

Spis treści


1.	Dokument potwierdzający zlecenie firmy, specyfikacja zadania projektowego	3
2.	Przedstawienie działania i wygląd.....	4
2.1	Panel logowania.....	4
2.2	Panel składania zamówienia.....	4
2.3	Lista zamówień, aktualizacja statusu zamówienia	6
2.4	Wiadomość e-mail.....	7
2.5	Baza danych	7
3.	Omówienie kodu	9
3.1	Plik odpowiedzialny za wyświetlanie odpowiednich stron	9
3.2	Strona logowania.....	9
3.3	Strona z formularzem	12
3.4	Strona z listą zamówień.....	15
3.5	Komponent odpowiedzialny za generowanie tabeli z zamówieniami.....	16
3.6	Pliki odpowiedzialne za łączenie z bazą danych	17


2. Przedstawienie działania i wygląd

2.1 Panel logowania

Samo logowanie odbywa się poprzez odpowiedni panel, gdzie dane logowania są odpowiednio dostosowane do stanowiska w firmie. W celu prezentacji zostało utworzone konto testowe posiadające wszystkie permisji.

Logowanie

 Login

 Hasło

Zaloguj



2.2 Panel składania zamówienia

Został zaprojektowany panel składania zamówienia, dostosowany do potrzeb zawartych w zleceniu.


 List


 Logout


Składanie zamówienia


Wybierz system


Wybierz artykuł

 Imie

 Nazwisko

 Adres zamieszkania

 Kod Pocztowy

 Dodatkowe informacje...

Wyślij formularz

➤ Opcja wybierz system :

Wybierz system
Alibi
Art1
Art2
Art3
Art4
Art5

➤ Opcja wybierz artykuł :

Wybierz artykuł
Art1
Art2
Art3
Art4
Art5

Warto wspomnieć iż sam formularz zostanie poprawnie przekazany tylko i wyłącznie gdy wszystkie dane zostaną poprawnie uzupełnione, brak jakichkolwiek danych spowoduje wywołanie odpowiedniego komunikatu informującego użytkownika o błędzie.

Składanie zamówienia

Coś poszło nie tak wpisz dane ponownie

Jeżeli operacja przebiegnie prawidłowo pracownik zostanie przeniesiony do kolejnego panelu, w którym są zamieszczone wszystkie informacje o zamówieniach dotychczasowo złożonych

2.3 Lista zamówień, aktualizacja statusu zamówienia

[Form](#)[Logout](#)

Lista zamówień

Imię	Nazwisko	Adres Zamieszkania	Kod Pocztowy	Artykuł	Dodatkowe informacje	Stan Zamówienia	Zmiana stanu Zamówienia
s	s	s	s	Amarant Art4	s	Wyprodukowano	Usuń
e	e	e	e	Alibi Art3	e	Zlecone do produkcji	Produkcja
dd	d	d	dd	Kairo Art3	d	Zlecone do produkcji	Produkcja
Jakub	Kawalerski	Test 1	88-100	Figaro Art5	Zamówienie testowe	Zlecone do produkcji	Produkcja

Panel lista zamówień zawiera wszystkie dotychczasowo złożone zamówienia za pośrednictwem systemu.

Użytkownik ma możliwość zmiany statusu zamówienia przy użyciu przycisku dostępnego w kolumnie „zmiana statusu zamówienia”.

Jakub	Kawalerski	Test 1	88-100	Figaro Art5	Zamówienie testowe	Produkcja	Wyprodukowano
-------	------------	--------	--------	-------------	--------------------	-----------	---------------

