Politechnika Świętokrzyska w Kielcach
Podsumowanie Projektu

Kierunek: Informatyka
Rok: 3
Semestr: 5
Skład zespołu: Patryk Grzywacz i Dominik Grudzień
Nazwa przedmiotu: Aplikacje Mobilne (Projekt)
Temat projektu: Aplikacja bankowa

Cel i założenia projektu

Celem projektu było opracowanie i zaimplementowanie aplikacji na temat wybranego systemu informacyjnego. Projekt musiał składać się z minimum 8 ekranów. W aplikacji należało wykorzystać trzy metody dostępu do danych za pomocą API (GET, POST, PUT) Kolejnym wymogiem była implementacja trzech nawigacji, które były omawiane na wykładzie Należało użyć wybranego frameworka z gotowymi kontrolkami (antDesign, nativeBase etc.) oraz skorzystać z wybranego sensora dostępnego z urządzenia (aparat, kamera, akcelerometr). Całość projektu miała być utrzymywana w repozytorium np. Github.

Użyte biblioteki i narzędzia w projekcie

Podczas pracy przy projekcie pomogły nam następujące biblioteki i narzędzia:

Biblioteki:

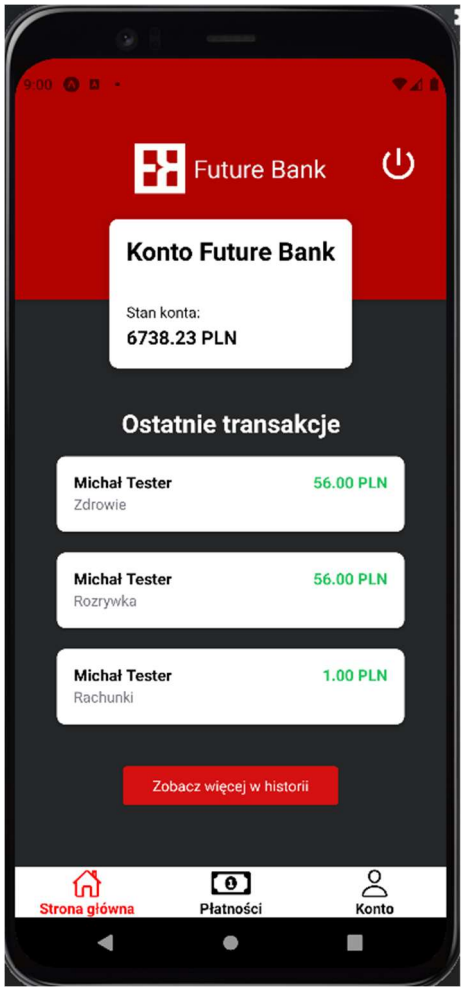
- Axios
- Expo
- Expo-secure-store
- Expo-barcode-scanner
- React-navigation (bottom-tabs, drawer, native-stack)
- Formik
- Yup
- React-toastify
- Jwt-Decode
- Moment
- Native-base

Narzędzia:

- *Webstorm 2021.2.1*

Sposób wykonania projektu

W aplikacji został zachowany duży poziom szczegółowości. Każde detale zostały dokładnie wystylizowane za pomocą styli z react native oraz komponentów i ich właściwości z biblioteki native-base



Widok startowy po zalogowaniu się użytkownika

Staraliśmy się unikać powtórzeń w kodzie za pomocą tworzenia custom hook'ów oraz stosowaniem typów generycznych.

```
import { useCallback, useEffect, useState } from 'react';
import axios from 'axios';

export const useFetchRawData = <T extends unknown>(endpoint: string, params?: any) => {
  const [rawData, setRawData] = useState<T>();
  const [isPending, setIsPending] = useState(false);

  const fetchData = useCallback(async () => {
    setIsPending(true);
    try {
      const { data } = await axios.get<T>(endpoint, { params });
      setRawData(data);
    } catch (e) {
      console.error(e);
    } finally {
      setIsPending(false);
    }
  }, [setIsPending, setRawData, endpoint, params]);
```

```
  useEffect(() => {
    fetchData().catch();
  }, [fetchData]);

  return { rawData, fetchData, isPending };
};
```

