Aplikacja webowa do obsługi rozgrywek tenisowych

(English title)

Marcin Wróbel

Praca inżynierska

Promotor: dr hab. Dariusz Biernacki

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

8 stycznia 2024

Streszczenie

Celem pracy jest zaprojektowanie i stworzenie aplikacji webowej umożliwiającej organizację rozgrywek tenisa ziemnego. Projekt obejmuje opracowanie responsywnego interfejsu użytkownika wraz z zapleczem. Każdy element aplikacji został skonteneryzowany i skonfigurowany do współpracy z resztą projektu.

TODO streszczenie po angielsku ...

Spis treści

Rozdział 1.

Wprowadzenie

Obsługa amatorskich lig tenisa ziemnego bardzo często polega na wykorzystaniu notesu i długopisu. W ramach tych lig zawodnicy są zazwyczaj zobowiązani do własnej organizacji meczów i pozyskiwania danych kontaktowych innych uczestników w celu ustalenia terminów spotkań. Pojawiła się potrzeba stworzenia nowoczesnego narzędzia do organizacji amatorskich rozgrywek ligowych. W poniższej pracy przedstawiono aplikację webową, która odpowiada na tę potrzebę.

Aplikacja umożliwia łatwe wprowadzanie wyników meczów, weryfikuje je oraz uwzględnia przypadki szczególne, takie jak walkover, czy krecz. Na podstawie wyników meczów sytuacja w grupie i drabince turniejowej jest obliczana automatycznie. Zawodnicy mają dostęp do danych kontaktowych osób, z którymi mają rozegrać mecz i są powiadamiani o zakończonych meczach i zmianach w tabeli. Aplikacja udostępnia także bogatą parametryzację, pozwalającą dostosować system do wymagań danej rozgrywki.

Co istotne, dzięki responsywnemu interfejsowi użytkownika, aplikacja jest przystosowana zarówno do urządzeń mobilnych, jak i stacjonarnych, Dzięki temu zawodnicy mają możliwość korzystania z systemu nie tylko w domu, ale również w przerwach w pracy, czy na korcie tenisowym. Aplikacja webowa została opublikowana i jest dostępna na stronie `www.rozgrywkitenisa.pl`.

Rozdział 2.

Opis zagadnienia

Na początku projektu stworzone zostały tylko podstawowe funkcjonalności. Z czasem zostały dokładane nowe pomysły i ostatecznie zrealizowano następujące funkcjonalności:

- Tworzenie rozgrywek, faza grupowa jest opcjonalna
- Liczba grup i zawodników w fazie pucharowej jest konfigurowalna. W pojedynczej rozgrywce liczba zawodników została ograniczona do 128, a liczba grup została ograniczona do 24. Limity zostały wybrane arbitralnie i mogą zostać zmienione. Niestety limity są konieczne ponieważ nie można pozwolić użytkownikowi na stworzenie rozgrywki o bardzo dużej (np. 10000) liczbie grup, lub zawodników. Przeglądarka nie potrafiłaby ich wyrenderować.
- Podczas tworzenia rozgrywki skład grup jest losowany. Losowanie jest wspomagane rankingiem. W przypadku rozgrywek bez fazy grupowej rozstawienie zawodników w fazie pucharowej ustalane jest na podstawie rankingu.
- zawodnicy mają dostęp do danych kontaktowych osób, z którymi będą rozgrywać mecz
- Wynik meczu może wpisywać organizator/sędzia, lub zawodnicy.
- Wynik meczu obsługuje przypadki szczególne:
  - walkower - w tabeli wynik liczony jest tak jakby zawodnik poddający się przegrał wszystkie gemy, aż do zwycięstwa w meczu drugiego zawodnika,
  - krecz - w tabeli wynik liczony jest tak jakby zawodnik poddający się przegrał wszystkie gemy od momentu wycofania się, aż do zwycięstwa w meczu drugiego zawodnika.
- Rejestracja użytkowników. Hasło można zresetować za pomocą emaila podenego podczas rejestracji.
- Obliczanie rankingu na podstawie wyników zakończonych rozgrywek

2.1. Generowanie grup

Zawodnicy są podzieleni na koszyki według rankingu. Dla rozgrywki o N zawodnikach i M grupach w pierwszym koszyku znajduje się M zawodników o najlepszym rankingu, w drugim koszyku kolejne M zawodników, itd. Wyjątkiem jest ostatni koszyk. Jeżeli $N \not\equiv 0 \pmod{M}$, wtedy liczba zawodników w ostatnim koszyku jest mniejsza, wynosi $N \bmod M$.

