

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

22 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

1. Wprowadzenie	9
1.1. Motywacja	9
1.2. Opis aplikacji CStrikers	9
1.3. Ważniejsze pojęcia i skróty	10
1.4. Porównanie z istniejącymi aplikacjami	11
1.4.1. Toornament	11
1.4.2. E-Sports Entertainment Association	11
1.5. Plan pracy	11
2. Opis wykorzystanych frameworków	13
2.1. Vue.js	13
2.1.1. Opis frameworku	13
2.1.2. Powody sprzyjające wyborowi Vue.js	16
2.1.3. Wykorzystanie w implementacji	16
2.2. Cycle.js	16
2.2.1. Strumienie	16
2.2.2. Opis frameworku	17
2.2.3. Powody sprzyjające wyborowi Cycle.js	18
2.2.4. Wykorzystanie w implementacji	19
3. Struktura i architektura aplikacji	21
3.1. Struktura aplikacji	21
3.2. Zastosowana architektura	22

3.3. Model pojęciowy [TODO]	23
3.4. Model C4 [TODO]	23
3.5. Testy jednostkowe komponentów	23
3.6. Metryka kodu	24
4. Aplikacja	25
4.1. Cele aplikacji	25
4.2. Opisy kolekcji Firestore Database	25
4.2.1. Users	25
4.2.2. Teams	26
4.2.3. Maps	26
4.2.4. Matches	26
4.2.5. Tournaments	26
5. Historyjki użytkownika	27
5.1. Gracz	27
5.2. Organizator turnieju	28
6. Implementacja aplikacji i jej przebieg	31
7. Podsumowanie	33
7.1. Wnioski i efekty pracy	33
7.2. Future work – plan rozwoju aplikacji	34
7.2.1. Możliwość zapisywania się na turnieje przez drużyny	34
7.2.2. Ustalenie terminu dla turnieju oraz meczów	34
7.2.3. Czat na żywo	34
7.2.4. Powiadomienia	34
7.2.5. Wsparcie dla dowolnej liczby drużyn	35
7.2.6. Udział w turniejach złożonych z drużyn o mniejszej liczbie graczy	35
7.2.7. Implementacja nowych systemów rozgrywania turniejów	35
7.2.8. Wydzielenie ról użytkowników	35
7.2.9. Obsługa serwerów gry CSGO w trakcie turniejów	35

<i>SPIS TREŚCI</i>	7
7.2.10. Optymalizacja zapytań do bazy danych	36
8. Opis techniczny	37
9. Polecenia aplikacji CStrikers	39
Bibliografia	41

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 ogromnych salach i się szukać, a informacje o wybranych mapach przekazywali administratorom po kilku minutach. Stworzenie aplikacji do organizacji turniejów było pomysłem, który miał wyjść naprzeciw oczekiwaniom i zarówno ułatwić udział w turniejach, jak i ich organizację.

1.2. Opis aplikacji CSStrikers

Aplikacja CSStrikers 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. Aplikacja również śledzi informacje o ilości wygranych rund przez obie strony na każdej mapie.

Aplikacja znajduje się w publicznym repozytorium dostępnym na GitHubie: <https://github.com/whiskeyo/csgo-tournament>.

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 zwykle 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. Counter-Terrorists (CT), Terrorists (T) — dwie strony możliwe do wyboru w CSGO. Obie drużyny w trakcie jednego meczu grają po obu stronach. Drużyny po rozegraniu połowy (15 rund lub 3 w przypadku dogrywki) zostają zamienione.

Definicja 3. 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 tego systemu są *Single Elimination* oraz *Knock-out*.

Definicja 4. System ligowy — sposób rozgrywania turnieju, w którym każda drużyna gra ze wszystkimi pozostałymi. Wyniki 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 5. 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 6. 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 7. Framework — szkielet definiujący strukturę aplikacji oraz sposób jej działania. Projekt CSStrikers wykorzystuje dwa frameworki front-endowe: Vue.js oraz Cycle.js.

1.4. Porównanie z istniejącymi aplikacjami

1.4.1. Toornament

Toornament to oprogramowanie działające jako usługa (*Software as a Service*) [1] służąca do organizacji turniejów w wielu grach dostępnych na różnych platformach. W darmowej wersji udostępnia narzędzia pozwalające na zarządzanie rozgrywkami, umożliwia zapisywanie się na turnieje przez drużyny, jednak ilość dużych biorących udział w jednym turnieju nie może przekroczyć 128. Implementacja aplikacji CStrikers nie posiada takiego limitu i w każdym turnieju może wziąć udział dowolna liczba drużyn.