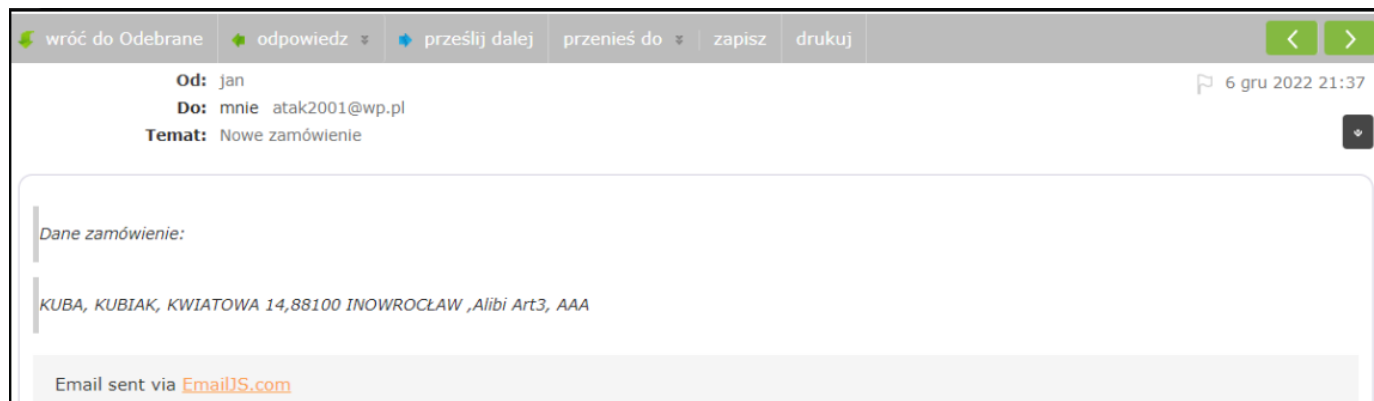
Opcja „wyprodukowano” jest to znak do kontaktu z klientem przez pracownika sklepu w celu ustalenia dostawy zamówionego produktu.

Jakub	Kawalerski	Test 1	88-100	Figaro Art5	Zamówienie testowe	Wyprodukowano	Usuń
-------	------------	--------	--------	-------------	--------------------	---------------	------

Jeżeli cała operacja została wykonana pomyślnie pracownik produkcji ma możliwość usunięcia danego zamówienia, które po kliknięciu opcji „usuń” zostanie przeniesione do archiwum, w tym przypadku odpowiedniej tabeli w bazie danych.

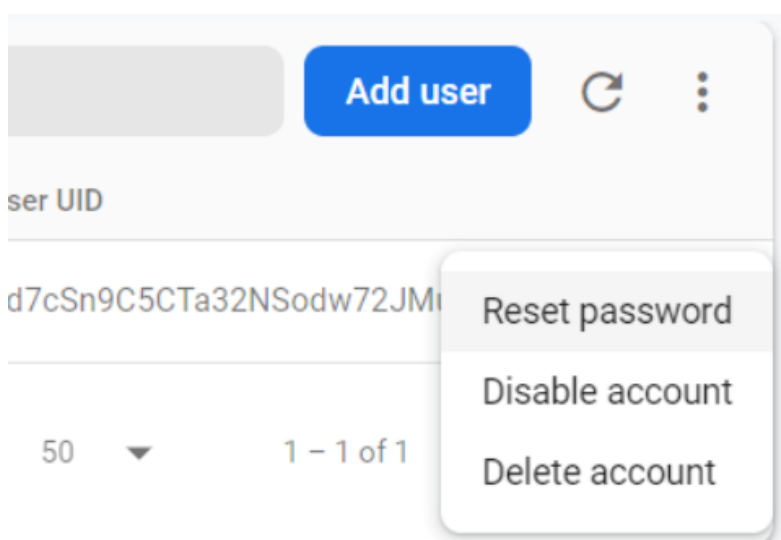
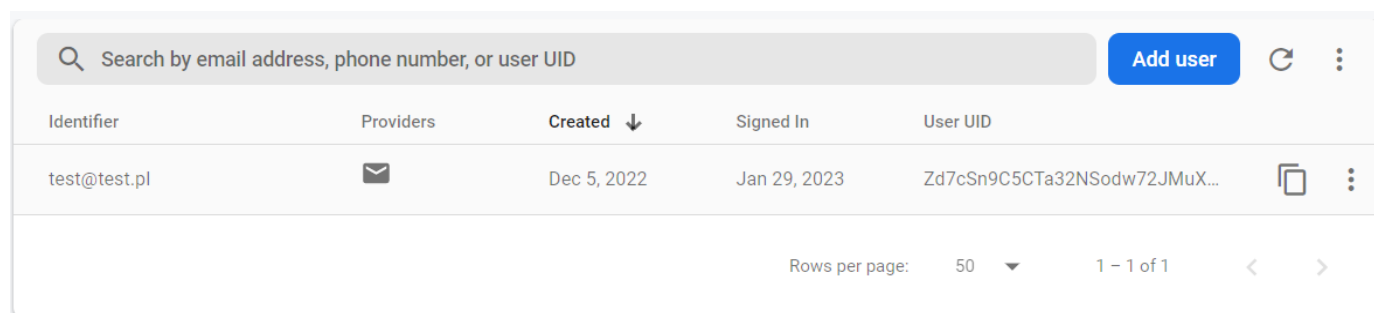
2.4 Wiadomość e-mail