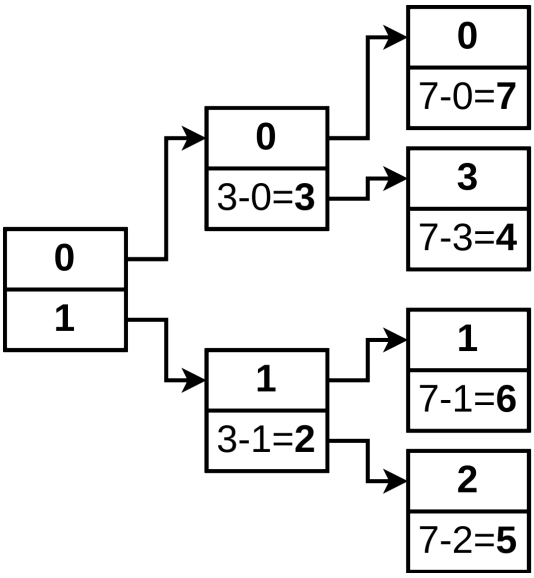
2.2. Generowanie drabinki fazy pucharowej

Na początku zawodnicy są sortowani. Jeżeli rozgrywka posiada fazę grupową, to kolejność zależy od wyników w grupie. Największe znaczenie ma pozycja w grupie, potem bilans meczów, bilans setów, a na końcu bilans gemów. Jeżeli rozgrywka składa się tylko z fazy pucharowej to kolejność zawodników zależy od pozycji w rankingu ogólnym.

Posortowanym zawodnikom przypisane zostają numery startowe od 0 do $N-1$, gdzie N to ilość zawodników. Numery startowe są następnie przypisywane do drabinki według poniższego algorytmu. Jego wizualizacja znajduje się na rysunku 2.1.

```
numeryStartowe=[0,1]
dopoki (numeryStartowe.rozmiar() < N) {
    poprzednieNumeryStartowe = numeryStartowe
    numeryStartowe = []
    najwiekszyNumer = poprzednieNumeryStartowe.rozmiar() * 2 - 1
    dla kazdego (ns w poprzednieNumeryStartowe) {
        numeryStartowe.dodaj(ns)
        numeryStartowe.dodaj(najwiekszyNumer - ns)
    }
}
zwroc numeryStartowe
```

Kolejne pary w zwróconej liście odpowiadają kolejnym parom w drabince fazy pucharowej. Przykładowo dla $N = 8$ zostanie zwrócona lista $[0, 7, 3, 4, 1, 6, 2, 5]$. Odpowiada to parom $[0, 7]$, $[3, 4]$, $[1, 6]$, $[2, 5]$.



Rysunek 2.1: Schemat generowania drabinki fazy pucharowej

Rozdział 3.

Opis aplikacji

TODO inne sekcje opisu ...

3.1. Instalacja

Aplikacja została przygotowana do uruchomienia w dwóch środowiskach:

- lokalnym - do testowania na urządzeniu użytkownika,
- produkcyjnym - gdy aplikacja jest publicznie dostępna

Instrukcje instalacji dla środowiska lokalnego i produkcyjnego różnią się, różnice są oznaczone.

3.1.1. Zmienne środowiskowe

Należy utworzyć pliki zawierające zmienne środowiskowe (`.env`). Pliki `.env.example` zawierają przykładowe wartości, które pozwalają na uruchomienie w środowisku lokalnym.

- ```
  cp .env.example .env
  ```
- ```
  cd frontend
  cp .env.example .env
  ```

Następnie można dostosować wartości do własnej konfiguracji.

