




**Uniwersytet Gdański**
**Wydział Matematyki, Fizyki i Informatyki**
**Instytut Informatyki**

# Plan zajeć dla firmy

**Weronika Nieżorawska**

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

**Gdańsk**

**28 czerwca 2024**

# Spis treści

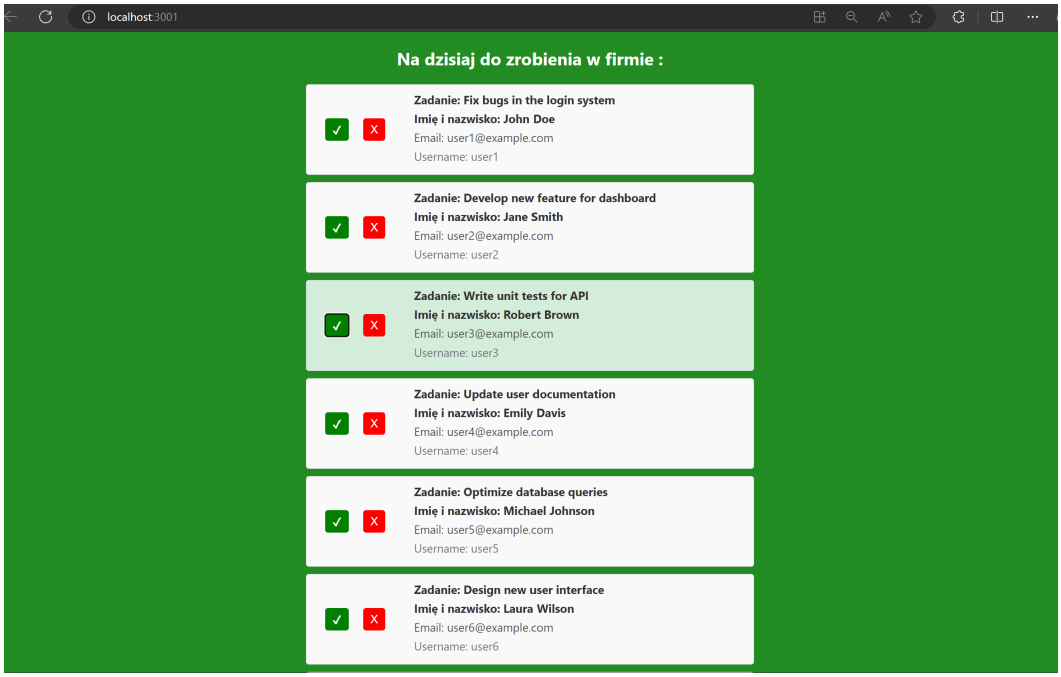
# 1 Opis projektu

Firma potrzebuje aplikacji, która bedzie pobierać zadania oraz dane pracowników z ich bazy danych i wyświetlać te informacje na stronie internetowej. Aplikacja ma umożliwiać użytkownikom zarzadzanie zadaniami pracowników poprzez nastepujace funkcje:

- Usuwanie zadań: Jeśli pracownik nie jest w stanie wykonać danego zadania, użytkownik może je usunać z listy.

- Oznaczanie wykonanych zadań: Jeśli pracownik wykonał zadanie, użytkownik może je oznaczyć jako wykonane, co wyróżni to zadanie w aplikacji i zapamieta jego stan.

Aplikacja ma na celu usprawnienie zarzadzania zadaniami pracowników oraz zapewnienie lepszego wgladu w realizacje zadań w firmie.




## 1.1 Opis architektury - 8 pkt

Projekt został stworzony z użyciem Minikube, który służy do tworzenia klastrów Kubernetes. W ramach tego klastra działaja trzy mikroserwisy, z których każdy jest oparty na Dockerze:

Backend: Mikroserwis backendowy napisany w JavaScript, korzystajacy z frameworka Express do komunikacji z baza danych. Backend jest odpowiedzialny za obsługe logiki biznesowej oraz interakcje z baza danych.

Baza danych: Mikroserwis z baza danych MongoDB, który działa w środowisku Dockerowym. MongoDB przechowuje wszystkie dane dotyczace zadań i pracowników.

Frontend: Mikroserwis frontendowy to aplikacja napisana w React. Łaczy sie ona z backendem, aby pobierać dane z bazy danych i odpowiednio je wyświetlać. Aplikacja frontendowa umożliwia również wykonywanie operacji na danych, takich jak oznaczanie zadań jako wykonane lub usuwanie zadań.

Dzieki zastosowaniu Kubernetes oraz Minikube, projekt jest skalowalny i łatwy do zarzadzania, co pozwala na efektywne wykorzystanie zasobów oraz łatwa integracje i utrzymanie mikroserwisów.

## 1.2 Opis infrastruktury - 6 pkt

Aplikacja bedzie działać w środowisku Kubernetes, zarzadzanym za pomoca miniKube, który tworzy lokalny klaster Kubernetes. Wewnatrz tego klastra uruchomione sa trzy mikroserwisy, każdy oparty na Dockerze.

