Implementacja aplikacji webowej służącej do obsługi turniejów do gry Counter Strike: Global Offensive

(Implementation of web application
used to organize tournaments for
Counter Strike: Global Offensive)

Tomasz Woszczyński

Praca inżynierska

Promotor: dr Wiktor Zychla

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

21 czerwca 2022

Streszczenie

Z roku na rok rośnie liczba organizowanych profesjonalnych turniejów w wielu grach, jednak narzędzia wspomagające organizację turniejów są zwykle niedostępne dla widzów. Implementacja tej aplikacji miałaby się przyczynić do ułatwienia organizacji turniejów dla każdego. Z założenia miałaby głównie wspierać:

- system ligowy, pucharowy i mieszany,
- rozgrywane mecze w formacie BO1/3/5 (Best of...),
- odrzucanie map przez kapitanów obu drużyn,
- możliwość tworzenia turniejów prywatnych i publicznych,
- panel administracyjny do zarządzania rozgrywkami.

Ponadto implementacja będzie korzystać z dwóch frameworków front-endowych: Vue oraz Cycle.js, które prezentują całkowicie odmienne podejście do programowania aplikacji webowych.

The number of organized professional tournaments in many games is growing every year, but the tools for organizing tournaments are usually inaccessible to spectators. Implementation of this application would contribute to enabling the organization of tournaments for everyone. By design it would mainly support:

- league, cup and mixed system,
- matches played in BO1/3/5 (Best of...) format,
- discarding of maps by the captains of both teams,
- possibility to create private and public tournaments,
- administration panel to manage the games.

Moreover, the implementation will use two front-end frameworks, i.e. Vue.js and Cycle.js, which present a completely different approach to web application programming.

Spis treści

Rozdział 1.

Wprowadzenie

1.1. Motywacja

W ciągu ostatnich lat brałem udział w kilkunastu amatorskich, jak i półprofesjonalnych turniejach offline w grze Counter Strike: Global Offensive i niejednokrotnie byłem świadkiem chaosu, który powstawał poprzez niepoprawne podejście organizatorów do organizacji zawodów. Zwykle wiązało się to z prowadzeniem wszelkich statystyk w niepoprawnie przygotowanych arkuszach kalkulacyjnych, albo co gorsza - na kartkach, bez żadnych narzędzi wspomagających automatyzację procesu. Kolejnym sporym problemem było wybieranie i odrzucanie map przez obie drużyny, zwykle gracze musieli chodzić po ogromych salach i się szukać, a informacje o wybranych mapach przekazywali administratorom po kilku minutach. Stworzenie aplikacji do organizacji turniejów było pomysłem, który miał wyjść na przeciw oczekiwaniom i zarówno ułatwić udział w turniejach, jak i ich organizację.

1.2. Opis aplikacji CStrikers

Aplikacja CStrikers sprawia, że organizatorzy mogą w łatwy sposób tworzyć turnieje różnych rodzajów, a gracze brać w nich udział. Najczęściej spotykanymi rodzajami turniejów są system pucharowy, system ligowy oraz system mieszany. Wszystkie mecze rozgrywane w turnieju są rozgrywane w jednym z systemów: Best Of 1, Best Of 3 albo Best of 5. Rozgrywane mecze są na mapach wybieranych przez obie drużyny - ich wybór odbywa się na żywo poprzez odrzucanie i wybieranie map przez kapitanów obu drużyn. Po każdym meczu uzupełniane są statystyki końcowe każdej z map, a następnie na ich podstawie wyznaczany jest wynik końcowy.

1.3. Ważniejsze pojęcia i skróty

Definicja 1. Counter Strike: Global Offensive — gra z gatunku first-person shooter stworzona przez Valve w 2012. roku. Do każdej rozgrywki przystępują dwie drużyny, każda z nich składa się z 5 graczy biorących aktywnie udział (często spotyka się zmienników, jednak w jednym momencie może grać tylko 5 osób). Celem każdej z drużyn jest wygranie 16 rund, lub więcej w przypadku dogrywki. W kolejnych rozdziałach tytuł gry skracany będzie do CSGO.

