UMCS

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE

Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Informatyka**

Mateusz Oćwieja

Nr albumu: 303814

System rekrutacji na studia z zastosowaniem algorytmu odroczonej akceptacji Gale'a-Shapleya

University recruitment system based on Gale-Shapley deferred acceptance algorithm

Praca licencjacka
napisana w Zakładzie Katedrze Oprogramowania Systemów Informatycznych

pod kierunkiem dr Anny Sasak-Okoń

LUBLIN ROK 2023

Spis treści

Wstęp

Studia wyższe wciąż cieszą się ogromną popularnością w kraju i na świecie. W ubiegłym roku na uczelnie wyższe w Polsce zostało przyjętych ponad 400 tys. osób [1]. Tym samym wielu młodych ludzi co roku stoi przed wyborem uczelni i wymarzonego kierunku studiów. Każdy z nich musi zmierzyć się z systemami elektronicznej rekrutacji kandydatów poszczególnych uczelni.

Celem tej pracy jest opracowanie koncepcji scentralizowanego systemu rekrutacji na studia oraz implementacja prototypu tego systemu, który zwiększy efektywność procesu rekrutacji i przyczyni się do zwiększenia transparentności i sprawiedliwości w tym procesie. Po pierwsze, aplikacja webowa umożliwi sprawne i szybkie przeprowadzenie rekrutacji, co z kolei przełoży się na lepsze wykorzystanie czasu przez wszystkich uczestników procesu rekrutacyjnego - od studentów, po pracowników uczelni odpowiedzialnych za przyjmowanie aplikacji. Po drugie, zastosowanie nowoczesnych technologii w procesie rekrutacji jest coraz bardziej powszechne, a opracowanie aplikacji webowej pozwoli na sprawne wdrożenie tego rozwiązania na uczelniach.

Pierwszy rozdział poświęcono analizie problemu prowadzenia zapisów na studia. Zdefiniowano kryteria i niezbędne cechy, jakimi powinien charakteryzować się scentralizowany system rekrutacyjny. Takie rozwiązanie stanowiłoby jednolity interfejs do prowadzenia oraz rozstrzygania zapisów na uczelnie w skali całego kraju. System ten miałby być uniwersalny dla wszystkich kierunków i specjalizacji, niezależnie od liczby uczelni. Porównano istniejące rozwiązania i wykazano ich wady oraz niedoskonałości. Przedstawiono również możliwe rozwiązanie i wizję takiego scentralizowanego systemu rekrutacji, który wykorzystywałby zaadaptowaną wersję algorytmu odroczonej akceptacji w zautomatyzowanym procesie zapisów.

W drugim rozdziale dokładnie omówiono algorytm Gale'a Shapleya, rozwiązujący problem stabilnego dopasowania. Zawarto opis i schemat działania algorytmu. Poza jego podstawową formą, przytoczono szereg innych wariantów i ich zastosowań, w tym wyróżniono ten wykorzystany w projekcie. Opisano wykorzystaną adaptację i zamieszczono fragment kodu ją implementujący. Na końcu rozdziału znajduje się przykład wykorzystania algorytmu w praktyce i analiza efektów uzyskanych w symulacji.

W trzecim rozdziale dokładnie omówiono technologie wykorzystane w ramach projektu. Przedstawiono główne narzędzia użyte podczas tworzenia projektu, skupiając się na ich cechach i zastosowaniach. Przeprowadzono analizę wad i zalet każdej z technologii, uzasadniono wybór i przedstawiono ich rolę w budowie i utrzymaniu implementowanego systemu.

W czwartym rozdziale przedstawiono kompleksowy opis systemu rekrutacji z dwóch perspektyw: użytkownika i technicznej. Opisano funkcjonalności od strony zwykłego użytkownika oraz przedstawiono strukturę aplikacji, bazę danych i klasy odpowiedzialne za funkcjonowanie systemu dla programistów.

Na końcu pracy znajduje się podsumowanie zrealizowanych prac wraz ze zbiorem wniosków płynących z przeprowadzonych testów. Zawarto w nim ocenę realizacji celu postawionego we wstępie oraz przedstawiono możliwości rozwoju systemu w przyszłości.

1. Analiza problemu

1.1. Opis i analiza zagadnienia

System rejestracji na studia to zbiór zasad, procedur oraz narzędzi służących do zarządzania procesem przyjmowania kandydatów na studia wyższe. Jego głównym celem jest selekcja kandydatów i wybór najlepszych z nich, tak aby mogli podjąć studia na danej uczelni. System rejestracji na studia odpowiada m.in. Za przyjmowanie i rozpatrywanie podaniami o przyjęcie.

System rejestracji na studia składa się z kilku etapów. Pierwszym z nich jest najczęściej składanie podania o przyjęcie na studia przez kandydata. Podanie to zawiera informacje o kandydacie, takie jak np. dane osobowe, wykształcenie czy osiągnięcia. Następnie, po sprawdzeniu podanych przez kandydata informacji, uczelnia przeprowadza postępowanie rekrutacyjne, w ramach którego sprawdza kwalifikacje kandydatów i wybiera najlepszych z nich. Ostatecznie, po zakończeniu postępowania rekrutacyjnego, uczelnia ogłasza listy kandydatów przyjętych i nieprzyjętych na studia.

Dzięki niemu możliwe jest sprawne i obiektywne wybieranie najlepszych kandydatów, co przyczynia się do podnoszenia jakości kształcenia na uczelni. Taki system odpowiada również za przeprowadzenie postępowań rekrutacyjnych zgodnie z obowiązującymi przepisami prawa, a także za przekazywanie informacji o wynikach postępowań rekrutacyjnych kandydatom. Ponadto, służy do gromadzenia i przetwarzania danych o kandydatach, co pozwala na lepsze zrozumienie ich potrzeb i oczekiwań oraz umożliwia uczelniom lepsze dostosowywanie oferty edukacyjnej do rynku pracy. W ostatnich latach coraz częściej do rejestracji na studia wykorzystuje się aplikacje webowe, które umożliwiają szybsze i łatwiejsze składanie podania o przyjęcie na uczelnię oraz zarządzanie całym procesem rekrutacyjnym.

Podsumowując, system rejestracji na studia jest ważnym elementem procesu naboru na studia wyższe, odpowiadającym za selekcję kandydatów i wybór najlepszych z nich, a także za przeprowadzenie postępowań rekrutacyjnych zgodnie z obowiązującymi przepisami.

1.2. Istniejące rozwiązania

Jedynym systemem, do którego można by przyrównać koncepcję scentralizowanego systemu rekrutacji na studia, do którego udało się dotrzeć autorowi, jest elektroniczny system rekrutacji do szkół ponadpodstawowych stosowany w Polsce. Przytoczony system działa niestety dla każdego z województw niezależnie oraz na różnych zasadach, stąd też nie można mówić o centralizacji na poziomie kraju, a co najwyżej okręgu administracyjnego. Kuratorium Oświaty w Warszawie udzieliło następującej informacji na temat algorytmu, jaki jest zastosowany w tymże systemie:

O przyjęciu kandydata do danej klasy decyduje liczba punktów, a nie miejsce oddziału na liście preferencji. Jeżeli dla ucznia a szkoła i jest szkołą pierwszego wyboru, ale uzyskał

mniej punktów niż uczeń B, który tę szkołę wybrał np. Na 3 miejscu to zostanie przyjęty uczeń B bo ma więcej punktów. System elektroniczny „schodzi" po kolei po pozycjach na liście preferencji ułożonej przez ucznia sprawdzając, do którego oddziału ma wystarczającą liczbę punktów, która go zakwalifikuje do przyjęcia, np. Do pierwszej klasy – za mało punktów, do drugiej klasy z listy – za mało punktów, do trzeciej klasy – wystarczająca liczba punktów. Uczeń zostaje zakwalifikowany do tej klasy, system nie sprawdza już kolejnych klas. Czyli uczeń może zostać zakwalifikowany tylko i wyłącznie do jednej klasy, nie blokuje miejsc w innych klasach. Ważne jest więc ułożenie listy preferencji dla danego ucznia. [20]

Na podstawie powyższego opisu można domniemywać, że u podstaw tego systemu leży algorytm Gale'a Shapleya. Najprawdopodobniej zastosowano wariant, w którym szkoły (klasy) składają propozycje uczniom jako pierwsze. Wynikiem tak sformułowanego algorytmu będzie stabilne dopasowanie najlepiej odpowiadające szkołom, a niekoniecznie uczniom (zjawisko opisane dalej w rozdziale 2.1). Nie jest więc to pożądana sytuacja z perspektywy użytkowników tego systemu, czyli uczniów. Można by zastosować inną wersję algorytmu, która nie faworyzowałaby szkół a wybory uczniów.

Istniejące systemy rekrutacji na studia nie są scentralizowane, a w zamian są wysoce spersonalizowane i dostosowane do potrzeb poszczególnych uczelni. Każda z nich ma swój własny system, który jest dostosowany do jej specyfiki i potrzeb. Dzięki temu uczelnie mogą swobodnie zarządzać procesem rekrutacyjnym i dostosowywać go do swoich potrzeb. Wiele z tych systemów zawiera masę multimediów, takich jak filmy czy piękne grafiki, które zachęcają studentów do wyboru danej uczelni. Są one przygotowywane z myślą o kandydatach i mają na celu pokazanie im, jakie możliwości oferuje dana uczelnia oraz mają zachęcić ich do podjęcia studiów właśnie tam.

Istniejące systemy rekrutacji na studia umożliwiają też dowolne zbieranie danych o kandydatach. Dzięki temu uczelnie mogą lepiej poznać potrzeby i oczekiwania kandydatów oraz dostosować swoją ofertę edukacyjną do rynku pracy, bez dzielenia się tymi danymi z innymi. Systemy te są więc ważnym narzędziem do zarządzania procesem rekrutacyjnym i optymalizacji kosztów związanych z przyjmowaniem kandydatów na studia. Istniejące rozwiązania umożliwiają składanie podaniami o przyjęcie za pośrednictwem aplikacji webowych, co z pewnością jest wygodne dla uczelni.

Niestety, mimo że w Polsce istnieje wiele uczelni i każda z nich ma swój własny elektroniczny system rekrutacji na studia, brakuje jednego, scentralizowanego systemu, który umożliwiałby kandydatom składanie podań o przyjęcie na różne uczelnie w jednym miejscu. W obecnej sytuacji kandydaci muszą zapoznawać się z poszczególnymi systemami rekrutacji na każdej uczelni, na którą chcą się zapisać, co jest nie tylko czasochłonne, ale i frustrujące. Kandydaci muszą zapoznawać się z różnymi systemami i składać te same dane w wielu miejscach, co jest nie tylko niewygodne, ale też może prowadzić do pomyłek czy nieporozumień. Ponadto, warto zwrócić uwagę na fakt, że złożenie aplikacji w takich systemach jest płatne, a więc jeśli kandydat aplikuje w wiele miejsc, musi się liczyć ze znaczącym kosztem takiej decyzji.

Jedną z wad istniejących rozwiązań w procesie rekrutacji na studia jest fakt, że rekrutacja składa się z wielu tur, w których najlepsi kandydaci mogą być naraz przyjęci na wiele kierunków. To może prowadzić do sytuacji, w której miejsca na niektórych kierunkach zostają tymczasowo zarezerwowane dla takich kandydatów, a innym osobom nie pozostaje nic innego, jak czekać albo zrezygnować z dalszego ubiegania się o przyjęcie na wybrany kierunek lub szukać innych możliwości. W takiej sytuacji kandydaci, którzy przez to nie dostali się na wybrane przez siebie kierunki, mogą być zmuszeni do zmiany swoich wyborów lub całkowitego rezygnowania ze studiów. Warto jednak pamiętać, że taki student, który dostał się na kilka specjalizacji, na końcu najprawdopodobniej wybierze tę, która jest dla niego najbardziej preferowana, co może oznaczać zwolnienie miejsca na pozostałych kierunkach. Wszystko to może prowadzić do nieefektywnego wykorzystania miejsc na uczelniach i niezadowolenia wśród kandydatów.

1.3. Założenia i wizja projektu

Projekt składa się zasadniczo z dwóch głównych części. Pierwsza z nich to aplikacja webowa, reprezentująca elektroniczny system rejestracji na studia, z interfejsem zarówno dla kandydatów, jak i dla przedstawicieli uczelni. Druga zaś stanowi serce całego systemu, którą jest implementacja algorytmu odpowiedzialnego za optymalne i stabilne powiązanie kandydatów z wybranymi przez nich kierunkami studiów. Stosując ten podział, dla każdej z części możemy wyróżnić ich główne założenia.

Aplikacja webowa

- Interfejs aplikacji powinien być zaprojektowany w taki sposób, aby korzystanie z niego było sprawne i wygodne
- Uwierzytelnianie użytkowników i przechowywanie ich danych powinno spełniać nowoczesne normy bezpieczeństwa
- Dla kandydatów i dla uczelni powinny istnieć oddzielne role, dające dostęp do oddzielnych komponentów systemu
- Kandydat powinien mieć możliwość:
    - Utworzenia konta i zalogowania się
    - Wprowadzenia swoich wyników w nauce i osiągnięć
    - Przejrzenia oferty uczelni i wyboru pożądanych kierunków
    - Uszeregowania swoich preferencji (odnośnie wybranych kierunków studiów) od najbardziej pożądanych do najmniej
- Uczelnia powinna mieć możliwość:
    - Wyznaczenia swojego moderatora, odpowiedzialnego za stronę uczelni i opisy kierunków
    - Zalogowania bez użycia hasła, za pomocą linku weryfikacyjnego e-mail
    - Dodawanie, usuwanie i edycję swoich kierunków studiów, dostęp do ich opisów a także parametrów przyjęcia (mnożników)
- Aplikacja powinna prezentować w przyjaznej formie wyniki rekrutacji

- Administrator aplikacji powinien mieć pełny dostęp do danych zawartych w bazie, a także do monitorowania ruchu, blokowania użytkowników, przywracania haseł, nadawania i odbierania roli moderatorom uczelni.

Algorytm

- Powinien być implementacją uogólnionej wersji algorytmu odroczonej akceptacji Gale'a-Shapleya
- Na wejściu powinien przyjmować dane w formie zdefiniowanej przez aplikację webową
- Na wyjściu powinien zwracać dane, możliwe do przetworzenia przez aplikację webową, celem ich dalszej prezentacji
- Dane wejściowe powinien podzielić na dwie grupy:
    - Kandydatów, z ich wynikami i preferencjami odnośnie wybranych kierunków
    - Uczelni, z ich kursami i specyfiką szeregowania studentów (wynikami z procesu rekrutacji, określonych za pomocą liczby uzyskanych punktów)
- Wynikiem powinna być lista dla każdego kierunku studiów, z zakwalifikowanymi na niej aplikantami
- Implementacja powinna być optymalna, a kod napisany zwięźle i w sposób czytelny oraz zrozumiały dla innych programistów

Celem wymienionych założeń jest zapewnienie przyjaznego w użytkowaniu systemu scentralizowanej rejestracji na studia, który spełni oczekiwania użytkowników. Należy jednak pamiętać, że ten system nie będzie w stanie zastąpić wszystkich istniejących implementacji, gdyż jest to jedynie skromna, funkcjonalna koncepcja takiego projektu.

2. Algorytm GS

Algorytm Gale'a-Shapleya jest algorytmem do rozwiązywania problemu małżeństwa (ang. stable marriage problem). Został opracowany przez dwóch amerykańskich matematyków: Davida Gale'a i Lloyda Shapleya w 1962 roku [2]. Ten algorytm należy do grupy algorytmów, których wynikiem jest stabilne dopasowanie (ang. stable matching). Stabilne dopasowanie to takie rozwiązanie problemu, w którym żadna para uczestników nie chciałaby zamienić się partnerami z inną parą, ponieważ każda z nich jest zadowolona ze swojego obecnego partnera.

Algorytm Gale'a-Shapleya jest również znany jako algorytm odroczonej akceptacji (ang. deferred acceptance algorithm). Jest to nazwa, która odnosi się do sposobu działania algorytmu, w którym uczestnicy nie podejmują natychmiastowej decyzji o przyjęciu lub odrzuceniu propozycji, ale zamiast tego odkładają swoją decyzję do momentu, gdy otrzymają wszystkie propozycje i mogą je porównać. Dzięki temu algorytm jest w stanie zapewnić stabilność rozwiązania, ponieważ każdy uczestnik ma możliwość zapoznania się z wszystkimi propozycjami i wybrania tej, która najbardziej mu odpowiada.

Algorytm ten jest często stosowany w modelowaniu rynków pracy i problemów związanych z asymetrią informacji. Jest również używany do rozwiązywania problemów związanych z przydzielaniem zasobów, takich jak przydział mieszkań czy przydział pacjentów do szpitali. Algorytm Gale'a-Shapleya ma kilka wariantów, w zależności od kontekstu problemu, który ma rozwiązać [3].

