

Wielowarstwowy system rekrutacji dla szkół z  
interfejsem webowym i aplikacją mobilną -  
analiza techniczna rozwiązania

Andrzej Westfalewicz, Filip Zyskowski

7 listopada 2019

# Spis treści

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Architektura</b>                              | <b>3</b>  |
| 1.1      | Baza danych . . . . .                            | 3         |
| 1.2      | API . . . . .                                    | 3         |
| 1.3      | Aplikacja webowa . . . . .                       | 3         |
| 1.4      | Aplikacja mobilna . . . . .                      | 3         |
| <b>2</b> | <b>Wzorce projektowe modułów</b>                 | <b>3</b>  |
| 2.1      | Baza danych . . . . .                            | 3         |
| 2.2      | Logika biznesowa . . . . .                       | 4         |
| 2.3      | API . . . . .                                    | 5         |
| 2.4      | Aplikacja webowa . . . . .                       | 5         |
| 2.5      | Aplikacja mobilna . . . . .                      | 5         |
| 2.6      | Testy . . . . .                                  | 5         |
| <b>3</b> | <b>Komunikacja</b>                               | <b>5</b>  |
| <b>4</b> | <b>Opis modułów</b>                              | <b>6</b>  |
| 4.1      | Logika biznesowa . . . . .                       | 6         |
| 4.1.1    | RecruitMe.Logic.Data . . . . .                   | 6         |
| 4.1.2    | RecruitMe.Logic.Logging . . . . .                | 6         |
| 4.1.3    | RecruitMe.Logic.Operations.Account . . . . .     | 6         |
| 4.1.4    | RecruitMe.Logic.Operations.Email . . . . .       | 8         |
| 4.1.5    | RecruitMe.Logic.Operations.Recruitment . . . . . | 8         |
| 4.1.6    | RecruitMe.Logic.Operations.Payments . . . . .    | 8         |
| 4.1.7    | RecruitMe.Logic.Operations.Admin . . . . .       | 8         |
| 4.1.8    | RecruitMe.Logic.Operations.Messages . . . . .    | 8         |
| 4.2      | Testy . . . . .                                  | 8         |
| 4.3      | API . . . . .                                    | 8         |
| 4.4      | Aplikacja webowa . . . . .                       | 9         |
| 4.4.1    | Komponenty . . . . .                             | 9         |
| 4.4.2    | Modele . . . . .                                 | 9         |
| 4.4.3    | Serwisy . . . . .                                | 9         |
| 4.5      | Aplikacja mobilna . . . . .                      | 10        |
| 4.5.1    | ViewModele . . . . .                             | 10        |
| 4.5.2    | Modele . . . . .                                 | 10        |
| 4.5.3    | Serwisy . . . . .                                | 10        |
| 4.5.4    | Magazyn danych . . . . .                         | 10        |
| <b>5</b> | <b>Opis UI/UX</b>                                | <b>11</b> |
| 5.1      | Aplikacja webowa . . . . .                       | 11        |
| 5.2      | Aplikacja mobilna . . . . .                      | 12        |
| <b>6</b> | <b>Interfejsy zewnętrzne</b>                     | <b>14</b> |
| 6.1      | SMTP . . . . .                                   | 14        |
| 6.2      | Integracja DotPay . . . . .                      | 14        |

|          |                                |           |
|----------|--------------------------------|-----------|
| <b>7</b> | <b>Wybór technologii</b>       | <b>14</b> |
| 7.1      | Języki programowania . . . . . | 14        |
| 7.2      | Użyte biblioteki . . . . .     | 14        |
| 7.2.1    | Aplikacja mobilna . . . . .    | 14        |
| 7.2.2    | Aplikacja webowa . . . . .     | 15        |
| 7.2.3    | API i logika . . . . .         | 15        |
| 7.2.4    | Testy . . . . .                | 16        |
| 7.3      | Platformy . . . . .            | 16        |
| 7.4      | System operacyjny . . . . .    | 16        |
| 7.4.1    | Aplikacja webowa . . . . .     | 16        |
| 7.4.2    | Aplikacja mobilna . . . . .    | 16        |
| 7.4.3    | API . . . . .                  | 16        |
| 7.4.4    | Baza danych . . . . .          | 17        |
| <b>8</b> | <b>Historia dokumentu</b>      | <b>17</b> |

# 1 Architektura

## 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.

## 1.2 API

Jest odpowiedzialne za dostęp do logiki biznesowej, zapis i odczyt danych, w szczególności zdjęć i dokumentów, przesłanych przez użytkowników. Umożliwia on pobranie przez przeglądarkę internetową Aplikacji webowej. Za pośrednictwem jego dokonywane są płatności przez bramkę płatniczą oraz wysyłane wiadomości e-mail. Do jego implementacji została wykorzystana technologia ASP.NET Core z uwagi na jej znajomość i popularność.