Definicja 2. System pucharowy — sposób rozgrywania turnieju polegający na rozgrywaniu bezpośrednich pojedynków. Zwycięzca spotkania przechodzi do kolejnej rundy, a pokonany odpada z turnieju. W turnieju może wziąć udział liczba drużyn będąca potęgą liczby 2. Często spotykanymi określeniami są Single Elimination oraz Knock-out.

Definicja 3. System ligowy — sposób rozgrywania turnieju, w którym każda drużyna gra ze wszystkimi pozostałymi. Wyniki końcowe ligi są obliczane na podstawie wygranych meczów, a w przypadku takiej samej liczby wygranych brane są pod uwagę również wygrane i przegrane rundy. Innymi określeniami na system ligowy są również system kołowy, każdy z każdym oraz round-robin.

Definicja 4. System mieszany — sposób rozgrywania turnieju będący połączeniem systemu ligowego oraz systemu pucharowego. Uczestnicy turnieju są rozdzielani do czterodrużynowych grup, w których rozgrywa się mecze w systemie kołowym, a następnie dwie najlepsze drużyny z każdej grupy przechodzą do kolejnego etapu rozgrywanego w systemie pucharowym.

Definicja 5. Best of N — określenie na typ pojedynczego meczu. N oznacza liczbę maksymalnie rozgrywanych map w bezpośrednim starciu, z czego aby wygrać cały mecz, należy wygrać $\left\lceil \frac{N}{2} \right\rceil$ map. W systemie BO1 kapitanowie odrzucają mapy dopóki nie zostanie ostatnia, na której będzie odbywał się mecz. Wybór rozgrywanych map w systemie BO3 odbywa się następująco: $B - B - P - P - B - B - P$, gdzie B oznacza mapę zbanowaną (odrzuconą), a P mapę wybraną. W systemie BO5 kapitanowie odrzucają po jednej mapie, a następnie wybierają kolejność rozgrywek na pozostałych mapach.

Definicja 6. Framework — szkielet definujący strukturę aplikacji oraz sposób jej działania. Projekt CStrikers wykorzystuje dwa frameworki front-endowe: Vue.js oraz Cycle.js.

Rozdział 2.

Opis wykorzystanych frameworków

2.1. Vue.js

Vue.js to framework napisany w języku JavaScript. Korzysta z architektury *Model-View-ViewModel*. Powstał on w celu stworzenia bardziej elastycznego narzędzia niż istniejące dotychczas frameworki React.js oraz Angular. Vue charakteryzuje bardzo niski próg wejścia (m.in. dzięki bardzo dobrej dokumentacji), bardzo przejrzysta struktura kodu oraz cechuje go intuicyjność. Głównymi funkcjonalnościami wyróżniającymi ten framework są:

- *declarative binding*, czyli rozszerzenie standardu HTML o składnię wzorców, umożliwiające opisywać wyjście HTML na podstawie stanu JavaScript.

- *reactivity* (reaktywność), czyli automatyczne śledzenie stanu JavaScript na podstawie zmian i aktualizowanie DOM.

Kolejnym dużym plusem jest zastosowanie *Single-File Components*, a więc komponentów zawierających się w jednym pliku `*.vue`. SFC pozwala na enkapsulację logiki komponetentu w JavaScript, wzorca strony w HTML oraz styli CSS w jednym pliku. Podejście do programowania SFC jest zalecane przez autorów frameworku.

Nie jest to jednak jedyna możliwość implementowania aplikacji we Vue.js. Drugim sposobem jest podzielenie komponentu na kilka plików (JS, HTML oraz CSS) i połączenie ich wszystkich poprzez wywołanie metody `createApp` z modułu `vue`, a następnie zamontowania jej do wybranego ID z pliku HTML. To podejście jest jednak używane znacznie rzadziej, jako że przenaszalność komponentów znacząco maleje, co sprawia, że w aplikacji mógłby powstać bardzo powtarzalny kod.

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Rysunek 2.1: Przykład prostego komponentu SFC [?]