Custom hook do pobierania danych z serwera

Skorzystaliśmy z pomocy kontekstu, który reprezentuje zalogowanego użytkownika. Przechowujemy w nim dane ściągnięte z bazy danych, funkcje refetchu danych, która posługuje się tokenem zapisanym w AndroidStorage, funkcje wylogowania użytkownika oraz stan-flagę sygnalizującą czy aktualnie jest pobierany użytkownik. Kontekst ułatwia nam rozpoznanie czy użytkownik jest zalogowany czy nie i na tej podstawie dobierane są odpowiednie widoki. Co więcej umożliwia nam łatwy dostęp do danych i operacji w głęboko zagnieżdżonych komponentach.

```
Import React, { createContext, FC, ReactNode, useCallback, useContext, useEffect, useState }
from "react";
import { CurrentUserContextModel } from "../interfaces/CurrentUserContextModel";
import { ClientModel } from "../interfaces/ClientModel";
import jwtDecode from "jwt-decode";
import { TokenData } from "../interfaces/TokenData";
import axios from "axios";
import { getRawToken } from "../utils/getRawToken";
import { deleteToken } from "../utils/deleteToken";
import { useToast } from "native-base";

const CurrentUserContext = createContext<CurrentUserContextModel | undefined>(undefined);

interface CurrentUserProviderProps {
  children: ReactNode;
}

const useFetchCurrentUser = () => {
  const [currentUser, setCurrentUser] = useState<ClientModel>();
  const [isPending, setIsPending] = useState(false);
  const toast = useToast();

  const fetchUser = useCallback(async () => {
    setIsPending(true);

    const token = await getRawToken();

    if (!token){
      setIsPending(false);
      return;
    }

    const decodedToken: TokenData = jwtDecode(token);

    try {
      const { data } = await axios.get<ClientModel>(`/clients/${decodedToken.userId}`);
      setCurrentUser(data);
    } catch € {
      console.error€;
    } finally {
      setIsPending(false);
    }
  }, [setIsPending, setCurrentUser]);

  useEffect(() => {
    fetchUser().catch();
  }, [setCurrentUser, fetchUser]);

  const handleLogout = async () => {
    setCurrentUser(undefined);
    await deleteToken();
    await fetchUser();
    delete axios.defaults.headers.common["Authorization"];
    toast.show({
      title: 'Dziękujemy za skorzystanie z naszych usług',
      status: 'info',
    });
```

```
  };

  return { fetchUser, currentUser, isPending, handleLogout };
};

export const useCurrentUser = () => useContext(CurrentUserContext) as CurrentUserContextModel;

const CurrentUserProvider: FC<CurrentUserProviderProps> = ({ children }) => {
  const currentUserData = useFetchCurrentUser();
  return (
    <CurrentUserContext.Provider value={currentUserData}>
      {children}
    </CurrentUserContext.Provider>
  );
};

export default CurrentUserProvider;
```

Kontekst zalogowanego użytkownika

Stosowanie zewnętrznych bibliotek do React Native ułatwiło rozwiązywanie problemów związanych z np. walidacją formularzy lub obsługą dat.

```
const validationSchema = Yup.object().shape({
  amount: Yup.number().required('Kwota jest wymagana').positive('Kwota nie może być ujemna'),
  category: Yup.string().required('Kategoria jest wymagana'),
  receiver_sender: Yup.string().required('Odbiorca jest wymagany'),
  title: Yup.string().required('Tytuł przelewu jest wymagany'),
  toAccountNumber: Yup.string().required('Number konta odbiorcy jest wymagany')
    .min(26, 'Numer konta posiada za mało cyfr')
    .max(26, 'Numer konta posiada za dużo cyfr')
});
```

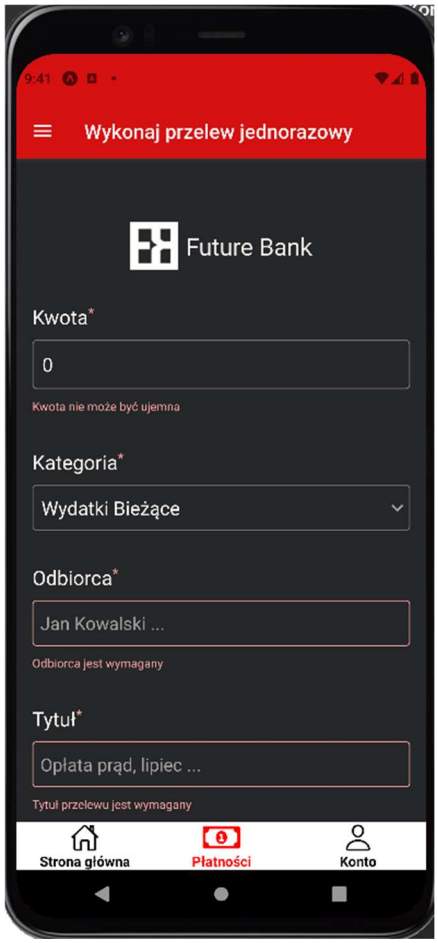
Schemat walidacyjny Yup do formularzu z nowym przelewem

```
<DateTimePicker
  value={moment(values.transferDate).toDate()}
  display="default"
  onChange={(event: any, date: Date | undefined) => {
    setShouldShowDatePicker(false);
    setFieldValue('transferDate', moment(date).format('YYYY-MM-DD'));
  }}
/>
```