2.1. Podstawowy algorytm GS

Załóżmy, że mamy dwie grupy osób: mężczyzn i kobiet, a każda osoba z jednej grupy ma swoje preferencje dotyczące osób z drugiej grupy. Chcemy stworzyć stabilne małżeństwa, gdzie każda osoba jest przypisana do jednej osoby z drugiej grupy. Algorytm Gale'a-Shapleya działa w następujący sposób:

1. Każdy mężczyzna i kobieta tworzą listę preferencji: mężczyźni w kolejności preferencji kobiet, a kobiety w kolejności preferencji mężczyzn.
2. Każdy mężczyzna wysyła propozycję do swojej ulubionej kobiety na liście preferencji.
3. Każda kobieta rozważa propozycję i odrzuca ją, jeśli otrzymała już lepszą propozycję. W przeciwnym razie kobieta akceptuje propozycję.
4. Jeśli kobieta odrzuciła propozycję, mężczyzna wysyła propozycję do swojego następnego wyboru na liście preferencji i proces się powtarza.
5. Algorytm kończy się, gdy każda kobieta ma już jednego partnera.

Ten algorytm gwarantuje, że otrzymamy stabilne pary, czyli takie, które nie mają motywacji, aby zmienić swojego partnera, ponieważ każdy preferuje swojego obecnego partnera nad każdym innym. Stabilność jest ważna, ponieważ w przypadku braku stabilności mogą wystąpić sytuacje, w których dwie osoby chcą zmienić swojego partnera, co prowadzi do niezadowolenia i konfliktów [4].

Podstawowa wersja algorytmu może zostać opisana w następujący sposób [5]:

```
initialize m∈M jako wolny
while ∃ wolny m posiadający k∈K, której może się oświadczyć:
     k := pierwsza kobieta z listy m, której się nie oświadczył
     if ∃ para (m', k) then
          if k woli m nad m' then
               m' staje się wolny
               (m, k) zostają zaręczeni
          endif
     else
          (m, k) zostają zaręczeni
     endif
     repeat
```

Powyższy pseudokod pokazuje w mocnym uproszczeniu, w jaki sposób działa algorytm (brakuje np. definicji listy preferencji), niemniej stanowi on klucz do zrozumienia jego implementacji, przedstawionej w dalszej części pracy.

Dowód

Dowód poprawności algorytmu polega na pokazaniu, że wynikowy zbiór dopasowań jest stabilny, czyli nie ma takich par, które chciałyby ze sobą być w parze bardziej niż z obecnym partnerem, ale nie są w dopasowaniu.

Aby udowodnić stabilność dopasowania, przyjmijmy, że istnieją dwie pary, (m, w) i (m', w'), które chciałyby ze sobą być w parze bardziej niż z obecnymi partnerami. Oznacza to, że m preferuje w' od swojej obecnej partnerki i w' preferuje m od swojego obecnego partnera, a także że w preferuje m' od swojego obecnego partnera i m' preferuje w od swojej obecnej partnerki.

Jeśli m proponował w' przed swoją obecną partnerką, to w' by go zaakceptowała, a następnie odrzuciła swojego obecnego partnera na rzecz m, co oznacza, że w' i m tworzą stabilną parę. Podobnie, jeśli w' proponowała m przed swoim obecnym partnerem, to m by ją zaakceptował, a następnie odrzucił swoją obecną partnerkę na rzecz w', co oznacza, że m i w' tworzą stabilną parę.

W ten sam sposób można wykazać, że pary (m', w) i (m, w') nie są stabilne. Wszystko to prowadzi do wniosku, że zbiór dopasowań wynikający z algorytmu Gale'a-Shapleya jest stabilny.

Przykład

Weźmy macierz opisującą preferencje trzech kobiet (a, b, c) i trzech mężczyzn (A, B, C) Niech pierwsza liczba oznacza ranking danego mężczyzny względem kobiety, a druga ranking danej kobiety względem mężczyzny.

	A	*B*	*C*
a	1,3	2,2	3,1
b	3,1	1,3	2,2
c	2,2	3,1	1,3

Dla n kobiet możemy wybrać po jednym z n mężczyzn na n! sposobów. dla n = 3 mamy 3! a więc 6. Zatem istnieje 6 możliwych par związków, z czego (dla powyższej macierzy) istnieją 3 stabilne dopasowania:

1. (a, A), (b, B), (c, C) - Każda kobieta wybiera najbardziej preferowanego mężczyznę (choć każdy mężczyzna ma najmniej preferowaną partnerkę)
2. (a, C), (b, A), (c, B) - Każdy z mężczyzn wybiera najbardziej preferowaną kobietę (choć każda kobieta ma najmniej preferowanego partnera)
3. (a, B), (b, C), (c, A) - Każdy jest ze swoim “drugim wyborem”

Na powyższym przykładzie widać, że dla jednego zbioru może istnieć wiele stabilnych dopasowań. To, dla kogo jest ono najkorzystniejsze, zależy od tego, kto składa propozycję. Jeśli to mężczyźni przedstawiają się kobietom, to uzyskane zostanie dopasowanie najbardziej korzystne dla mężczyzn (i vice versa).

Wniosek jest taki, że niezależnie w jaki sposób uruchomimy nasz algorytm, otrzymamy stabilne dopasowanie – jest ono gwarantowane. Niemniej należy pamiętać o fakcie, że jedna strona może być z tego dopasowania bardziej zadowolona niż druga.

2.2. Rozszerzenia algorytmu GS

Algorytm Gale'a-Shapleya ma kilka wariantów, które rozwiązują różne problemy. Wśród nich możemy wymienić takie jak:

- 1-to-N Matching Problem - dotyczy przyporządkowania jednej kobiety do N mężczyzn.
- N-to-M Matching Problem - dotyczy przyporządkowania dowolnej liczby mężczyzn do dowolnej liczby kobiet.
- Partial Matching Problem - dotyczy znalezienia jak największej liczby stabilnych par, ale nie jest konieczne, aby każdy mężczyzna miał partnera lub każda kobieta miała partnera.
- Stable Roommates Problem - dotyczy znalezienia stabilnego rozwiązania dla grupy osób, które muszą dzielić ze sobą mieszkanie w parach.
- Hospitals/Residents Problem - dotyczy przyporządkowania mieszkańców do szpitali w oparciu o preferencje mieszkańców i dostępność miejsc w szpitalach.

- College Admissions Problem - dotyczy przyporządkowania uczniów do szkół zgodnie z preferencjami uczniów i wymaganiami szkół

Istnieje wiele więcej wariantów tego algorytmu, gdyż jest on w sporej mierze podatny na modyfikacje. Jest to jedna z jego zalet, gdyż możliwe jest dostosowanie potrzeb i wymagań konkretnej sytuacji do kształtu algorytmu. W każdym z tych problemów algorytm działa w podobny sposób, ale z odpowiednimi modyfikacjami w zależności od kontekstu problemu.

2.3. Dostosowanie wersji algorytmu do problemu

W przypadku tego projektu mamy do czynienia z przydziałem studentów do uczelni. W tym rozwiązaniu uczestnikami są studenci i uczelnie (konkretne kierunki na uczelniach) – odpowiednio można ich traktować jako *mężczyźni* i *kobiety*. Charakterystyka problemu znajduje swoje rozwiązanie w wariancie algorytmu pod nazwą *College Admissions Problem.*

Każda uczelnia tworzy listę swoich preferencji co do studentów, podobnie jak każdy student tworzy listę swoich preferencji co do uczelni (oddzielne dla każdego kierunku studiów). Przez listę preferencji studentów rozumiemy zwykłą kolejkę priorytetową, w której studenci układają uczelnie (kierunki) zgodnie z ich własnymi preferencjami. Przez listę preferencji uczelni rozumiemy konkretny wzór, na podstawie którego obliczana jest pozycja kandydata w rankingu. W obecnej wersji na wzór składa się suma kolejnych mnożników wyników matur, uzyskanych przez kandydatów.

Algorytm Gale'a-Shapleya dla tego problemu działa na podobnej zasadzie, co w przypadku problemu stabilnego małżeństwa. Każdy student proponuje uczelniom swoją kandydaturę, a następnie uczelnie dokonują wyboru i akceptują najlepszych kandydatów. Proces ten powtarza się, aż do momentu, gdy nie będzie już wolnych miejsc lub każdy student i uczelnia znajdą swojego partnera (lub grupę partnerów).

Jednakże, w przypadku problemu dopasowywania studentów do uczelni, istnieje dodatkowy wymóg, którym jest pojemność uczelni. To oznacza, że nie każdy student może zostać przyjęty na każdą uczelnię. W takim przypadku, pojawia się potrzeba wprowadzenia dodatkowych warunków do algorytmu, aby uwzględnić ograniczenia pojemnościowe. W związku z tym każda uczelnia powinna prowadzić listę studentów, którzy zostali przyjęci, a także listę studentów, którzy otrzymali propozycję, ale zostali odrzuceni (zaimplementowana w nieco inny sposób). W przypadku, gdy liczba studentów, którzy zostali przyjęci, osiągnie maksymalną pojemność uczelni, nie powinna ona już akceptować dalszych propozycji.

Wynikiem działania algorytmu jest stabilne dopasowanie między studentami a uczelniami, uwzględniające ograniczenia pojemnościowe uczelni. Algorytm ten może być z powodzeniem stosowany do wielu różnych problemów dopasowania w praktyce, takich jak rekrutacja na studia, rekrutacja pracowników czy dobieranie pacjentów do lekarzy.

Dowód

Dowód poprawności algorytmu w wersji *College Admissions Problem* polega na pokazaniu, że wynikowe dopasowanie jest stabilne, a także że nie istnieje lepsze dopasowanie, które uwzględniałoby pojemność uczelni.

Aby pokazać stabilność wynikowego dopasowania, przyjmijmy, że istnieją dwa stabilne dopasowania między studentami a uczelniami: D1 i D2. Z założenia o stabilności, w każdym z tych dopasowań, nie ma pary studentów i uczelni, którzy chcieliby pozostać razem w innym dopasowaniu. Oznacza to, że każdy student i uczelnia są zadowoleni ze swojego obecnego partnera w dopasowaniu. Załóżmy teraz, że istnieje student S i uczelnia U, którzy nie są razem w żadnym stabilnym dopasowaniu, ale chcieliby być razem. Oznacza to, że w dopasowaniu D1, student S jest przydzielony do uczelni U', a w dopasowaniu D2, uczelnia U przyjmuje studenta S' (innego niż S). Ponieważ S i U chcą być razem, S preferuje U przed U', a U preferuje S przed S'. Zatem, w obu dopasowaniach D1 i D2, student i uczelnia chcieliby pozostać razem. W tym przypadku, dopasowanie D1 lub D2 nie jest stabilne, co stoi w sprzeczności z naszym założeniem.

Pokażmy teraz, że nie istnieje lepsze dopasowanie, które uwzględniałoby pojemność uczelni. Przypuśćmy, że istnieje lepsze dopasowanie D', w którym każda uczelnia ma taką samą liczbę studentów lub więcej niż w wynikowym dopasowaniu D. Oznacza to, że w D' każdy student, który jest w D, pozostaje przy swojej uczelni lub zostaje przyjęty na innej uczelni. Jeśli każda uczelnia w D' ma więcej lub tyle samo studentów, to każdy student w D ma tę samą lub lepszą uczelnię w D'. Zatem D nie jest gorsze od D'. Jeśli jakieś uczelnie w D' mają mniej studentów niż w D, to musiały one odrzucić niektórych studentów, którzy zostali przyjęci do tych uczelni w D. Oznacza to, że te uczelnie były w stanie zaakceptować studentów, którzy zostali odrzuceni w D', a co za tym idzie, mogą one jeszcze bardziej zwiększyć swoją pojemność. W ten sposób można doprowadzić do uzyskania dopasowania D', w którym każda uczelnia ma co najmniej tyle samo studentów, co w D, co oznacza, że D nie jest gorsze niż D'. W ten sposób pokazaliśmy, że wynikowe dopasowanie D jest stabilne i nie istnieje lepsze dopasowanie, które uwzględniałoby pojemność uczelni [6].

Podsumowując, algorytm Gale'a-Shapleya z uwzględnieniem pojemnościowych ograniczeń prowadzi do stabilnego dopasowania między studentami a uczelniami, które uwzględnia te ograniczenia. A zatem algorytm w zmodyfikowanej wersji pozostaje stabilny.

2.4. Implementacja algorytmu

Do implementacji algorytmu zastosowano język TypeScript i wykorzystano paradygmat programowania obiektowego. Na przygotowany program składa się kilka klas modelujących rzeczywisty problem, który rozwiązuje algorytm GS, w tym klasy reprezentujące uczelnię, poszczególne kierunki i kandydata. Sam program nie jest częścią aplikacji webowej, chociaż dzięki wyborowi języka TypeScript możliwe będzie w przyszłości połączenie części algorytmicznej z resztą projektu.

Kwintesencja algorytmu została zawarta w pliku *algorithm.ts* i przedstawia ona szkielet działania całego programu. Poniżej znajduje się omawiany fragment:

```
/**Runs Gale-Shapley algorithm. Updates property colleges, from
which results may be extracted later. */
run(): void {
 let free_students = [...this.getStudentsList()];
 while (free_students.length>0) {
  const fs = [...free_students];
  free_students = [];
  fs.forEach((student) => {
   const pref: Pref | undefined = student.currentPref();
   if (typeof pref === undefined) return;
   const coll: College = this.colleges.get(pref.getCollegeId());
   const cour: Course = coll.getCourse(pref.getCourseId());
   if (!cour.isFull()) {
      cour.enrollStudent(student.getId());
   } else {
      const bStudent: Student = this.students
                                  .get(cour.getLastEnrolled());
      if (!bStudent) return;
      if (cour.getScore(bStudent.getId()) <
                cour.getScore(student.getId())) {
            cour.removeLastEnrolled();
            cour.enrollStudent(student.getId());
            student = bStudent;
      }
      student.shiftPrefs();
      if (student.hasPrefsLeft()) {
            free_students.push(student);
      }
   }
  });
 }
}
```

W pierwszym kroku tworzona jest zmienna *free_students*, w której przechowywana będzie lista studentów, którzy nie zostali przypisani do żadnego z kursów. Na początku, żaden z kandydatów nie jest przypisany, zatem dla potrzeb pierwszej iteracji, do zmiennej *free_students* zostają przypisani wszyscy studenci. Dopóki lista ta nie jest pusta, co jest sprawdzane w warunku pętli *while*, wykonywane są kolejne iteracje programu.

Tworzona jest kopia zmiennej *free_students* pod nazwą *fs*, po której to wykonaniu zmienna *free_students* jest zastępowana pustą tablicą. Nie zmienia ona swojej funkcji i wciąż

przechowuje listę nieprzypisanych kandydatów. W dalszej części algorytmu będzie ona uaktualniana, gdy okaże się, że jakiś kandydat jest nieprzypisany.

Rozpoczyna się zasadnicza pętla algorytmu *forEach*, iterująca po kolejnych studentach ze zbioru *fs*. Dla każdego studenta po kolei wykonywane są kolejne kroki:

Do zmiennej *pref* pobierany jest obiekt reprezentujący aktualną (najwyższą nieodrzuconą) preferencję studenta. O ile istnieje, pobierane są informacje o ID uczelni oraz ID kursu. Następnie, na podstawie uzyskanych identyfikatorów, w zmiennej *cour* zapisywany jest obiekt klasy *Course*, a w zmiennej *coll* obiekt klasy *College* odpowiadający konkretnej uczelni, do której ten kurs należy.

W tym miejscu pojawia się instrukcja warunkowa *if* sprawdzająca, czy w danym kursie są jeszcze wolne miejsca. Jeśli kurs nie jest pełny, obecny student zostaje do niego tymczasowo przypisany, a algorytm przechodzi do iteracji z kolejnym studentem. W przeciwnym wypadku przechodzimy do klauzuli *else*, która oznacza, że w obecnej chwili wszystkie miejsca w tym kursie są zajęte.

Z listy (kolejki priorytetowej) studentów zapisanych na ten kurs, pobierany jest do *Student* ostatni student (z najgorszym wynikiem, najmniej preferowany przez uczelnię). W instrukcji warunkowej *if* porównywane są wyniki osiągnięte przez *student* oraz *bStudent*.

W przypadku, gdy *student* względem tego kursu uzyskał lepszą ocenę uczelni niż *bStudent*, zostają oni zamienieni miejscami. Oznacza to, że *student* zostaje wpisany na listę kandydatów danego kursu zamiast tego drugiego. Na koniec tego kroku, do zmiennej *student*, przypisuje się *bStudent*, co pozwala uniknąć w dalszej części zdublowanego kodu. Warto zaznaczyć, że w zależności od uzyskanej liczby punktów, *student* znajdzie się na odpowiedniej pozycji danej listy, wynikającej z kolejki priorytetowej (nie koniecznie ostatniej).

W przeciwnym przypadku, gdy *bStudent* uzyskał lepszy wynik, krok z poprzedniego akapitu nie jest wykonywany.

Dalej, dochodzimy do momentu, gdy odpowiedni student został już na pewno dopisany do listy na odpowiedniej pozycji, a w zmiennej *student* znajduje się obiekt, reprezentujący kandydata, który nie zakwalifikował się do danego kursu. W związku z tym, kolejnym krokiem algorytmu jest usunięcie obecnej preferencji, z listy tego kandydata. Jeśli kandydat posiada jeszcze inne preferencje na swojej liście (sytuacja, w której ich nie wyczerpał), zostaje on na nowo dopisany do listy *free_students i* weźmie udział w kolejnych iteracjach algorytmu.

W przeciwnym razie zachodzi scenariusz, w którym zgłoszone przez studenta preferencje, wyniki jego matur, jak i mnożniki ustalone przez uczelnię, nie pozwoliły na dostanie się na żaden z wybranych kierunków studiów. Tym samym, nie pojawi się on na liście żadnego z kursów jako ich kandydat.