Poniżej przedstawiono przykładową wiadomość po wypełnieniu formularza przez pracownika :



2.5 Baza danych

➤ Zarządzanie kontami :



➤ Złożone zamówienie :

<https://zpo-projekt-default-rtdb.europe-west1.firebaseio.com>

<https://zpo-projekt-default-rtdb.europe-west1.firebaseio.com/>

orders +

- NMxehb00RBq9NbJ3SJU
 - additional: "Zamówienie testowe"
 - adres: "Test 1"
 - id: 0.25110342979709843
 - item: "Figaro Art5"
 - name: "Jakub "
 - postal: "88-100"
 - stan: "Wyprodukowano"
 - surname: "Kawalerski"
- NMyD5XeLaYfaGsnLaD0
 - additional: "ff"

<https://zpo-projekt-default-rtdb.europe-west1.firebaseio.com>

- name: "Jakub "
- postal: "88-100"
- stan: "Wyprodukowano"
- surname: "Kawalerski"
- NMyD5XeLaYfaGsnLaD0 +
 - additional: "ff"
 - adres: "ff"
 - id: 0.8056149573523299
 - item: "Figaro Art1"
 - name: "ff"
 - postal: "ff"
 - stan: "Wyprodukowano"
 - surname: "ff"

3. Omówienie kodu

3.1 Plik odpowiedzialny za wyświetlanie odpowiednich stron

App.js -kod definiuje komponent React nazywany App. Komponent używa komponentu Routes z biblioteki routingu do zdefiniowania trzech możliwych tras. Pierwsza trasa, o ścieżce "/", renderuje komponent LoginPage. Dwie pozostałe trasy, o ścieżkach "/form" i "/list", renderują odpowiednio komponenty FormPage i ListPage, ale tylko wtedy, gdy w pamięci lokalnej jest wartość dla klucza "token".

```
import './App.css';
import LoginPage from './pages/LoginPage';
import { Route, Routes } from 'react-router-dom';
import ListPage from './pages/ListPage';
import FormPage from './pages/FormPage';

function App() {
  return (
    <div className="App">
      <Routes>
        <Route exact path="/" element={<LoginPage />}></Route>
        {localStorage.getItem( key: "token") !== null && (
          <Route path="/form" element={<FormPage />}></Route>
        )}
        {localStorage.getItem( key: "token") !== null && (
          <Route path="/list" element={<ListPage />}></Route>
        )}
      </Routes>
    </div>
  );
}
export default App;
```

3.2 Strona logowania

Kod definiuje funkcjonalny komponent w React o nazwie LoginPage. Komponent korzysta z hooków useNavigate i useDispatch z bibliotek @reach/router i react-redux, odpowiednio. Komponent również korzysta z hooka useState, aby zarządzać stanem lokalnym, konkretnie wartościami pól wejściowych logowania i hasła oraz flagą wskazującą, czy dane logowania są poprawne.

Komponent posiada funkcję submitHandler, która jest wywoływana podczas zdarzenia wysłania formularza. Funkcja wysyła żądanie POST do API z wartościami logowania i hasła z pól wejściowych, a jeśli API zwraca pomyślną odpowiedź, wywołuje akcję logowania użytkownika. Jeśli API zwraca błąd, w konsoli jest wyświetlona wiadomość błędu.