Toornament pozwala na wykupienie dodatkowych planów rozszerzających możliwości organizatorów. Są to między innymi rozszerzenie ilości uczestników turniejów, dodanie informacji o sponsorach w (pakiet Starter), wykupienie własnej domeny przygotowanej pod organizację jednego turnieju (pakiet Tourney), czy nawet uzyskanie dostępu do API w celu stworzenia własnej usługi na podstawie Toornament (pakiet API) [2].

1.4.2. E-Sports Entertainment Association

E-Sports Entertainment Association (ESEA) to aplikacja umożliwiająca graczom na udział w pojedynczych meczach, ligach oraz turniejach. Udostępnia ona stronę internetową, na której użytkownicy mają dostęp do turniejów, statystyk oraz forum, oraz aplikację będącą jednocześnie oprogramowaniem służącym do szukania gier, dołączania do meczów oraz odrzucania map, jednak ma w sobie wbudowany anti-cheat [3] (tj. system wykrywający osoby używające oprogramowania, które wspomaga ich grę).

Niestety ESEA nie udostępnia użytkownikom narzędzi do organizacji własnych lig ani turniejów. Możliwości użytkowników serwisu ograniczają się jedynie do brania w nich udziału, jednak uczestnictwo w ESEA jest płatne [4].

Aplikacja CStrikers nie posiada narzędzi do wykrywania niechcianego oprogramowania, iż jest aplikacją internetową nieingerującą w pliki użytkownika. Pozwala jednak każdemu użytkownikowi na stworzenie drużyny i wzięcie udziału w rozgrywkach bądź stworzenie własnego turnieju w jednym z trzech systemów.

1.5. Plan pracy

W rozdziale drugim opisano zasadę działania, wykorzystane architektury frameworków Vue.js oraz Cycle.js oraz uzasadniono wybór każdego z nich.

W rozdziale trzecim opisano strukturę aplikacji i jej architekturę. Opis zawiera informacje o katalogach, jakie można znaleźć w repozytorium, mówi o wykorzystanej architekturze, prezentuje diagram modelu pojęciowego oraz model C4, ponadto znajdują się w nim informacje na temat pokrycia kodu testami jednostkowymi oraz metryki kodu.

W rozdziale czwartym opisano cele aplikacji, kolekcje Firestore Database.

W rozdziale piątym zaprezentowano historyjki użytkownika wraz ze zrzutami ekranu aplikacji.

W rozdziale szóstym opisano przebieg implementacji aplikacji CStrikers wraz z pomysłami na wykorzystanie różnych technologii i problemami, jakie wystąpiły przy ich wyborze. Ponadto znajdują się w nim informacje na temat procesu testowania aplikacji oraz sporządzania dokumentacji.

W rozdziale siódmym omówiono wnioski wynikające z wyboru dwóch frameworków do pracy nad jednym projektem, również wyciągnięto wnioski na podstawie doświadczeń przy implementacji. Zaprezentowano też *future work*, czyli pomysły na rozwój aplikacji w przyszłości o dodatkowe funkcjonalności.

W ostatnim, ósmym rozdziale przedstawiono opis techniczny aplikacji, kroki wymagane do skonfigurowania środowiska oraz możliwe do wywołania polecenia.

Rozdział 2.

Opis wykorzystanych frameworków

2.1. Vue.js

2.1.1. Opis frameworku

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 komponentu w JavaScript, wzorca strony w HTML oraz stylów 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 [5]