Opisana metoda *run()* nie zwraca żadnej wartości. Wynikiem jej działania jest zaktualizowane pole *colleges*, zawierające informacje o wszystkich uczelniach i ich kursach, które to w trakcie działania algorytmu zostały zaktualizowane o listę kandydatów.

2.5. Działanie algorytmu w praktyce

Przygotowanie danych do testu

Po zaimplementowaniu algorytmu pojawiło się pytanie o to, jak w zasadzie poradzi on sobie z rzeczywistymi danymi i czy uzyskane przez niego wyniki będą w jakikolwiek sposób przydatne. Aby zweryfikować założenia teoretyczne w praktyce, należało przygotować szereg testów, które odpowiadałyby na te pytania. Wstępne wyniki z małych i średnich testów prowadzonych w trakcie implementacji były zadowalające, jednak wciąż pozostawała kwestia większych testów odpowiadających realiom, w jakich scentralizowany system rekrutacji na studia miał funkcjonować.

Pierwszym rzeczywistym testem miało być sprawdzenie, jak algorytm zachowa się dla jak najbardziej realnych danych, przy zapisach prowadzonych na terenie całego kraju. Plik testowy był generowany z wykorzystaniem mechanizmów randomizacji, dzięki czemu można było wielokrotnie generować podobne testy. W tym procesie wykorzystano garść narzędzi dostarczanych przez statystykę, aby jak najlepiej oddać naturę dużej populacji i wyborów dokonywanych przez jednostki. Między innymi do generowania liczby kierunków na uczelni, wyników matur czy preferencji studentów wykorzystano różne rozkłady: Gaussa, beta, trójkątny i jednostajny [7]. Wszystko po to, aby wygenerowane dane w większym stopniu przypominały rzeczywiste dane, możliwe do uzyskania przez system, a zatem lepiej odpowiadały na pytanie o zasadność stosowania algorytmu.

Test został wygenerowany w oparciu o poniższe sztywne założenia:

- Liczba studentów = 268257 (oficjalna liczba maturzystów w 2022 roku [17])
- Liczba uczelni = 43 (oficjalna liczba uniwersytetów w Polsce [18], nie liczba uczelni, gdyż ich istnieje ponad 400, co byłoby trudne w analizie)
- Liczba przedmiotów maturalnych = 20 (3 podstawowe + 17 rozszerzeń, zgodnie z CKE [19])

Ponadto do generowania testu wykorzystano kilka dynamicznych parametrów:

- Liczba kierunków na uczelni = (30, 150, 60)
- Dopuszczalna liczba studentów na kierunku = (15, 200, 40)
- Liczba matur, do których podszedł kandydat = (4, 9, 4)
- Liczba kierunków, na które zapisał się kandydat = (1, 20, 4)
- Liczba mnożników na kierunku = (2, 8, 3)

Powyższe parametry określają przyjęty rozkład danych wartości w konkretnej populacji. Sprecyzowane są przez trzy wartości (p, k, m), gdzie: p - początek przedziału, k – koniec przedziału, m – modalna rozkładu trójkątnego. Przykładowo: liczba matur, do których podszedł kandydat, określona jako (4, 9, 4) oznacza, że każdy kandydat podszedł

przynajmniej do 4 przedmiotów (3 podstawowe i 1 rozszerzenie) a maksymalnie zdawał 9 przedmiotów (3 podstawowe i 6 rozszerzeń) - co zgodne jest z wytycznymi CKE. Ostatnia wartość oznacza, że maturzyści najczęściej zdawali 4 przedmioty.

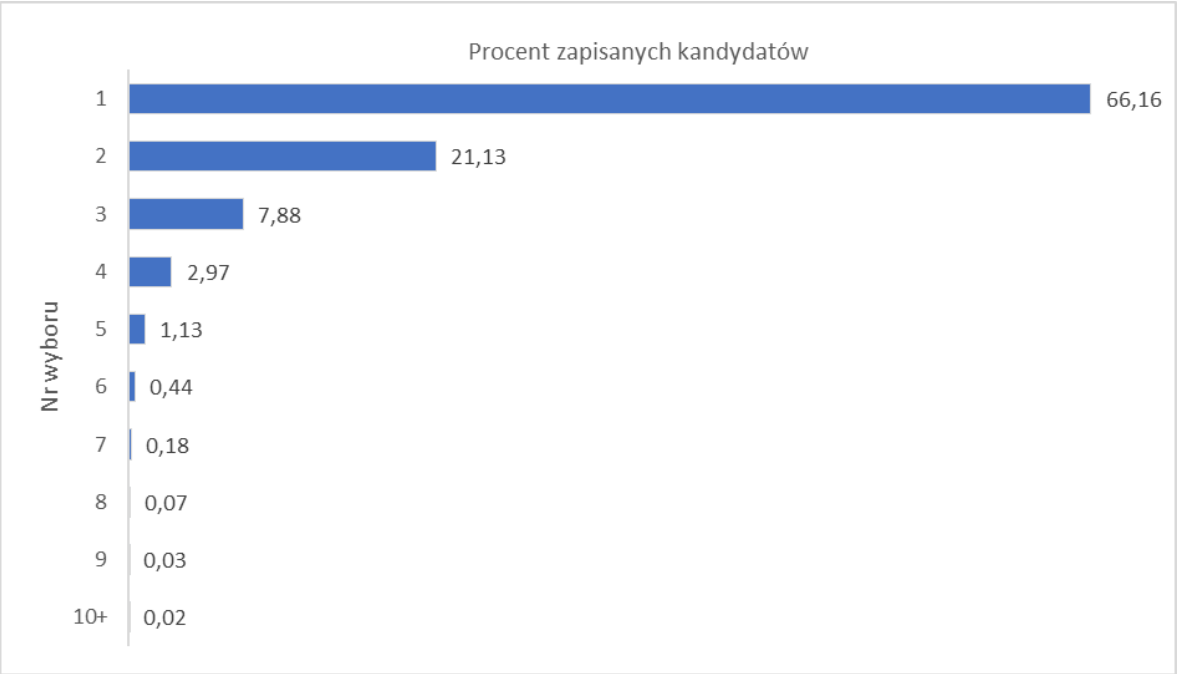
Na podstawie sprecyzowanych parametrów wykorzystując skrypt napisany w Pythonie, otrzymano plik z danymi, który zawierał: łącznie3427 kierunków studiów przypisanych do 43 uczelni, na których znalazło się 291307 miejsc dla 268257 kandydatów. Dane kandydatów wypełniono łącznie 1394092 wynikami matur i wygenerowano dla nich 2101407 preferencji co do kierunków. Całość, wraz z zapisaniem tych danych do pliku JSON, zajęła 91.34 sekundy.

Analiza wyników dla testowych danych

Co ciekawe, mimo obszernego pliku testowego(515.6 MB) i sporej ilości zawartych w nim danych, uruchomienie algorytmu wykonało się bardzo sprawnie. Zasadnicza część działania algorytmu (poza wczytywaniem i zapisem) zajęła zaledwie 1054 ms. Wyniki, jakie uzyskano wskutek działania algorytmu Gale'a Shapleya, przedstawiają się następująco:

- 97.58%kandydatów zostało zapisanych ogółem
- 66.16% zapisanych kandydatów było z pierwszego wyboru
- 95.16% zapisanych kandydatów było z co najwyżej trzeciego wyboru
- 10.14% miejsc na uczelniach pozostało wolnych (część była nadmiarowa)
- 47:1 wynosił maksymalny stosunek chętnych kandydatów do zapisanych na jednym kierunku
- 3.8:1 wynosił minimalny stosunek chętnych kandydatów do zapisanych na jednym kierunku
- 127 wolnych miejsc pozostało najwięcej na pojedynczym kierunku
- 1 kandydat został zapisany z 15, najdalszego wyboru

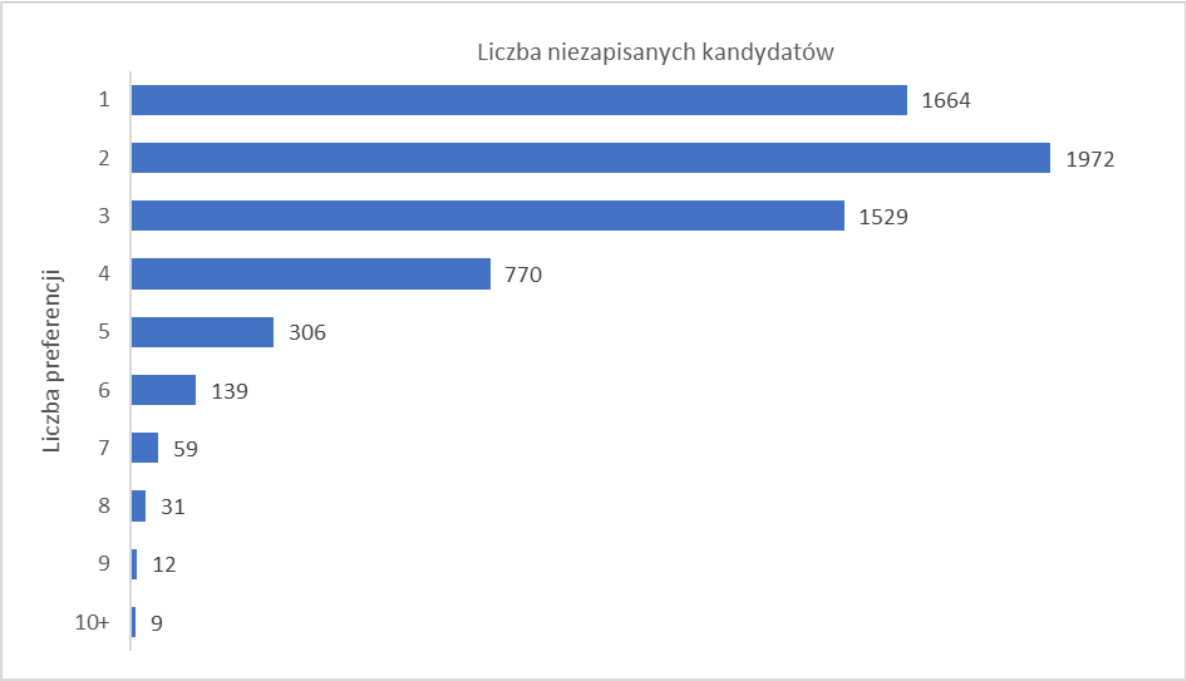
Poniższy wykres przedstawia z których pozycji preferencji kandydaci byli zapisywani przez algorytm na wybrane przez siebie kierunki:



Rys.1 Wykres decydującej pozycji preferencji dla zapisanych kandydatów

Jak widać po powyższym wykresie, osiągnięcie stabilnego dopasowania w przypadku tak zróżnicowanych danych wcale nie wymagało poświęcenia pierwszych wyborów dla większości kandydatów. Algorytm świetnie się sprawdził, jeśli chodzi o przydział miejsc dla zdefiniowanych preferencji. Samo zapisanie 97,6% kandydatów wydaje się dobrym wynikiem, a ponadto tak wielu z nich zostało zapisanych na te kierunki, na które najbardziej chcieli się dostać. Widać też, że nawet w przypadku zajęcia miejsc przez lepszych studentów na niektórych kierunkach algorytm dobrze poradził sobie z przypisaniem słabszym uczniom ich dalszych preferencji.

Następny wykres przedstawia liczbę preferencji, jakie zgłosili kandydaci, którzy nie zostali zapisani na żaden z wybranych kierunków studiów:



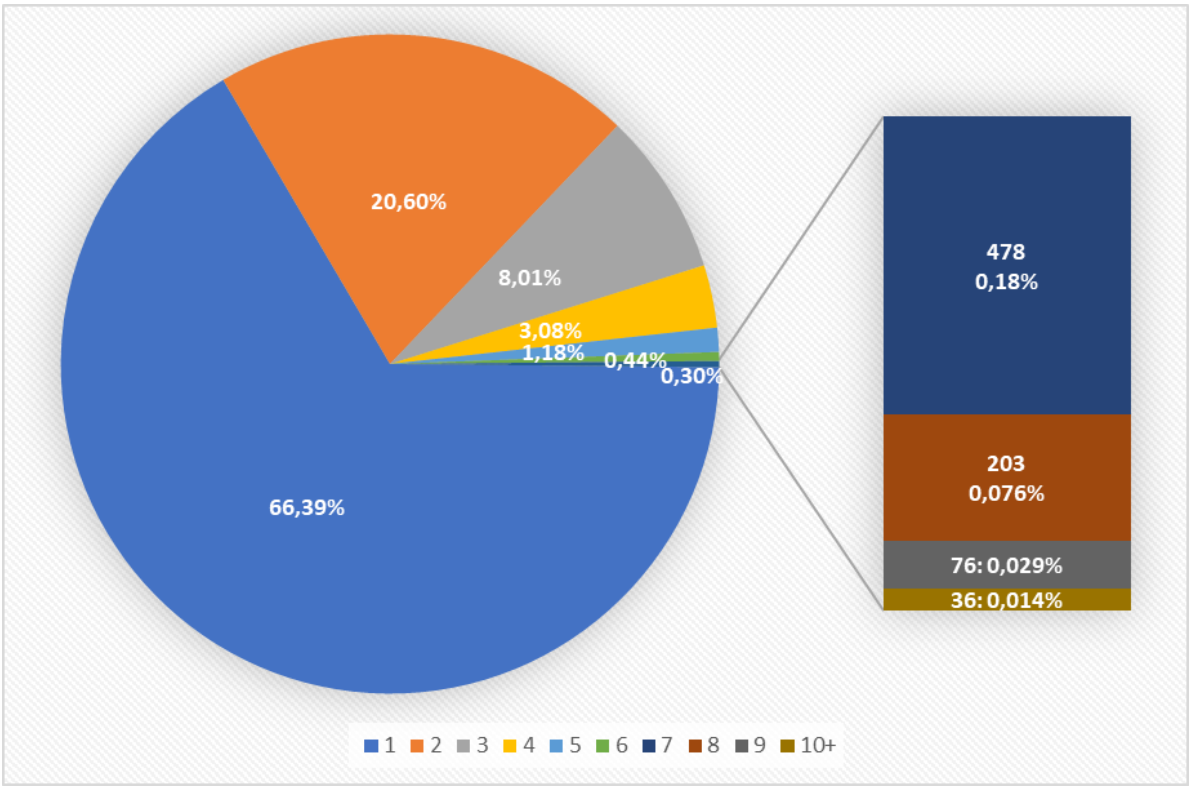
Rys.2 Liczba zgłoszonych preferencji przez niezapisanych kandydatów

Spoglądając na powyższy wykres, należy uwzględnić fakt, że studenci najczęściej zgłaszali 4 preferencje, gdyż taka była zdefiniowana wartość modalna w rozkładzie trójkątnym dla liczby preferencji. Mając to na uwadze, można bardzo dobrze zauważyć, jak zwiększenie zgłoszonych liczby preferencji, zwiększa szanse kandydata na zapisanie się na dowolny kierunek. Większość kandydatów, którzy się nie zapisali, posiadała małą liczbę preferencji i zapewne słabe wyniki matur w stosunku do wybranych kierunków. Osoby, które posiadały między jedną, a trzema preferencjami stanowiły 45% całej populacji kandydatów, natomiast posiadają udział w wysokości 80% wśród osób niezapisanych, co jest najlepszym dowodem na to, że większa liczba preferencji sprzyja kandydatom, a ograniczenie ich liczby poważnie wpływa na szanse stania się studentem.

Biorąc dotychczasowe wyniki pod uwagę, przygotowano drugi plik testowy, bardzo podobny do pierwszego, narzucając jednak na kandydatów wymóg, aby zgłaszali oni co najmniej 3 preferencje. Pozostałe parametry nie uległy zmianom. Podczas generowania testu zmieniła się część losowych parametrów, w tym uległa minimalnemu zwiększeniu liczba dostępnych miejsc na kierunkach z 291307 do 294443 (+3136). Wskutek wprowadzonego wymogu uległa zmianie również liczba preferencji do tego stopnia, że maksymalny stosunek chętnych kandydatów do zapisanych na jednym kierunku wzrósł do 71:1. Większa liczba preferencji przełożyła się również na dłuższy czas działania algorytmu, który wyniósł 1203 ms (+14%).

W wyniku działania algorytmu zostało zapisanych 265936 z 268257, a więc 99.13% wszystkich kandydatów. Stanowi to znaczący wzrost w porównaniu do wyniku 97.58%, uzyskanego w pierwszym teście. Nawet mimo zwiększenia liczby miejsc na kierunkach,

końcowa liczba wolnych miejsc jest mniejsza niż w poprzednim teście i wynosi 28507 (9.68%) w porównaniu do 29541 (10.14%).



Rys.3 Wykres pozycji preferencji, z której kandydaci zostali zapisani

Z powyższego wykresu widać, że poprawie uległy wszystkie wartości. Nałożenie dodatkowego ograniczenia na kandydatów w trakcie definiowania ich preferencji poprawiło sprawność całego algorytmu i przełożyło się na uzyskanie lepszych wyników pod każdym względem, zarówno dla przyszłych studentów, jak i dla uczelni.