Komponent posiada również funkcję autoLogoutHandler, która ustawia timer dla funkcji auto-wylogowania na podstawie odpowiedzi API. Komponent posiada jeszcze dwie funkcje: changeLoginHandler i changePasswordHandler, które są wywoływane podczas zmiany pól wejściowych logowania i hasła i aktualizują odpowiednie wartości w stanie lokalnym.

```
const submitHandler = (event) => {
  event.preventDefault();

  fetch(
    input: "https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassw
    init: {
      method: "POST",
      body: JSON.stringify( value: {
        email: loginInputValue,
        password: passInputValue,
        returnSecureToken: true,
      }),
      headers: {
        "Content-Type": "application/json",
      },
    }
  )
  .then((res : Response ) => {
    if (res.ok) {
      return res.json();
    } else {
      return res.json().then((data) => {
        setDataIsCorrect( value: false);
        console.log(data);
        throw new Error("Authentication Failed!");
      });
    }
  })
  .then((data : Response ) => {
    console.log("zalogowano");
    const expirationTime = new Date(
      value: new Date().getTime() + +data.expiresIn * 1000
    );
    dispatch(authActions.login(data.idToken));
    autoLogoutHandler(expirationTime);
    navigate("/form");
  })
  .catch((err) => {
    console.log(err.message);
  });
};

const autoLogoutCalculator = (expirationTime) => {
  const currentTime = new Date().getTime();
  const adjExpirationTime = new Date(expirationTime).getTime();

  const remainingDuration = adjExpirationTime - currentTime;
  return remainingDuration;
};
```



```

const autoLogoutHandler = (expirationTime) => {
  const remainingDuration = autoLogoutCalculator(expirationTime);
  setTimeout( handler: () => {
    navigate("/");
    dispatch(authActions.logout());
  }, remainingDuration);
};

const changeLoginHandler = (event) => {
  setLoginInputValue(event.target.value);
};

const changePasswordHandler = (event) => {
  setPassInputValue(event.target.value);
};

```

➤ Komponent React LoginPage wygląda następująco:

```

return (
  <div className={classes.container}>
    {/* <section>
      <div className={classes.skewed}></div>
    </section> */}
    <form onSubmit={handleSubmit}>
      <h1>Logowanie</h1>
      {!dataIsCorrect && (
        <div className={classes.error}>
          <p className={classes.errorMessage}>Podano zły login lub hasło</p>
        </div>
      )}
      <Input
        placeholder="Login"
        icon={faUser}
        type="text"
        onChange={changeLoginHandler}
        value={loginInputValue}
        isCorrect={dataIsCorrect}
      />
      <Input
        placeholder="Hasło"
        icon={faLock}
        type="password"
        onChange={changePasswordHandler}
        value={passInputValue}
        isCorrect={dataIsCorrect}
      />
      <input type="submit" value="Zaloguj" />
    </form>
    <div className={classes.divider}>
      <svg
        data-name="Layer 1"
        xmlns="http://www.w3.org/2000/svg"
        viewBox="0 0 1200 120"
        preserveAspectRatio="none"
      >
        <path
          d="M0,0V46.29c47.79,22.2,103.59,32.17,158,28,70.36-5.37,136.33-33.31,
          opacity=".25"
          className={classes.fill}
        ></path>
        <path
          d="M0,0V15.81c13,36.92,27.64,56.86,47.69,72.05,99.41,111.27,165,111,
          opacity=".5"
          className={classes.fill}
        ></path>
        <path
          d="M0,0V5.63c149.93,59,314.09,71.32,475.83,42.57c43-7.64,84.23-20.12,
          className={classes.fill}
        ></path>
      </svg>
    </div>
  </div>
);

```

3.3 Strona z formularzem

Wszystkie produkty wybierane poprzez tworzenia zamówienia w formularzu :

```
const categories = [
  { name: "Alibi" },
  { name: "Figaro" },
  { name: "Magnum" },
  { name: "Amarant" },
  { name: "Diana" },
  { name: "Kairo" },
  { name: "Mistic" },
  { name: "Andre" },
  { name: "Carla" },
  { name: "Elizabeth" },
  { name: "Intro" },
  { name: "Lilly" },
  { name: "Picasso" },
  { name: "Preston" },
  { name: "Rico" },
  { name: "Salwador" },
  { name: "Verso" },
  { name: "Virdi" },
  { name: "Vinci" },
];
```

```
const categories2 = [
  { name: "Art1" },
  { name: "Art2" },
  { name: "Art3" },
  { name: "Art4" },
  { name: "Art5" },
  { name: "Art6" },
  { name: "Art7" },
  { name: "Art8" },
  { name: "Art9" },
  { name: "Art10" },
  { name: "Art11" },
  { name: "Art12" },
  { name: "Art13" },
  { name: "Art14" },
  { name: "Art15" },
];
```

```
const FormPage = () => {
  const { sendRequest } = useHttp(addOrder);
  const navigate = useNavigate();
  const [categoryInputValue, setCategoryInputValue] = useState( initialState: "" );
  const [categorySecondInputValue, setCategorySecondInputValue] = useState( initialState: "" );
  const [descriptionInputValue, setDescriptionInputValue] = useState( initialState: "" );
  const [nameInputValue, setNameInputValue] = useState( initialState: "" );
  const [surnameInputValue, setSurnameInputValue] = useState( initialState: "" );
  const [adresInputValue, setAdresInputValue] = useState( initialState: "" );
  const [postalCodeInputValue, setPostalCodeInputValue] = useState( initialState: "" );

  const [descriptionIsCorrect, setDescriptionIsCorrect] = useState( initialState: true );
  const [nameIsCorrect, setNameIsCorrect] = useState( initialState: true );
  const [surnameIsCorrect, setSurnameIsCorrect] = useState( initialState: true );
  const [adresIsCorrect, setAdresIsCorrect] = useState( initialState: true );
  const [postalIsCorrect, setPostalIsCorrect] = useState( initialState: true );

  const changeCategoryHandler = (category) => {
    setCategoryInputValue(category);
  };
};
```