## 1.3 Aplikacja webowa

Umożliwia dostęp do systemu przez przeglądarkę internetową. Akcje użytkownika są interpretowane przez aplikację i wywołują odpowiednie akcje w API. Aby dane prezentowane na stronie były takie same na wszystkich urządzeniach, aplikacja nie przechowuje żadnych danych - są one pobierane z API. Zaimplementowana w popularnym frameworku Vue.js.

## 1.4 Aplikacja mobilna

Umożliwia dostęp do systemu przez telefon z systemem Android. Na tych urządzeniach nadal można się uzyskać dostęp do systemu poprzez mobilną przeglądarkę internetową, lecz aplikacja mobilna zapewnia znacznie większy komfort użytkowania na urządzeniach mobilnych. Akcje użytkownika są interpretowane przez aplikację i wywołują odpowiednie akcje w API. Zaimplementowana w frameworku NativeScript z nakładką umożliwiającą tworzenie aplikacji tak, jak w frameworku Vue.js.

# 2 Wzorce projektowe modułów

## 2.1 Baza danych

Baza została zaprojektowana zgodnie z podejściem *code-first* z wykorzystaniem Entity Framework Core dla baz w systemie MySQL. Oznacza to tyle, że najpierw opisujemy schemat tabeli jako klasa w języku C#, a potem jest ona mapowana na tabelę w bazie danych. Pozwala to również na modyfikowanie schematu bazy danych poprzez modyfikacje powyższych klas, za pomocą mechanizmu migracji dostarczonego przez Entity Framework. Pliki migracyjne reprezentujące historię

zmian oraz ostateczny stan bazy danych są w katalogu Migrations wewnątrz projektu RecruitMe.Web.