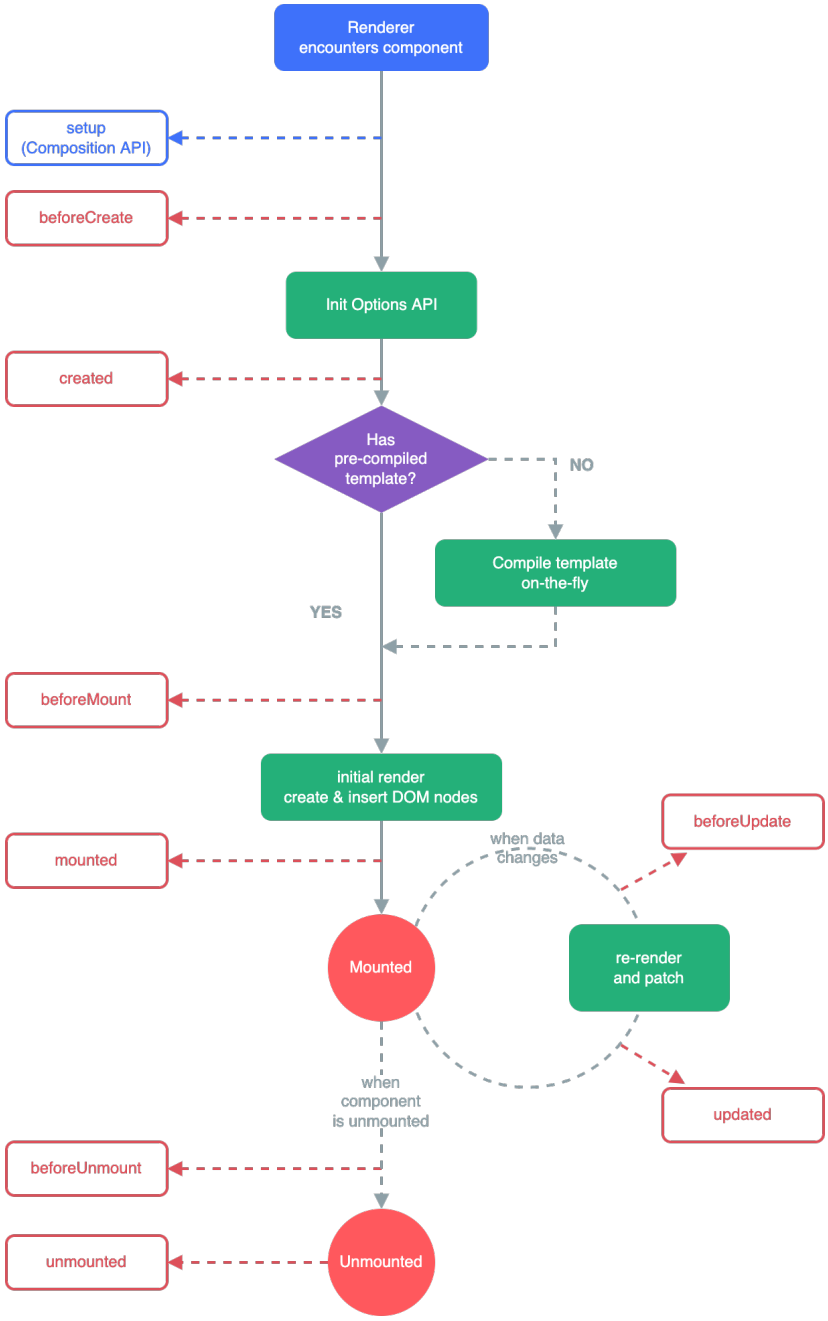
```
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')
```

```
<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```

Rysunek 2.2: Analogiczny komponent podzielony na pliki JS oraz HTML [1]

Ważną częścią Vue.js jest cykl życia komponentów. Każda instancja komponentu przechodzi przez serię kroków inicializacyjnych przy stworzeniu, a sam framework udostępnia możliwość dodania kodu do istniejących *lifecycle hooks*. Dzięki temu programista może w łatwy sposób wpłynąć na działanie komponentu, wywołując w odpowiednich momentach zawołania API do bazy danych, inicializując wartości domyślne danych komponentów i inne.

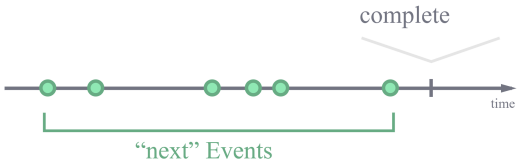


Rysunek 2.3: Cykl życia komponentu Vue.js wraz z *lifecycle hooks* [2]

2.2. Cycle.js

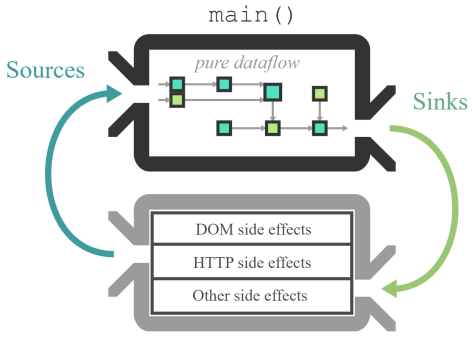
Cycle.js to funkcyjny i reaktywny framework JavaScript. Funkcyjność sprawia, że kod jest bardziej przewidywalny, a reaktywność pozwala na rozdzielenie kodu. Aplikacje Cycle.js są w pełni stworzone z funkcji, dzięki czemu na podstawie wejścia (argumentów funkcji) jest generowane przewidywalne wyjście, bez niepożądanych efektów I/O.

Elementem składowym są reaktywne strumienie z bibliotek takich jak `xstream`, `RxJS`, czy `Most.js`. Strukturyzacja aplikacji Cycle.js za pomocą strumieni usuwa problemy dotyczące zmian z zewnątrz, jako że wszystkie zmiany danych są wewnątrz strumieni. Strumień w `xstream` jest ciągiem wydarzeń, którym może wyemitować zero lub więcej wydarzeń oraz może się skończyć lub nie skończyć. Zakończenie się strumienia oznacza wykonanie jakiegoś działania lub wyemitowanie błędu.



Rysunek 2.4: Zasada działania strumieni [5]

Abstrakcja frameworku wydziela funkcję `main`, której argumentami są efekty odczytów ze świata zewnętrznego (*sources*), a wyjściem są efekty zapisów w celu wpłynięcia na świat zewnętrzny (*sinks*). Efekty I/O są zarządzane przez sterowniki (*drivers*), a więc wtyczki obsługujące efekty DOM, zapytania HTTP, itp.



Rysunek 2.5: Schemat przepływu danych w aplikacji Cycle.js [6]

Cycle.js wykorzystuje architekturę *Model-View-Intent*. Architektura ta spełnia główne założenia *Model-View-Controller*, jest reaktywna i funkcyjna. Reaktywność jest spowodowana tym, że *Intent* obserwuje użytkownika (akcje wywołane z komputera), *Model* obserwuje *Intent*, *View* obserwuje *Model* oraz użytkownik obserwuje *Intent*. Funkcyjność jest wynikiem tego, iż każdy z komponentów jest wyrażony poprzez funkcje na strumieniach.

Rysunek 2.6: Dekompozycja funkcji `main` na `intent`, `model`, `view` [7]

