**Dokumentacja Techniczna**

**aplikacji**

Spis treści

[1. Cel aplikacji 2](#_Toc110445626)

[2. Zastosowane technologie 2](#_Toc110445627)

[3. Działanie aplikacji 3](#_Toc110445628)

# Cel aplikacji

Celem projektu było stworzenie aplikacji webowej, służącej do zarządzania pracownikami wewnątrz firm poprzez tworzenie ich spisów w bazie danych. Aplikacja umożliwia zapisywanie pracowników wewnątrz istniejących już w bazie danych firm oraz utworzenie nowej firmy wraz z pracownikami.

# Zastosowane technologie

Aplikacja została napisana z wykorzystaniem narzędzia Spring Boot. Narzędzie to pozwoliło wprowadzić domyślne konfigurację framework’u Spring oraz stworzyć odpowiednią dla aplikacji webowej strukturę projektu, która prezentuje się w następujący sposób:

* Warstwa modelu danych,
* Warstwa repozytorium,
* Warstwa serwisowa,
* Warstwa wymiany danych.

Warstwa modelu danych składa się z klas odpowiedzialnych za reprezentacje obiektów, na których aplikacja będzie operować, w przypadku omawianej aplikacji są to **Company** oraz **Employee**. Warstwa repozytorium umożliwia zapis tworzonych encji wewnątrz bazy danych. Składa się ona z interfejsu **JpaRepository**, rozszerzającego inne interfejsy, który dostarcza metody służące do wykonywania operacji na bazie danych. W aplikacji została zastosowana pamięciowa baza danych którą zapewnia zaimplementowana **H2 Database**. Pamięciowa baza danych zapisuje dane lokalnie w pamięci RAM komputera. Jednakże jest to jedynie zastosowanie chwilowe, ponieważ w przypadku wykorzystaniu aplikacji przez firmę konieczne będzie przejście na nie pamięciową bazę danych. Ze względu na wprowadzenie do projektu framework’u **Hibernate**, przejście na zewnętrzną bazę danych jest możliwe poprzez zmianę wartości URL w pliku **application.yaml** oraz zaimplementowanie odpowiednich bibliotek odpowiadających za obsługę wybranej przez nas bazy danych. Warstwa serwisowa zawiera główną logikę aplikacji. To właśnie w niej znajdują się implementacje metod z warstwy repozytorium, które umożliwiają wykonywanie wszelkich operacji. Dostęp do metod z repozytorium zapewnia wstrzyknięcie **beana** (specyficzny dla framework’u **Spring** rodzaj klasy, wewnątrz globalnego kontekstu), jakim jest odpowiednie repozytorium do serwisu. Ostatnia warstwa, czyli warstwa wymiany danych, jak sama nazwa wskazuje, służy do wymiany danych za pomocą zapytań **HTTP**. Warstwa ta podczas wykonywania zapytań wykorzystuje metody zaimplementowane wcześniej w warstwie serwisowej aplikacji a dostęp do nich zapewnia wstrzyknięty **bean**, w postaci odpowiedniego serwisu.

# Działanie aplikacji

Działanie aplikacji zostało przedstawione na podstawie funkcjonalności umożliwiającej zapisanie w bazie danych nowej firmy wraz z pracownikiem, którego wcześniej w niej nie było, oraz z pracownikiem, który jest już zapisany. Pierwszym krokiem będzie wysłanie zapytania **HTTP** typu **POST** pod odpowiedni adres URL, tutaj będzie to: http://localhost:8080/companies. W ciele zapytania w formacie **JSON** wpisujemy interesujący nas obiekt, w tym przypadku firmę wraz z listą pracowników których jeszcze nie ma wewnątrz bazy danych. Ciało zapytania będzie zawierało następujący plik **JSON**:

{

    "name": "Company",

    "employees": [

        {

        "firstName": "Jan",

        "secondName": "Kowalski",

        "salary": 1.0

        },

        {

        "firstName": "Piotr",

        "secondName": "Ostry",

        "salary": 2.0

        }

    ]

}

Następnie zapytanie **HTTP** zostaje przechwycone przez **DispatcherServlet**, który odpowiedzialny jest za kierowanie pracy programu do odpowiednich komponentów programu. Wykonuje on wszelkie potrzebne procesy potrzebne do przetworzenia zapytania oraz zwrócenia odpowiedniego widoku. Aby odnaleźć odpowiedni stworzony przez nas kontroler, **DispatcherServlet** wykorzystuje **HandlerMapping**, który wyszukuje odpowiedni kontroler na podstawie przypisanego do niego adresu URL. Odnaleziony kontroler wygląda w następujący sposób:

*@PostMapping*

*public ResponseEntity<Company> save(@RequestBody Company company) {*

*return ResponseEntity.ok(companyService.save(company));*

*}*

Po tym, jak HandlerMapping odnajdzie odpowiedni kontroler, wykorzystywany jest HandlerAdapter, który wstawi odpowiednie argumenty metody oznaczone adnotacjami, w tym przypadku jest to adnotacja **@RequestBody**. Jeżeli podczas wysyłania danych do kontrolera nie wystąpi żaden błąd, to format **JSON** zostanie **zdeserializowany** (przetworzenie postaci tekstowej na obiekt w **Javie**), po czym nastąpi przesłanie obiektu do kolejnej warstwy (tj. serwisowej). Następnym krokiem będzie przesłanie danych do warstwy bazodanowej. Po tym następuje zapis obiektu do bazy danych, co skutkuje dopisaniem id do wysłanych przez nas obiektów. Informacja zwrotna po wykonanym zapytaniu wygląda następująco:

{

    "id": 1,

    "name": "Company",

    "employees": [

        {

            "id": 1,

            "firstName": "Jan",

            "secondName": "Kowalski",

            "salary": 1.0

        },

        {

            "id": 2,

            "firstName": "Piotr",

            "secondName": "Ostry",

            "salary": 2.0

        }

    ]

}

Jak można zauważyć, zarówno do dodanej firmy jak i pracowników zostały dodane kolejne wartości **ID**. Dzięki zastosowaniu adnotacji **@Cascade** podczas tworzenia połączenia między encją **Company** oraz **Employee**, podani wewnątrz firmy pracownicy zostali automatycznie dodani do bazy danych. W przypadku podania w zapytaniu **ID** pracownika, który już istnieje to zostanie on nadpisany, z kolei podanie nie istniejącej wartości **ID** zapisze podanego pracownika z **ID** następującym po ostatnim zapisanym pracowniku.