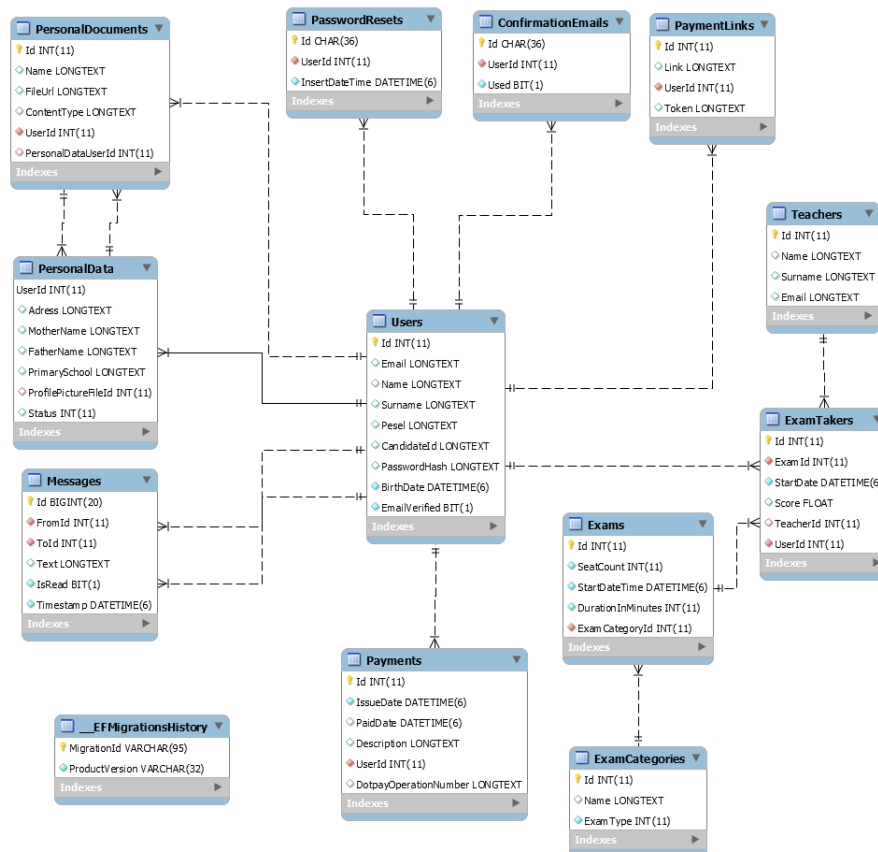


Figure 1: Obecny diagram schematu bazy

## 2.2 Logika biznesowa

- Logika biznesowa została zorganizowana według reguły rozdzielania komend i zapytań (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) oraz zapytania, które zwracają dane. Stosując się do tej reguły system można lepiej optymalizować i taniej utrzymywać.
- Jeśli pomiędzy akcjami są jakieś zależności (jedna agreguje wiele innych lub działa na wyniku poprzedniej) zgodnie z wzorcem projektowym wstrzykiwania zależności (dependency injection, DI) akcje nie są tworzone wewnątrz

klasy, lecz przekazywane do jej wnętrza przez konstruktor. Zastosowanie takiego wzorca umożliwia testowanie kodu.

- Część klas potrzebnych w warstwie logiki musi być zdefiniowana w warstwie API, dlatego stworzone zostały abstrakcyjne klasy i interfejsy, które są używane w miejscu gotowych klas w zgodzie z zasadą programowania do interfejsu (programming to interface).

## 2.3 API

Zadaniem API jest pośredniczenie pomiędzy aplikacjami frontendowymi, a warstwą logiki. Aby nie zużywać transferu danych, część akcji jest agregowanych w jedną np.: zapis danych kandydata i zapytanie o nowe wartości. Metody kontrolerów wywoływane na żądania na dane endpointy definiują interfejs systemu - możliwe akcje użytkowników.

Dodatkowo API jest odpowiedzialne za autentykację użytkowników wykonujących żądania do systemu, czyli za tworzenie i walidację JWT, według standardu autentykacji OpenID Connect.

## 2.4 Aplikacja webowa

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

- Model - żądania do API,
- View - kod TypeScript,
- View-Model - kod html/css.

## 2.5 Aplikacja mobilna

Z racji, że aplikacja mobilna jest zaimplementowana w frameworku NativeScript z nakładką umożliwiającą pisanie aplikacji w Vue.js, został wykorzystany wzorzec MVVM.

## 2.6 Testy

Testują kluczowe elementy systemu (np. płatności).

# 3 Komunikacja

Komunikacja pomiędzy aplikacjami frontendowymi a API będzie się odbywała poprzez protokół HTTPS.

Połączenie pomiędzy warstwy API a bazą danych będzie obsługiwane przez Entity Framework Core, który to obsługuje połączenia z bazami używając frameworka ADO.NET. Z kolei ADO.NET ustanawia połączenie poprzez sockety TCP/IP.

## 4 Opis modułów

### 4.1 Logika biznesowa

Moduł logiki biznesowej znajduje się w projekcie `RecruitMe.Logic`. Został podzielony na części (domeny) według zadań, które dana część wykonuje. Implementacja każdej domeny jest umieszczona w osobnej przestrzeni nazw.

#### 4.1.1 `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 już wyżej.

#### 4.1.2 `RecruitMe.Logic.Logging`

Zawiera klasy służące do logowania. Wybrana klasa jest wstrzykiwana do klas, które coś logują, aby ujednolicić proces zbierania logów.

#### 4.1.3 `RecruitMe.Logic.Operations.Account`

Zawiera klasy implementujące proces rejestracji, logowania, sprawdzanie ID użytkownika itd. Proces rejestracji, przypomnienia loginu, resetu hasła opisują poniższe diagramy aktywności.