Model-View-Intent nie narzuca dekompozycji funkcji `main`, a więc podział na *Model*, *View* oraz *Intent* nie jest konieczny. Jest to jednak wydajny sposób na organizację kodu i wydzielenie poszczególnych odpowiedzialności.

Cycle.js jest bardzo lekką biblioteką z dużymi możliwościami, dzięki czemu udostępnia programistom zestaw funkcji do tworzenia własnych sterowników i dostosowywania ich pod własne potrzeby. Widoki są generowane poprzez stworzenie *virtual tree* za pomocą tagów `virtual-hyperscript`.

Rozdział 3.

Struktura i architektura aplikacji

3.1. Struktura aplikacji

Katalogiem głównym aplikacji jest `app`. Znajdują się w nim różnorakie pliki konfiguracyjne - projektowe, bazodanowe, czy dotyczące edytora kodu. Znajdują się w nim również dwa katalogi: katalog `public` zawierający plik `index.html` specyfikujący tag `<div id='app'></div>`, wykorzystywany przez Vue w celu wyrenderowania aplikacji, jak i katalog `src` zawierający wszystkie pliki źródłowe. Znajdują się w nim katalogi:

- `api` — katalog zawierający implementację funkcji dotyczących zapytań CRUD do bazy danych wykorzystywanych przez komponenty,
- `assets` — katalog zawierający style CSS,
- `components` — katalog zawierający komponenty paska nawigacyjnego (*navbar*) oraz stopki,
- `configs` — katalog zawierający plik konfiguracyjny Firebase oraz wrapper wykorzystujący konfigurację w celu uzyskania obiektu bazy danych,
- `router` — katalog z plikiem `vue-router` obsługujący przekierowania,
- `store` — katalog z plikiem `vuex` przechowujący globalny stan aplikacji,
- `services` — katalog zawierający komponenty Cycle.js, generatory obiektów, typy, funkcje pomocnicze,
- `tests` — katalog zawierający testy jednostkowe
- `views` — katalog zawierający komponenty Vue.js

3.2. Zastosowana architektura

Aplikacja CStrikers wykorzystuje dwa frameworki front-endowe: Vue.js oraz Cycle.js. Oznacza to, że wykorzystywane są architektury *MVVM* oraz *MVI*, opisane w poprzednim rozdziale. Bazą danych jest Firestore Database, udostępnione funkcje są używane w celu zaimplementowania wspólnego, reużywalnego API dla wszystkich komponentów.

Plik `main.js` odpowiada za inicializajcę aplikacji, wraz z użyciem `Vuex` służącego do przechowywania globalnego stanu we Vue, oraz z `vue-router` służącego do obsługi przekierowań.

Bazowym komponentem wykorzystanym przez aplikację jest `App.vue` - udostępnia on style oraz ogólny widok aplikacji (*wrapper*). Pozostałe komponenty zaimplementowane we Vue korzystają z SFC i są wydzielone do osobnych plików w katalogu `views`. W celu zachowania spójności aplikacji, komponenty Cycle.js są wstrzykiwane do Vue poprzez handlery, które wywołują funkcję `run(main, drivers)`. W ten sposób uzyskano jednolity layout.