Z pewnością nałożenie na kandydatów wymogu zgłaszania większej liczby preferencji sprawiłoby, że więcej z nich dostawałoby się na wybrane kierunki. Jednak kwestia czy faktycznie te dziesięć czy więcej preferencji byłoby rzeczywistym oddaniem zainteresowań kandydatów? Wynik oceniający skuteczność algorytmu, o ile od strony matematycznej byłby wyśmienity, to odczucia osób korzystających z systemu mogłyby nie być pozytywne. Z tego względu w implementowanym przeze mnie systemie postanowiłem nie nakładać żadnych ograniczeń czy limitów na kandydatów, a powyższe uwagi pozostawić w sferze rozważań teoretycznych.

3. Zastosowane technologie

Ta część jest przeznaczona głównie dla programistów lub osób chcących dowiedzieć się więcej o całym projekcie. Zawiera techniczny opis aplikacji, wykaz użytych narzędzi wraz z uzasadnieniem wyboru, a także opis struktury projektu.

Głównym celem przy tworzeniu tej aplikacji było skierowanie jej do jak największej liczby odbiorców, dlatego zdecydowałem się na stworzenie strony internetowej dostępnej na różnych urządzeniach i przeglądarkach. Po przeanalizowaniu założeń i wizji projektu, zdecydowałem się na wybór technologii i języka programowania najlepiej pasujących do tworzenia takiej aplikacji.

Spośród wielu dostępnych na rynku technologii, wybrałem TypeScript jako główny język programowania, ponieważ jest nadzbiorem języka JavaScript, a dodaje do niego typowanie statyczne, co umożliwia lepsze zarządzanie kodem i eliminację błędów na etapie pisania kodu. Do tworzenia interfejsu użytkownika wybrałem bibliotekę React.js, pozwalającą na efektywne zarządzanie stanem aplikacji i tworzenie dynamicznych stron internetowych, wraz z frameworkiem Bootstrap, umożliwiającym stworzenie responsywnego interfejsu o przyjaznym wyglądzie, który dostosowuje się do różnych rozmiarów ekranów. Platforma Firebase została wybrana jako baza danych i rozwiązanie z zakresu uwierzytelniania. Firebase jest szczególnie przydatny przy tworzeniu aplikacji, które wymagają szybkiego dostępu do danych i integracji z innymi usługami, a także przy aplikacjach, które wymagają skalowalności i niezawodności, co było kluczowe przy moich założeniach. W kwestii implementacji algorytmu narzucającym się wyborem był język TypeScript, aby zachować spójność z całym projektem i być może w przyszłości wykorzystać algorytm jako funkcję chmury. Chciałem, aby środowisko programistyczne było otwarte, dlatego wybrałem VisualStudio z systemem Linux jako główne narzędzia do pracy, które są niezwykle lekkie, stabilne i oferują możliwość personalizacji, co jest kluczowe, aby proces wytwarzania oprogramowania przebiegał sprawnie.

Warto również zwrócić uwagę na kilka wad związanych z wyborem takiego stosu technologicznego. Po pierwsze, korzystając z Firebase jako platformy do przechowywania danych, trzeba pamiętać, że jest to usługa chmurowa i w przypadku utraty połączenia z siecią, użytkownicy nie będą mieli dostępu do aplikacji. Po drugie, stosowanie gotowych rozwiązań takich jak React.js i Bootstrap może ograniczać elastyczność i możliwości personalizacji interfejsu użytkownika, co może być problematyczne w przypadku projektów wymagających bardziej unikalnego i skomplikowanego interfejsu. Wreszcie, korzystanie z otwarto-źródłowych narzędzi takich jak VisualStudio na systemie Linux, może wymagać większej wiedzy technicznej i umiejętności, co może stanowić barierę dla początkujących programistów. Niemniej, żaden stos technologiczny nie jest idealny, a świadomość wad i ograniczeń pozwala na ich skuteczne przezwyciężanie w trakcie projektowania i wdrażania rozwiązań informatycznych [8].

3.1. Typescript

TypeScript jest językiem programowania stworzonym przez firmę Microsoft [9], który rozszerza możliwości JavaScript. Sam JavaScript jest językiem skryptowym, który jest głównie stosowany do tworzenia interaktywnych elementów stron internetowych. Pozwala na dodawanie dynamicznych elementów, takich jak animacje, formularze, gry i wiele innych, bez konieczności odświeżania całej strony. Jest on obecnie jednym z najpopularniejszych języków programowania na świecie i jest wbudowany we wszystkie przeglądarki internetowe. TypeScript z kolei umożliwia statyczne typowanie, co oznacza, że typy danych są sprawdzane podczas kompilacji, a nie podczas działania programu. Dzięki temu możliwe jest wykrywanie błędów w kodzie na etapie jego tworzenia, co ułatwia jego późniejsze utrzymanie. TS rozszerza JS o takie elementy jak interfejsy, klasy czy enkapsulację. Dzięki nim programiści mogą korzystać z zaawansowanych technik programowania obiektowego, takich jak dziedziczenie czy polimorfizm. To szczególnie przydatne, gdyż lepsza organizacja i struktura kodu ułatwiają jego utrzymanie i rozwój. TS jest w pełni kompatybilny z JS i może być bezpośrednio uruchamiany w przeglądarce lub na innych urządzeniach. Jest on coraz bardziej popularny wśród programistów, zarówno przy tworzeniu aplikacji internetowych, jak i na serwerze. W ostatnich latach był językiem, który najszybciej zyskiwał popularność (wzrost z 12% użycia w 2017 do 34% w 2022) [10]. Wspierany jest przez wiele narzędzi i środowisk programistycznych, a także przez popularne biblioteki, takie jak Angular czy React.

Ze względu na przewidywany wielomiesięczny okres implementacji, zastosowanie TypeScript miało na celu minimalizację możliwych błędów, które mogą wyniknąć z przeciążenia czy przeoczeń. Miało to zagwarantować utrzymanie spójności kodu i zapewnić o jego wyższej jakości w dłuższym okresie.

```
class Person {
 constructor(public name: string, public age: number) {}

  greet(): string {
 return `Hello, my name is ${this.name}
           and I am ${this.age} years old.`;
  }
}

// Przykładowe użycie klasy Person
const person = new Person("John", 25);
const greeting = person.greet();
console.log(greeting);
```

Przykładowy kod w języku TypeScript

Powyższy kod demonstruje deklarację klasy *Person* posiadającej dwa pola:*name* (string) i *age* (number).Klasa posiada metodę *greet()* zwracającą wartość typu string. Dalsza część kodu to już czysty JavaScript.

3.2. React.js

React jest biblioteką JavaScript stworzoną przez firmę Facebook, która służy do tworzenia interfejsów użytkownika (UI). Pozwala na tworzenie aplikacji internetowych opartych o architekturę opartą na komponentach. Komponenty to funkcje lub klasy, które służą do opisywania fragmentów interfejsu użytkownika. Każdy komponent jest odpowiedzialny za renderowanie określonej części aplikacji, takiej jak przycisk, formularz czy lista. Komponenty mogą być również używane do tworzenia innych komponentów, co umożliwia ich modularne tworzenie i łatwe utrzymanie. React używa abstrakcji virtual DOM (wirtualnego drzewa DOM), która pozwala na szybkie renderowanie interfejsu użytkownika. Virtual DOM jest wirtualnym odwzorowaniem struktury dokumentu HTML, które jest porównywane z poprzednim stanem, aby zidentyfikować zmiany, które należy wprowadzić. Dzięki temu React.js może szybko aktualizować tylko te fragmenty interfejsu, które rzeczywiście uległy zmianie, co zapewnia wysoką wydajność aplikacji. Biblioteka ta jest popularna wśród programistów zarówno ze względu na swoją prostotę, jak i możliwość tworzenia aplikacji internetowych o dużej skali [11]. Jest on również często używany w połączeniu z innymi technologiami, takimi jak Redux do zarządzania stanem aplikacji oraz TypeScript w celu uzyskania lepszego typowania i organizacji kodu.

Ze względu na cel, jaki został postawiony w założeniach projektu, a mianowicie maksymalizację doświadczeń użytkownika poprzez prostotę i szybkość działania dynamicznego systemu, to właśnie React wydawał się najlepszym rozwiązaniem. Dodatkowo modułowy charakter aplikacji idealnie wpasował się w wykorzystanie komponentów, na których ta biblioteka jest oparta. Ponadto, jest to bardzo popularne narzędzie posiadające wiele opracowań i gotowych rozwiązań, co w przypadku budowania systemu od fundamentów było niemniej istotne.

```
import React, { useState } from 'react';

const ToggleText = () => {
const [isVisible, setIsVisible] = useState(true);
const handleToggle = () => {
 setIsVisible(!isVisible);
 };
return (
 <div>
  <button onClick={handleToggle}>Toggle Text</button>
   {isVisible && <p>Some text to toggle</p>}
 </div>
 );
};
export default ToggleText;
```

Przykładowy kod z użyciem biblioteki React.js

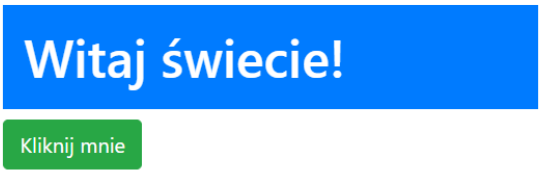
Powyższy kod demonstruje deklarację dynamicznego komponentu wyświetlającego przycisk <*button*>oraz paragraf tekstu <*p*>. Wykorzystano dynamiczne możliwości JSX i React, aby kliknięcie w przycisk zmieniało widoczność tekstu (bez konieczności odświeżania strony).

3.3. Bootstrap

Bootstrap to biblioteka szablonów front-end oparta na HTML, CSS i JavaScript, która służy do szybkiego tworzenia responsywnych i atrakcyjnych stron internetowych oraz aplikacji mobilnych. Bootstrap zapewnia gotowe stylizacje i szablony elementów interfejsu, takich jak nagłówki, przyciski, formularze, tabele, navbary, carousel'e itp., dzięki czemu możliwe jest szybkie tworzenie prototypów i aplikacji bez konieczności projektowania interfejsu od podstaw. Umożliwia elastyczne rozmieszczanie elementów na stronie za pomocą klas CSS. Model ten pozwala na tworzenie responsywnych layoutów, które dostosowują się do szerokości ekranu urządzenia [12].

Narzędzie zostało wybrane ze względu na swoją prostotę i elastyczność. Niski próg wejścia, w porównaniu do bliźniaczych rozwiązań, przechylił szalę na korzyść Bootstrap'a. Zastosowanie go jako podstawy projektu od strony front-end, miało na celu szybkie stworzenie wstępnego projektu i sprawne przełożenie go na finalny wygląd aplikacji, co było możliwe dzięki gotowym komponentom i zestawom komponentów, proponowanym przez samą firmę wprowadzającą Bootstrap. Ponadto, gdyby w przyszłości zaszła potrzeba rozszerzenia wyglądu strony, to narzędzie było idealne, gdyż jest łatwo rozszerzalne i może być dostosowywane do indywidualnych potrzeb projektu za pomocą własnych styli CSS.

```
<h1 class=”bg-primary text-white p-3">Witaj świecie!</h1>
<button class=”btn btn-success">Kliknij mnie</button>
```



Rys.4 Przykładowy kod HTML z wykorzystaniem Bootstrap i wygenerowany efekt

Powyższy kod demonstruje prosty przykład użycia biblioteki Bootstrap. Pokazuje, jak w prosty sposób dodając do obiektu predefiniowane klasy, można szybko uzyskać wizualny efekt.

3.4. Firebase

Firebase to platforma backendowa stworzona przez Google [13], która umożliwia tworzenie aplikacji webowych, mobilnych oraz gier. Jest to narzędzie do szybkiego i prostego budowania aplikacji z wykorzystaniem chmury. Firebase oferuje wiele usług, które pozwalają na łatwe tworzenie i zarządzanie aplikacjami, a także udostępnia narzędzia do analizowania

zachowania użytkowników i poprawiania ich doświadczenia. W projekcie wykorzystano trzy narzędzia proponowane przez Firebase, które opisano poniżej.

Firebase Cloud Functions

Funkcje serverless to rodzaj usługi chmury, która pozwala na uruchamianie kodu bez konieczności zarządzania serwerem. W projekcie wykorzystano Firebase Cloud Functions, czyli narzędzie pozwalające na uruchamianie kodu JavaScript na serwerach Google Cloud. Dzięki temu rozszerzeono funkcjonalność aplikacji webowej.

Funkcje Firebase Cloud mogą być uruchamiane na różne sposoby, np. na zdarzenie z Firebase Realtime Database, co zostało wykorzystane w projekcie. Funkcje te mogą mieć bardzo szerokie zastosowanie, ale w strukturze tej aplikacji służą one wyłącznie do nasłuchiwania zmian w bazie danych i na tej podstawie uaktualnianie jej w innym miejscu. Warto dodać, że funkcje te operują na bazie Realite Database z prawami dostępu administratora, więc mają nieograniczony dostęp do zapisu i odczytu danych.

Firebase Authentication

System uwierzytelniania został oparty o Firebase Authentication, czyli usługę udostępnianą przez Firebase, która umożliwia bezpieczne uwierzytelnianie użytkowników w aplikacjach mobilnych i webowych. Umożliwia ona implementację różnych metod uwierzytelniania, w tym logowania za pomocą adresu e-mail i hasła, logowanie za pomocą kont społecznościowych (np. Facebook, Google), czy uwierzytelnianie za pomocą tokenów SMS.

Firebase Realtime Database

Firebase Realtime Database to serwis bazy danych przechowywujący dane w formacie JSON, który pozwala na szybki i łatwy dostęp do danych przez różne platformy, takie jak aplikacje mobilne lub webowe. Baza danych jest synchronizowana w czasie rzeczywistym, co oznacza, że wszystkie połączenia z bazą danych otrzymują automatycznie aktualizacje ze zmianami w danych. Jest to baza danych NoSQL (Not Only SQL), która pozwala na przechowywanie i odczyt danych w sposób inny niż relacyjne bazy danych SQL, z czego korzysta coraz więcej współczesnych projektów. Baza danych tego typu posiada szereg zalet:

- Elastyczność - pozwala na szybkie i łatwe dodawanie nowych kolumn do bazy danych bez potrzeby zmieniania schematu
- Skalowalność - pozwala na rozszerzenie bazy danych wraz z rosnącym obciążeniem
- Dostępność - udostępnia dostęp do danych za pomocą różnych interfejsów, takich jak RESTful API

Wykorzystanie tych trzech technologii zapewniło spójny zbiór narzędzi odpowiedzialny za część serwera, czyli tak zwany back-end. Zagwarantowało to

ujednolicenie sposobu przechowywania danych i brak konieczności tworzenia adapterów. Wszystkie trzy systemy wzajemnie ze sobą współpracują i umożliwiają dostęp do serwisu logowania i bazy danych, przechowujących w bezpieczny sposób informacje o klientach oparty o niezawodne systemy firmy Google.

3.5. Visual Studio

Visual Studio to zintegrowane środowisko programistyczne (IDE) firmy Microsoft [14], które służy do tworzenia oprogramowania dla systemów operacyjnych Windows, Linux, MacOS i innych. IDE zawiera szeroki zestaw narzędzi do projektowania, debugowania, testowania i wdrażania aplikacji, a także obsługuje wiele języków programowania. VS poza podstawowymi funkcjonalności posiada szereg wtyczek i rozszerzeń, które pozwalają na dostosowanie go do indywidualnych potrzeb programisty lub zespołu.

Wcześniejsze doświadczenia w pracy z tym środowiskiem, a także jego lekkość i możliwość personalizacji przeważyły o wyborze. Co prawda wybór edytora tekstu zależy głównie od preferencji, ale skróty klawiaturowe i wbudowane narzędzia, jakie posiada to środowisko, czynią go bardzo wygodnym w użytkowaniu, a przez to jednym z lepszych przy wytwarzaniu oprogramowania.

3.6. Linux – Ubuntu

Linux jest bardzo popularny wśród programistów [15], głównie ze względu na to, że na wiele pozwala i nie ogranicza programisty (w przeciwieństwie do najbardziej popularnego Windowsa). Ubuntu jest dystrybucją systemu Linux, która jest szczególnie przyjazna dla użytkowników, oferując intuicyjny interfejs graficzny oraz wsparcie techniczne i dokumentację. Dostarcza szereg narzędzi oraz aplikacji do codziennego użytku i umożliwia sprawne instalowanie nowego oprogramowania. Ubuntu jest również znana ze swojej dbałości o bezpieczeństwo i stabilność systemu.

Zdecydowanie na korzyść Ubuntu wpłynęło również to, że zarządzanie wszystkimi innymi narzędziami zastosowanymi w projekcie było możliwe z poziomu konsoli. Dzięki temu praca była szybka i wygodna, co pozwoliło na zaoszczędzenie czasu, który byłby potrzebny na naukę korzystania z innych narzędzi.

Podsumowanie

Wybór takiego zestawu narzędzi do pracy nad projektem z pewnością miał wiele zalet, zwłaszcza biorąc pod uwagę ich nowoczesność, popularność wśród developerów czy wzajemną kompatybilność. Powyższy stos technologiczny pozwolił ostatecznie na sprawne utworzenie projektu w zgodzie z założeniami i efektywne zarządzanie nim. Niemniej, spoglądając na szeroki rynek dostępnych programów, języków i bibliotek nie sposób nie dojść do wniosku, że niniejszy projekt można było zrealizować z powodzeniem z użyciem innego zestawu narzędzi [16].Nie świadczy to jednak o złym czy dobrym wyborze narzędzi, a jedynie oznacza, że jest to przede wszystkim kwestia indywidualnych preferencji.