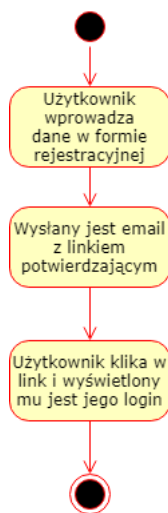


Figure 2: Diagram aktywności procesu rejestracji

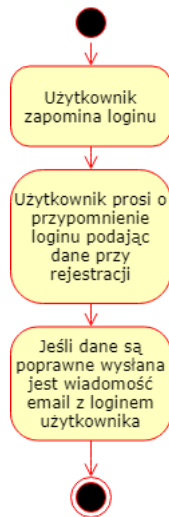


Figure 3: Diagram aktywności procesu przypomnienia loginu

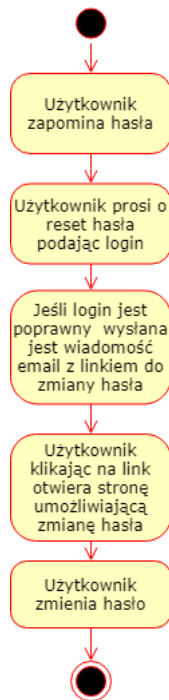


Figure 4: Diagram aktywności procesu resetu hasła



#### **4.1.4 RecruitMe.Logic.Operations.Email**

Zawiera klasy implementujące wysyłanie wiadomości email. Wysyłanie maili odbywa się za pomocą protokołu SMTP. Skrzynka, z której są wysyłane wiadomości może być podmieniona, jednak na czas rozwijania aplikacji został użyty adres "RecruitMeSystem@gmail.com".

#### **4.1.5 RecruitMe.Logic.Operations.Recruitment**

Zawiera klasy implementujące proces rekrutacji jak przesłanie danych, przesłanie dokumentów, zdjęcia lub rozmowy (chat) ze szkołą. Pliki będą przechowywane na dysku twardym poza bazą danych. Sposób zapisywania plików jest wstrzykiwany do domeny z projektu API.

#### **4.1.6 RecruitMe.Logic.Operations.Payments**

Zawiera klasy implementujące proces płatności. Klasy będą implementować operacje na bramce płatniczej DotPay. W bazie będą przechowywane tylko dane potrzebne do zweryfikowania, że na konto bankowe organizatora rekrutacji wpłynęły odpowiednie środki (to znaczy, że nie będziemy przechowywali takich danych jak numer karty kredytowej czy numeru CVV).

Proces wykonania transakcji, podobnie do procesu autentykacji OAuth, polega na przekierowaniu użytkownika na stronę DotPay z linkiem na który ma powrócić po dokonaniu transakcji oraz z zaszyfrowanymi informacjami.

#### **4.1.7 RecruitMe.Logic.Operations.Admin**

Zawiera klasy implementujące działania administratora, w szczególności edycję obiektów w bazie danych, generowanie i wczytywanie kart egzaminacyjnych.

#### **4.1.8 RecruitMe.Logic.Operations.Messages**

Zawiera klasy implementujące funkcjonalność chatu, dla kandydatów jak i dla administratora.

### **4.2 Testy**

Do testów sprawdzających poprawność działania systemu i jego poszczególnych modułów zaliczamy testy jednostkowe oraz integracyjne. Każdy z testów zostanie przydzielony do odpowiedniej przestrzeni nazw względem domen, które testują.

### **4.3 API**

Moduł API odpowiada za udostępnienie komunikacji pomiędzy modulem aplikacji webowej/mobilnej a modulem logiki biznesowej. Znajduje się on w folderze *RecruitMe.Web/*.

Z aplikacji webowej/mobilnej wysyłane jest zapytanie na adres serwera systemu o jakieś dane. Moduł API przechwytuje to żądanie, sprawdza poprawność parametrów zapytania i, gdy spełnione są odpowiednie warunki (choćby takie jak to, czy użytkownik jest zalogowany lub ma odpowiednie uprawnienia do wykonania danego zapytania), odsyła do aplikacji proszone dane lub informację, że jakaś czynność została wykonana pomyślnie. W przeciwnym wypadku moduł API zwraca aplikacji informację o nieudanym wykonaniu żądania wraz z odpowiednim kodem i informacją błędu.

## 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 ścieżki adresu do odpowiedniego komponentu aplikacji oraz ogólna konfiguracja startowa aplikacji.

### 4.4.1 Komponenty

Folder *components/* zawiera komponenty renderowane do widoków w aplikacji. Są one podzielone na foldery względem ścieżki, po jakiej się do nich można 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
- \*.cs - stylowanie widoku (w większości zastąpione używając gotowych komponentów z paczki vuetify)

### 4.4.2 Modele

W folderze *models/* znajdują się definicje modeli danych potrzebnych do wykonywania poprawnych zapytań do API oraz otrzymywania poprawnych odpowiedzi z API.

Każdy plik z o końcówce *.model.ts* zawiera co najmniej interfejs, który specyfikuje format, w jakim powinniśmy wysłać zapytanie i otrzymać odpowiedź.