3.1.2. Instalacja zależności

Aby zbudować i uruchomić projekt należy zainstalować `Docker` zgodnie z instrukcjami zawartymi w oficjalnej dokumentacji
`https://docs.docker.com/`
`https://docs.docker.com/engine/install/`
Użytkownik musi należeć do grupy `docker`

```
sudo usermod -aG docker nazwa_uzytkownika
```

3.1.3. Budowanie

Aby zbudować projekt należy wykonać poniższe polecenie:

```
# srodowisko produkcyjne
docker compose build
# srodowisko lokalne
docker compose -f docker-compose-localhost.yml build
```

3.1.4. Uruchamianie

```
# srodowisko produkcyjne
docker compose up -d
# srodowisko lokalne
docker compose -f docker-compose-localhost.yml up -d
```

Flaga `-d` nie jest wymagana, uruchamia projekt w tle.

3.1.5. Wyłączanie

```
docker compose down
```

Rozdział 4.

Architektura aplikacji

4.1. Struktura aplikacji

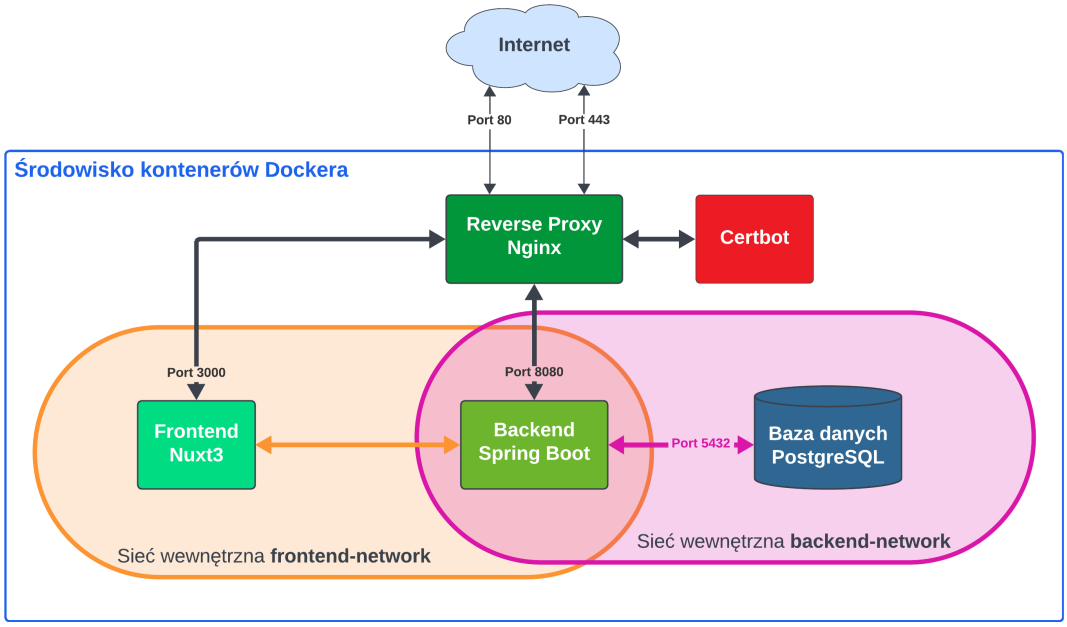
Aplikacja została podzielona na 5 głównych części:

- **frontend** - cześć aplikacji widoczna dla użytkownika
- **backend** - odpowiada za przekazywanie i odbieranie danych od frontendu, odczytuje i zapisuje dane do bazy danych
- **baza danych** - przechowuje dane aplikacji
- **reverse proxy** - wysyła, odbiera, przekazuje zapytania do odpowiednich części aplikacji
- **narzędzie do odnawiania certyfikatów SSL/TLS** - automatyzuje proces odnawiania certyfikatów SSL/TLS

Każda część aplikacji jest opisana w następnej sekcji.