4. System rekrutacji

Ten rozdział został poświęcony opisowi systemu rekrutacji zarówno od strony użytkowników systemu, jak i struktury projektu. Pierwsza część jest przeznaczona dla zwykłych użytkowników systemu i zawiera ogólny opis, do czego służy aplikacja oraz stanowi podręcznik jej użytkowania. Druga część natomiast jest kierowana dla programistów, chcących dowiedzieć się więcej o całym systemie, jego strukturze i zastosowanych rozwiązaniach.

4.1. Podręcznik dla użytkowników

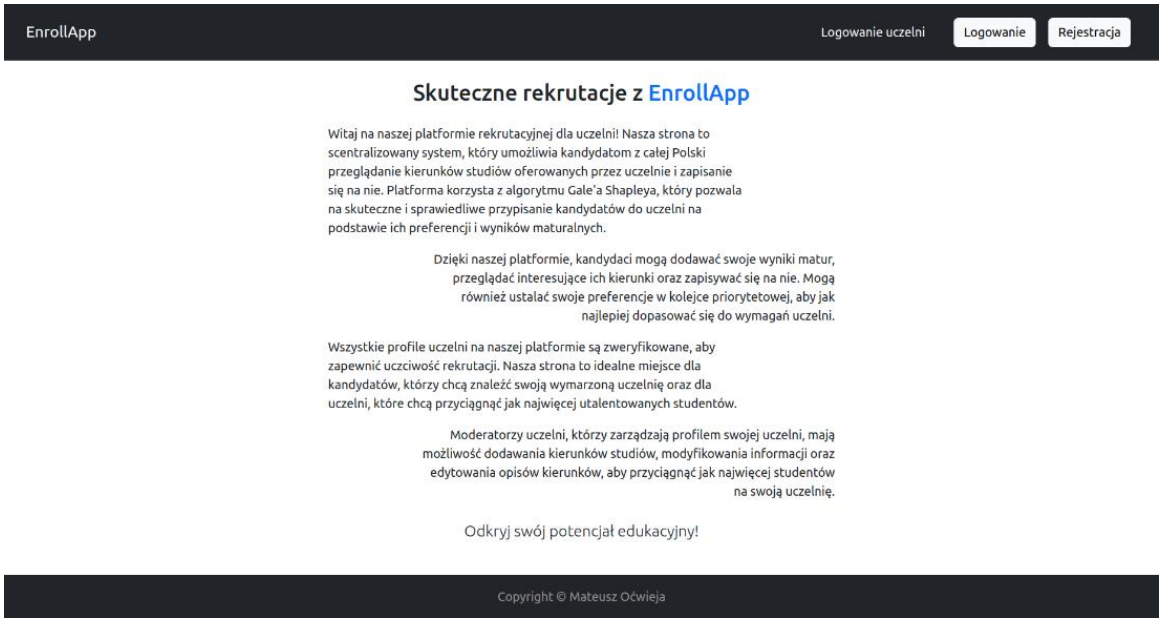
Na początku warto zaznaczyć, że każda strona wyświetlana przez aplikację składa się z trzech nierozłącznych komponentów, od góry kolejno:

- Pasek nawigacji (navbar) - zawiera linki do różnych sekcji aplikacji, umożliwia szybkie przemieszczanie się po witrynie
- Główna zawartość strony (main content) - jej kształt, wygląd, informacje i zawartość zmieniają się w zależności od podstrony, która aktualnie jest otwarta
- Stopka strony internetowej (footer) - zawiera dodatkowe informacje, w tym o autorze

Taki podział ułatwia poruszanie się po aplikacji i poprawia jej czytelność.

4.1.1. Część wspólna

Wchodząc na stronę internetową aplikacji, użytkownikowi ukazuje się *landing page*. Na domowej stronie aplikacji widnieją podstawowe informacje o aplikacji oraz skrócona instrukcja korzystania z niej. Poniżej widnieje zrzut ekranu, jaki ukazuje się przy pierwszym wejściu na stronę.

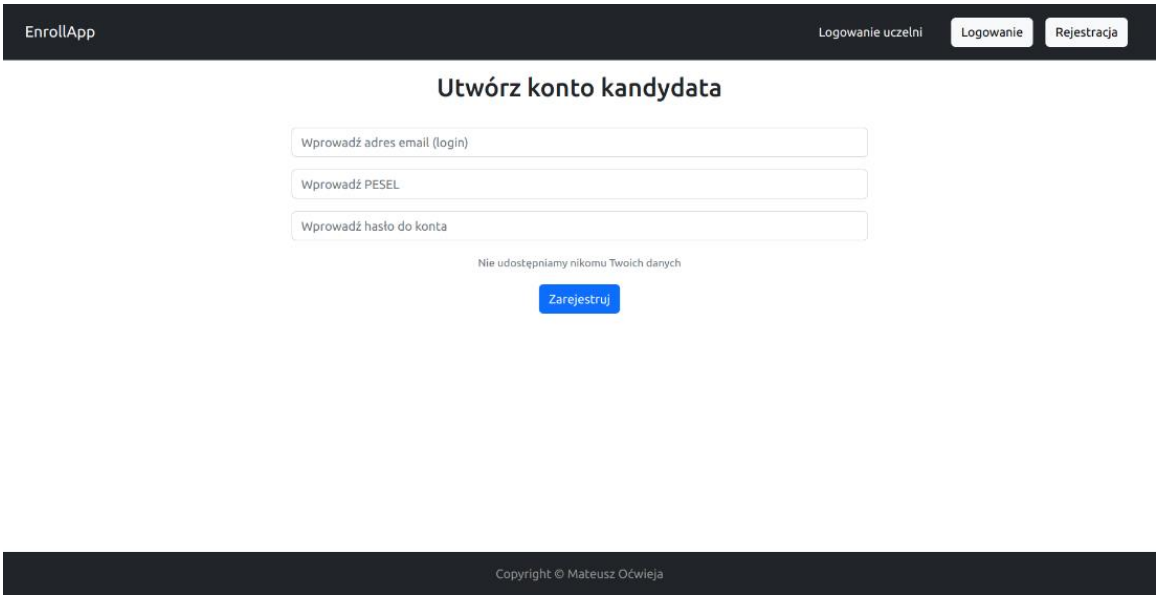


Rys.5 Widok strony domowej

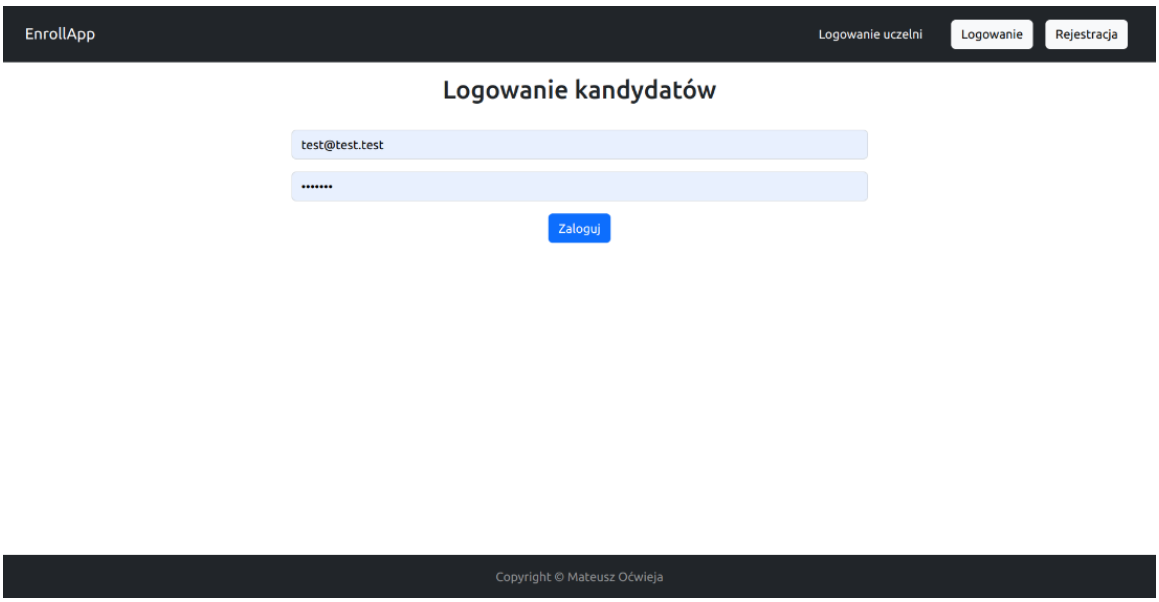
W górnym pasku dostępne są 3 przyciski odpowiedzialne za weryfikację użytkownika. Kliknięcie przeniesie użytkownika do odpowiednich okien, odpowiedzialnych za rejestrację kandydatów, logowanie kandydatów lub do okna z logowaniem dla moderatorów uczelni. Jeśli korzystamy z wersji mobilnej przeglądarki lub nasze okno przeglądarki jest zbyt wąskie, wówczas przyciski będą niewidoczne, a dostęp do nich zapewnia rozwijane menu, którego przycisk akcji pojawia się w prawym górnym rogu.

Logowanie i rejestracja kandydatów

Przechodząc przyciskiem *Logowanie* lub *Rejestracja* zostaniemy przeniesieni ze strony domowej odpowiednio na adres */auth/login* lub */auth/register*, co zostanie wyświetlone na pasku adresu w przeglądarce. Zamiast strony domowej, wyświetli się odpowiedni komponent, umożliwiający logowanie lub rejestrację.



Rys. 6 Widok okna rejestracji nowych kandydatów



Rys.7 Widok okna logowania kandydatów

Do rejestracji wykorzystujemy rzeczywisty adres e-mail, numer Pesel oraz hasło. Wszystkie te pola są sprawdzane pod względem poprawności, a użytkownik jest informowany, jeśli podał niepoprawny adres e-mail lub pesel czy też, gdy jego hasło nie spełnia wymagań bezpieczeństwa (np. co do długości). Do logowania wykorzystujemy adres e-mail i hasło podane w procesie rejestracji.

Jeśli podane przez użytkownika dane w tym procesie są poprawne, zostaną przesłane do systemu autoryzacyjnego. Po zatwierdzeniu przez system zostanie zalogowany do aplikacji (nawet jeśli dopiero się zarejestrował). Ponadto, jego dane zostaną zapamiętane w pamięci przeglądarki, dzięki czemu nie będzie się on musiał logować za każdym razem, gdy uruchomi aplikację, gdyż ten proces nastąpi automatycznie. Wylogowanie się z aplikacji lub wyczyszczenie danych przeglądarki zatrzyma ten proces.

Logowanie moderatorów uczelni

Przechodząc przyciskiem *Logowanie uczelni*, zostaniemy przeniesieni ze strony domowej na adres */auth/signInLink*, co zostanie wyświetlone na pasku adresu w przeglądarce. Dostęp do tej strony mają wszyscy niezalogowani użytkownicy.

W skład komponentu wchodzą 2 przyciski odpowiedzialne za akcję. Przycisk *Wygeneruj link* spowoduje otwarcie okna z polem do wpisania adresu e-mail. Adres e-mail musi być uprzednio zweryfikowanym mailem przez administratora strony (uczelnia musi się z nim wcześniej skontaktować: szczegóły w części dla programistów). Jeśli użytkownik poda właściwy adres e-mail, który został zweryfikowany przez administratora, otrzyma komunikat ze strony, że link do logowania został przesłany na wskazany adres. W przeciwnym wypadku, ukaże się komunikat o braku autoryzacji z prośbą o kontakt z administratorem.

W przypadku, gdy aplikacja poinformowała użytkownika o sukcesie, powinien on sprawdzić swoją skrzynkę pocztową, gdzie powinna czekać na niego wiadomość z linkiem do logowania. Po przejściu w ten link użytkownik powróci na stronę */auth/signInLink?key*, gdzie w linku został zaszyty klucz (ang. key) do weryfikacji. Po wybraniu przycisku *Zaloguj* klucz weryfikacyjny zostanie sprawdzony, a użytkownik zalogowany. Jeśli użytkownik wszedł na dany link z innego urządzenia niż z tego, na którym wysyłał link, zostanie poproszony dodatkowo o potwierdzenie adresu e-mail, pod rygorem odmowy autoryzacji.

4.1.2. Dla kandydatów na studia

Po zalogowaniu się do systemu i uzyskaniu roli kandydata aplikacja odświeży swój widok i podmieni komponent paska nawigacji. W wersji dla zalogowanych kandydatów, z prawej strony zamiast trzech przycisków służących do logowania widnieje jeden przycisk *Wyloguj* służący do wylogowania aktualnego zweryfikowanego użytkownika z jego konta. Obok znajduje się pole tekstowe zawierające informację o aktualnie zalogowanym użytkowniku, stanowiące komunikat powitalny, gdzie nazwę użytkownika stanowi

identyfikator adresu e-mail (ciąg znaków do @). Natomiast strona domowa również ulegnie zmianie i będzie się ukazywała wersja z informacjami dla kandydatów na studia.

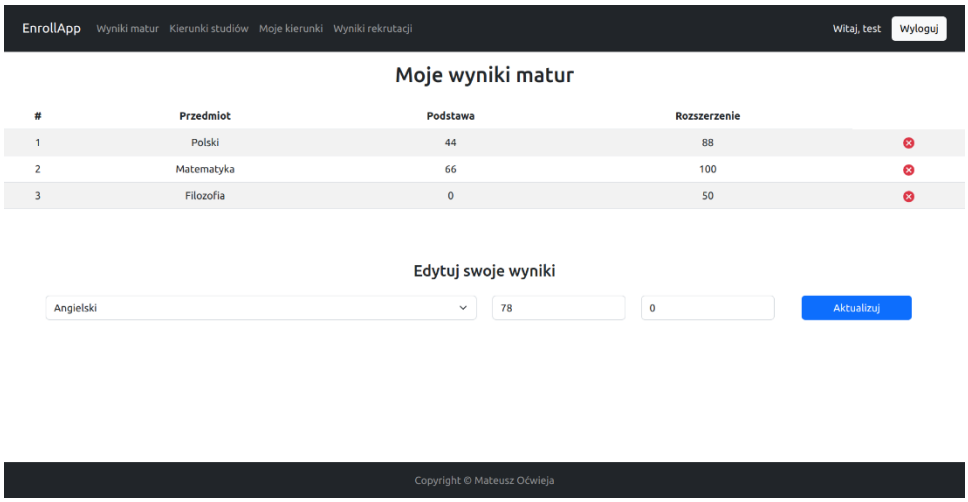
Rys.8 Widok strony domowej kandydatów

Na początku paska nawigacji znajduje się pięć odnośników, kolejno:

- EnrollApp– zarówno link jak i nazwa aplikacji, przenosi do strony domowej dla kandydatów
- Wyniki matur – link do widoku z wynikami egzaminu maturalnego kandydata
- Kierunki studiów – link do przeglądarki uczelni i dostępnych kursów
- Moje kierunki – link do widoku uszeregowanych preferencjami kierunków, na które zapisał się kandydat
- Wyniki rekrutacji – link do widoku podsumowującego wyniki rekrutacji

Wyniki matur

W tej zakładce znajduje się narzędzie, do wprowadzania i prezentacji wyników egzaminów maturalnych.

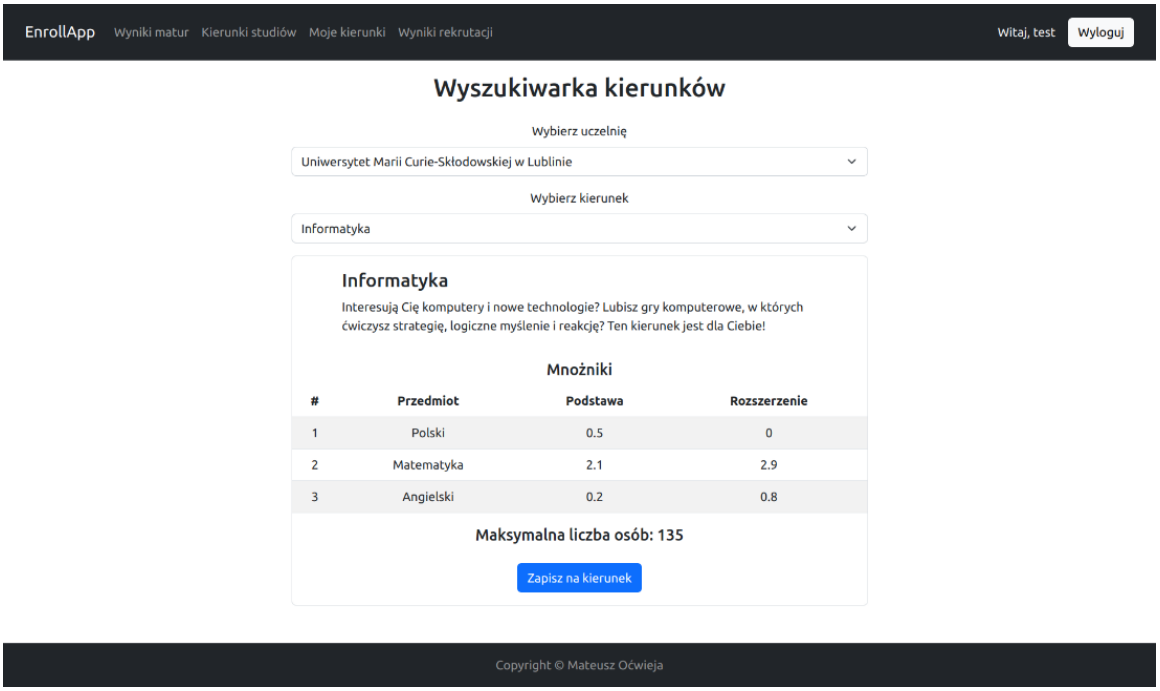


Rys.9 Widok strony Wyniki matur

Kandydat może wybrać przedmiot maturalny z rozwijanej listy i ręcznie wprowadzić w kolejnych komórkach wyniki z części podstawowej i części rozszerzonej egzaminu. Po zatwierdzeniu wynik ten pojawi się w tabeli wszystkich wyników i zostanie zapisany w bazie danych. W przypadku wykrycia pomyłki istnieje możliwość usunięcia danego rekordu, co spowoduje usunięcie go z bazy danych i opcja wyboru tego przedmiotu znowu będzie dostępna na liście przedmiotów maturalnych.