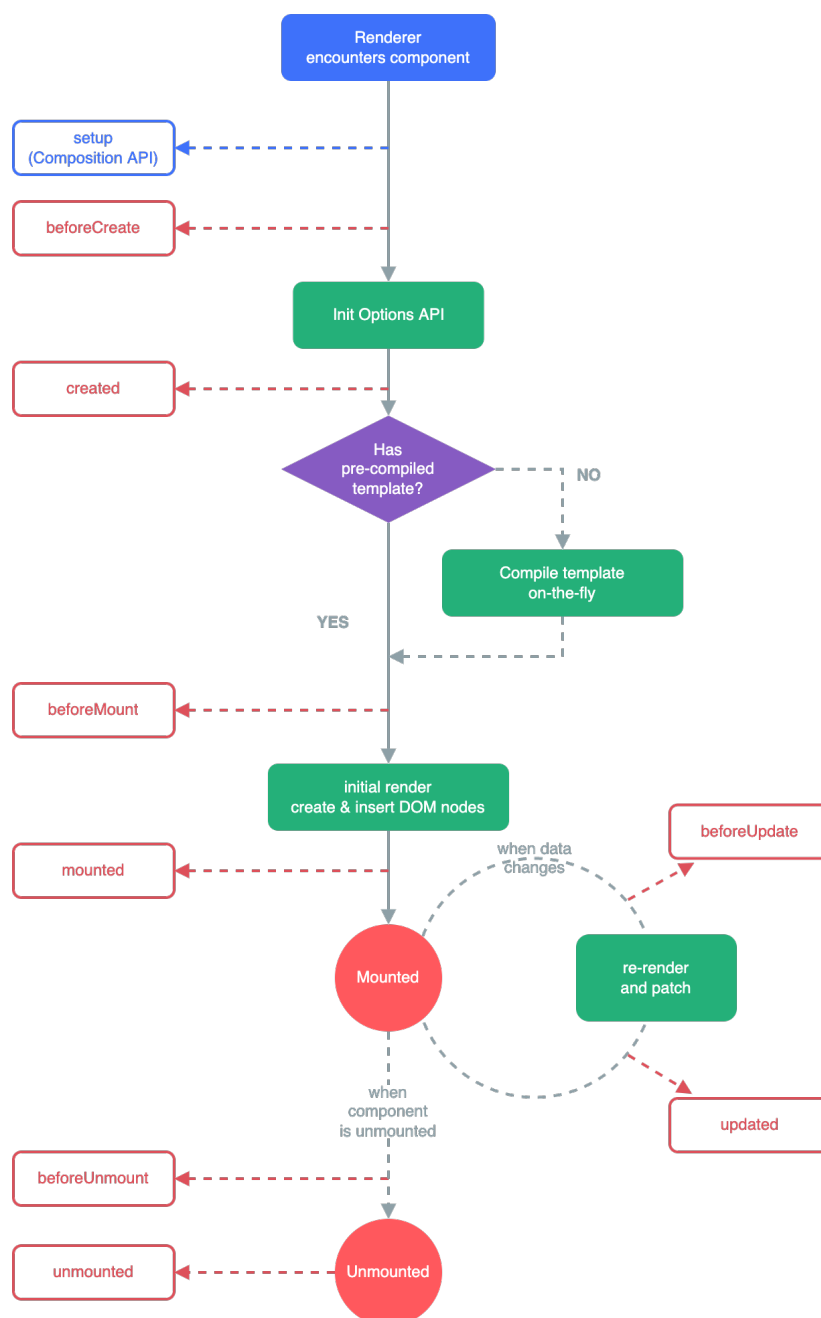
```
import { createApp } from 'vue'

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 [5]

Ważną częścią Vue.js jest cykl życia komponentów. Każda instancja komponentu przechodzi przez serię kroków inicjalizacyjnych 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 zwołania API do bazy danych, inicjalizując wartości domyślne danych komponentów i inne.



Rysunek 2.3: Cykl życia komponentu Vue.js wraz z *lifecycle hooks* [6]

2.1.2. Powody sprzyjające wyborowi Vue.js

Vue.js wybrano z kilku powodów.

Pierwszym z nich jest jasno określony cykl życia komponentów, co sprawia, że w łatwy sposób można podzielić wywołania API przypisujące wartości polom określonym w sekcji `data`. Dobrym tego przykładem jest sposób, w jaki są pobierane dane w pliku `MatchRoom.vue` – `hook beforeCreate` pobiera wszystkie dane dotyczące meczu, są to: nazwy drużyn, członkowie obu zespołów, możliwe mapy do wyboru. Następnie w `hooku created` wywoływana jest funkcja `onSnapshot`, która w czasie rzeczywistym nasłuchuje zmian dotyczących zmieniających się map, dzięki czemu liczba zapytań jest ograniczona do minimum.

Drugim powodem przemawiającym za wyborem Vue są *Single-File Components*. Dzięki nim panuje znacznie większy porządek w repozytorium, a każdy plik jest odpowiedzialny za jedną rzecz.

Kolejnym powodem jest dostęp do wartości z sekcji `data`. Są to obiekty proxy, więc wyciągnięcie z nich danych nie stanowi problemu. Pozwala to również na wykorzystanie bardzo czytelnych funkcji, które mogą być reużywalne na poziomie całego projektu, co obrazuje część funkcji z `utils.js`.

Ostatnim, lecz nie mniej ważnym powodem, jest dostęp do przejrzystej dokumentacji, w której są opisane wszystkie aspekty języka. Twórcy Vue stworzyli również kilkanaście paczek NPM pozwalających na rozszerzenie możliwości Vue o kolejne funkcjonalności, takie jak `Vuex`, czy `Vue-router`.

2.1.3. Wykorzystanie w implementacji

Vue.js został wykorzystany do zaimplementowania dwóch komponentów: paska nawigacyjnego wraz z obsługą rejestracji nowych użytkowników i logowania oraz obsługi turniejów i meczów, tj. tworzenie turniejów, lista wszystkich stworzonych turniejów, pokój meczowy ze wsparciem odrzucania i wybierania map w czasie rzeczywistym, czy lista meczów i statystyk na temat map.

W przypadku komponentów zaimplementowanych w `Cycle.js`, Vue został wykorzystany do wstrzykiwania kodu uruchamiającego komponenty.

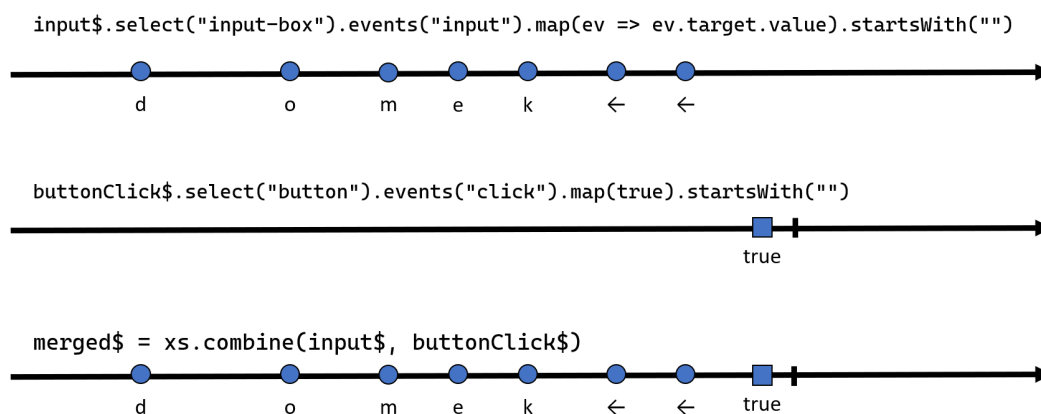
2.2. Cycle.js

2.2.1. Strumienie

Elementem składowym `Cycle.js` są reaktywne strumienie z bibliotek takich jak `xstream`, `RxJS`, czy `Most.js`.

