PAI projekt indywidualny 2 dokumentacja

1. Identyfikacja zagadnienia biznesowego

Zaprojektowanie i zaimplementowanie aplikacji internetowej, która umożliwia ogłaszanie przetargów przez dowolną instytucję oraz wzięcie w nich udziału, również przez dowolną instytucję.

Aplikacja rozwiązuje i wyczerpuje podstawowe wymagania zagadnienia biznesowego. Może być ona jednak dalej rozwijana o kolejne funkcjonalności, takie jak np. konta lub panel administracyjny.

2. Wymagania systemowe i funkcjonalne

Do działania aplikacja wymaga systemu z dostępem do sieci Internet oraz środowiskiem uruchomieniowym Node.js w wersji co najmniej 16.

Zaproponowane rozwiązanie składa się z relacyjnej bazy danych MySQL, serwera back-end napisanego w technologii Node.js + Express.js oraz serwera front-end w bibliotece React.js.

Aplikacja została stworzona w architekturze klient-serwer za pomocą wzorca Model-View-Controller. Model oraz kontroler zaimplementowane są w Node.js + Express.js, a widok w React.js.

Komunikacja między klientem a serwerem jest asynchroniczna. Wysyłane komunikaty są w standardzie REST API.

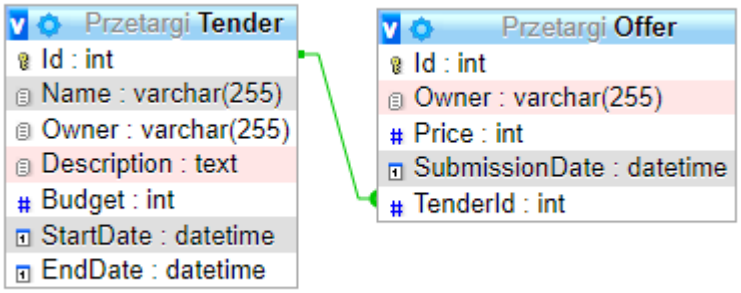
3. Analiza zagadnienia i jego modelowanie

Baza danych składa się z dwóch tabel:

- Tender – zawiera informacje o przetargach

- Offer – zawiera informację o ofertach

Tender jest w relacji jeden-do-wielu z Offer.



Kontrolery:

- TenderController – zawiera funkcje zwracające dane o przetargach oraz pozwalające na dodanie nowego przetargu

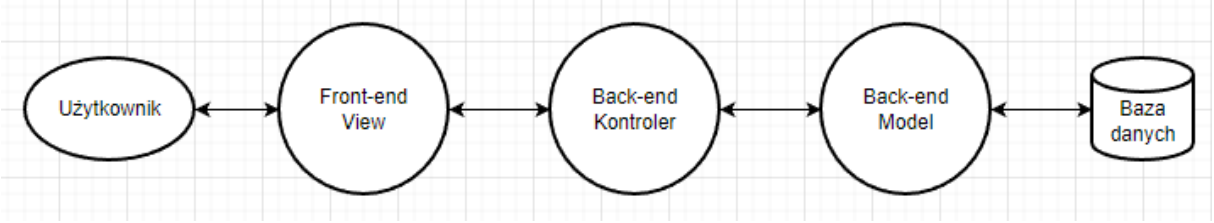
- OfferController – zawiera funkcje zwracające oferty dla danego przetargu oraz pozwalające na dodawanie nowej oferty dla danego przetargu

Modele:

- TenderModel – pośrednik między kontrolerem a bazą danych, zwraca dane z tabeli Tender oraz dodaje nowe

- OfferModel - pośrednik między kontrolerem a bazą danych, zwraca dane z tabeli Offer oraz dodaje nowe

Ogólny schemat przepływu danych



4. Implementacja

Front-end

W implementacji wykorzystano bibliotekę React.js. Routing obsługiwany jest za pomocą *react-routing-dom*. Każda strona jest osobnym komponentem, który zawiera funkcję zwracającą kod HTML. Dane pobierane są za pomocą funkcji fetch().