Kod tworzy funkcjonalność strony formularza. Używa komponentu useHttp do wysłania żądania API i komponentu useNavigate do nawigacji. Siedem stanów (categoryInputValue, categorySecondInputValue, descriptionInputValue, nameInputValue, surnameInputValue, adresInputValue, postalCodeInputValue) jest ustawianych i aktualizowanych przez funkcje obsługujące zdarzenia. Kolejne siedem stanów (descriptionIsCorrect, nameIsCorrect, surnameIsCorrect, adresIsCorrect, postalIsCorrect) jest ustawianych na true lub false w zależności od poprawności wprowadzonych danych. Funkcja submitHandler wysyła formularz po naciśnięciu przycisku i sprawdza, czy wszystkie pola są wypełnione, a jeśli nie, ustawia stan error na true.

```
const submitHandler = (event) => {
  event.preventDefault();
  if (nameInputValue === "") {
    setNameIsCorrect( value: false);
    localStorage.setItem("error", true);
    return;
  }
  if (surnameInputValue === "") {
    setSurnameIsCorrect( value: false);
    localStorage.setItem("error", true);
    return;
  }
  if (adresInputValue === "") {
    setAdresIsCorrect( value: false);
    localStorage.setItem("error", true);
    return;
  }
  if (postalCodeInputValue === "") {
    setPostalIsCorrect( value: false);
    localStorage.setItem("error", true);
    return;
  }
  if (descriptionInputValue === "") {
    setDescriptionIsCorrect( value: false);
    localStorage.setItem("error", true);
    return;
  }
  if (categoryInputValue === "" || categorySecondInputValue === "") {
    localStorage.setItem("error", true);
    return;
  }
}
```

Kod ten tworzy losowy identyfikator i wysyła żądanie zawierające dane dotyczące produktu (dodatkowy opis, adres, id, nazwa produktu, kod pocztowy, stan produkcji, nazwisko). Następnie tworzy parametry szablonu wiadomości e-mail i wysyła wiadomość e-mail na określony adres. Po wysłaniu wiadomości e-mail, dane dotyczące błędu są usuwane z pamięci lokalnej i użytkownik jest przekierowywany do listy. Jeśli w pamięci lokalnej znajduje się informacja o błędzie, zostanie wyświetlony komunikat błędu na stronie.

```
const id = Math.random();
sendRequest( requestData: {
  additional: descriptionInputValue,
  adres: adresInputValue,
  id: id,
  item: categoryInputValue + " " + categorySecondInputValue,
  name: nameInputValue,
  postal: postalCodeInputValue,
  stan: "Zlecone do produkcji",
  surname: surnameInputValue,
});

var templateParams = {
  name: "atak2001@wp.pl",
  name2: "michal-kurtys@wp.pl",
  message: `${nameInputValue}, ${surnameInputValue}, ${adresInputValue}, ${postalCodeInputValue}, ${categoryInputValue + " " + categorySecondInputValue}, ${descriptionInputValue}`,
};

emailjs
  .send(
    {
      serviceID: "service_4umyyvd",
      templateID: "template_ugs705d",
      templateParams,
      publicKey: "-XzRK96iwbkhHOAfM"
    }
  )
  .then(
    function (response : EmailJSResponseStatus) {
      console.log("SUCCESS!", response.status, response.text);
    },
    function (error) {
      console.log("FAILED...", error);
    }
  );

localStorage.removeItem( key: "error");
navigate("/list");
};

let error;
if (localStorage.getItem( key: "error")) {
  error = (
    <p className={classes.error}>Coś poszło nie tak wpisz dane ponownie</p>
  );
}
```

3.4 Strona z listą zamówień

Ten kod reprezentuje komponent "ListPage" dla strony wyświetlającej listę zamówień. Komponent używa hooka "useHttp" do pobierania zamówień z API za pomocą funkcji "sendRequest".

