Dokumentacja

1. Identyfikacja zagadnienia biznesowego

Celem projektu jest umożliwienie ogłaszania przetargów przez dowolną instytucję oraz wzięcie w nich udziału, również przez dowolną instytucję/osobę. Użytkownik ma możliwość przeglądu dostępnych aukcji w jednym miejscu i zgłoszenie swojej oferty w zainteresowanym przez niego przetargu. Obecnie na rynku istnieje bardzo mało aplikacji internetowych umożliwiających płynne poruszanie się po dostępnych na rynku w danym momencie przetargów. Proponowana aplikacja rozwiązuje główne zagadnienia biznesowe, jednak nie wyczerpuje w pełni możliwości, które aplikacja tego rodzaju może posiadać. Jest tylko jej elementem, dzięki któremu możemy dalej rozwijać i udoskonalać stworzony serwis.

2. Wymagania systemowe i funkcjonalne

Nasz system skierowany jest do ludzi, w każdym wieku. Zawiera on intuicyjny i nowoczesny interfejs, z którego z łatwością zdołają korzystać zarówno osoby młode jak i starsze. Nie klasyfikuje ludzi ze względu na płeć, wykształcenie czy narodowość. Do posługiwania się aplikacją wystarczy podstawowa znajomość obsługi komputera. Użytkownik aplikacji to dowolna osoba wyrażająca chęć korzystania z naszego systemu, nie jest wymagane posiadanie zarejestrowanego konta na naszej platformie. Każdy jest traktowany jak gość.

Głównymi funkcjonalnościami aplikacji są:

● Możliwość tworzenia przetargów.

● Możliwość tworzenia ofert do przetargów.

● Możliwość przeglądania wszystkich dostępnych w tej chwili przetargów, do których można złożyć ofertę.

● Możliwość przeglądania wszystkich zamkniętych już przetargów.

● Możliwość przeglądania wszystkich ofert w zamkniętym przetargu, które nie przekraczają maksymalnej wartości przetargu i są posortowane od najkorzystniejszej do najmniej korzystnej oferty w danym przetargu.

Zagrożenia wynikające z użytkownika aplikacji:

● Wprowadzenie błędnych danych przy tworzeniu przetargu lub oferty.

● Ataki botów.

● Niecenzuralne nazwy przetargów lub ofert.

● Zamknięcie karty w przeglądarce, kiedy użytkownik jest w trakcie tworzenia przetargu lub oferty.

● Utrata dostępu do Internetu w trakcie korzystania z aplikacji.

● Awaria systemu

Aplikacja opiera się na ogólnym modelu architektury klient – serwer w której, klient wysyła żądania do serwera, a ten zwraca mu w odpowiedzi żądane dane. Aplikacja została stworzona w oparciu o wzorzec projektowy Model-Widok-Kontroler (z ang. Model - View - Controller). Koncepcja tego modelu opiera się na podziale aplikacji na trzy niepowiązane ze sobą warstwy.

● Model - reprezentacja logiki aplikacji, która definiuje elementy, ich atrybuty oraz relacje pomiędzy nimi. Może opisywać strukturę bazy danych.

● Widok - wyświetlanie części modelu, warstwa prezentacji, może pobierać dane z modelu.

● Kontroler - logika działania, obsługuje żądania użytkownika, zwykle pośredniczy w przesyłaniu danych pomiędzy kontrolerem, a modelem.

Aplikacja wykorzystuje relacyjną bazę danych - MySQL, która jest uruchomiona na lokalnym komputerze.

3. Analiza zagadnienia i jego modelowanie

Cechy charakterystyczne serwera:

● Po otrzymaniu żądania, przetwarza go i wysyła odpowiedź.

● Pasywny.

● Oczekuje na żądania klientów.

Cechy charakterystyczne klienta:

● Przesyła zapytanie do serwera.

● Aktywny.

● Po wysłaniu zapytania do serwera, oczekuje na jego odpowiedź.

Komunikacja odbywa się za pomocą konkretnego protokołu, który jest zrozumiały dla obu stron na przykład protokół HTTP, SMTP, FTP. Architektura ta ma za zadanie udostępnić informacje użytkownikowi bez względu na jego lokalizację w sieci.