Strumień to obiekt *A* będący obserwatorem innego obiektu *B*. Wszystkie akcje zapisywane są jako ciąg wydarzeń. Oznacza to, że w przypadku wybranej akcji na obiekcie *B* (np. kliknięcie przycisku czy wpisanie tekstu do okienka), obiekt ten wyemituje akcję, która zostanie zapisana w strumieniu w wybranym momencie czasu. Na takim strumieniu można wywołać różne operacje w celu przekształcenia go na oczekiwany efekt, np. zmapować go na jakąś wartość.

Na poniższym schemacie znajduje się przykład działania strumieni. Na wejściu znajdują się dwa strumienie: `input$` nasłuchujący (obserwujący) wydarzenia typu `input` w elemencie o identyfikatorze `input-box`, a `buttonClick$` oczekuje wydarzenia `click` w elemencie o identyfikatorze `button`, a następnie się kończy. Po połączeniu tych strumieni zostanie wywołana funkcja wykorzystująca ze strumienia `input$` tekst "dom", a kliknięcie przycisku może wysłać te dane przez API do bazy danych.



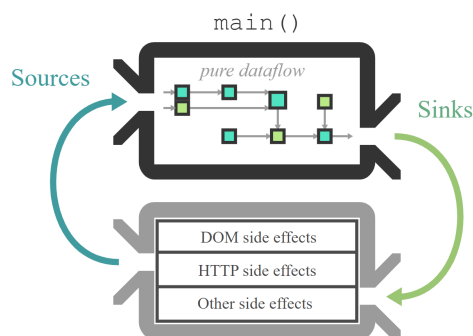
Rysunek 2.4: Działanie strumieni [9] obsługujących pole tekstowe oraz przycisk na podstawie przykładzie działania z pliku `createTeamCycle.js` [opracowanie własne]

Strumienie w `xstream` mogą wyemitować zero lub więcej wydarzeń. Mogą one być również niekończące się (strumień `input$` z przykładu) lub kończące się. Zakończenie strumienia oznacza wykonanie jakiegoś działania, lub w przypadku niepowodzenia, wyemitowanie błędu.

2.2.2. Opis frameworku

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. Strukturyzacja aplikacji Cycle.js za pomocą strumieni usuwa problemy dotyczące zmian z zewnątrz, jako że wszystkie zmiany danych są wewnątrz strumieni.

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 [10]

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` [11]

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`.

2.2.3. Powody sprzyjające wyborowi Cycle.js

Głównym powodem, dla którego zdecydowano się na wybór Cycle.js, jest zupełnie odmienne podejście do programowania, niż w przypadku Vue.

Wykorzystanie strumieni sprawia, że wszystkie dane wewnątrz strumienia są niezmiennie z zewnątrz, co pozytywnie wpływa na bezpieczeństwo aplikacji. Każdy strumień jest również innym ciągiem wydarzeń. W ten sposób można łatwiej uniknąć błędów, które w przypadku programowania strukturalnego są do wyłapania jedynie poprzez testy jednostkowe.

Cycle.js jest dla autora nowym frameworkiem, więc opanowanie go do stopnia wystarczającego do zaimplementowania komponentu do obsługi drużyn stanowiło wyzwanie oraz okazję do poszerzenia wiedzy.

2.2.4. Wykorzystanie w implementacji

Cycle.js został wykorzystany do zaimplementowania komponentów dotyczących obsługi drużyn. Są to: tworzenie nowej drużyny, lista wszystkich istniejących drużyn, jak i szczegółowe informacje o członkach zespołu oraz turniejach, w których wybrana grupa brała udział.

