. Po zalogowaniu powinno się wykonać poniższy skrypt, w którym należy zmienić hasło „Tester!123” na zgodne z hasłem w `appsettings.json`.

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'Tester!123';
CREATE USER 'recruitme'@'%' IDENTIFIED BY 'Tester!123';
GRANT ALL PRIVILEGES ON *.* TO 'recruitme'@'%';
quit
```

W ten sposób skonfigurowana baza danych jest gotowa na połączenia. Można się do niej podłączyć używając programów typu MySQL Workbench i konfigurować według własnych potrzeb. Katalog bazy Aplikacji zostanie stworzony przy pierwszym włączeniu Serwera API.

Serwer API

Aby uruchomić Serwer API najpierw należy utworzyć jego obraz. Dzieje się to w krokach zdefiniowanych w pliku Dockerfile. Aby uruchomić proces kompilacji należy, będąc w folderze `src/`, wywołać komendę:

```
docker build -t rm-api-image .
```

Proces kompilacji może być czasochłonny, zwłaszcza za pierwszym razem. Kiedy się skończy, należy otrzymany obraz uruchomić polecienniem:

```
docker run -d --net=recruit-me-network --name rm-api
-p 80:80 -p 443:443 --expose=80 --expose=443 rm-api-image
```

Po chwili serwer powinien być gotowy na przyjmowanie połączeń, co można sprawdzić polecienniem:

```
docker logs rm-api
```

Przykładowy wynik polecenia:

```
Now listening on: https://0.0.0.0:443
Now listening on: http://0.0.0.0:80
Application started. Press Ctrl+C to shut down.
```

Czynności po wdrożeniu

Pierwszą czynnością po wdrożeniu Systemu, którą należy wykonać natychmiast po pierwszym włączeniu Aplikacji, jest zmiana hasła Administratora z domyślnego ZMIEN_TO_HASŁO. Dokonuje się tego analogicznie do funkcjonalności resetowania hasła przez zwykłych Użytkowników. Wiadomość e-mail z linkiem do zmiany hasła przyjdzie na adres wskazany w appsettings.json. Po zmianie hasła można przystąpić do wprowadzania danych: nauczycieli, Egzaminów i ich kategorii.

6.4. Elementy nie pokryte instrukcją

6.4.1. Publikacja Aplikacji w sklepie Google Play

Organizator chcąc upublicznić Aplikację mobilną w sklepie Google Play, musi uiścić opłatę rejestracyjną konta deweloperskiego. Dlatego tworzenie takiego konta oraz publikacja Aplikacji w sklepie nie jest pokryta w instrukcji.

6.4.2. Konfiguracja sieci Serwera

Konfiguracja sieci, do której podłączony jest Serwer API, zależy w pełni od osoby instalującej System. Oczywiście, aby System był użytkowany przez potencjalnych Kandydatów, należy wykupić domenę internetową, aby ukryć adres IP przed Użytkownikiem. Konfiguracja sieci również ma bardzo duży wpływ na bezpieczeństwo Systemu. Zaleca się skonfigurowanie jej tak, aby baza danych nie była dostępna spoza sieci lokalnej. Dodatkowo, aby zapewnić płynne działanie Systemu, powinno się zastosować usługi blokujące ataki typu odmowy usługi (DDoS) lub brute force.

6.4.3. Założenie konta produkcyjnego Dotpay

Do poprawnego działania Systemu Administrator jest zobowiązany do założenia konta produkcyjnego w serwisie Dotpay. Samodzielne utworzenie konta zwiększa bezpieczeństwo i zmniejsza prawdopodobieństwo dostania się go w niepowołane ręce. Administrator nadal jest zobowiązany dostarczyć do Aplikacji wszystkie potrzebne dane, jakie posiada właściciel konta w serwisie Dotpay.

7. Testy

7.1. Testy automatyczne

Testy automatyczne znajdują się w projekcie *RecruitMe.Logic.Tests*. Kod podzielony jest na 3 główne przestrzenie nazw:

- UnitTests – zawiera testy jednostkowe sprawdzające działanie pojedynczych metod lub klas,
- IntegrationTests – zawiera testy integracyjne sprawdzające bardziej globalne aspekty Systemu, np.: sprawdzenie konfiguracji kontenera wstrzykiwania zależności,
- Helpers – zawiera części wspólne dla testów lub metody pomocne pozwalające na czytelniejsze testy.

Testy zostały napisane z użyciem frameworka nUnit. Dla klas napisanych z pomocą tego narzędzia charakterystyczne są dwa atrybuty metod: *Setup* – wskazująca metodę, która uruchamia się przed wykonaniem każdego testu, oraz *Test* – oznaczająca pojedynczy test.

W pojedynczym pliku testu znajdzie się jedna metoda *Setup* oraz co najmniej jedna metoda *Test*.

7.1.1. Testy jednostkowe

Struktura folderów wewnętrz folderu UnitTests odpowiada strukturze folderów projektu logiki (*RecruitMe.Logic*). To znaczy, że dla klasy testowej, istnieje klasa testowana w tym samym folderze co plik testu, ale w projekcie logiki.

W metodzie *Setup* tworzona jest zawsze atrapa (ang.: *mock*) bazy danych, w której znajdują się tabele używane przez testowane funkcjonalności. Tak stworzona atrapa pozwala w łatwy i powtarzalny sposób testować kod, bez konieczności uruchamiania całej bazy danych dla testów. Wstrzykiwanie atrap pozwala również wyeliminować inne potencjalne źródła błędów, które nie wynikają z błędnego kodu testowanej metody.