```
<template>
  <div id="create-team-cycle"></div>
</template>

<script>
import { run } from "@cycle/run";
import { makeDOMDriver } from "@cycle/dom";
import { main } from "../services/createTeamCycle.js";

export default {
  name: "TeamCreateHandler",
  mounted: function () {
    run(main, {
      DOM: makeDOMDriver("#create-team-cycle"),
    });
  },
};
</script>
```

Rysunek 3.1: Przykład implementacji wstrzykiwania kodu Cycle.js do Vue.js

Rozdział 4.

Aplikacja

4.1. Cele aplikacji

- Tworzenie kont użytkowników oraz możliwość logowania się.
- Tworzenie drużyn przez istniejących użytkowników.
- Przegląd informacji o drużynach, ich członkach oraz turniejach w jakich brały udział.
- Możliwość tworzenia turniejów CSGO w różnych systemach.
- Możliwość ukrycia turnieju z listy wszystkich turniejów.
- Odrzucanie i wybieranie map przez kapitanów drużyn biorących udział w meczu odbywające się *real-time*.
- Możliwość zapisania wyniku spotkania.
- Śledzenie postępów turnieju na podstawie listy wyników oraz tabeli.

4.2. Opisy kolekcji Firestore Database

Firestore Database jest bazą danych mającą podobną strukturę do MongoDB.

4.2.1. Users

Kolekcja korzystająca z kolekcji Users wbudowanej w Firebase Authentication. Przechowuje informacje o użytkownikach: ich pseudonim, imię i nazwisko, email oraz pole uid będące referencją.

4.2.2. Teams

Kolekcja zawierająca informacje o stworzonych drużynach. Polami dokumentów są: nazwa drużyny, ID użytkownika z kolekcji users będącego kapitanem drużyny oraz tablicę ID członków drużyny.

4.2.3. Maps

Kolekcja zawierająca mapy, na których rozgrywane są mecze.

4.2.4. Matches

Kolekcja przechowująca dokumenty dotyczące meczów. Zawiera pola będące ID obu drużyn, typ meczu, tablicę z wybranymi mapami, tablicę z odrzuconymi mapami, tablicę wyników każdej z wybranej map, liczbę wygranych map przez obie drużyny oraz nazwę drużyny, która wygrała.

4.2.5. Tournaments

Kolekcja przechowująca dokumenty dotyczące turniejów. Zawiera pola będące ID organizatora, informacje o widoczności turnieju, system w jakim rozgrywany jest turniej, typ rozgrywanych meczów (B01/BO3/BO5), tablicę ID drużyn biorących udział w turnieju, tablicę ID meczów rozgrywanych w ramach turnieju oraz nazwę drużyny, która wygrała turniej.

4.3. Historyjki użytkownika

4.3.1. Gracz

Gracz — osoba chcąca rozgrywać mecze w turnieju.

- Jako gracz, chcę zarejestrować się w aplikacji, żeby inny gracz mógł mnie dodać do swojej drużyny.
- Jako gracz, chcę stworzyć drużynę, żeby móc wziąć udział w turnieju.
- Jako gracz, chcę mieć możliwość łatwego odrzucania i wybierania map w meczach, w których biorę udział, żeby móc grać na mapach, które preferuje mój zespół.
- Jako gracz, chcę mieć możliwość wpisywania wyników spotkań, które rozgrywam, żeby wiedzieć w którym miejscu w tabeli/w której rundzie jest moja drużyna.

4.3.2. Organizator turnieju

Organizator turnieju — użytkownik aplikacji chcący stworzyć turniej.

- Jako organizator turnieju, chcę mieć możliwość wyboru drużyn, które wezmą udział w moim turnieju, żeby mieć pewność, że żadna niezaproszona drużyna nie weźmie w nim udziału.
- Jako organizator turnieju, chcę mieć możliwość ukrycia turnieju, żeby rozgrywki były prywatne.