Pierwszy mikroserwis to backend napisany w JavaScript, wykorzystujacy Express do komunikacji z baza danych.

Drugi mikroserwis to baza danych MongoDB działajaca w środowisku Kubernetes.

Trzeci mikroserwis to frontend, bedacy aplikacja React, która łaczy sie z backendem w celu uzyskania danych z bazy i ich odpowiedniego wyświetlenia oraz obsługi interakcji użytkownika.

W opisie infrastruktury ważne jest również uwzglednienie efektywnego zarzadzania zasobami, takimi jak sieci i pamieć masowa, aby zapewnić optymalna wydajność i skalowalność aplikacji.

## 1.3 Opis komponentów aplikacji - 8 pkt

Aplikacja składa sie z trzech głównych komponentów: serwisów backend, bazy danych i aplikacji frontend.

### 1.3.1 Serwis Backend

Serwis backendowy został napisany w jezyku JavaScript z użyciem frameworka Express. Jego zadaniem jest komunikacja z baza danych oraz dostarczanie danych do aplikacji frontendowej. Backend jest odpowiedzialny za pobieranie zadań oraz danych pracowników z bazy danych, a także za obsługe logiki biznesowej, takiej jak usuwanie lub zatwierdzanie zadań. Serwis backendowy jest wdrażany jako kontener Docker i zarzadzany przez Kubernetes, co zapewnia jego skalowalność i niezawodność.

### 1.3.2 Baza Danych

Komponent bazy danych wykorzystuje MongoDB jako system zarzadzania baza danych. MongoDB przechowuje wszystkie zadania oraz informacje o pracownikach. Baza danych jest wdrażana jako osobny mikroserwis w Kubernetes, co umożliwia jej łatwe skalowanie oraz zarzadzanie zasobami. MongoDB jest również uruchamiane w kontenerze Docker, co ułatwia zarzadzanie jego wersjami oraz konfiguracja.

### 1.3.3 Aplikacja Frontend

Aplikacja frontendowa została stworzona z użyciem React. Jej głównym celem jest prezentacja danych pobranych z serwisu backendowego oraz interakcja z użytkownikiem.

Użytkownicy moga przegladać zadania, usuwać je lub oznaczać jako wykonane. Aplikacja frontendowa łaczy sie z backendem za pomoca API, a backend przekazuje jej dane z bazy danych. Podobnie jak inne komponenty, frontend jest wdrażany jako kontener Docker i zarzadzany przez Kubernetes, co zapewnia jego wysoka dostepność i łatwość w utrzymaniu.

### 1.3.4 Wdrażanie i Konfiguracja

Wszystkie trzy komponenty sa wdrażane za pomoca manifestów Kubernetes, które definiuja sposób ich uruchomienia, konfiguracje oraz zasoby potrzebne do działania. Każdy mikroserwis jest uruchamiany w osobnym kontenerze Docker, a Kubernetes zarzadza ich skalowaniem oraz dystrybucja zasobów. Konfiguracja poszczególnych komponentów jest przechowywana w plikach konfiguracyjnych Kubernetes, co umożliwia łatwe zarzadzanie i modyfikacje ustawień w razie potrzeby.

### 1.3.5 Zarzadzanie

Kubernetes zapewnia zaawansowane mechanizmy zarzadzania aplikacja, takie jak automatyczne skalowanie, równoważenie obciażenia, odtwarzanie po awarii oraz zarzadzanie konfiguracja. Dzieki temu każdy z komponentów aplikacji może być łatwo monitorowany, skalowany oraz aktualizowany bez przestojów w działaniu. Dodatkowo, wykorzystanie Docker do konteneryzacji poszczególnych komponentów pozwala na zachowanie spójności środowisk oraz łatwe wdrażanie aplikacji na różnych platformach.

## 1.4 Konfiguracja i zarzadzanie - 4 pkt

W tej sekcji przedstawione zostana szczegółowe informacje dotyczace konfiguracji i zarzadzania aplikacja na poziomie klastra Kubernetes.

### 1.4.1 Konfiguracja klastra Kubernetes

Klaster Kubernetes jest zainstalowany i zarzadzany za pomoca narzedzia Minikube, które umożliwia tworzenie i zarzadzanie lokalnym klastrem Kubernetes. Wszystkie mikroserwisy sa wdrażane jako Deploymenty, co zapewnia automatyczne skalowanie, aktualizacje i wysoka dostepność aplikacji. Każdy mikroserwis ma swój własny Deployment, który określa liczbe replik, zasoby potrzebne do działania oraz kontener Docker, który ma zostać uruchomiony.