Przykładowe pobieranie danych, które otrzymuje informację o wszystkich zakończonych przetargach:

```
export const fetchPastTenders = async() => {
    return fetch( input: 'http://localhost:3051/api/v1/tender/past')
        .then(response => response.json());
};
```

Do przechowywania danych w komponentach, przekierowań oraz zgłaszania potrzeby pobrania danych wykorzystano reactowe hooki.

Przykładowy komponent wyświetlający wszystkie aktywne przetargi w liście:

```
export const TenderActiveDetails = () => {
    const [ isLoaded, setIsLoaded ] = useState( initialState: false);
    const [ tender, setTender ] = useState( initialState: []);
    const params = useParams();
    const navigate = useNavigate();

    useEffect( effect: () => {
        fetchTender(params.id)
            .then((t) => {
                if(t.length === 0) {
                    setIsLoaded( value: true);
                    return;
                }
                else if(new Date(t[0].EndDate) < new Date()) {
                    navigate('/przetargi/zakonczone/' + t[0].Id);
                }
                setTender(t[0]);
                setIsLoaded( value: true);
            })
    }, deps: [navigate, params.id])

    if(!isLoaded) {
        return <>Ładowanie...</>;
    } else if(tender.length === 0) {
        return <div className='wrong-id-error'>Przetarg o danym id nie istnieje!</div>
    } else {
        return (
            <div className='active-tender-details-page'>
                <h1 className='tender-name'>{tender.Name}</h1>
                <h3 className='owner-title'>Zgłaszający przetarg</h3>
                <p className='owner'>{tender.Owner}</p>
                <h3 className='description-title'>Opis</h3>
                <p className='description'>{tender.Description}</p>
                <NavLink to={`/oferty/${tender.Id}`} className='add-offer'>Dodaj ofertę</NavLink>
            </div>
        );
    }
};
```

Back-end

Do implementacji użyto środowiska Node.js i Express.js. Routing obsługiwany jest za pomocą paczki *router*. Skonfigurowano również *cors*, w taki sposób, że zapytania przyjmowane są tylko z http://localhost:3000.

Istnieją dwa modele: OfferModel i TenderModel odpowiedzialne za odczyt i zapis danych z tabel dla ofert i przetargów. Baza danych to MySQL z dwiema tabelami Offer i Tender. Do tworzenia zapytań nie wykorzystano ORM, zapytania są zapisane w języku SQL.

Przykładowa asynchroniczna funkcja w modelu, która zwraca informacje o jednym przetargu (lub błąd):

```
const getOneTender = async (id) => {
    const sql = 'SELECT * FROM Tender WHERE Id = "' + id + '";';
    let r;
    try {
        const [rows] = await pool.query(sql);
        r = rows;
    } catch (err) {
        console.error(err);
        r = err;
    }
    return r;
};
```

Istnieją również dwa kontrolery, które przyjmują zapytania, wywołują funkcje z modeli oraz zwracają wynik w postaci odpowiedzi HTTP z odpowiednim statusem i danymi zakodowanymi w JSON.

Przykładowa funkcja w kontrolerze, zwracająca informacje o przetargu o podanym id:

```
const getOneTender = async (req, res) => {
    const tender = await TenderModel.getOneTender(req.params.tenderId);
    res.status(200).json(tender);
};
```

Front-end uruchamiany jest na porcie 3000, back-end na porcie 3051.

5. Podsumowanie

Aplikacja spełnia podstawowe założenia zagadnienia biznesowego. W trakcie pracy nie musiałem zmagać się z żadnych dużym problemem.

System może być dalej rozwijany, między innymi o:

- system logowania i autoryzacji

- bardziej rozbudowany front-end

- deployment na chmurę.

Autor:

Łukasz Wójcik