- Jako organizator turnieju, chcę mieć możliwość wyboru typu rozgrywanych meczów, żeby kontrolować czas trwania i balans drużyn w turnieju.
- Jako organizator turnieju, chcę mieć możliwość wyboru systemu, w którym będzie rozgrywany turniej, żebym mógł spełnić swoje wymagania.
- Jako organizator turnieju, chcę mieć podgląd statystyk wszystkich rozgrywanych meczów, żeby kontrolować ich stan.
- Jako organizator turnieju, chcę wiedzieć kim jest zwycięzca turnieju, aby nie musieć przeglądać wszystkich spotkań i liczyć wyników manualnie.

Rozdział 5.

Implementacja aplikacji i jej przebieg

Implementacja od początku sprawiała problemy, jednak po czasie udało się je rozwiązać i ukończyć aplikację zgodnie z założeniami.

Pierwszym pomysłem było stworzenie aplikacji opartej o Express.js oraz backendzie w JavaScript, jednak konfiguracja Vue.js oraz Cycle.js działających jednocześnie była bardzo kłopotliwa, gdyż na tamten moment część paczek była niekompatybilna. Podjąłem wtedy decyzję o zmianie Express.js na Pythona oraz Django, z wykorzystaniem Django Templates. Pomimo ukończonego wcześniej projektu w Pythonie, moja znajomość języka była zbyt niska, aby napisać łatwo rozszerzalny kod, więc po zaimplementowaniu dwóch komponentów kod był praktycznie nierozszerzalny, na co w dużej mierze wpływała ilość zagnieżdżeń oraz wzajemnych referencji encji.

Ostatnim rozwiązaniem, było napisanie aplikacji we Vue.js oraz Cycle.js, wykorzystując przy tym bazę danych Firestore Database. To pozwoliło na stworzenie API wykorzystywanego przez oba frameworki, dzięki czemu implementacja nie sprawiała aż tak dużych problemów. Największą trudność stanowiło jednak zaplanowanie struktur kolekcji, ich referencji oraz kroków implementacyjnych. Przez brak organizacji i konkretnego planu działania, musiałem wielokrotnie zmieniać kod i rozszerzać kolekcje, aby były kompatybilne ze wszystkimi działaniami, jakie można było podjąć w aplikacji.

Testowanie aplikacji ograniczone zostało do funkcji, które nie należały do bibliotek udostępniających m.in. obsługę baz danych, czy tworzenia V-DOM we Vue czy Cycle.js gdyż ich implementacje są w pełni przetestowane.

Dokumentacja aplikacji była sporządzana na bieżąco, w trakcie programowania - plusem takiego podejścia jest możliwość odnoszenia się do różnych funkcji bez konieczności analizowania ich działania, jednak przy częstych zmianach kodu wymagało to dodatkowego czasu.

Rozdział 6.

Podsumowanie

Pierwszym krokiem w implementacji powinno być zaplanowanie struktury aplikacji, przygotowanie kolekcji obejmujących wszystkie możliwe rozwiązania oraz konfiguracja środowiska. Pominięcie któregoś z tych punktów może sprawić, że pisanie kodu będzie bardzo trudne, a potencjalne modyfikacje w przyszłości - niemożliwe. Bardzo ważne jest również podejście *Test-Driven Development* - dzięki temu udało się uniknąć niepotrzebnego debugowania, które przy częstych zmianach kodu bez wykorzystania TDD jest praktycznie nieodłączną częścią.