Jeśli pojawia się błąd, to info na stronie zmienia się na "Wystąpił błąd". Jeśli status jest w trakcie wczytywania, info na stronie zmienia się na "Wczytywanie zamówień". Jeśli status jest ukończony, a nie ma żadnych zamówień, info na stronie zmienia się na "Brak zamówień".

Kiedy status jest ukończony i istnieją zamówienia, komponent wyświetla tabelę zawierającą te zamówienia i przypisuje funkcję "onUpdateHandler" do przycisku umożliwiającego odświeżenie listy zamówień.

```
const ListPage = () => {
  const {
    sendRequest,
    status,
    data: loadedItems,
    error,
  } = useHttp(getAllItems, { startWithPending: true });

  useEffect(
    () => {
      sendRequest();
      localStorage.removeItem("error");
    },
    [sendRequest]
  );

  let info;
  if (status === "pending") {
    info = <h2 style={{ margin: "20px" }}>Wczytywanie zamówień</h2>;
  }

  if (error) {
    info = <h2 style={{ margin: "20px" }}>Wystąpił błąd</h2>;
  }

  if (status === "completed" && (!loadedItems || loadedItems.length === 0)) {
    info = <h2 style={{ margin: "20px" }}>Brak zamówień</h2>;
  }

  const onUpdateHandler = () => {
    sendRequest();
  };

  return (
    <>
      <Navigation />
      <h1 className={classes.title}>Lista zamówień</h1>
      {info}
      {status === "completed" &&
        (!loadedItems || loadedItems.length === 0) && (
          <Table data={loadedItems} onUpdate={onUpdateHandler} />
        )}
    </>
  );
};
```

3.5 Komponent odpowiedzialny za generowanie tabeli z zamówieniami

W kodzie wyżej widać, że funkcja `sendRequest` jest wywoływana dwa razy - raz w `useEffect` wewnątrz komponentu `Table` i raz w funkcji `onUpdateHandler` wewnątrz komponentu `ListPage`. Ta funkcja wysyła żądanie HTTP i pobiera dane. W związku z tym, aby pobrać dane dwukrotnie, trzeba powtórzyć proces wywołania `sendRequest` dwukrotnie.

Funkcja `sendRequest` jest wywoływana, aby pobrać identyfikatory zamówień z bazy danych w komponentcie `Table`, a następnie w funkcji `onUpdateHandler` wewnątrz komponentu `ListPage`, aby odświeżyć listę zamówień. W obu przypadkach celem jest pobranie aktualnych danych z bazy danych.

```
const Table = (props) => {
  const { sendRequest, data: loadedOrders } = useHttp(getOrderIDs, { startWithPending: true });

  useEffect( effect: () => {
    sendRequest();
  }, deps: [sendRequest]);

  const deleteHandler = async (id) => {
    let orderDataFinal = [];
    for (const key in loadedOrders) {
      let orderData = [];
      orderData.push(loadedOrders[key]);
      orderData.map((item) => {
        if (item.id === id) {
          orderDataFinal.push(key);
          return;
        }
      });
    }

    const response = await fetch(
      {
        input: `https://zpo-projekt-default-rtdb.europe-west1.firebaseio.com/orders/${orderDataFinal}.json`,
        init: {
          method: "DELETE",
          body: null,
          headers: {
            "Content-Type": "application/json",
          },
        },
      },
    );
    props.onUpdate();
  };

  const updateHandler = async (id) => {
    let newStateValue;
    let orderDataFinal = [];
    for (const key in loadedOrders) {
      let orderData = [];
      orderData.push(loadedOrders[key]);
      orderData.map((item) => {
        if (item.id === id) {
          orderDataFinal.push(key);
          if (item.stan === "Zlecone do produkcji") {
            newStateValue = "Produkcja";
          } else if (item.stan === "Produkcja") {
            newStateValue = "Wyprodukowano";
          }
          return;
        }
      });
    }
  };
}
```

```
const response = await fetch(
  input: `https://zpo-projekt-default-rtdb.europe-west1.firebaseio.com/orders/${orderDataFinal}.json`,
  init: {
    method: "PATCH",
    body: JSON.stringify( value: { stan: newStateValue } ),
    headers: {
      "Content-Type": "application/json",
    },
  },
);
props.onUpdate();
};
```