### 4.4.3 Serwisy

Folder *services/* zawiera pliki o końcówce *.service.ts*, które zawierają logikę łączenia się do API i zarządzania otrzymanymi z API danymi. Wykorzystywane są głównie w plikach \*.ts komponentów aplikacji.

## 4.5 Aplikacja mobilna

W głównym folderze projektu aplikacji mobilnej (Recruit.MobileApp/app/) mamy uporządkowaną strukturę plików i folderów odpowiadającą poszczególnym modułom zaimplementowanych w aplikacji mobilnej.

### 4.5.1 ViewModele

W folderze *components/* są wszystkie viewmodele używane w aplikacji. Znajdziemy tam pliki z rozszerzeniem *.vue*, które są rozmieszczone w odpowiednich podfolderach oznaczających w jakich operacjach będą się pojawiały poszczególne widoki (plik *Login.vue* będzie znajdował się w podfolderze *account/*, gdyż akcja logowania jest związana z operacjami na koncie użytkownika).

Pliki zawierają trzy główne sekcje rozdzielone znacznikami wymienionymi poniżej

- `<template>...</template>` - rozmieszczenie komponentów na ekranie telefonu
- `<script>...</script>` - dane i logika modelu
- `<style>...</style>` - wystylowanie komponentów

### 4.5.2 Modele

W folderze *models/* znajdują się definicje modeli danych potrzebnych do wykonywania poprawnych zapytań do API oraz otrzymywania poprawnych odpowiedzi z API.

Każdy plik z rozszerzeniem *.ts* zawiera co najmniej interfejs, który specyfikuje format, w jakim powinniśmy wysłać zapytanie i otrzymać odpowiedź.

### 4.5.3 Serwisy

Folder *services/* zawiera pliki o rozszerzeniu *.ts*, które zawierają logikę łączenia się do API i zarządzania otrzymanymi z API danymi. Niektóre pliki są pogrupowane w foldery, gdy pewne logiki należą do jednej kategorii (serwis użytkownika i serwis przechowywania danych logowania znajdują się w jednym folderze).

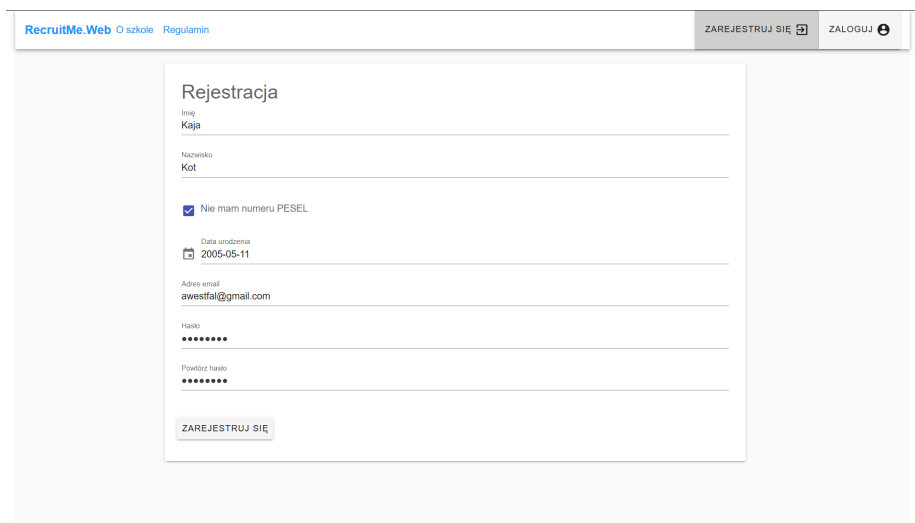
### 4.5.4 Magazyn danych

W folderze *RecruitMe.MobileApp/* znajduje się plik o nazwie *store.ts* będący wynikiem zastosowania biblioteki Vuex, który przechowuje dane pomiędzy widokami aplikacji. Obecnie jest wykorzystywany do utrzymywania tokenu zalogowanego użytkownika.

## 5 Opis UI/UX

### 5.1 Aplikacja webowa

Aplikacja webowa będzie korzystać 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 jak i małych monitorach. W razie potrzeby niektóre elementy będą dopasowywane do stylu systemu za pomocą stylów CSS.