Cechy architektury klient-serwer

● współpraca przez sieć,

● przenośność,

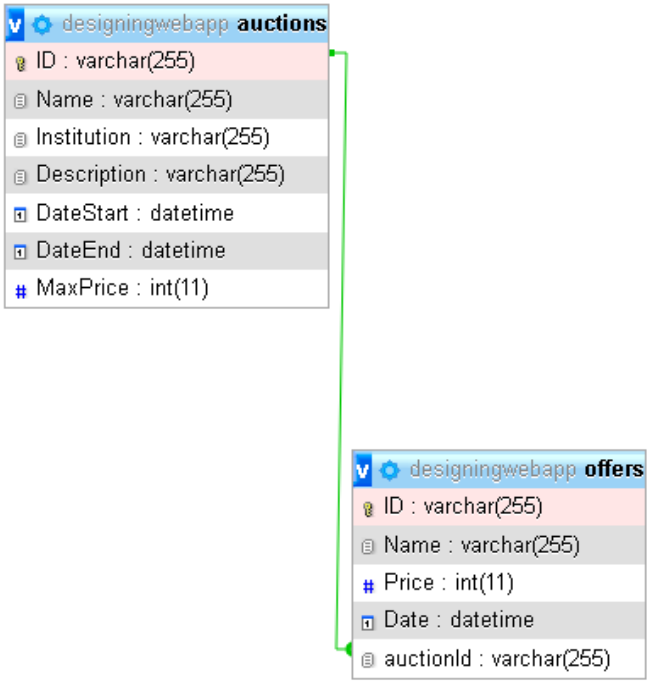
● udostępnianie zasobów,

● skalowalność aplikacji,

● niezależność do lokalizacji.

Backend:

Serwer jest bezpośrednio połączony z bazą danych. Jest podzielony na konfigurację bazy danych, stworzenie modelu przetargu i oferty oraz kontrolerów, które są odpowiedzialne za odpowiednie funkcjonalności aplikacji. Dodatkowo zawiera ścieżki, przez które następuje komunikacja z aplikacją klienta. Dane wrażliwe są ukryte w specjalnym pliku.



Rysunek 1. Diagram bazy danych wygenerowany w aplikacji phpmyadmin

W powyższym diagramie zilustrowano następujące encje:

● auctions - przechowuje informacje o przetargach.

● offers - przechowuje informacje o ofertach.

Jeden przetarg może mieć wiele ofert. Każda z tabel posiada unikalny klucz, który nie może być pusty i generowany jest na serwerze za pomocą biblioteki uuid. Dodatkowo użyte zostały klucze obce w celu określenia związki zachodzących pomiędzy encjami.

Opis atrybutów:

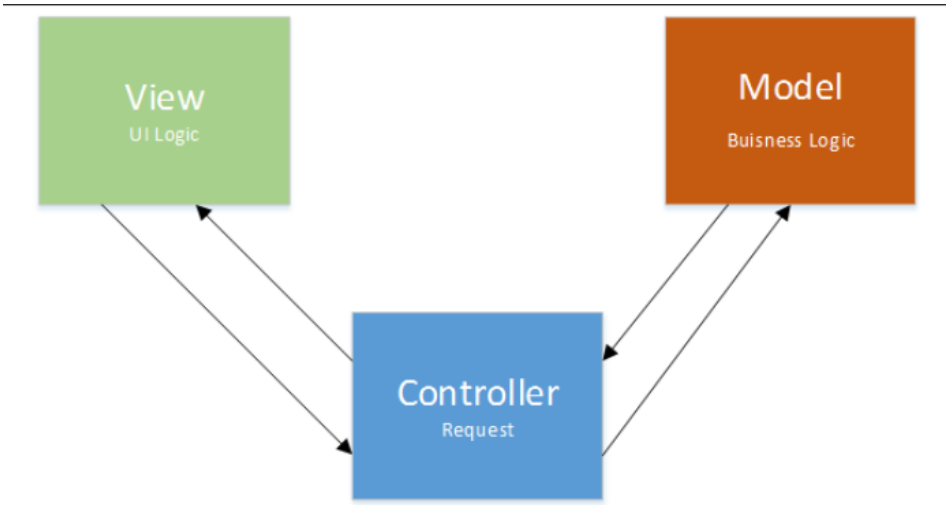
Tabela auctions:

- ID - jest to unikalny i nie pusty identyfikator użytkownika będący kluczem tabeli.
- Name – przechowuje nazwę przedmiotu przetargu.
- Institution – przechowuje nazwę instytucji zamawiającej.
- Description – opis przedmiotu przetargu.
- DateStart – data i godzina rozpoczęcia przetargu.
- DateEnd – data i godzina zakończenia przetargu.
- MaxPrice – maksymalna wartość jaką może zapłacić zamawiający za realizację przedmiotu zamówienia.

Tabela offers:

- ID - jest to unikalny i nie pusty identyfikator użytkownika będący kluczem tabeli.

- Name – przechowuje nazwę składającego ofertę
- Price – zawiera wartość oferty
- Date – zapisuje datę i czas złożenia oferty



Rysunek 2. Schemat przepływu danych w aplikacji za pomocą architektury MVC.

Frontend:

Aplikacja kliencka jest podzielona na komponenty. Funkcjonalności aplikacji podzielone są na podstrony, a zarządzanie stanem jest możliwe dzięki kontekstowi. Zawiera ona jeden obiekt (http), który umożliwia szybką i bezpieczną komunikację z serwerem. Aplikacja jest przygotowana na obsługiwanie błędów przychodzących z aplikacji serwerowej.

4. Implementacja

Serwer został napisany w środowisku Node.js z użyciem Express.js. Serwer łączy się z zewnętrzną bazą danych za pomocą biblioteki Sequelize:

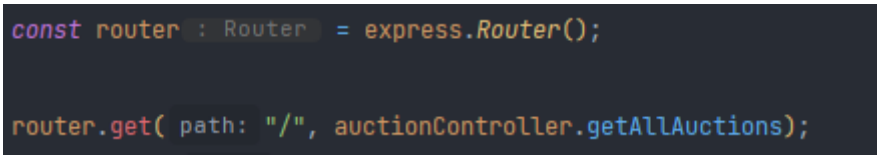
```
const sequelize : Sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
});

sequelize
  .authenticate()
  .then(() : void => {
    console.log("Connected to database...");
  })
  .catch(err => {
    console.log("Error with connecting to database!!!" + err);
  });
```

Rysunek 3. Łączenie serwera z bazą danych

Zostały utworzone modele danych odpowiadające poszczególnym encjom w bazie danych. Przykład modelu Auctions pokazano na Rys. 4.

Rysunek 4.

Endpointy zostały podzielone na Route'y przy pomocy Express.js. Przykład route dla modelu Auctions (Rysunek 5):



Rysunek 5.

Każdy route posiada swój kontroler odpowiedzialny za daną funkcjonalność aplikacji, przykład z rysunku powyżej, patrz Rysunek nr 6:

```
const getAllAuctions = async (req, res) : Promise<void> => {
  let auctions : Model[] = await Auction.findAll()
    .then(auctions => {
      res.status(200).send({
        success: "Success",
        message: "auctions",
        auctions: auctions,
      });
    })
    .catch(err => {
      res.send({
        error: err,
      });
    });
};
```

Rysunek 6. Pobranie wszystkich przetargów w serwisie

Do bezpiecznej wymiany danych pomiędzy stronami służy biblioteka cors, a do obsługi żądań HTTP POST została użyta biblioteka body-parser. Dodatkowo biblioteka Nodemon zapewnia wprowadzanie zmian na serwerze w trybie rzeczywistym.

```
app.use(cors());
```

Rysunek 7. Przykład wykorzystania biblioteki cors.