Kierunki studiów

W tej zakładce znajduje się przeglądarka kursów prowadzonych przez autoryzowane uczelnie.



Rys.10 Widok strony Kierunki studiów

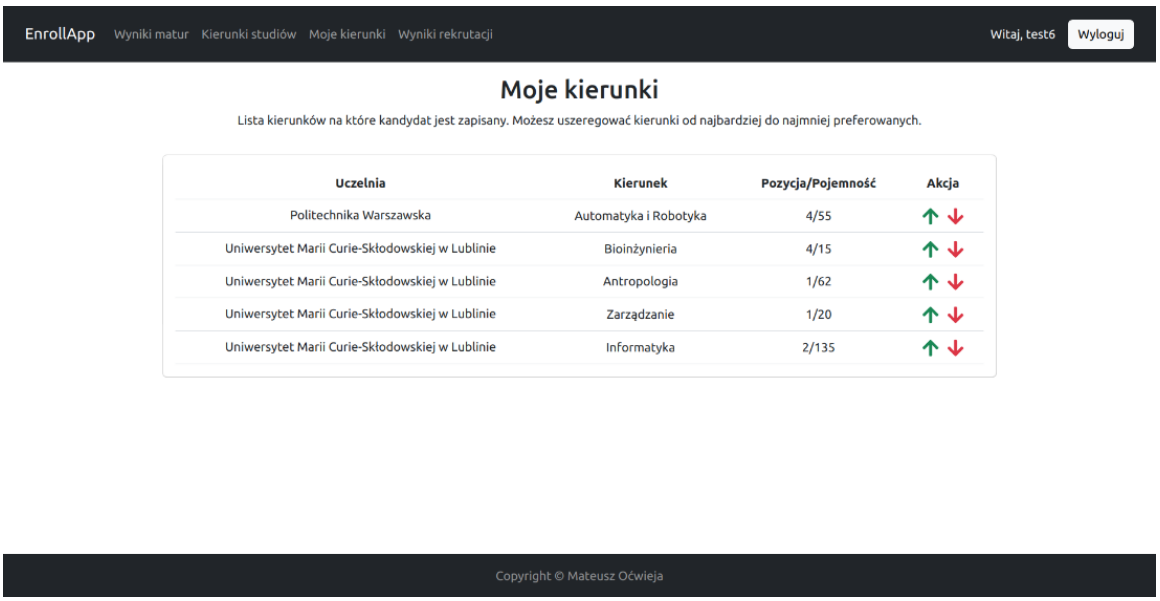
Korzystając z listy rozwijanej, kandydat może wybrać interesujący go uniwersytet. Wybór konkretnego uniwersytetu spowoduje pojawienie się kolejnej listy rozwijanej z kierunkami na danej uczelni. Po wyborze, pojawi się karta prezentująca dany kierunek, w której skład wchodzą:

- Nazwa kierunku
- Krótki opis
- Informacja o mnożnikach (dla konkretnych wyników egzaminów maturalnych)
- Liczba miejsc na kierunku
- Przycisk do rejestracji

Kliknięcie przycisku *Zapisz na kierunek* spowoduje dopisanie użytkownika na listę kandydatów oczekujących na przyjęcie wybranego kierunku studiów (o ile użytkownik nie był zapisany już wcześniej) i spowoduje przeniesienie go do kolejnej strony.

My preferences

W tej zakładce znajduje się narzędzie służące określeniu preferencji użytkownika, odnośnie do wybranych kursów.



Rys.11 Widok strony Moje kierunki

Całe narzędzie jest bardzo proste i intuicyjne w obsłudze. Wyświetlana jest lista kierunków, na które użytkownik złożył swoją aplikację. Nazwy kierunków są wyświetlane jeden pod drugim. Na samej górze znajdują się kierunki najbardziej preferowane, a najniżej najmniej. Celem użytkownika jest takie ułożenie kursów, przy użyciu strzałek, aby jak najlepiej oddawały jego osobiste preferencje.

Dodatkowo obok każdego kursu jest wyświetlana informacja o liczbie miejsc na danym kierunku i relatywnej pozycji zalogowanego użytkownika, względem pozostałych. Przykładowo, informacja 6/20 oznacza, że użytkownik zajmuje 6 miejsce

w rankinguna kierunku, na którym jest 20 miejsc. Jeśli więc wybierze ten kierunek jako najbardziej preferowany, otrzyma gwarancję, że to na niego się dostanie, choć jego pozycja w rankingu może się jeszcze zmienić (gdy kolejni kandydaci będą zapisywani na ten sam kierunek).

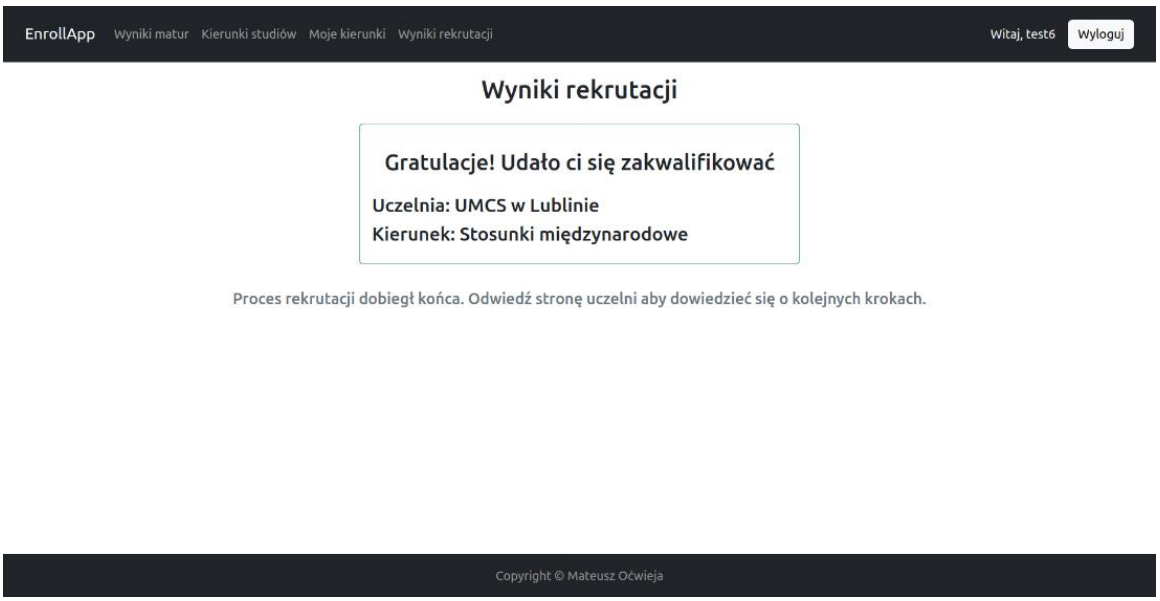
Uwaga: Jeśli pozycja kandydata jest dalsza, niż liczba miejsc na kierunku, to nie przekreśla to jego szans i wciąż istnieje prawdopodobieństwo, że zostanie zakwalifikowany.

Wyniki rekrutacji

W tej zakładce znajduje się widok z wynikami. W zależności od aktualnego stanu wyświetlana jest odpowiednia informacja. Wyróżnione są trzy przypadki:

- Rejestracja jeszcze się nie rozpoczęła
- Rejestracja się zakończyła i student został zakwalifikowany
- Rejestracja się zakończyła i student nie został zakwalifikowany

Dla każdej z powyższych zakładek wyświetlana jest odpowiednia zawartość. Przykładowo, dla sytuacji, w której kandydat poprawnie się zapisałoraz został pomyślnie zakwalifikowany, w centrum okna pojawia się komunikat zawierający nazwę tej uczelni i tej kierunek, na który dostał się kandydat.



Rys.12 Widok okna Wyniki rekrutacji

Powyżej przedstawiono widok dla sytuacji, w której kandydat został zakwalifikowany na jeden z kierunków. Z ekranu możemy odczytać, że jest to kierunek *Stosunki międzynarodowe* na uczelni o nazwie *UMCS w Lublinie*. Na stronie nie wyświetlają się na tym etapie żadne dodatkowe informacje, poza krótką notką informującą, że proces rekrutacji dobiegł końca oraz sugerującą następne kroki. Pozostałe scenariusze, dla których kandydat się nie zakwalifikował, lub gdy wyniki nie są gotowe, wyglądają analogicznie.

4.1.3. Dla moderatorów uczelni

Po zalogowaniu się do systemu i uzyskaniu roli moderatora aplikacja odświeży swój widok i podmieni komponent paska nawigacji. W wersji dla zalogowanych moderatorów uczelni, z prawej strony zamiast trzech przycisków służących do logowania widnieje jeden przycisk *Wyloguj* służący do wylogowania aktualnego użytkownika z konta. Obok znajduje się pole tekstowe z informacją, jaki użytkownik jest aktualnie zalogowany, wraz z komunikatem powitalnym, gdzie nazwę użytkownika stanowi identyfikator adresu e-mail (ciąg znaków do @). Natomiast strona domowa również ulegnie zmianie i będzie się ukazywała wersja z informacjami dla moderatorów uczelni.

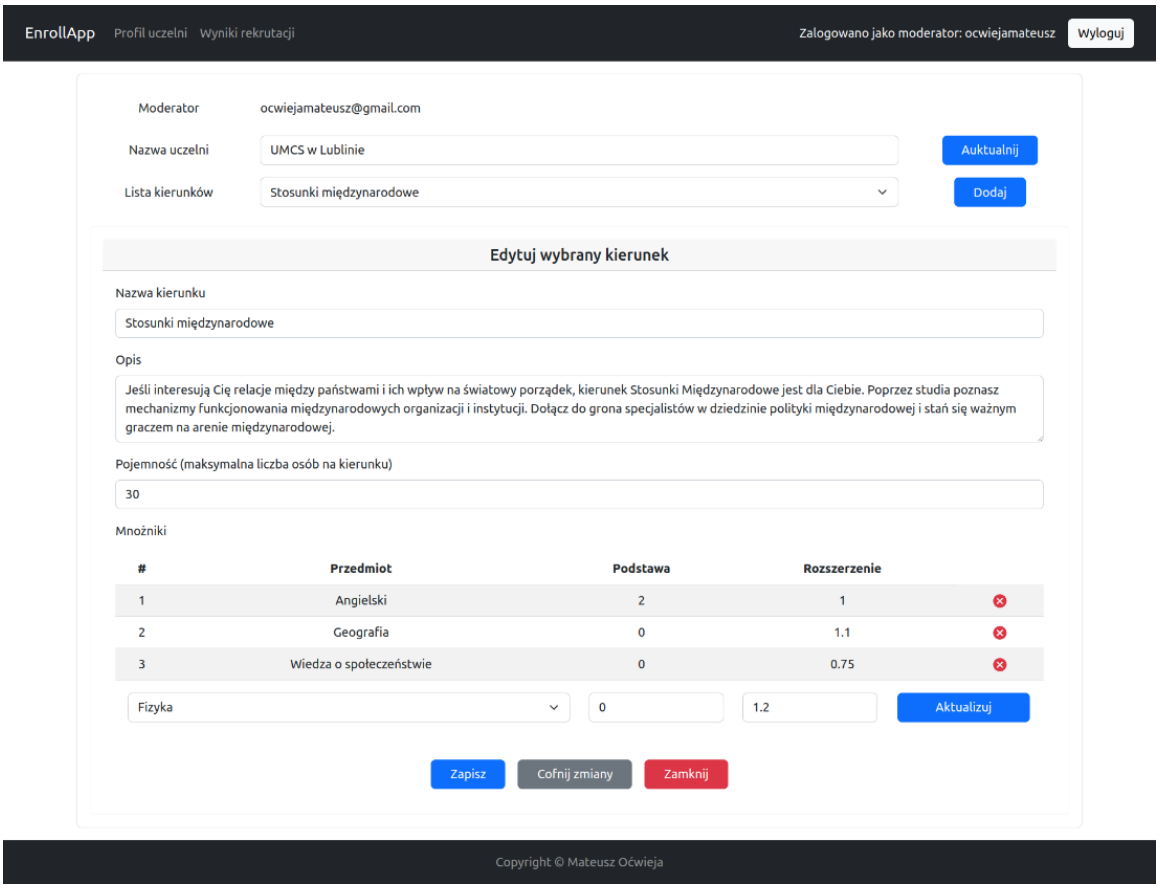
Rys.13 Widok strony domowej dla moderatorów

Na początku paska nawigacji znajdują się trzy odnośniki, kolejno:

- EnrollApp– zarówno link jak i nazwa aplikacji, przenosi do strony domowej dla moderatorów
- Profil uczelni– link do ekranu z danymi o uczelni, w tym o dostępnych kierunkach studiów
- Wyniki rekrutacji – link do widoku podsumowującego wyniki rekrutacji

Profil uczelni

Widok ten zawiera wszystkie informacje o uczelni, do której moderator ma dostęp. Wszystkie dane, z wyłączeniem konta moderatora, są modyfikowalne.



Rys.14 Widok edytowania kierunku studiów w zakładce Profil uczelni

Po pierwszym zalogowaniu się na konto moderator powinien ustawić adekwatną nazwę uczelni. Później, w miarę potrzeby, można ją zmienić. Na początku lista z wyborem kierunków będzie pusta i zapełni się, dopiero gdy moderator doda nowy kurs przyciskiem *Dodaj*. Po kliknięciu w ten przycisk pojawi się formularz, który należy przed zatwierdzeniem uzupełnić. Pola takie jak nazwa kursu, liczba miejsc i mnożniki są obowiązkowe i nie mogą pozostać puste. Pole z opisem jest opcjonalne i jeśli pozostawimy puste, kandydaci nie zobaczą żadnego opisu. Formularz jest sprawdzany pod kątem błędów i nieprawidłowych wartości, a moderator jest powiadamiany, gdy próbuje zatwierdzić niepoprawny formularz. Jeśli wszystko jest poprawne, po wybraniu przycisku *Zapisz,* kurs zostanie zapisany i wysłany do bazy danych - od tej chwili użytkownicy będą mogli na niego aplikować.

Dodany wcześniej kurs możemy edytować, wybierając go z rozwijanej listy *Lista kierunków*. Wybór konkretnego kursu spowoduje ukazanie się komponentu, który moderator zna z dodawania kursu, z tą jednak różnicą, że formularz będzie uzupełniony o aktualne dane dotyczące kursu. Formularz można dowolnie edytować (pamiętając o obowiązkowych polach). Dane nie zostaną wysłane, zanim moderator nie zatwierdzi ich przyciskiem *Zapisz*. Natomiast, jeśli moderator chce zrezygnować i bezpiecznie anulować wprowadzone w formularzu zmiany, zanim je zapisze, może wybrać *Zamknij*, który spowoduje zamknięcie komponentu bez wprowadzania zmian.