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 inicjalizację 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 handlersy, 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 uruchamiającego komponent Cycle.js do Vue.js w aplikacji CStrikers

3.3. Model pojęciowy [TODO]



Rysunek 3.2: Model pojęciowy w aplikacji CStrikers

3.4. Model C4 [TODO]

...

3.5. Testy jednostkowe komponentów

Aplikacja zawiera 3 zestawy testowe, w których zawarte są łącznie 62 testy. Decyzja na stworzenie testów do tych konkretnych plików została podjęta z kilku powodów:

1. Generator obiektów `objectGenerators.js` tworzy dane, które następnie są

wysyłane do bazy danych Firebase – aby wszystkie dokumenty w kolekcjach były zgodne ze sobą, należy mieć pewność, że obiekty są tworzone w poprawny sposób.

2. Funkcje pomocnicze z pliku `utils.js` są wykorzystywane w wielu miejscach jako podstawowa część obsługi danych, czy wyciągania z nich odpowiednich informacji. Każda zmiana w tych funkcjach mogłaby wywołać niepożądane działanie w innym miejscu w kodzie, a dzięki testom jednostkowym taka sytuacja nie wystąpi.
3. Wiele funkcji korzysta z bibliotek dostępnych w NPM, są one przetestowane i ich testowanie wiązałoby się z niepotrzebną duplikacją kodu podobnego do tego, który stworzyli autorzy wybranych paczek.

Plik	Wyrażenia		Gałęzie		Funkcje		Linie	
<code>objectGenerators.js</code>	100%	65/65	100%	8/8	100%	13/13	100%	60/60
<code>types.js</code>	100%	5/5	100%	0/0	100%	0/0	100%	5/5
<code>utils.js</code>	100%	140/140	100%	102/102	100%	25/25	100%	121/121

Tablica 3.1: Tabela przedstawiająca pokrycie kodu testami

3.6. Metryka kodu

Język	Pliki	Puste linie	Linie z komentarzami	Linie kodu
JavaScript	19	269	430	2033
Vue.js Component	17	90	9	1523
CSS	1	6	0	33
ŁĄCZNIE	37	365	439	3589

Tablica 3.2: Tabela przedstawiająca statystyki dotyczące kodu napisanej aplikacji

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.

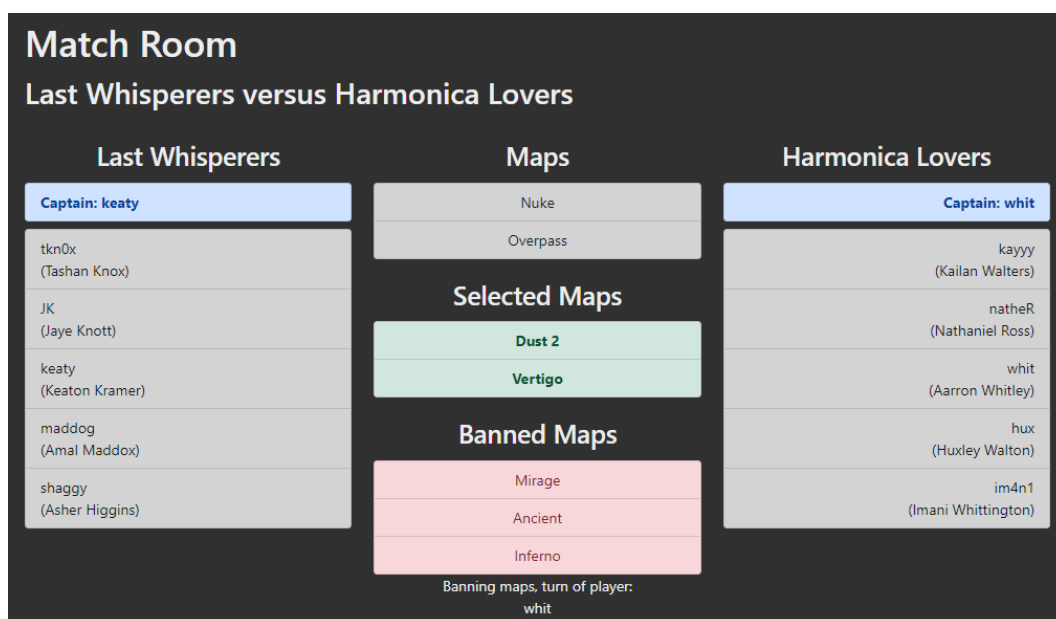
Rozdział 5.

Historyjki użytkownika

5.1. 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.



Rysunek 5.1: Widok fazy odrzucania i wybierania map przez kapitanów

- Jako gracz, chcę mieć możliwość wpisywania wyników spotkań, które rozgrywam, żeby wiedzieć w którym miejscu w tabeli/w której rundzie jest moja drużyna.

Map	1st box	2nd box	3rd box	4th box
Dust 2	16	3	15	4
Vertigo	16	11	17	10
Overpass	←	→	CT	T

Save scores

1st box: score of Last Whisperers
 2nd box: score of Harmonica Lovers
 3rd box: rounds won by both teams on CT side
 4th box: rounds won by both teams on T side

Rysunek 5.2: Widok okienka z częściowo uzupełnionymi wynikami

- Jako gracz, chcę mieć dostęp do podejrzenia statystyk wszystkich rozgrywanych meczów na różnych mapach, żeby móc przeanalizować która strona jest lepszym wyborem do gry.

Maps			
Map	Matches played	Rounds won by T	Rounds won by CT
Nuke	6	39 (29.5%)	93 (70.5%)
Mirage	4	34 (38.2%)	55 (61.8%)