Frontend został napisany w React.js na komponentach funkcyjnych. Do zarządzania stanem aplikacji wykorzystany został kontekst. Każdy komponent zawiera swoje własne style wykorzystując do tego metodę makeStyles() z biblioteki Material UI. Do komunikacji z serwerem używamy biblioteki axios.

```
export const http : AxiosInstance = axios.create({
  baseURL: BASE_URL,
  headers: {
    "Content-Type": "application/json",
  },
  transformRequest: [
    data => {
      return JSON.stringify(data);
    },
  ],
  transformResponse: [
    data => {
      return JSON.parse(data);
    },
  ],
});
```

Rysunek 8. Wykorzystanie biblioteki axios

```
export const createOffer = async (id, data) : Promise<AxiosResponse<any>> => {
  return http.post( url: `/offer/${id}`, data);
};

export const fetchAuctionData = async id => {
  return http.get( url: `/auction/${id}`);
};

export const AuctionsContextProvider = ({ children }) => {
  const [auctions : any[] , setAuctions] = useState( initialState: []);
  const [closedAuctions : any[] , setClosedAuctions] = useState( initialState: []);
  const [auction : {} , setAuction] = useState( initialState: {});

  const providerValue : {...} = {
    auctions,
    setAuctions,
    closedAuctions,
    setClosedAuctions,
    auction,
    setAuction,
  };

  return <AuctionsContext.Provider value={providerValue}>{children}</AuctionsContext.Provider>;
};
```

Rysunek 9. Fragment kodu AuctionsContext odpowiedzialnego za obsługę stanu aplikacji

```
return (
  <BrowserRouter>
    <AppContextProvider>
      <ThemeProvider theme={theme}>
        <Box
          sx={{
            display: "flex",
            flexDirection: "column",
            minHeight: "100vh",
          }}
        >
          <Header />
          <Routes>
            <Route exact path="/" element={<HomeScreen />} />
            <Route exact path="/auctions" element={<AuctionsScreen />} />
            <Route exact path="/auctions/closed" element={<ClosedAuctionsScreen />} />
            <Route exact path="/auction/:id" element={<AuctionDetailsScreen />} />
            <Route path="/auctions/add" element={<AddAuctionScreen />} />
            <Route path="*" element={<ErrorScreen />} />
          </Routes>
          <Footer />
        </Box>
      </ThemeProvider>
    </AppContextProvider>
  </BrowserRouter>
);
};
```

Rysunek 10. Fragment kodu app.js odpowiedzialny za obsługę podstron aplikacji

Rysunek 11. Fragment kodu AuctionsScreen.jsx odpowiedzialny za pobranie danych o trwających przetargach i zapisanie danych do zmiennej w globalnym stanie.

5. Podsumowanie

Minimalny cel aplikacji został osiągnięty w 100%. Serwis poprawnie dodaje nowe przetargi i zapisuje oferty w relacyjnej bazie danych. Implementacja odbyła się bez większych trudności. Mały problem mógł zostać napotkany przy próbie rozplanowania i implementacji poprawnej bazy danych. Oczywiście perspektywa rozwoju jak najbardziej istnieje.

Możliwości rozwoju:

- Możliwość założenia konta w serwisie
- Możliwość widoku administratora
- Możliwość wyszukiwania i filtrowania przetargów/ofert w przypadku bardzo dużych ilości danych
- Obsługa wszystkich błędów mogących wystąpić podczas użytkowania aplikacji
- Napisanie dużej ilości testów, które pomogą w izolacji aplikacji przed pojawieniem się niepożądanych błędów.
- Możliwość blokowania użytkowników, za nieodpowiednie zachowanie w serwisie
- Obsługa niecenzuralnych słów w aplikacji
- Możliwość edycji i usunięcia dodanego przez siebie przetargu lub oferty
- Podgląd dodanych przez użytkownika przetargów/ofert
- Stworzenie regulaminu serwisu
- Sekcja FAQ
- Możliwość kontaktu z administratorem serwisu
- Refaktoryzacja kodu.
- Dodanie możliwości przetłumaczenia strony na kilka wybranych przez siebie języków.
- Możliwość dodawania zdjęć do przedmiotu przetargu

Dokumentacja została utworzona przez:
Paweł Kapusta