W przypadku gdy moderator chciałby anulować wprowadzone zmiany, bez zamykania komponentu, może wybrać opcję *Cofnij zmiany*, która przywróci stan formularza z momentu

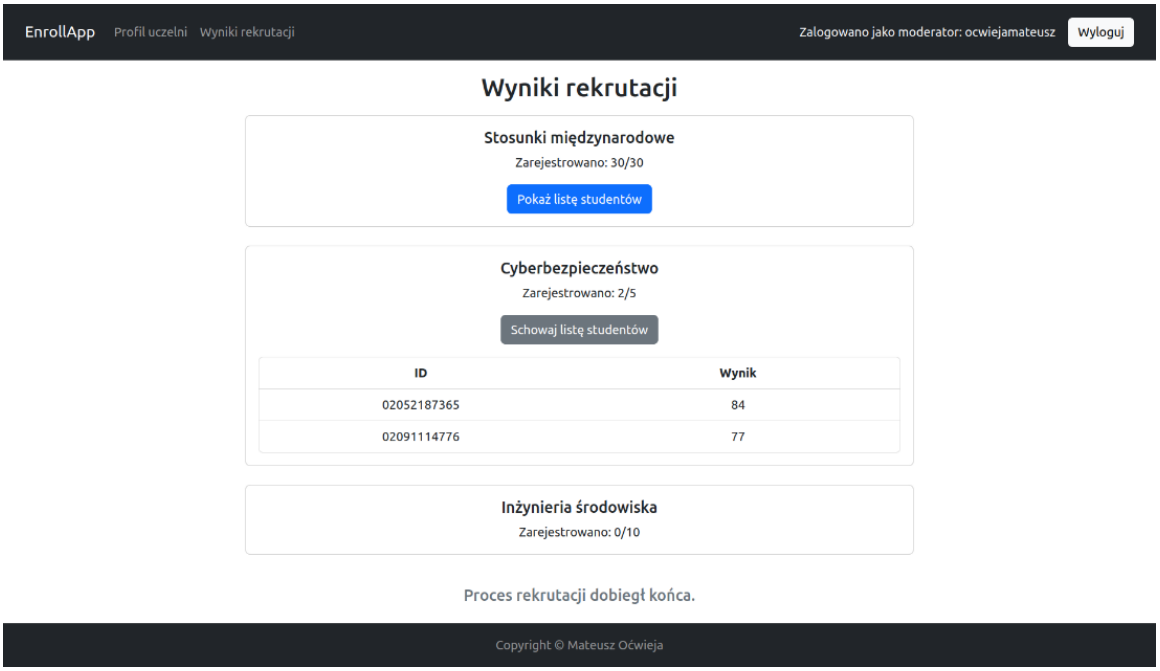
wybrania kursu do edycji. Ponadto, opcję *Cofnij zmiany*, można zastosować po wybraniu opcji *Zapisz*, co spowoduje przywrócenie formularza do stanu, w jakim znajdował się przy poprzednim zapisie. Sam przycisk *Cofnij zmiany* nie powoduje zatwierdzenia formularza, a jedynie wczytuje jego poprzednią wersję. Opcja ta może być przydatna, gdy moderator omyłkowo wyśle niepoprawne dane do bazy, lub w sytuacji gdy użytkownik chce bezpiecznie powrócić do poprzedniej wersji.

Wyniki rekrutacji

W tej zakładce znajduje się widok z wynikami. Wyróżnione są dwa przypadki:

- Rejestracja jeszcze się nie rozpoczęła
- Rejestracja się zakończyła i listy studentów są gotowe

W zależności od aktualnego stanu wyświetlana jest odpowiednia informacja.



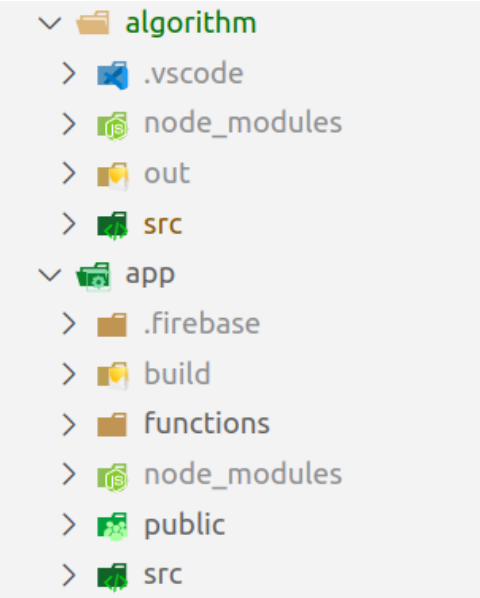
Rys. 15 Widok moderatora po zakończonym procesie rekrutacji

W widoku wyróżnione są wszystkie kierunki dostępne na uczelni. Dla każdego z nich widoczna jest liczba zarejestrowanych kandydatów na liczbę dostępnych miejsc. Jeśli przynajmniej jeden kandydat został zapisany, dostępny jest przycisk do wyświetlania listy zapisanych studentów.

4.2. Informacje dla programistów

4.2.1. Struktura projektu

Cały projekt posiada dwie części spajające go w logiczną całość: część do zbierania informacji i część do ich przetwarzania. Ta pierwsza zawiera się w aplikacji webowej, druga zaś została zaszyta lokalnie i odpowiedzialna jest za realizację algorytmu. Projekt został zaimplementowany przy zachowaniu tego logicznego podziału.



Rys.16 Ogólna struktura projektu z wyróżnieniem podfolderów

Cały kod został zawarty w jednym folderze, zawierającym dwa podfoldery. W pierwszym podfolderze *algorithm* znajduje się kod odpowiedzialny za algorytm, w drugim podfolderze *app*: część odpowiedzialna za aplikację webową.

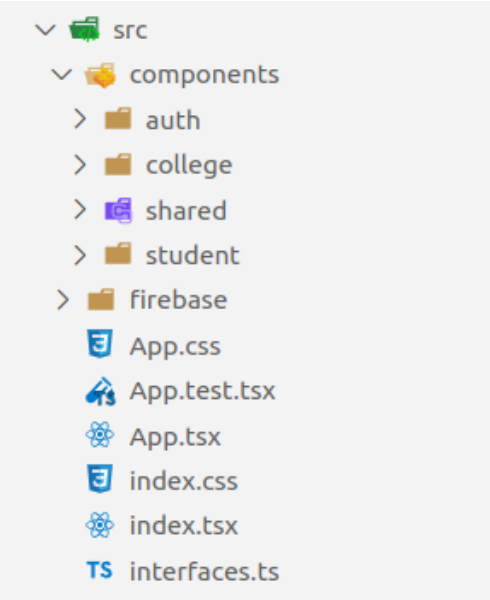
Z powyższych warto wyróżnić foldery, które zawierają:

- algorithm/out - produkt działania kompilatora i algorytmu
- app/funcions - funkcje serverless (wysyłane na serwer)
- app/build- zbudowany i zoptymalizowany kod aplikacji
- app/public - pliki dostępne dla użytkowników, niezbędne do działania strony internetowej
- */src - kod źródłowy

Aplikacja webowa

Kod źródłowy aplikacji webowej znajduje się w folderze *app*. Kod napisany bezpośrednio przez programistę pojawia się w pod folderach *functions* oraz *src*. Pozostałe foldery czy pliki konfiguracyjne, są wymagane przez narzędzia użyte w projekcie i są niezbędne do poprawnego funkcjonowania aplikacji. Aplikacja jest uruchamiana przez środowisko uruchomieniowe Node.js na podstawie napisanego lokalnie kodu, zawartego w folderze src. W trakcie działania korzysta z rozwiązań dostarczonych przez Firebase (autoryzacja, baza danych, funkcje serverless).

Zaglądając do folderu *src*, możemy dostrzec poniższą strukturę:



Rys.17 Widok drzewa katalogów kodu aplikacji webowej

Wszystkie powyższe pliki (w tym komponenty z nierozwiniętych podfolderów) posiadają jedno z trzech rozszerzeń:

- .css - pliki kaskadowych arkuszy stylów (CSS), odpowiedzialne za formatowanie strony internetowej
- .ts - pliki Typescript
- .tsx

Rozszerzenie .tsx oznacza, że plik jest w formacie TypeScript JSX. JSX to składnia rozszerzająca JavaScript, która pozwala na umieszczanie elementów XML (lub podobnych) w kodzie JavaScript. TypeScript to natomiast rozszerzenie JavaScript, które pozwala na użycie statycznego typowania i innych cech programowania nieobecnych w czystym JavaScript. Kiedy plik z rozszerzeniem .tsx jest kompilowany, jest konwertowany do czystego JavaScript, który jest następnie interpretowany przez przeglądarkę lub uruchamiany przez Node.js. Plik typu .tsx jest przeznaczony dla React. Zawiera składnię JSX i służy do tworzenia komponentów Reactowych. Komponenty Reactowe (funkcje lub klasy) pozwalają na dzielenie interfejsu użytkownika na mniejsze, bardziej złożone części. Każdy komponent jest odpowiedzialny za wyświetlanie określonej części interfejsu użytkownika i może przyjmować dane jako argumenty (tzw. propsy) oraz stan (zmienna wewnątrz komponentu) i na ich podstawie generować i renderować interfejs.

Główny element (tzw. root), od którego pochodzą wszystkie pozostałe komponenty, jest generowany i renderowany w pliku *index.tsx:*

```
const root = ReactDOM.createRoot(
 document.getElementById("root") as HTMLElement
);
root.render(
 <React.StrictMode>
  <BrowserRouter> <App/> </BrowserRouter>
 </React.StrictMode>
);
```

W skrócie kod ten tworzy element domowy 'root' i przypisuje do niego naszą aplikację Reactową z routerem, która jest renderowana w przeglądarce. Komponent <App/> jest głównym komponentem aplikacji, który z kolei jest zwracany w App.tsx.

W pliku *App.tsx* zawarta została cała logika aplikacji. Plik ten domyślnie eksportuje funckjęApp(), która zwraca element JSX. Stanowi on element nadrzędny dla wszystkich pozostałych komponentów. Jego role to:

- Przetrzymywanie i synchronizacja z bazą kluczowych danych (o uczelniach, kursach, studentach, przedmiotach)
- Przetrzymywanie i aktualizacja stanów obiektów (kluczowych danych, użytkownika)
- Przejmowanie odpowiedzialności za pod komponenty (wykonywanie za nich niektórych zadań)
- Odpowiadanie na żądania od komponentów (o logowanie, rejestrację, zmianę danych w aplikacji czy bazie)
- Wyświetlanie poprawnego interfejsu (poprzez zarządzanie innymi komponentami i przekazywanie im odpowiednich stanów)
- Wyświetlanie odpowiednich komponentów (na podstawie stanu aplikacji)
- Zarządzanie routingiem (nawigacją po stronie internetowej)
- Dbanie o bezpieczeństwo aplikacji

Wszystkie pozostałe komponenty wyświetlane są w <App/>. Znajdują się one w folderze *components*. Zostały one skategoryzowane względem swojej logicznej funkcji i zgrupowane w podfoldery:

- Auth – związane z autoryzacją
  - LoginForm – logowanie studentów
  - RegisterForm – rejestracja studentów
  - SignIn – logowanie moderatorów uczelni
  - Logout – wylogowanie użytkowników
  - Header – podstawowy *Header* dla niezalogowanych użytkowników
  - PrivateRoute – komponent ścieżki chronionej, w zależności od stanu autoryzacji użytkownika zwróci odpowiedni dla niego komponent
- College - dla moderatora uczelni
  - Profile – zawiera pełny komponent przedstawiający profil uczelni
  - CardCourse – karta edycji kursu, zawarta w komponencie *Profile*
  - HeaderCollege - zamiennik *Header* gdy zalogowany jest moderator uczelni
- Student - wyświetlane dla zalogowanych studentów

- Scores – wyniki studenta (korzysta z *ScoresTable* i *ScoresForm*)
- ScoresTable – tabela przedstawiająca wyniki studenta
- Majors - przeglądarka uczelni i kursów
- Major – zwraca informacje o pojedynczym kursie (zawarta w *Majors*)
- Preferences – wyświetla preferencje studenta (lista komponentów *Preference*)
- Preference - pojedyncza preferencja, wyświetla nazwę kursu i przyciski góra/dół
- Results – wyniki rekrutacji
- HeaderStudent – zamiennik *Header*, gdy zalogowany jest student
- Shared - współdzielone przez różne komponenty
- Home – strona domowa
- Footer – stopka strony internetowej (wyświetlana nieustannie w *App*)
- ScoreForm – wielokrotnie używany komponent do wprowadzania wyników lub mnożników
- ScoresTableRow – wykorzystywany w *ScoresTable*, jej pojedynczy wiersz
- Page404 – strona z komunikatem o błędzie *adres nieosiągalny*

Niektóre z powyższych komponentów są używane wielokrotnie, w różnych miejscach aplikacji. Współdzielą między sobą konkretne stany i dzielą się odpowiedzialnością. Gdyby spróbować narysować graf zależności między nimi, przypominałby strukturę drzewiastą, której korzeniem byłby komponent App.

Zastosowanie takiej struktury pozwala na łatwiejszą nawigację i rozumienie kodu, a także na szybsze rozwiązywanie problemów i utrzymanie aplikacji. Komponenty, które są używane wielokrotnie, są wykorzystywane jako komponenty wyższego poziomu, które zawierają mniejsze komponenty. To z kolei pozwala na lepszą modularność. Dzięki temu, gdyby w przyszłości zaszła konieczność zmiany w jednym z komponentów, nie trzeba by przeszukiwać całego kodu, a jedynie zmienić konkretny komponent i zaktualizować jego gałąź.

System uwierzytelniania

W projekcie zastosowano dwie metody autoryzacji, oparte o logowanie za pomocą konta e-mail.

- Email Sign-in, polegającą na uwierzytelnianiu użytkownika za pomocą adresu e-mail i hasła.
- Email Link Sign-in, podobną do Email Sign-in, ale po podaniu adresu e-mail, Firebase wysyła link bezpośrednio do użytkownika. Po kliknięciu na link użytkownik jest automatycznie zalogowany i może korzystać z aplikacji. Dzięki temu rozwiązaniu, użytkownik nie musi pamiętać swojego hasła.

Dwie metody uwierzytelniania zostały wykorzystane również do rozdzielenia sposobów logowania dla studentów i dla moderatorów. Logowanie za pomocą adresu e-mail i hasła jest dostępne tylko dla studentów, a logowanie za pomocą linku jest dostępne tylko dla moderatorów.

W aplikacji webowej (gdy użytkownik jest zalogowany) jest tworzony obiekt sesji, który zawiera informacje o stanie zalogowania użytkownika i przypisany mu token dostępu. Ten token jest przechowywany w przeglądarce użytkownika i może być używany do uwierzytelniania użytkownika przy każdym żądaniu do zabezpieczonych zasobów w aplikacji. Obiekt sesji zawiera informacje o zalogowanym użytkowniku, w tym jego identyfikator czy adres e-mail. Dostęp do tych informacji jest zagwarantowany przez interfejs API, dostarczany przez Firebase Authentication.

Wykorzystanie tego interfejsu API możemy znaleźć w aplikacji w kilku miejscach, przykładowo w *auth/signIn.tsx*:

- Auth - klasa główna Firebase Authentication, udostępniająca interfejsy do różnych metod uwierzytelniania
- isSignInWithEmailLink - funkcja sprawdza czy link logowania jest poprawny
- sendSignInLinkToEmail - funkcja wysyła link logowania na podany adres e-mail.
- signInWithEmailLink - funkcja loguje użytkownika za pomocą dostarczonego linku logowania

Wykorzystanie Firebase Authentication, pozwoliło na szybkie i łatwe zintegrowanie funkcjonalności uwierzytelniania użytkowników do aplikacji, przy użyciu gotowego i eleganckiego interfejsu API, bez konieczności pisania kodu od podstaw. Ponadto, zapewniło sprawną integrację z bazą danych, przy zachowaniu zasad bezpieczeństwa, również dostarczaną przez Firebase.

Funkcje chmury (serverless)

Użyta w projekcie funkcja serverless, o nazwie *onPrefAdd***,** nasłuchuje zmian w bazie danych, w lokalizacji przechowującej preferencje każdego ze studentów. Gdy pojawi się tam nowa wartość (kandydat złoży aplikację na konkretny kierunek uczelni), funkcja jest *triggerowana*.

Schemat jej działania wygląda następująco:

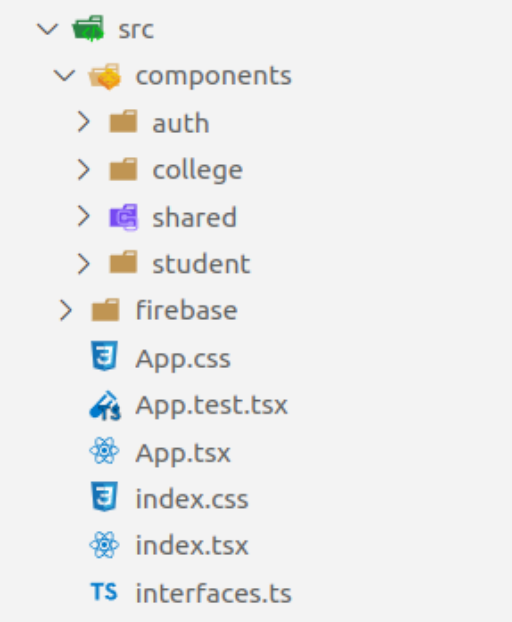
1. Przechwytuje informacje o danych, które uległy zmianie
2. Uzyskuje identyfikatory: użytkownika, uczelni, kursu
3. Uzyskuje informacje o mnożnikach konkretnego kursu
4. Uzyskuje informacje o wynikach konkretnego użytkownika
5. Oblicza wynik uzyskany przez studenta, na podstawie danych z 3. i 4.
6. Wpisuje studenta na listę kandydatów kursu, w postaci krotki (*userID, userScore*)

Takie rozwiązanie pozwoliło na odseparowanie części logiki z samej aplikacji. Ponadto, tym sposobem rozwiązano problem praw dostępu do zapisu w bazie danych, gdyż student zapisuje tylko dane na swoim profilu (do czego ma uprawnienia), a informacja o tym jest przenoszona na profil uczelni (do której praw zapisu nie posiada). Ponadto, ze strony użytkownika nie ma możliwości „podkręcenia” swojego wyniku, jaki jest wpisywany na liście kandydatów kursu, gdyż odpowiada za to oddzielna funkcja, której użytkownik nie

widzi i nie może w sposób niepowołany uzyskać do niej dostępu, co poprawia bezpieczeństwo aplikacji.