```
public class LoginTests
{
    Mock<BaseDbContext> DbContext { get; set; }

    [SetUp]
    public void Setup()
    {
        DbContext = StartupHelper.GetSetUpAsyncMethod(t => t.Users,
            GetUserCollection());
    }
}
```

Listing 7.1: Metoda *Setup()* uruchamiająca się przed testami z klasy *LoginTests*

W metodach oznaczonych atrybutem *[Test]* znajduje się logika pojedynczego testu. Testy są najczęściej napisane w strukturze AAA (ang. Arrange, Act, Assert), czyli podzielone na 3 części: przygotowanie obiektów, wykonanie testowanego kodu i sprawdzenia wyników.

```
[ Test ]
public async Task ShouldReturnExistingUserOnValidCandidateIdAndPassword()
{
    ILogger logger = new ConsoleLogger();
    LoginRequestValidator validator = new LoginRequestValidator();
    PasswordHasher passwordHasher = new PasswordHasher();
    LoginDto loginDto = new LoginDto()
    {
        CandidateId = "pawjum000" , Password = "goodPassword" };

    var query = new LoginUserQuery(logger , validator ,
        DbContext.Object , passwordHasher);
    var result = await query.Execute(loginDto);

    Assert.AreEqual(result , GetUserCollection()[0]);
}

[ Test ]
public void ShouldThrowUnauthorizedAccessExceptionWhenWrongDataPassed()
{
    ILogger logger = new ConsoleLogger();
    LoginRequestValidator validator = new LoginRequestValidator();
    PasswordHasher passwordHasher = new PasswordHasher();
    LoginDto loginDto = new LoginDto()
    {
        CandidateId = "jankow000" , Password = "badPassword" };