Zastosowanie biblioteki moment do formatowania dat w komponencie DateTimePicker

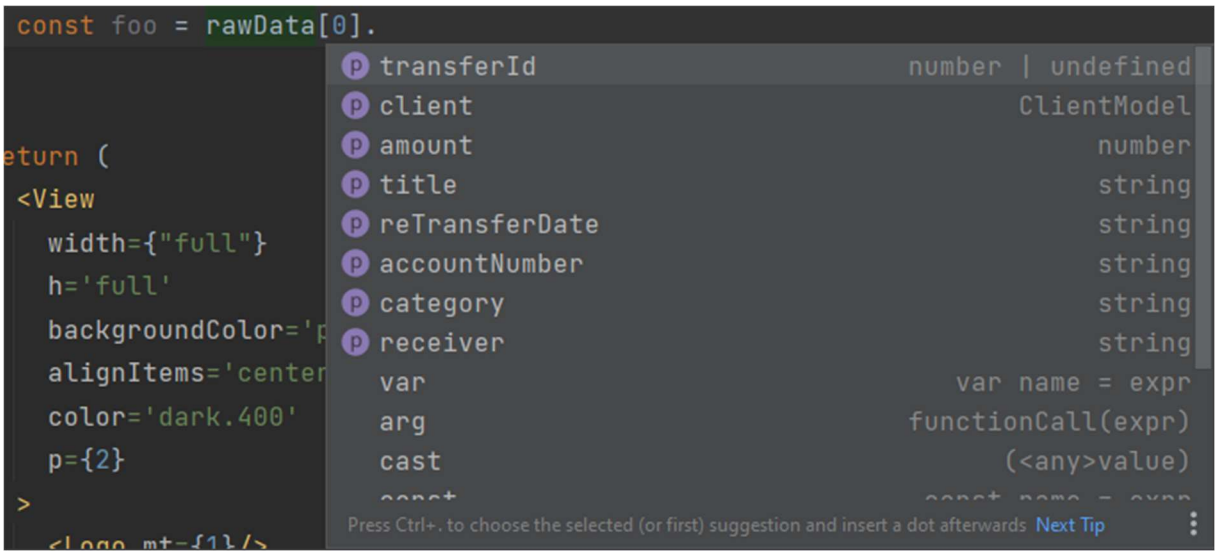
```
<Formik<TransferData>
  initialValues={initialFormikValues}
  onSubmit={handleSubmit}
  validationSchema={validationSchema}
>
  <TransferForm
    transferRouteName={transferRouteName}
    initialValue={route.params.initialAmount}
  />
</Formik>
```

Załączenie walidatora do formularza



Efekt walidacji

Dzięki językowi TypeScript oraz interfejsom mogliśmy w łatwy sposób otypować dane. Usprawniło to pracę oraz czytelność kodu.



Zastosowanie interfejsu do pobranych danych z bazy

W aplikacji korzystamy z naszego backendu. Został on dopasowany do naszego tematu projektu. Dodatkowo działa on na chmurze Heroku. Link do backendu (Swagger): https://pip-backend-server.herokuapp.com/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config
Skorzystaliśmy z kontrolerów takich jak:

- Static resources controller
- Transfer controller
- Client controller
- Cyclical transfer controller

Wnioski

Projekt udało się wykonać w całości. Wszystkie nasze założenia zostały zawarte w aplikacji. Zajęcia projektowe pomogły nam poznać React'a oraz pokazały jak wygląda projektowanie aplikacji mobilnych za pomocą tej biblioteki.