### 1.4.2 Manifesty Kubernetes

Każdy komponent aplikacji (backend, frontend, baza danych) jest zdefiniowany za pomoca manifestów Kubernetes w formacie YAML. Poniżej znajduje sie przykładowy manifest dla serwisu backendowego:

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
name: backend-deployment
spec:
replicas: 3
selector:
matchLabels:
app: backend
template:
metadata:
labels:
app: backend
spec:
containers:
- name: backend
image: my-backend-app
ports:
- containerPort: 4000
```

Podobne manifesty istnieja dla serwisu frontendowego oraz bazy danych MongoDB.

### 1.4.3 Zarzadzanie konfiguracja

Do zarzadzania konfiguracja aplikacji wykorzystywane sa ConfigMap i Secret. ConfigMap przechowuje dane konfiguracyjne, które sa współdzielone pomiedzy kontenerami, takie jak adresy URL, porty, itp. Secret jest używany do przechowywania wrażliwych informacji, takich jak dane uwierzytelniajace bazy danych. Poniżej znajduje sie przykładowa ConfigMap dla adresów URL:

```
apiVersion: v1
kind: ConfigMap
metadata:
name: app-config
data:
DATABASE_URL: mongodb://mongo:27017/mydatabase
BACKEND_URL: http://backend:4000
```

### 1.4.4 Monitorowanie i logowanie

Monitorowanie aplikacji jest realizowane za pomoca Prometheus oraz Grafana, które umożliwiaja zbieranie metryk i tworzenie wizualizacji. Logi aplikacji sa gromadzone przez Elasticsearch, Logstash i Kibana (ELK stack), co umożliwia centralne zarzadzanie logami i szybkie wykrywanie problemów.

### 1.4.5 Automatyczne skalowanie i aktualizacje

Kubernetes zapewnia mechanizmy automatycznego skalowania, które pozwalaja na dynamiczne dostosowywanie liczby replik mikroserwisów w zależności od obciażenia. Ak-

tualizacje aplikacji sa realizowane za pomoca strategii rolling update, która zapewnia stopniowe wdrażanie nowych wersji bez przestojów w działaniu aplikacji.

### 1.4.6 Bezpieczeństwo

Bezpieczeństwo aplikacji jest zapewniane poprzez ograniczenie dostepu do zasobów za pomoca RBAC (Role-Based Access Control), izolacje kontenerów oraz użycie Secret do przechowywania danych uwierzytelniajacych. Dodatkowo, wszystkie komunikacje pomiedzy mikroserwisami moga być zabezpieczone za pomoca TLS/SSL.

Konfiguracja i zarzadzanie aplikacja na poziomie klastra Kubernetes zapewnia wysoka dostepność, skalowalność oraz bezpieczeństwo, co pozwala na efektywne wdrażanie i utrzymanie aplikacji.

## 1.5 Zarzadzanie błedami - 2 pkt

Aplikacja obsługuje błedy różnych zapytań i wyświetla w konsoli użytkownika wszelkie nieudane lub nieprawidłowe operacje.

## 1.6 Skalowalność - 4 pkt

Skalowalność w moim projekcie jest zarzadzana za pomoca Kubernetes i jego Horizontal Pod Autoscaler (HPA). HPA automatycznie dostosowuje liczbe replik mikroserwisów backendowych i frontendowych w odpowiedzi na zmieniajace sie obciażenie CPU, co zapewnia wydajność i elastyczność aplikacji. Dzieki temu, gdy zapotrzebowanie na zasoby wzrasta, Kubernetes zwieksza liczbe instancji kontenerów, a gdy zapotrzebowanie spada, redukuje ich liczbe. Taka konfiguracja zapewnia, że aplikacja może obsłużyć wieksza liczbe użytkowników bez degradacji wydajności, jednocześnie optymalizujac wykorzystanie zasobów. W efekcie, moja aplikacja jest w stanie dynamicznie reagować na zmienne warunki pracy, zapewniajac ciagłość i niezawodność działania.

## 1.7 Wymagania dotyczace zasobów - 2 pkt

Wymagania dotyczace zasobów:

- Backend: Minimalne: 0.5 CPU, 512 MB RAM, Maksymalne: 1 CPU, 1 GB RAM

- Frontend: Minimalne: 0.5 CPU, 512 MB RAM, Maksymalne: 1 CPU, 1 GB RAM

- Baza danych: Minimalne: 0.5 CPU, 512 MB RAM, Maksymalne: 1 CPU, 1 GB RAM

Wszystkie powinny wykonywać sie w akceptowalnym dla użytkownika czasie, poniżej 100 ms.

## 1.8 Architektura sieciowa - 4 pkt

Wszystkie serwisy w klastrze komunikuja sie ze soba w sieci lokalnej localhost. Serwisy sa skonfigurowane na działanie w nastepujacych portach:

- Backend nasłuchuje na porcie 3050
- Frontend na porcie 3000
- Baza danych na porcie 27017

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

# Literatura

[1] Docker Documentation, *https://docs.docker.com/*, 2013.

[2] Kubernetes Documentation, *https://kubernetes.io/docs/home/*, 2024.

[3] MongoDB, *https://www.mongodb.com/*, 2024.

[4] Create react app, *https://create-react-app.dev/docs/getting-started/*, 2022.

[5] React-JS, *https://pl.legacy.reactjs.org/*, 2024.

[6] WebApi, *https://developer.mozilla.org/en-us/docs/web/api*, 1998.