    Assert.ThrowsAsync<UnauthorizedAccessException>(async () =>
        await new LoginUserQuery(logger , validator , DbContext.Object ,
            passwordHasher).Execute(loginDto));
}
```

Listing 7.2: Przykładowe testy

Pierwsza część polega na przygotowaniu przypadku testowego: zawiera deklarację i inicjalizacje obiektów potrzebnych w metodzie. Tutaj używane są klasy bez wewnętrznej logiki lub przetwarzane już w innych testach, np.: wstawiane są obiekty bezpośrednio do „zamockowanej” bazy danych.

Druga i trzecia część to wywołanie testowanej funkcjonalności i sprawdzenie rezultatów. W zależności od pożdanego wyniku kod może być wykonany na dwa sposoby. Jeśli sprawdzany będzie wynik metody, jest ona wywoływana bezpośrednio w kodzie testu oraz jej wynik jest porównywany z wartością oczekiwanaą odpowiednią metodą z klasy *Assert* (*Assert.AreEqual*, *Assert.True*, *Assert.NotNull* itd.). Ta sytuacja jest widoczna w teście pierwszym na Listingu 7.2.

Jeśli natomiast sprawdzane jest, czy metoda rzuci wyjątek (test 2.), wywołuje się ją przekazującą

7.1. TESTY AUTOMATYCZNE

ając do `Assert.ThrowsAsync` asynchroniczną metodę anonimową uruchamiającą tę metodę, jak w drugim teście na Listingu 7.2.

Popularną konwencją w testach jest pisanie długich i opisowych nazw funkcji testujących, dzięki czemu wiadomo, co jest sprawdzane w teście i jaki jest jego oczekiwany wynik.

7.1.2. Testy integracyjne

Testy integracyjne mają szerszy zakres niż testy jednostkowe i wykonują się zazwyczaj we własnym środowisku np. testy nie „mockują” używanych tabel w bazie danych, ale mają uruchomioną własną bazę. Kiedy system implementowany jest według reguły CQRS, testy integracyjne zazwyczaj też testują pojedynczą metodę podobnie jak testy integracyjne, jednak nic nie stoi na przeszkodzie, aby sprawdzały kilka klas jednocześnie i ich względową integrację lub inne aspekty projektu. W Systemie został napisany test, który sprawdza poprawność kontenera wstrzykiwania zależności. Uruchamiana jest z funkcji testowej metoda `ConfigureServices` klasy `Startup`, która m.in. konfiguruje kontener. Następnie kontener jest proszony o instancję każdego dodanego serwisu, który w swojej pełnej nazwie zawiera frazę „RecruitMe”. Jeśli nie można uzyskać dowolnego serwisu test się nie powodzi.

7.1.3. Uruchomienie środowiska testowego

Zintegrowane środowisko deweloperskie (IDE)

Używając środowiska programistycznego *Visual Studio*, można łatwo sprawdzić, czy testy w projekcie są napisane poprawnie oraz czy kod logiki jest zgodny z tymi testami.

Po otworzeniu solucji i uruchomieniu testów (np.: klikając prawym klawiszem myszy na projekt testów (`RecruitMe.Logic.Tests`) a następnie *Run tests*) wyświetli się okno „Eksplorator testów” (ang. *Test Explorer*) z listą wszystkich dostępnych testów. Po zakończeniu wykonywania się testów przy każdym elemencie na liście pojawi się zielone lub czerwone kółko. Pojawienie się czerwonego kółka przy dowolnym z testów oznacza, że testy nie przechodzą poprawnie i wymagana jest interwencja w kodzie projektu (albo w samej logice testów czy projektu).

Linia poleceń

Opcjonalnie testy można przeprowadzić, uruchamiając w terminalu polecenie `dotnet test RecruitMe.sln` (będąc w głównym folderze rozwiązania). Wymagane jest wtedy jednak przywrócenie i zbudowanie całego projektu odpowiednio poleceniami `dotnet restore` oraz `dotnet build`.

Automatyczne testowanie z Bitbucket pipeline