4.1.1. Docker

W celu utrzymania dobrej współpracy pomiędzy wszystkimi elementami aplikacji zdecydowałem się użyć Dockera[1]. Każda z 5 części aplikacji jest opakowywana w osobny kontener, czyli lekką maszynę wirtualną, która zawiera tylko potrzebne do jej działania zależności. Konfiguracja całego środowiska znajduje się w pliku `docker-compose.yml`. Najważniejsze elementy zostały pokazane na rysunku 4.1.



Rysunek 4.1: Struktura kontenerów Dockera

4.2. Kontenery Dockera

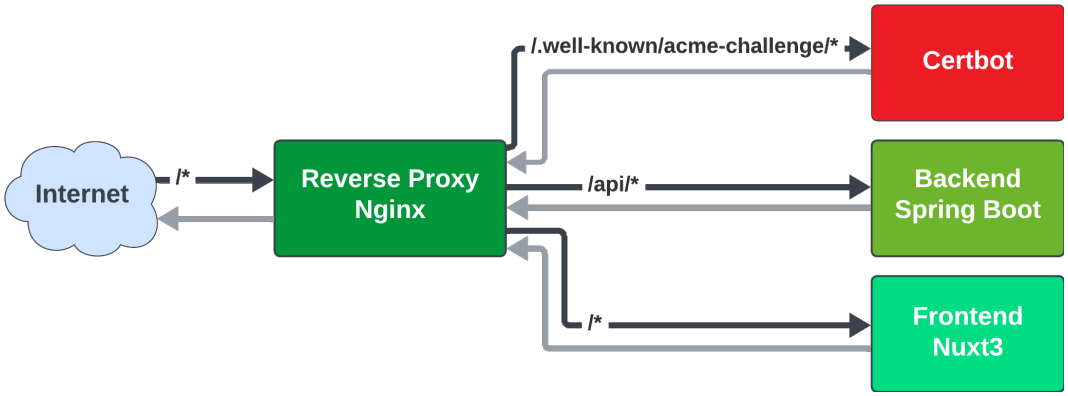
4.2.1. Reverse Proxy - Nginx

Nginx[2] działa jako Reverse Proxy i jest odpowiedzialny za przekierowywanie zapytań http i https do odpowiednich części aplikacji. W folderze `nginx/config/` znajdują się 3 pliki konfiguracyjne. Podczas działania aplikacji używany jest dokładnie jeden z nich.

- `nginx_init_ssl_certificate.conf` jest używany podczas pierwszego uruchomienia w środowisku produkcyjnym, obsługuje tylko zapytania http i pozwala kontenerowi Certbot(4.2.5.) na pobranie certyfikatu SSL/TLS. Jest potrzebny, ponieważ nie jest możliwe obsługiwanie zapytań https bez poprawnego certyfikatu SSL/TLS.

- `nginx.conf` jest używany w środowisku produkcyjnym. Definiuje przekierowywanie zapytań do odpowiednich kontenerów. W tej konfiguracji nieszyfrowana komunikacja za pomocą http jest przekierowywana do szyfrowanej komunikacji za pomocą https.

- `nginx_localhost.conf` jest używany w środowisku lokalnym.



Rysunek 4.2: Przepływ zapytań https zdefiniowanych w pliku `nginx.conf`

Na rysunku 4.2 pokazano schemat przepływu zapytań https dla pliku `nginx.conf`. W przypadku zapytania http reverse proxy nie przekazuje zapytania do kolejnego kontenera, a odpowiada klientowi od razu przekierowaniem do https. Dla zapytań https zdefiniowane są następujące reguły:

1. Jeżeli url zaczyna się od `/.well-known/acme-challenge/` przekaż zapytanie do kontenera Certbot.

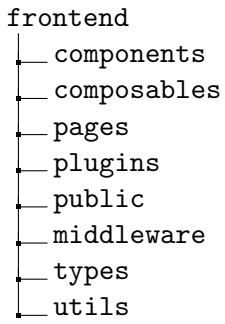
2. Jeżeli url zaczyna się od `/api` przekaż zapytanie do kontenera backend.