3.6 Pliki odpowiedzialne za łączenie z bazą danych

Jest funkcja custom hooka "useHttp" używana do wysyłania żądań HTTP w aplikacji React. Hook używa "useReducer" do zarządzania stanem żądania HTTP i udostępnia informacje o stanie, danych odpowiedzi i błędzie jako obiekt. Funkcja "sendRequest" jest callbackiem, który można użyć do wysłania żądania HTTP za pomocą funkcji "requestFunction" przekazanej jako argument. Gdy żądanie zostanie wysłane, hook zmienia stan i udostępnia informacje o wyniku żądania.

components/Table.js

```
function httpReducer(state, action) {
  if (action.type === 'SEND') {
    return {
      data: null,
      error: null,
      status: 'pending',
    };
  }

  if (action.type === 'SUCCESS') {
    return {
      data: action.responseData,
      error: null,
      status: 'completed',
    };
  }

  if (action.type === 'ERROR') {
    return {
      data: null,
      error: action.errorMessage,
      status: 'completed',
    };
  }

  return state;
}

function useHttp(requestFunction, startWithPending : boolean = false) {
  const [httpState, dispatch] = useReducer(httpReducer, initializerArg: {
    status: startWithPending ? 'pending' : null,
    data: null,
    error: null,
  });

  const sendRequest = useCallback(
    callback: async function (requestData) {
      dispatch({ type: 'SEND' });
      try {
        const responseData = await requestFunction(requestData);
        dispatch({ type: 'SUCCESS', responseData });
      } catch (error) {
        dispatch({
          type: 'ERROR',
          errorMessage: error.message || 'Something went wrong!',
        });
      }
    },
    [requestFunction]
  );

  return {
    sendRequest,
    ...httpState,
  };
}
```

Plik z kodem JavaScript, który eksportuje trzy funkcje do interakcji z bazą danych Firebase Real-time Database:

- `getAllItems`: Ta funkcja pobiera wszystkie dane zamówień z bazy danych Firebase Real-time Database, konwertuje je na format JSON i zwraca jako tablicę obiektów. Jeśli odpowiedź nie jest pomyślna, rzuca błąd z komunikatem błędu.
- `addOrder`: Ta funkcja tworzy nowe zamówienie poprzez wysłanie żądania POST do bazy danych Firebase Real-time Database z danymi zamówienia. Jeśli odpowiedź nie jest pomyślna, rzuca błąd z komunikatem błędu.
- `getOrderIDs`: Ta funkcja pobiera wszystkie dane zamówień z bazy danych Firebase Real-time Database, konwertuje je na format JSON i zwraca je. Jeśli odpowiedź nie jest pomyślna, rzuca błąd z komunikatem błędu.

lib/ api.js

```
export async function getAllItems() {
  const response = await fetch( input: `${FIREBASE_DOMAIN}/orders.json` );
  const data = await response.json();

  if (!response.ok) {
    throw new Error(data.message || "Could not fetch items.");
  }

  const transformedItems = [];
  for (const key in data) {
    const itemObj = {
      id: key,
      ...data[key],
    };

    transformedItems.push(itemObj);
  }

  return transformedItems;
}
```



```
export async function addOrder(orderData) {
  const response = await fetch( input: `${FIREBASE_DOMAIN}/orders.json`, init: {
    method: "POST",
    body: JSON.stringify(orderData),
    headers: {
      "Content-Type": "application/json",
    },
  });
  const data = await response.json();

  if (!response.ok) {
    throw new Error(data.message || "Could not create quote.");
  }

  return null;
}

export async function getOrderIDs() {
  const response = await fetch( input: `${FIREBASE_DOMAIN}/orders.json`);
  const data = await response.json();

  if (!response.ok) {
    throw new Error(data.message || "Could not fetch items.");
  }
  return data;
}
```

W dokumentacji zostały opisane wszystkie najważniejsze elementy aplikacji.