Program obsługujący algorytm

Część programu obsługującego algorytm Gale'a Shapleya jak i kod źródłowy odpowiedzialny za jego działanie został zawarty w jednym z dwóch głównych podfolderów o nazwie *algorithm.*



Rys.17 Widok struktury katalogu algorithm

W tym podfolderze, poza plikami konfiguracyjnymi i bibliotekami, warte uwagi są foldery *src, out,* oraz plik skryptowy *run.sh*. W folderze *src* znajdują się pliki odpowiedzialne za algorytm oraz podfolder *data*, który zawiera pliki json z danymi odnośnie uczelni i studentów (zrzut bazy danych, z kolekcji *data*). Pliki źródłowe w formacie ts (Typescript) to:

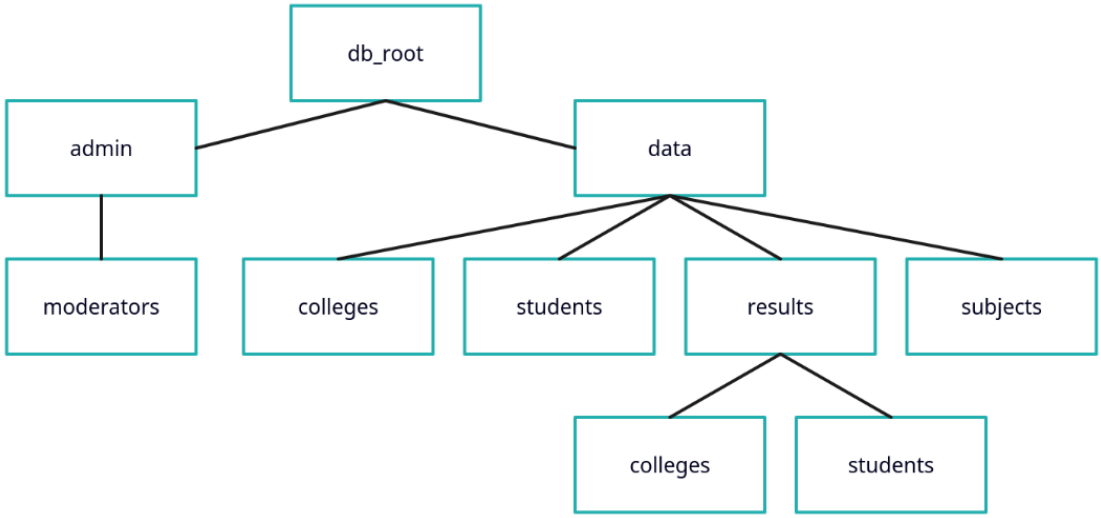
- main – odpowiada za uruchomienie i przebieg programu
- algorithm - zawiera klasę *GSAlgorithm*, serce całego algorytmu
- helper - pomocnicza klasa, odpowiedzialna za zapis / odczyt plików
- college - wykorzystywana przez *GSAlgorithm*, zawiera klasy logicznie wykorzystywane przez uczelnię: *College*, *Course*, *PriorityQueue*
- student - wykorzystywana przez *GSAlgorithm*, zawiera klasy logicznie reprezentujące kandydata na studenta: *Student*, *Preference*

Plik skryptowy *run.sh* służy do kompilacji kodu napisanego w języku TypeScript do języka JavaScript i uruchomienia tego kodu za pomocą Node.js. Proces działa następująco: kompilator TypeScript jest uruchamiany na folderze *src i* przetwarza pliki TypeScript na pliki JavaScript. Wynikiem tego procesu jest folder *out*. Następnie Node.js uruchamia program,

którego plikiem wykonawczym jest plik *out/main.js*, odpowiadający oryginalnemu plikowi *src/main.ts*. Program przyjmuje na wejściu plik z danymi z bazy (zawarty w folderze *data*), którego nazwę można zdefiniować w pliku *main* przy zmiennej *inFile*. Na konsoli wyświetlany jest szereg komunikatów informujący o kolejnych krokach działania programu i na końcu, w przypadku poprawnego zakończenia, ukazuję się komunikat potwierdzający poprawne zapisanie pliku wynikowego. Wynikiem działania algorytmu jest plik, do którego ścieżkę możemy zdefiniować w pliku *main*, używając zmiennej *outFile*. Wszystko zostało zorganizowane w przemyślany sposób, dzięki czemu plik wynikowy został specjalnie przygotowany i można go bezpośrednio zaimportować do bazy danych do kolekcji *data* po zakończeniu pracy algorytmu. W ten sposób wszystkie dane w bazie zostaną zaktualizowane, a wyniki rekrutacji staną się dostępne.

4.2.2. Baza danych

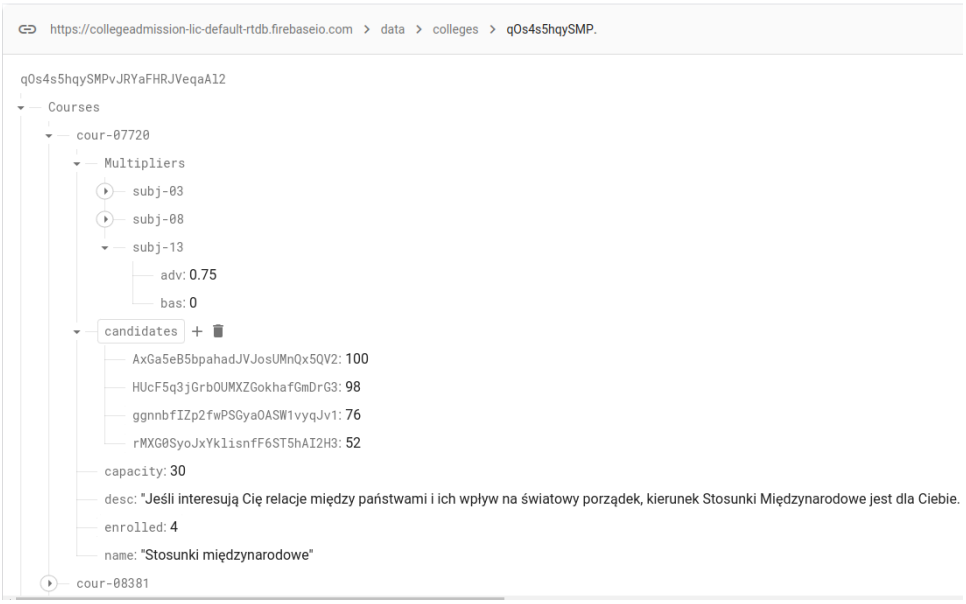
Jako baza danych została wykorzystana baza NoSQL Fireabase Realtime Database. Są w niej przechowywane wszystkie aktualne dane z aplikacji. Struktura tych danych (uproszczona do głównych kolekcji) została przedstawiona na poniższym grafie.



Rys. 18 Uproszczona struktura bazy danych

Dane w tej bazie są przechowywane w postaci kolekcji i mogą być reprezentowane przez pliki typu JSON. Jest to nowoczesne podejście w tworzeniu bazy danych, które pozwala na sprawne zarządzanie bazą i jednoczesny dostęp do danych z wielu urządzeń, bez obawy o awarie czy niespójność informacji. Każda zmiana w bazie (tj. dodanie nowych danych, modyfikacja czy usunięcie) powoduje zmianę w stanie bazy i wywołuje tzw. *event listener*, który jest wykorzystywany w aplikacjach klientów, a więc dane pozostają cały czas aktualne.

Naturalnie każda z kolekcji w grafie jest zbudowana w inny sposób tak, aby przechowywała informacje dla niej przeznaczone. Dla przykładu, poniżej przedstawiono analizę zawartości dokumentu reprezentującego profil wybranej uczelni.



Rys.19 Dane przechowywane przez pojedynczy dokument w /data/colleges

Powyższy dokument, reprezentujący uczelnię (*college*), składa się z kolekcji *Courses* oraz wartości *name* reprezentującej nazwę uczelni, w tym przypadku jest to „UMCS w Lublinie” (niewidoczne na zrzucie ekranu). W kolekcji *Courses* dokumenty są reprezentowane poprzez numer identyfikacyjny unikalny dla każdego z kierunków studiów. Dalej, dla każdego z tych dokumentów możemy wyróżnić:

- Multipliers - kolekcję zawierającą mnożniki dla poszczególnych przedmiotów maturalnych
- Candidates - kolekcję z ID kandydatów (FK) oraz ich wyniki obliczone w oparciu o mnożniki
- Capacity – pole reprezentujące maksymalną liczbę kandydatów
- Desc – pole z opisem kierunku, zdefiniowany przez moderatora uczelni
- Enrolled – pole z liczbą chętnych kandydatów
- Name – pole zawierające nazwę kierunku

Jak widać po powyższym przykładzie, dane przechowywane w bazie Realtime Database są przechowywane w sposób mocno zagnieżdżony. Aby zarządzać tak zapisanymi danymi, konieczne było stworzenie szerokiej logiki, obsługującej powiązane informacje z różnych kolekcji, dbającej o zachowanie ciągłej spójności. Dokładne rozwiązania można znaleźć w kodzie źródłowym, który został załączony razem z pracą do systemu APD.

W tym miejscu należałoby poruszyć temat bezpieczeństwa przechowanych danych oraz dostępu do nich. Jak wspomniano wcześniej, za proces uwierzytelniania odpowiada Firebase Authentication, który można uznać za bezpieczny. System ten pozwala również na uwierzytelnianie użytkowników i na podstawie dostarczanych przez niego informacje zarządzanie dostępem do bazy danych. Mechanizm ten został wykorzystany w tym projekcie, a reguły odpowiedzialne za dostęp zostały przedstawione poniżej.

```
"rules": {
  "data": {
      ".read": true,
      "subjects": {".write": false},
      "colleges": {
            "$uid": {".write": "$uid === auth.uid"}
      },
      "students": {
            ".read": "auth != null",
            "$uid": {
                  ".read": "$uid === auth.uid",
                  ".write": "$uid === auth.uid"
            }
      },
      "results": {
            ".read": "auth != null",
            ".write": false,
            "students": {
                  "$uid": {".read": "$uid === auth.uid"},
                  "colleges": {
                        "$uid":{".read": "$uid === auth.uid"}
                  },
            }
      }
  },
  "admin": {
      ".read": true,
      ".write": false
  }
}
```

Rys.20 Reguły dostępu do bazy danych

Zbiór reguł dostępu do Realtime Database jest przechowywany na serwerze w postaci pliku JSON. Odpowiada on za nadawanie praw odczytu *.read* oraz zapisu *.write* do odpowiednich miejsc w bazie danych. Reguły są dziedziczone.

Przykładowo, dla kolekcji *subjects* wszyscy mają prawo odczytu, ale nikt nie może nadpisywać zawartych tam danych. Z kolei dla kolekcji *data/students* odczyt listy zawierającej ID studentów jest dostępny dla wszystkich zalogowanych użytkowników systemu, jednak do indywidualnego profilu studenta ma dostęp wyłącznie jego właściciel, posiadający pełne uprawnienia do zapisu i odczytu w obrębie własnego dokumentu.

Podsumowanie

W niniejszej pracy przedstawiono scentralizowany system rekrutacji na studia oparty na algorytmie odroczonej akceptacji Gale'a Shapleya, który ma na celu zwiększenie efektywności i przejrzystości procesu rekrutacji oraz zapewnienie sprawiedliwego rozwiązania w tym zakresie.

Analizując problem rekrutacji na studia i brak istniejącego scentralizowanego systemu, określono podstawowe założenia oraz wybrano algorytm, który zapewnił sprawiedliwość i optymalność owego procesu. Na potrzeby pracy przeanalizowano podobny, istniejący system stosowany w zapisach do szkół średnich, zwracając uwagę na jego wady i niedociągnięcia, identyfikując obszary wymagające poprawy.

Następnie omówiono zarówno teoretyczną, jak i praktyczną stronę algorytmu Gale'a Shapleya, skupiając się na jego wykorzystanej wersji. Przedstawiono różne warianty algorytmu i przeprowadzono symulację procesu rekrutacji w Polsce. Wykorzystanie algorytmu Gale'a Shapleya do postawionego problemu rekrutacji okazało się skuteczne i efektywne. Dla losowo wygenerowanych danych 43 uczelni, 3427 kierunków które obejmowały około 30000 miejsc dla prawie 270000 zgłoszonych kandydatów. Algorytm, biorąc pod uwagę zgłoszone przez nich preferencje i uzyskane wyniki z matur dokonał zapisów uzyskując około 97% skuteczność ogólną w tym 66% kandydatów zapisano na kierunek będący ich pierwszym wyborem. Rezultaty uzyskano w około 1054ms. Wyniki symulacji zaprezentowane w pracy wskazują na wyraźny potencjał wdrożenia algorytmu w praktyce.

W pracy opisano również wykorzystane technologie i narzędzia, a także uzasadniono ich wybór. Wykorzystane narzędzia pozwoliły na ukończenie niniejszego projektu, a praca z nimi była wygodna i nie sprawiała problemów, więc był to trafny wybór.

Ostatni rozdział skupił się na zaimplementowanej aplikacji, stanowiącej scentralizowany system rekrutacji. W części dla użytkowników zawarto instrukcję korzystania z systemu i opisano jego możliwości, a w części dla programistów przedstawiono wykorzystane rozwiązania, działanie wybranych funkcjonalności i rozpisano całą strukturę systemu.

Podsumowując całość należy stwierdzić, że cel pracy został osiągnięty poprzez stworzenie scentralizowanego systemu rekrutacji na studia oraz jego prezentację. Aplikacja webowa i algorytm obsługujący zapisy zostały poprawnie zaimplementowane i przetestowane, a przedstawiona koncepcja systemu może być dalej rozwijana i wykorzystywana jako nowoczesna alternatywa dla obecnych rozwiązań.

Wnioski z analizy niniejszego systemu i porównania go z istniejącymi systemami powinny być wzięte pod uwagę przez odpowiednie instytucje i mogą przyczynić się do poprawy jakości i przejrzystości procesu na wszystkich poziomach edukacji. Ponadto, innowacyjne podejście do zaimplementowanego systemu może stać się wzorcem dla innych procesów naboru, nie tylko w obszarze edukacji.

Bibliografia

1. Ministerstwo Edukacji Narodowej. (2022). *Wyniki rekrutacji na studia w roku akademickim 2021/2022 w uczelniach nadzorowanych przez Ministra Edukacji i Nauki*. https://www.gov.pl/web/edukacja-i-nauka/wyniki-rekrutacji-na-studia-w-roku-akademickim-20212022-w-uczelniach-nadzorowanych-przez-ministra-edukacji-i-nauki (Dostęp: 04.01.2023)
2. Gale, D., & Shapley, L. S. (1962). *College Admissions and the Stability of Marriage*. The American Mathematical Monthly, 69(1), 9–15. doi: https://doi.org/10.2307/2312726
3. Kazuo, I., & Shuichi, M. (2008). *A Survey of the Stable Marriage Problem and Its Variants*. doi: https://doi.org/10.1109/ICKS.2008.7
4. Mairson, H. (1992). *The Stable Marriage Problem*, The Brandeis Review V.12: no.1
5. OpenGenus: *Gale-Shapley Algorithm*. https://iq.opengenus.org/gale-shapley-algorithm/ (Dostęp: 21.02.2023)
6. Fudenberg, D., & Tirole, J. (1991). *Game theory*.
7. James W. Jawitz. (2004). *Moments of truncated continuous univariate distributions*. doi: https://doi.org/10.1016/j.advwatres.2003.12.002
8. Zametti, F. (2020). *Modern Full-Stack Development*.
9. Microsoft Corporation: *TypeScript Documentation*. https://www.typescriptlang.org/docs/ (Dostęp: 02.03.2023)
10. Curry, D. (2021). *Cloud Data Insights*. https://www.clouddatainsights.com/typescript-fastest-growing-programming-language-javascript-most-popular/ (Dostęp: 02.03.2023)
11. Meta Platforms (2023): *React Docs*. https://pl.reactjs.org/ (Dostęp: 21.03.2023)
12. Bootstrap (2023): *Bootstrap Docs*. https://getbootstrap.com/docs/5.2 (Dostęp: 04.03.2023)
13. Firebase (2023): *Firebase Docs*. https://firebase.google.com/docs (Dostęp: 17.03.2023)
14. Microsoft (2023): *Visual Studio Code Docs*. https://code.visualstudio.com/docs (Dostęp: 21.03.2023)
15. Canonical (2022): *Ubuntu*. https://ubuntu.com/ (Dostęp: 22.03.2022)
16. DAC.digital (2022): *Technology stack in a nutshell*. https://dac.digital/what-is-a-tech-stack-technology-stack-in-a-nutshell/ (Dostęp: 22.03.2022)
17. Centralna Komisja Egzaminacyjna (2022): *Sprawozdanie z egzaminu maturalnego*. https://cke.gov.pl/images/_EGZAMIN_MATURALNY_OD_2015/Informacje_o_wynikach/2022/sprawozdanie/ (Dostęp: 23.03.2022)
18. Opinie o Uczelniach (2022): *Uniwersytety*. https://opinieouczelniach.pl/typy-studiow/uniwersytety/ (Dostęp: 23.03.2022)
19. Centralna Komisja Egzaminacyjna (2022): *Informator o egzaminie maturalnym*. https://cke.gov.pl/egzamin-maturalny/egzamin-maturalny-w-formule-2023/informatory/ (Dostęp: 23.03.2022)
20. Wybieram Szkołę (2019): *Algorytm Rekrutacji Elektronicznej*. https://www.wybieramszkole.pl/blog/31/ (Dostęp: 27.03.2023)