Overpass	3	42 (56.0%)	33 (44.0%)
Ancient	6	57 (37.5%)	95 (62.5%)
Vertigo	2	14 (46.7%)	16 (53.3%)
Dust 2	4	45 (58.4%)	32 (41.6%)
Inferno	7	35 (32.7%)	72 (67.3%)

Rysunek 5.3: Widok tabeli z informacjami dotyczącymi liczby spotkań oraz wygranych rund po obu stronach na wszystkich mapach

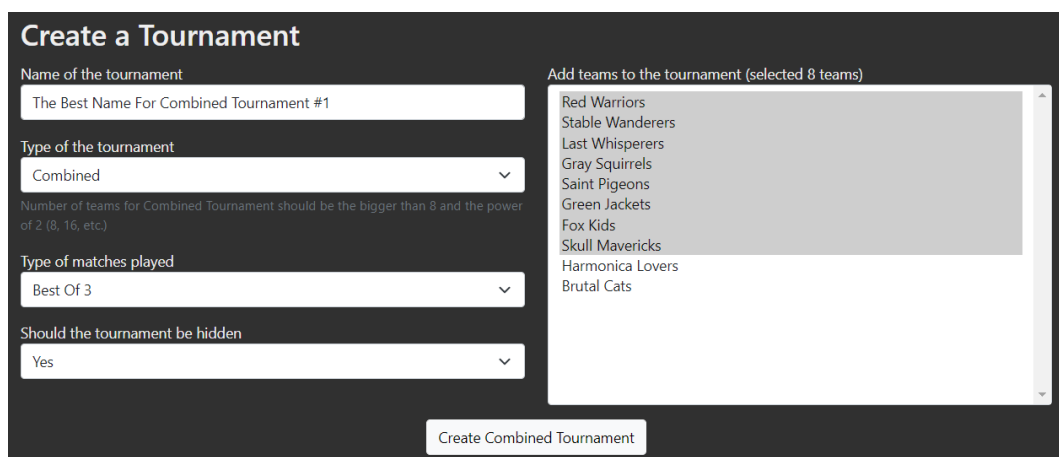
5.2. Organizator turnieju

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.



Create a Tournament

Name of the tournament
The Best Name For Combined Tournament #1

Type of the tournament
Combined

Number of teams for Combined: Tournament should be the bigger than 8 and the power of 2 (8, 16, etc.)

Type of matches played
Best Of 3

Should the tournament be hidden
Yes

Add teams to the tournament (selected 8 teams)

- Red Warriors
- Stable Wanderers
- Last Whisperers
- Gray Squirrels
- Saint Pigeons
- Green Jackets
- Fox Kids
- Skull Mavericks
- Harmonica Lovers
- Brutal Cats

Create Combined Tournament

Rysunek 5.4: Widok panelu tworzenia turnieju wraz z wybraną nazwą, systemem turnieju, typem meczu, widocznością oraz drużynami biorącymi udział

- 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.

Tournament Details

The Best League #1

Organiser

duchowe50k (Tomasz Woszczyński)
Contact email: duchowe50k@gmail.com

Type of the tournament

League (All versus All), matches played in Best Of 3 system

Score Table

#	Team	W	L	RW	RL
1	Saint Pigeons	4	0	158	94
2	Skull Mavericks	2	2	140	121
3	Fox Kids	2	2	126	126
4	Green Jackets	2	2	97	126
5	Gray Squirrels	0	4	78	132

Matches

- [Saint Pigeons vs. Skull Mavericks \(2:1\)](#)
- [Green Jackets vs. Fox Kids \(2:1\)](#)
- [Skull Mavericks vs. Gray Squirrels \(2:1\)](#)
- [Saint Pigeons vs. Green Jackets \(2:1\)](#)
- [Gray Squirrels vs. Fox Kids \(0:2\)](#)
- [Skull Mavericks vs. Green Jackets \(2:0\)](#)
- [Green Jackets vs. Gray Squirrels \(2:0\)](#)
- [Fox Kids vs. Saint Pigeons \(0:2\)](#)
- [Gray Squirrels vs. Saint Pigeons \(0:2\)](#)
- [Fox Kids vs. Skull Mavericks \(2:1\)](#)

Winner of the tournament

Fox Kids

Rysunek 5.5: Widok szczegółowych informacji o wybranym turnieju

Rozdział 6.

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 7.

Podsumowanie

7.1. Wnioski i efekty pracy

Pierwszym krokiem w implementacji powinno być zaplanowanie struktury aplikacji, przygotowanie kolekcji obejmujących wszystkie możliwe rozwiązania oraz konfiguracja środowiska. Pominiecie 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.

7.2. Future work – plan rozwoju aplikacji

Aplikacja w aktualnym stanie obsługuje jedynie podstawowe funkcjonalności wymagane do prawidłowej organizacji turniejów i lig. W poniższych sekcjach znajdują się propozycje rozszerzeń, które miałyby usprawnić działanie CStrikers i zwiększyć możliwości.

7.2.1. Możliwość zapisywania się na turnieje przez drużyny

