



Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Aplikacja do gry w Gomoku

Dawid Kalinowski

Projekt z przedmiotu technologie chmurowe
na kierunku informatyka profil praktyczny
na Uniwersytecie Gdańskim.

Gdańsk
26 czerwca 2024

Spis treści

1	Opis projektu	2
1.1	Opis architektury - 6 pkt	2
1.2	Opis infrastruktury - 6 pkt	2
1.3	Opis komponentów aplikacji - 6 pkt	2
1.4	Konfiguracja i zarządzanie - 4 pkt	3
1.5	Zarządzanie błędami - 2 pkt	3
1.6	Skalowalność - 4 pkt	3
1.7	Wymagania dotyczące zasobów - 2 pkt	3
1.8	Architektura sieciowa - 4 pkt	3

1 Opis projektu

Firma zajmująca się grami planszowymi, Board Games Interactive, zleciła stworzenie aplikacji webowej, która będzie docelowo obsługiwała rozgrywki w przeróżnych grach planszowych, między graczami z całego świata. Na sam początek przedsiębiorstwo postanowiło zapewnić graczom możliwość gry w gre "Gomoku". Serwis ma działać w przeglądarce, zarówno na komputerach, jak i na urządzeniach mobilnych. System musi zapewnić płynną rozgrywkę dla tysięcy graczy jednocześnie.

1.1 Opis architektury - 6 pkt

Architektura aplikacji została zaimplementowana w środowisku Kubernetes, co w przyszłości umożliwi łatwiejsze skalowanie aplikacji oraz wdrażanie nowych zmian. Składa się z trzech oddzielnych komponentów: Backendu w Node.js, Frontendu w React.js oraz bazy danych MongoDB. Każdy z nich jest osobnym kontenerem w klastrze Kubernetes.

1.2 Opis infrastruktury - 6 pkt

Loadbalancer dla frontu i backu, ClusterIP dla mongo

1.3 Opis komponentów aplikacji - 6 pkt

Aplikacja składa się z trzech głównych komponentów:

- **Backend (Node.js):** Backend aplikacji został wyprodukowany w Node.js przy użyciu frameworka Express.js. Odpowiada za obsługę użytkowników oraz gier, między innymi poprzez pobieranie oraz wysyłanie informacji z / do bazy danych w MongoDB. Backend udostępnia RESTful API, które obsługuje operacje takie jak rejestracja, logowanie użytkowników, tworzenie, dołączanie do gier, w przypadku Gomoku również stawianie pionków na planszy. Panel administratora udostępnia również takie funkcjonalności, jak edycja poszczególnych instancji gier, manipulowanie uprawnieniami oraz danymi zwykłych użytkowników. Do zarządzania stanem sesji użytkowników zastosowano tokeny JWT, a logowanie użytkowników jest zabezpieczone funkcją haszującą, a dokładniej bcrypt. Aplikacja umożliwia również czatowanie z innymi użytkownikami na wspólnym forum, za pomocą protokołu MQTT.
- **Frontend (React.js):** Frontend aplikacji został zaimplementowany w bibliotece React.js. Użytkownik może, korzystając z intuicyjnych komponentów, takich jak pola tekstowe, przyciski, poruszać się po aplikacji, w szczególności dołączać do gier oraz grać. Aplikacja posiada przejrzystą planszę do gry w Gomoku, która po zmianie stanu planszy na bieżąco się odświeża. Frontend komunikuje się z Backendem za pomocą zapytań HTTP, przysyłając i odbierając dane w formacie JSON. Aplikacja jest również przystosowana pod urządzenia mobilne, uzależniając rozmiar poszczególnych komponentów od maksymalnej rozdzielczości okna.
- **Baza danych MongoDB:** Do przechowywania informacji o wszystkich użytkownikach, oraz toczonych grach w aplikacji, wykorzystano MongoDB, bazy danych NoSQL.

Baza przechowuje dane użytkowników, informacje zarówno o wszystkich aktualnie toczonych meczach, jak i historycznych gier.

Każdy komponent jest skonfigurowany w osobnym pliku YAML dla Kubernetes i jest wdrażany jako oddzielny pod.

1.4 Konfiguracja i zarządzanie - 4 pkt

Konfiguracja aplikacji odbywa się poprzez pliki YAML dla Kubernetes, które definiują deploymenty i serwisy. Deployment zarówno backendu, jak i frontendu, definiuje obraz Docker, m.in. liczbę replik - w tym przypadku 3 - oraz zmienne środowiskowe, w tym URI do bazy danych MongoDB. Usługa zapewnia zewnętrzny dostęp do backendu i frontendu poprzez LoadBalancer. Baza danych również jest w 3 replikach, a komunikacja odbywa się za pomocą ClusterIP.

1.5 Zarządzanie błędami - 2 pkt

1.6 Skalowalność - 4 pkt

Za skalowalność aplikacji jest odpowiedzialny Horizontal Pod Autoscaler (HPA). Jest skonfigurowany do monitorowania backendu, i zmienia liczbę jego replik w zakresie liczb 3-10, w zależności od obciążenia procesora. Aktualnie jego średnie zużycie jest ustawione na poziomie 60

1.7 Wymagania dotyczące zasobów - 2 pkt

Każdy komponent ma określone wymagania dotyczące zasobów:

- **Backend:** 1 CPU, 512MB RAM
- **Frontend:** 0.5 CPU, 256MB RAM
- **Baza danych:** 2 CPU, 1GB RAM

Oczekiwania dotyczące wydajności obejmują czas odpowiedzi poniżej 200ms dla większości zapytań.

1.8 Architektura sieciowa - 4 pkt

Każdy odnośnik, który znajdzie się w literaturze musi mieć swoje odwołanie w projekcie. - 2 pkt

Literatura

- [1] *Dokumentacja express js* <https://expressjs.com/>.
- [2] *Dokumentacja js-cookie* <https://www.npmjs.com/package/js-cookie>.
- [3] *Dokumentacja kubernetes* <https://kubernetes.io/docs/home/>.
- [4] *Dokumentacja mongodb* <https://www.mongodb.com/docs/>.
- [5] *Dokumentacja mongoose* <https://mongoosejs.com/docs/>.
- [6] *Dokumentacja react* <https://legacy.reactjs.org/docs/getting-started.html>.