The image shows a web browser window displaying a registration form. The browser's address bar shows 'RecruitMe.Web' with links to 'O szkole' and 'Regulamin'. The page has a header with 'ZAREJESTRUJ SIĘ' and 'ZALOGUJ' buttons. The main content is a form titled 'Rejestracja' with the following fields: 'Imię' (filled with 'Kaja'), 'Nazwisko' (filled with 'Kot'), a checkbox 'Nie mam numeru PESEL' (checked), 'Data urodzenia' (calendar icon, filled with '2005-05-11'), 'Adres email' (filled with 'awestfal@gmail.com'), 'Hasło' (masked with dots), and 'Powtórz hasło' (masked with dots). A 'ZAREJESTRUJ SIĘ' button is at the bottom of the form.

Figure 5: Przykładowy wygląd strony rejestracji

RecruitMe.Web
O szkole
Regulamin
CHAT
EGZAMINY
PŁATNOŚCI
PROFIL
WYLOGUJ

### Profil

Dane podstawowe

Login  
kajkot000

Imię  
Kaja

Nazwisko  
Kot

Adres e-mail  
awestfal@gmail.com

Data urodzenia  
25.08.2004

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


Dane dodatkowe

Adres  
Warszawa

Imię i nazwisko rodzica 1  
Jarek

Imię i nazwisko rodzica 2

Zdjęcie profilowe

IMG\_20190721\_133812.jpg


WCZYTAJ ZDJĘCIE
APARAT

Dokumenty

Wybierz dokument
Świadcstwo.pdf

PRZEŚLIJ DOCUMENT

Figure 6: Przykładowy wygląd strony profilu kandydata

## 5.2 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 będzie się posługiwała 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 zobaczy duże logo aplikacji i pod spodem dwa przyciski prowadzące do ekranu logowania oraz ekranu rejestracji. W ekranie logowania użytkownik oprócz podania danych do autoryzacji, będzie mógł przypomnieć sobie login i/lub hasło. W ekranie rejestracji będzie widoczny formularz z polami do wypełnienia oraz przycisk do ukończenia procesu rejestracji.

Po zalogowaniu użytkownik zostanie przeniesiony na ekran kandydata, na którym znajdzie informacje dotyczące egzaminów, na które jest zapisany. Z tego widoku, łatwo będzie w stanie się przenieść do innych funkcjonalności systemu za pomocą paska nawigacji. Użytkownik będzie widział przypomnienie u góry ekranu, jeżeli nie potwierdzi swojego adresu e-mail.