W tym momencie implementacja CStrikers obejmuje jedynie tworzenie turniejów z aktualnej puli zarejestrowanych drużyn. To usprawnienie miałyby to zmienić – organizator mógłby stworzyć turniej bez wyboru uczestników, a następnie dowolne drużyny otrzymałyby możliwość przystąpienia do turnieju.

7.2.2. Ustalenie terminu dla turnieju oraz meczów

Aktualnie turniej po stworzeniu automatycznie dostaje status `ONGOING`, pary meczowe są generowane i uczestnicy mogą od razu przystąpić do procesu odrzucania i wybierania map. Ta zmiana miałaby umożliwić tworzenie turniejów, które rozpoczynałyby się w wybrany dzień o wybranej godzinie. Do tego czasu użytkownicy mogliby zapisywać się na turniej, a pary meczowe byłyby generowane w momencie, gdy rozgrywki się rozpoczynają.

7.2.3. Czat na żywo

Zwykle gry i aplikacje mają funkcjonalność czatu w celu ułatwienia komunikacji między obiema drużynami. Rozszerzenie to ułatwiłoby również kontakt do osoby odpowiedzialnej za organizację czy przydzielenie serwerów do meczów np. poprzez komendę `/support [wiadomość]`. Czat na żywo mógłby również dotyczyć pojedynczych użytkowników w postaci prywatnych wiadomości.

7.2.4. Powiadomienia

Opisane wcześniej propozycje opierają się o funkcjonalnościach działających w czasie rzeczywistym. Pasek z listą powiadomień ułatwiłby użytkownikom śledzenie zmian dziejących się w czatach oraz wydarzeniach, które go dotyczą.

7.2.5. Wsparcie dla dowolnej liczby drużyn

Bieżąca implementacja wspiera organizację turniejów w systemie pucharowym oraz w trybie mieszanym jedynie dla ilości drużyn będącej potęgą liczby 2. Z tego powodu nie ma możliwości organizacji turnieju z dowolną liczbą drużyn, co w przypadku niektórych turniejów może być uciążliwe. Dodanie "wirtualnych przeciwników" w postaci wolnego losu rozwiązałoby ten problem, dzięki czemu ilość drużyn nie byłaby wymaganiem przy organizacji.

7.2.6. Udział w turniejach złożonych z drużyn o mniejszej liczbie graczy

Najbardziej popularnymi turniejami są turnieje, w których drużyny składają się z pięciu aktywnych graczy oraz zmienników. Ta funkcjonalność jednak mogłaby się zmienić dzięki dodaniu wsparcia dla drużyn o mniejszej liczbie graczy. Turnieje 1 na 1 mogłyby rozgrywać się na mapach `aim_*` (małe mapy, zwykle rundy kończą się w kilkanaście sekund), turnieje 2 na 2 oraz 3 na 3 na mapach z trybu *Wingman* (jeden z trybów gry w CSGO), a 4 na 4 na standardowych mapach.

7.2.7. Implementacja nowych systemów rozgrywania turniejów

Obecnie implementacja wspiera trzy systemy: pucharowy, ligowy oraz łączony. Propozycja miałaby rozszerzyć możliwe do wyboru systemy o system szwajcarski oraz *double-elimination*, czyli system wspierający drabinę przegranych. W przypadku istniejących już systemów ligowego i łączonego można by było obsłużyć przypadek remisu w fazie grupowej (jednak nie jest to często spotykane w turniejach CS:GO).