Kod projektu był przechowywany w repozytorium na portalu Bitbucket, który pozwala na skonfigurowanie automatycznego uruchamiania testów na nowym kodzie za pomocą mechanizmu pipelines. Został on wykorzystany podczas tworzenia Systemu i kazda migawka (ang. commit) trafiająca na gałąź *dev* lub *master* (od momentu uruchomienia automatycznych testów) była automatycznie sprawdzana, czy nowy kod się kompliluje i przechodzi testy.

7.2. Test manualne

Środowisko przedprodukcyjne

Poza testami automatycznymi oraz manualnymi w czasie wytwarzania Systemu, wprowadzone zostało środowisko „przedprodukcyjne” (ang. staging), na którym testowane były aspekty wydajności Systemu, jego zachowanie w środowisku sieciowym i poprawność instrukcji instalacji. Niestety, środowisko nie posiadało publicznego adresu IPv4 (testy odbyły się na IPv6), więc część funkcjonalności, jak np.: potwierdzenie płatności z bramki płatniczej nie mogła być w ten sposób przetestowana. Mimo wszystko znalezione zostało kilka problemów, nie pojawiających się w pierwszej wersji na urządzeniach developerskich, na których przypadkiem były zainstalowane potrzebne narzędzia.

Testowanie czytania kart egzaminacyjnych

Testy czytania wypełnionych kart egzaminacyjnych pokazały pełną skuteczność dla poprawnie wypełnionych kart. Błędne czytanie było efektem zlej jakości zdjęcia karty (dlatego zalecane jest dokonanie jej skanu) lub wypełnieniem punktu za słabo i za jasnym kolorem. System nie wykrywa braku oceny na karcie – dla danego uczestnika egzaminu zawsze wybierany jest najciemniej zamalowany punkt, co w przypadku, kiedy żaden nie jest zamalowany, oznacza wybranie losowego. System poradził sobie też z przypadkami, kiedy punkty nie były zamalowane całkowicie lub wychodziły lekko poza granicę konturu. Część skanów użytych do testów znajduje się w folderze `/doc/exam-sheets-tests`.

8. Podsumowanie

Stworzony System spełnia swoje założenia i wszystkie funkcjonalności zostały zaimplementowane. Jeśli Aplikacje zostaną wdrożone poprawnie, oferują swoim Użytkownikom podstawowe funkcje systemu rekrutacyjnego, ale również pomocnicze narzędzia oraz dodatkowy stopień automatyzacji w procesie rekrutacji. Przewagą RecruitMe nad podobnymi produktami jest integracja płatności w Systemie, bezpośredni chat pomiędzy Kandydatami a Administratorem i wykorzystanie kart egzaminacyjnych. Tego typu produktów podstawowe systemy często nie dostarczają, ponieważ korzystanie z nich może być zbyt skomplikowane. Z drugiej strony przez wprowadzenie tych funkcjonalności wymagane jest posiadanie wiedzy informatycznej, a nawet programistycznej, do pełnej instalacji Systemu, co może zniechęcić szkoły do korzystania z produktu.

Dzięki wysokiej jakości kodu projekt może być łatwo rozwijany przez dowolnego programistę, nie tylko z zespołu twórców. Chociaż produkt jest gotowy do użytku, niektóre funkcjonalności mogą zostać ulepszone, aby podwyższyć jego atrakcyjność. Na przykład strona panelu Administratora jest w pełni funkcjonalna, ale inaczej zaprojektowana poprawiłaby User Experience Administratora. Innym możliwym ulepszeniem jest użycie biblioteki SignalR, która umożliwia przesyłanie notyfikacji na Aplikację webową, dzięki którym chat działałby płynniej. Dodawanie nowych funkcjonalności najlepiej wykonać po pierwszych opiniach od Użytkowników, aby mieć wiedzę o podstawowych danych statystycznych, np.: jak popularna jest Aplikacja mobilna albo jak często Kandydaci wysyłają wiadomości poprzez chat, i na ich podstawie stwierdzić, jakie nowe funkcjonalności są potrzebne Użytkownikom.