Programowanie z użyciem dwóch frameworków jest skomplikowane. Narzucając Vue.js jako główną bibliotekę w projekcie, trudno było sprawić, aby Cycle.js zaczął działać tak, jak powinien. Po kilkunastu różnych próbach udało się je jednak ze sobą połączyć poprzez wstrzyknięcie funkcji `run(main, drivers)` do handlera we Vue. Niestety nie można rozdzielić tych frameworków całkowicie - globalny stan z `vuex` przechowuje dane o zalogowanym użytkowniku, a `vue-router` obsługuje przekierowania, a więc konieczne było importowanie bibliotek spoza Cycle.js, aby komponenty spełniały swoje zadania.

Kolejnym problemem wynikającym z użycia dwóch frameworków jest konieczność ciągłego przestawiania swojego podejścia. Różni się nie tylko składnia, czy wykorzystywane biblioteki, ale też architektura. W przypadku Vue.js działanie opiera się na obiektach, ich przekazywanie oraz manipulacja nie są skomplikowane. Cycle.js działa w oparciu o MVI, a wszelkie zmiany są oparte na podejściu funkcyjnym i strumieniach.

W aplikacjach znacznie większych niż CStrikers łączenie różnych frameworków frontendowych nie jest najlepszym pomysłem. Istniejące frameworki mają bardzo szerokie zastosowanie, często rozszerzane przez dodatkowe, niewymagane biblioteki, dzięki czemu ograniczenie się do jednego nie przyczyni się do pogorszenia jakości aplikacji - wręcz przeciwnie, korzystając tylko z jednego frameworku programiście będzie znacznie łatwiej pisać kod zgodny z praktykami dobrego programowania oraz szybciej zyska biegłość, co znacząco przyspieszy implementację.

Rozdział 7.

Opis techniczny

Aplikacja została napisana w języku Vue.js (wersja 3.0.0) oraz Cycle.js (`@cycle/dom` w wersji 23.1.0 oraz `@cycle/run` w wersji 5.7.0). Backend aplikacji obsługujący zapytania bazodanowe oraz logikę aplikacji został napisany w JavaScript. Testy jednostkowe zostały zaimplementowane przy użyciu Jest (wersja 28.1.1), a dokumentacja wygenerowana przez JSDoc (wersja 3.6.10). Za bazę danych posłużył Firestore Database dostarczany przez Firebase.

Kod aplikacji znajduje się w repozytorium GitHub:
`https://github.com/whiskeyo/csgo-tournament`

Rozdział 8.

Polecenia aplikacji CStrikers

Zestaw najważniejszych poleceń znajduje się w pliku `README.md`, są to m.in:

- instalacja vue-cli: `sudo npm install -g @vue/cli`
- instalacja paczek NPM: `cd app; npm install`
- uruchomienie serwera developerskiego: `npm run dev`
- kompilacja projektu z możliwością hostowania na serwerach: `npm run build`
- uruchomienie lintera wskazującego błędy w kodzie: `npm run lint`
- uruchomienie formattera: `npm run format`
- uruchomienie testów jednostkowych: `npm run test`
- wygenerowanie pokrycia kodu: `npm run coverage`
- wygenerowanie dokumentacji: `npm run docs`

Bibliografia

[1] Vue.js - Introduction, Single-File Components - https://vuejs.org/guide

[2] Vue.js - Lifecycle Diagram - https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram

[3] Vue Router - Getting Started - https://router.vuejs.org/guide/

[4] Vuex - Getting Started - https://vuex.vuejs.org/guide/

[5] Cycle.js - Streams - https://cycle.js.org/streams.html

[6] Cycle.js - Dataflow - https://cycle.js.org/#-dataflow

[7] Cycle.js - Model-View-Intent - https://cycle.js.org/model-view-intent.html

[8] Cycle.js - Getting started - https://cycle.js.org/getting-started.html

[9] Cycle.js - API reference - https://cycle.js.org/api/index.html#api-reference

[10] Firestore Database - Documentation - https://firebase.google.com/docs/firestore