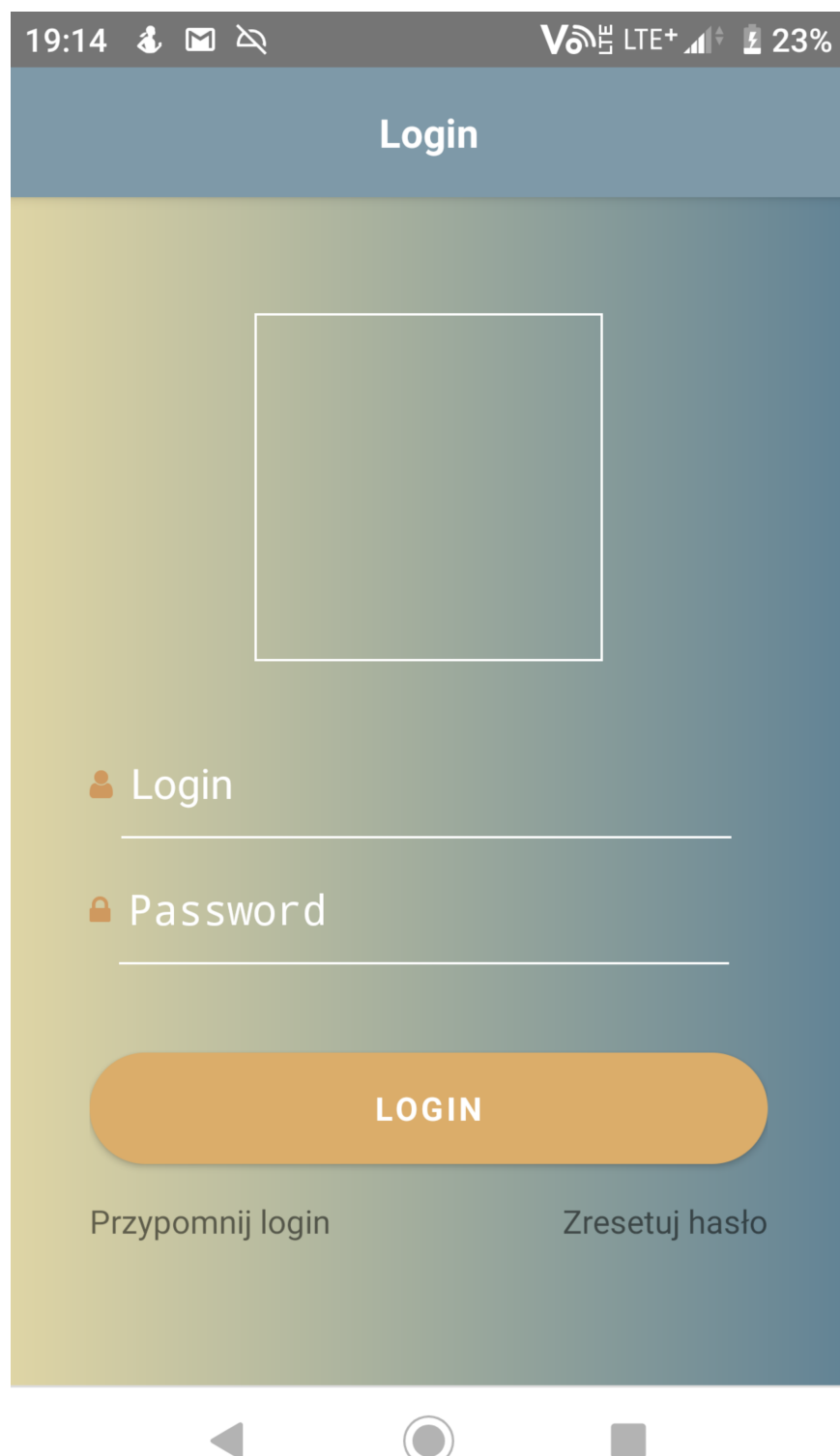


Figure 7: Przykładowy wygląd strony logowania w aplikacji mobilnej

## 6 Interfejsy zewnętrzne

### 6.1 SMTP

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

### 6.2 Integracja DotPay

Potrzebne będzie zaimplementowanie protokołów i procesu dokonywania płatności dostarczane przez DotPay. Po wstępnej analizie proces ten jest podobny do dobrze znanego procesu OAuth2. System DotPay nie implementuje żadnego ogólnodostępnego standardu, jednak ich standard jest dobrze udokumentowany i z przykładami. [17]

## 7 Wybór technologii

### 7.1 Języki programowania

Do implementacji systemu zostały wykorzystane następujące języki programowania:

- C# (warstwa logiki biznesowej, API, testy logiki i API)
- TypeScript/JavaScript (aplikacja webowa i mobilna oraz ich testy)
- SQL (język do komunikowania się z bazą danych, głównie autogenerowany przez Entity Framework)
- Python (skrypt wczytujący karty egzaminacyjne)

### 7.2 Użyte biblioteki

#### 7.2.1 Aplikacja mobilna

Głównym frameworkiem pozwalającym na budowę systemu na telefony z systemem Android jest NativeScript [1]. W aplikacji mobilnej jest wykorzystywany jego plugin nazwany NativeScript-Vue, pozwalający na pisanie aplikacji w sposób bardzo podobny do tego, jakbyśmy projektowali aplikację webową za pomocą frameworka Vue.js [2].

Innymi bibliotekami użytymi w aplikacji mobilnej są:

- axios - do wykonywania zapytań HTTP [3]
- nativescript-background-http - do ściągania zdjęć z serwera [4]
- nativescript-camera - do robienia zdjęć [5]
- nativescript-feedback - wyświetlanie powiadomień w aplikacji [6]

- nativescript-imagepicker - do wyboru zdjęcia z galerii [7]
- nativescript-local-notifications - pokazywanie powiadomień o przychodzących wiadomościach [8]
- nativescript-ui-sidedrawer - panel boczny aplikacji [9]
- nativescript-urlhandler - wykrywanie powrotu do aplikacji mobilnej z płatności [10]
- rxjs - śledzenie zmian komponentów po zalogowaniu [11]
- vue-class-component, vue-property-decorator - poprawna obsługa TypeScriptu dla NativeScripta [12] [13]
- vuex - przechowywanie informacji zalogowanego użytkownika w trakcie działania aplikacji [14]

### 7.2.2 Aplikacja webowa

Analogicznie głównym frameworkiem użytym 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,
- vuetify-datetime-picker
- oraz inne zaimportowane poprzez zależności lub udział w procesie ładowania: inversify, node-gyp, ajv, popper.js, ts-loader

Biblioteki wymagane podczas kompilacji i rozwoju aplikacji: aspnet-webpack, awesome-typescript-loader, bootstrap, copy-webpack-plugin, css-loader, deep-merge, 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.