Tworzenie systemu wymagało poznania nowych technologii m.in. Dockera i NativeScripta, procesów np.: implementowania aplikacji mobilnych i tworzenia architektury, a nawet języka programowania - Pythona. Projekt okazał się również większy niż było to przypuszczane na początku w fazie planowania i analizy, co jest cennym doświadczeniem do przyszłych wycen zadań i projektów.

Bibliografia

- [1] OMR - Wikipedia, wolna encyklopedia
<https://pl.wikipedia.org/wiki/OMR> (dostęp: styczeń 2020)
- [2] Scott Chacon, Ben Straub *Pro Git* (2014): Gałęzie Gita - Czym jest gałąź
[https://git-scm.com/book/pl/v2/Gałęzie-Gita-Czym-jest-gałąz](https://git-scm.com/book/pl/v2/Ga%C5%82ezie-Gita-Czym-jest-ga%C5%82a%C5%82) (dostęp: styczeń 2020)
- [3] FluentValidation - A popular .NET library for building strongly-typed validation rules
<https://fluentvalidation.net> (dostęp: styczeń 2020)
- [4] Filtry w ASP.NET Core
<https://docs.microsoft.com/pl-pl/aspnet/core/mvc/controllers/filters>
(dostęp: styczeń 2020)
- [5] Konfiguracja w ASP.NET Core
<https://docs.microsoft.com/pl-pl/aspnet/core/fundamentals/configuration>
(dostęp: styczeń 2020)
- [6] Wstrzykiwanie zależności w ASP.NET Core
<https://docs.microsoft.com/pl-pl/aspnet/core/fundamentals/dependency-injection>
(dostęp: styczeń 2020)
- [7] Integracja techniczna Dotpay
https://www.dotpay.pl/developer/doc/api_payment/pl/index.html (dostęp: styczeń 2020)
- [8] Opis pełnej instalacji środowiska NativeScript
<https://docs.nativescript.org/start/quick-setup#full-setup> (dostęp: styczeń 2020)
- [9] Opis stworzenia certyfikatu aplikacji mobilnej
<https://docs.nativescript.org/tooling/publishing/publishing-android-apps> (dostęp: styczeń 2020)
- [10] Nate Barbettini *The Little ASP.NET Core Book* (2018)
<https://recaffeinate.co/book/> (dostęp: styczeń 2020)
- [11] Adrian Rosebrock: Bubble sheet multiple choice scanner and test grader using OMR, Python and OpenCV
<https://www.pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/> (dostęp: styczeń 2020)

Spis rysunków

4.1	Pierwszy widok po otworzeniu aplikacji mobilnej	23
4.2	Formularz rejestracji	24
4.3	Formularz rejestracji aplikacji mobilnej	25
4.4	Ekran logowania	26
4.5	Ekran logowania w aplikacji mobilnej	26
4.6	Formularz przypomnienia loginu	27
4.7	Formularz przypomnienia loginu w aplikacji mobilnej	27
4.8	Formularz resetowania hasła	28
4.9	Formularz resetowania hasła w aplikacji mobilnej	28
4.10	Ekran profilu Kandydata	29
4.11	Ekran profilu Kandydata w aplikacji mobilnej	30
4.12	Uzupełnianie dodatkowych danych	31
4.13	Profil Kandydata z pozytywnym statusem rekrutacji	32
4.14	Profil Kandydata ze statusem rekrutacji w aplikacji mobilnej	32
4.15	Ekran statusu płatności	33
4.16	Ekran statusu płatności w aplikacji mobilnej	34
4.17	Ekran egzaminów	34
4.18	Ekran egzaminów w aplikacji mobilnej	35
4.19	Ekran chatu	36
4.20	Ekran chatu w aplikacji mobilnej	36
4.21	Główny ekran panelu administratora	37
4.22	Przykładowe dodawanie Egzaminu do Systemu	37
4.23	Ekran edycji Użytkownika	38
4.24	Ekran listy konwersacji	39
4.25	Ekran edycji egzaminu	39
4.26	Przykładowa karta egzaminacyjna	40
5.1	Schemat architektury systemu	41
5.2	Diagram schematu bazy	43
5.3	Diagram integracji z serwisem Dotpay [7]	51
5.4	Przykładowy wygląd strony rejestracji	54
5.5	Przykładowy wygląd strony na mniejszym ekranie	55
5.6	Przykładowy wygląd strony logowania w aplikacji mobilnej	56

Spis listingów

5.1	Przykładowa operacja: LoginUserQuery	49
5.2	Zawartość pliku Dockerfile	53
7.1	Metoda Setup() uruchamiająca się przed testami z klasy LoginTests	67
7.2	Przykładowe testy	68

Spis załączników

1. Płyta CD z pracą dyplomową w wersji elektronicznej oraz plikami źródłowymi projektu w folderze *repozytorium*.