7.2.8. Wydzielenie ról użytkowników

W kodzie aplikacji każdy użytkownik jest sobie równy, nie ma podziału na zwykłych użytkowników, moderatorów, administratorów. Zmiana ta miałaby wprowadzić role możliwe do przypisania użytkownika, dzięki czemu zarządzanie tym, co się dzieje w aplikacji, zależałoby od tego, jakie uprawnienia ma zalogowany użytkownik.

7.2.9. Obsługa serwerów gry CSGO w trakcie turniejów

Aktualnie implementacja obejmuje stworzenie meczów turniejowych oraz umożliwienie odrzucania i wybierania map kapitanom. Organizatorzy muszą jednak sami skontaktować się z drużynami, aby wysłać im adres IP serwera, na którym będzie odbywać się rozgrywka. Nowa funkcjonalność pozwoliłaby organizatorowi podać listę dostępnych serwerów, nowy algorytm korzystający z paczki `rcon-srcds` [15]

mógłby dobierać dostępne serwery i przypisywać je do meczów, a następnie za pomocą *RCON* (*remote console*) konfigurować serwer.

7.2.10. Optymalizacja zapytań do bazy danych

Obecnie baza danych w dużej mierze działa na zasadzie jednokrotnego pobierania i zapisywania danych, przez co komponenty dotyczące np. tworzenia drużyn i turniejów, jak i wyświetlania ich, są aktualne w momencie wejścia na wybraną podstronę. Firebase udostępnia jednak funkcję `onSnapshot` [16], która otwiera socket nasłuchujący na zmiany w wybranej kolekcji, a następnie w czasie rzeczywistym aktualizuje zawartość pobranych danych.

Rozdział 8.

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 9.

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] Toornament - About - https://www.toornament.com/en_US/about
- [2] Toornament - Pricing - https://www.toornament.com/en_US/pricing
- [3] ESEA - Client - <https://play.esea.net/client>
- [4] ESEA - Subscription plans - <https://play.esea.net/subscribe>
- [5] Vue.js - Introduction, Single-File Components - <https://vuejs.org/guide>
- [6] Vue.js - Lifecycle Diagram - <https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram>
- [7] Vue Router - Getting Started - <https://router.vuejs.org/guide/>
- [8] Vuex - Getting Started - <https://vuex.vuejs.org/guide/>
- [9] Cycle.js - Streams - <https://cycle.js.org/streams.html>
- [10] Cycle.js - Dataflow - <https://cycle.js.org/#-dataflow>
- [11] Cycle.js - Model-View-Intent - <https://cycle.js.org/model-view-intent.html>
- [12] Cycle.js - Getting started - <https://cycle.js.org/getting-started.html>
- [13] Cycle.js - API reference - <https://cycle.js.org/api/index.html#api-reference>
- [14] Firestore Database - Documentation - <https://firebase.google.com/docs/firestore>
- [15] RCON library for NodeJS - <https://www.npmjs.com/package/rcon-srcds>
- [16] Get realtime updates with Cloud Firestore - <https://firebase.google.com/docs/firestore/query-data/listen>