### 7.2.3 API i logika

Do zaimplementowania logiki systemu oraz API będą wykorzystane następujące frameworki:

- 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, QRCoder, ZXing - biblioteki służące do generowania i czytania dokumentów,
- oraz liczne pomocnicze biblioteki z środowiska .NET Core.

#### 7.2.4 Testy

W testach będzie użyte kilka bibliotek:

- NUnit - framework do testów jednostkowych
- Moq - framework do tworzenia atrap obiektów do testów

### 7.3 Platformy

Środowiskiem wykonawczym aplikacji jest Docker, który jest wspierany na wszystkich popularnych platformach.

### 7.4 System operacyjny

#### 7.4.1 Aplikacja webowa

Z aplikacji webowej możemy korzystać na każdym systemie operacyjnym, który posiada przeglądarkę zgodną z wymaganiami niefunkcjonalnymi aplikacji.

#### 7.4.2 Aplikacja mobilna

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

#### 7.4.3 API

Api jest napisane na platformę .NET Core która jest dostępna na wszystkich głównych systemach operacyjnych: Windows, Linux i macOS.[15] Należy jednak pamiętać, że maszyna musi mieć dużą moc obliczeniową oraz wysoką prędkość łącza internetowego, najlepiej łącze symetryczne.

#### 7.4.4 Baza danych

Serwer MySQL jest dostępny w wszystkich głównych systemach operacyjnych: Windows, Linux, macOS i inne.[16] Należy jednak pamiętać, że maszyna musi mieć dużą moc obliczeniową oraz wysoką prędkość łącza internetowego, najlepiej łącze symetryczne. Powinna również być w tej samej sieci co API - VPN lub LAN.

## 8 Historia dokumentu

| Autor                                    | Data       | Wersja | Wprowadzone zmiany                  |
|--|------------|--------|-------------------------------------|
| Andrzej Westfalewicz,<br>Filip Zyskowski | 04.11.2019 | v0.1   | Pierwsza wersja dokumentu           |
| Andrzej Westfalewicz,<br>Filip Zyskowski | 05.11.2019 | v0.1   | Poprawki pierwszej wersji dokumentu |

## Odwołania

- [1] Strona frameworka NativeScript <https://www.nativescript.org/>
- [2] Strona frameworka Vue.js <https://vuejs.org/>
- [3] axios <https://github.com/axios/axios>
- [4] nativescript-background-http <https://github.com/NativeScript/nativescript-background-http>
- [5] nativescript-camera <https://github.com/NativeScript/nativescript-camera>
- [6] nativescript-feedback <https://github.com/EddyVerbruggen/nativescript-feedback>
- [7] nativescript-imagepicker <https://github.com/NativeScript/nativescript-imagepicker>
- [8] nativescript-local-notifications <https://github.com/eddyverbruggen/nativescript-local-notifications>
- [9] nativescript-ui-sidedrawer <https://www.npmjs.com/package/nativescript-ui-sidedrawer>
- [10] nativescript-urlhandler <https://github.com/hyper2k/nativescript-urlhandler>
- [11] rxjs <https://github.com/ReactiveX/RxJS>
- [12] vue-class-component <https://github.com/vuejs/vue-class-component>
- [13] vue-property-decorator <https://github.com/kaorun343/vue-property-decorator>
- [14] vuex <https://github.com/vuejs/vuex>
- [15] Strona główna oprogramowania .NET Core <https://dotnet.microsoft.com/download>
- [16] Wikipedia: MySQL <https://pl.wikipedia.org/wiki/MySQL>

- [17] Dotpay - Technical manual for payments implementation  
[https://www.dotpay.pl/developer/doc/api\\_payment/en/](https://www.dotpay.pl/developer/doc/api_payment/en/)