3. W przeciwnym przypadku przekaż zapytanie do kontenera frontend.

Wszystkie odpowiedzi na zapytania przekazywane są do reverse proxy, a reverse proxy przekazuje je dalej do nadawcy zapytania.

4.2.2. Frontend - Nuxt3, Vue.js

Frontend został napisany we frameworku Nuxt3 [3], który oparty jest na Vue.js. Użyłem języka TypeScript, ponieważ wprowadza on statyczne typowanie. TypeScript w porównaniu do JavaScriptu ułatwił pisanie kodu, dzięki lepszym podpowiedziom środowiska programistycznego oraz pozwolił zapobiegać błędom, szybciej ostrzegając o problemach podczas kompilacji. Do utrzymania jednolitego formatowania kodu źródłowego użyłem narzędzia Prettier [4]. Kod źródłowy frontendu znajduje się w folderze `frontend/`. W projekcie struktura plików frontendu oparta jest na standardach zdefiniowanych w dokumentacji Nuxt3.



Rysunek 4.3: Zarys struktury plików

Zawartość tych folderów jest następująca:

- `components/` zawiera elementy interfejsu użytkownika (np. tabela wyników, element tablicy wyników, drabinka turniejowa)
- `composables/` zawiera elementy logiki, które posiadają swój stan i są używane w wielu miejscach. Moja aplikacja definiuje `AuthStatus`, który jest odpowiedzialny za przechowywanie informacji o tym, czy użytkownik jest zalogowany, czy nie.
- `pages/` struktura adresów url odpowiada plikom w tym folderze (np. `rozgrywkitenisa.pl/rozgrywki/123` $\longrightarrow$ `pages/rozgrywki/[id].vue`)
- `plugins/` zawiera konfigurację dwóch pluginów (FontAwesome [6] (dostarcza ikony aplikacji), VueDatePicker [7] (Dostarcza narzędzie do wybierania daty)
- `public/` zawiera statyczne pliki, które nie są zmieniane podczas kompilacji i są przekazywane jako niezmienione klientowi
- `middleware/` zawiera middleware, które odpowiedzialne są za modyfikowanie zapytań przed przekazaniem ich dalej. Moja aplikacja używa tylko 1 middleware `logged.in.ts`, który przekierowuje niezalogowanych użytkowników do strony logowania
- `types/` zawiera pliki definiujące typy języka TypeScript
- `utils/` zawiera funkcje i stałe używane w co najmniej 2 plikach aplikacji

Oprócz wyżej wymienionych technologii zastosowałem bibliotekę TailwindCSS [8] do stylizacji aplikacji. W pliku `tailwind.config.js` znajduje się konfiguracja tej biblioteki, wraz z dedykowaną dla tego projektu paletą kolorów. Wielokrotnie używałem prefiksów dla stylów oferowanych przez TailwindCSS, które aktywują/dezaktywują dany styl w zależności od wielkości ekranu. Dzięki temu aplikacja jest responsywna i dostosowana do komputerów i urządzeń mobilnych.

4.2.3. Backend - Spring Boot

Backend napisałem używając frameworka Spring Boot. Do zarządzania zależnościami i do automatyzacji budowania użyłem narzędzia Maven.
Kod backendu został podzielony na warstwy:

- **kontrolery** - odpowiedzialne za przetworzenie parametrów żądania, wysłanie odpowiedzi
- **serwisy** - odpowiedzialne za logikę biznesową backendu
- **repozytoria** - odpowiedzialne za komunikację z bazą danych

Dodatkowo każdy obiekt przechowywany w bazie danych posiada klasy odpowiedzialne za ich działanie:

- **encje** - definiują pola, cechy obiektu przechowywanego w bazie danych
- **obiekty** transferu danych (DTO) - definiują obiekty przesyłane z, lub do serwera
- **maper** - klasy służące do konwertowania obiektów między encjami, a obiektami transferu danych

Przykładowo dla obiektu odpowiedzialnego za rozgrywkę (`Tournament`) zdefiniowane są następujące klasy:

- `TournamentController` - kontroler
- `TournamentMapper` - maper
- `TournamentRepository` - repozytorium
- `TournamentService` - serwis
- `Tournament` - encja
- `TournamentCreateDto` - obiekt transferu danych wysyłany przez frontend podczas żądania stworzenia rozgrywki
- `TournamentBasicDto` - obiekt transferu danych zawierający tylko podstawowe informacje o rozgrywce, jest używany podczas wysyłania listy wszystkich rozgrywek

4.2.4. Baza danych - PostgreSQL

Do przechowywania danych wybrałem PostgreSQL. Zdecydowałem się wybrać tą technologię ze względu na to, że chciałem użyć relacyjnej bazy danych, która jest wydajna, niezawodna i aktywnie rozwijana. PostgreSQL spełnia wszystkie te wymagania.

Konfiguracja bazy danych jest zarządzana przez backend i narzędzie Flyway [10]. W folderze `backend/src/main/resources/db/migration` znajdują się pliki z poleceniami SQL definiujące migracje pomiędzy wersjami. Gdy schemat bazy danych był zmieniany dokładałem kolejny plik odpowiedzialny za migrację. Przy uruchomieniu backendu Flyway automatycznie sprawdza, czy zostały dodane jakieś pliki migracyjne. Jeżeli tak, to dokonuje migracji. Dzięki temu baza danych używana lokalnie oraz baza danych na serwerze w wersji produkcyjnej utrzymują ten sam schemat bazy danych.

4.2.5. Narzędzie do odnawiania certyfikatów SSL/TLS - Certbot

Dla bezpieczeństwa użytkowników połączenie z stroną internetową powinno być szyfrowane. Do nawiązania szyfrowanego połączenia używane są certyfikaty SSL/TLS, które zawierają klucz publiczny i inne dane, takie jak okres ważności, dane właściciela, wystawcy. Każdy ma możliwość wystawić własny certyfikat, ale przeglądarka będzie ufać tylko certyfikatom wydanym przez zaufane (według twórców przeglądarki) instytucje. Świadomy użytkownik znając wystawcę certyfikatu może ręcznie dodać go do listy zaufanych certyfikatów. Takie rozwiązanie jest właściwe tylko wtedy, gdy nie chcemy powierzać generowania kluczy zewnętrznej instytucji i użytkownicy mogą potwierdzić jego prawdziwość (np. gdy wystawiamy certyfikat aplikacji webowej do użytku wewnętrznego firmy). W związku z tym, że aplikacja webowa będąca tematem pracy będzie publicznie dostępna, generowanie certyfikatów SSL/TLS powierzyłem organizacji Let's Encrypt [12]. Let's Encrypt wystawia darmowe certyfikaty SSL/TLS oraz umożliwia automatyzację tego procesu. Kontener Certbot oparty jest na narzędziu o tej samej nazwie [11]. Odpowiada on za automatyczne odnawianie certyfikatów poprzez wysyłanie zapytania do Let's Encrypt i odpowiadanie serwerom tej organizacji na zapytania weryfikujące - czy prośbę o wydanie certyfikatu wysłał właściciel domeny rozgrywkitenisa.pl.

Bibliografia

[1] Dokumentacja Dockera, https://docs.docker.com/

[2] Dokumentacja Nginx, https://docs.nginx.com/

[3] Nuxt3, https://nuxt.com/

[4] Prettier https://prettier.io/

[5] Nuxt3 - dokumentacja struktury plików w folderze pages/, https://nuxt.com/docs/guide/directory-structure/pages

[6] FontAwesome https://fontawesome.com/docs

[7] VueDatePicker https://vue3datepicker.com/

[8] TailwindCSS https://tailwindcss.com/

[9] SpringBoot https://spring.io/projects/spring-boot/

[10] Flyway https://flywaydb.org/

[11] Certbot https://certbot.eff.org/

[12] LetsEncrypt https://letsencrypt.org/pl/