Dokumentacja

System do zarządzania boiskiem szkolnym

Develop Team

January 2021

Spis treści

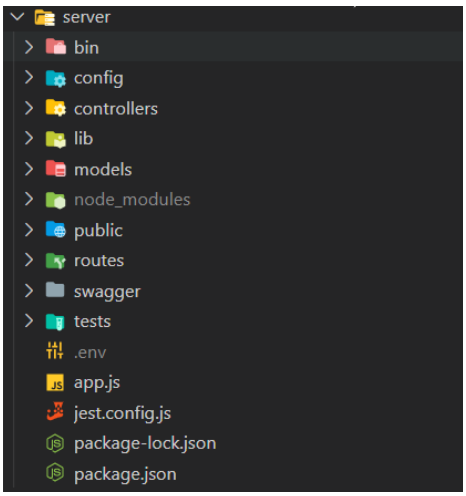
Wstęp

Głównym założeniem aplikacji jest rezerwowanie boiska za pomocą strony internetowej. Zapewnia logowanie lokalne oraz poprzez CAS. Cały projekt składa się z

- Aplikacji Bacekndowej opatej o Node.js
- Aplikacji frontendowej opartej o React.js
- Aplikacji frontendowej dla administratora opartej o ReactAdmin.js

Aplikacja korzysta z architektury REST oraz jest dostosowana do urządzeń mobilnych

1 Backend

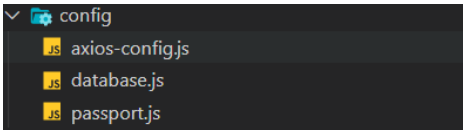


Rysunek 1: Struktura backendu

Część backendowa w aplikacji do zarządzania boiskiem szkolnym jest napisany w języku JavaScript oparta o framework Express.js.

1.1 Katalog config

W tym katalogu mieszczą się trzy pliki



Rysunek 2: Katalog config

1.1.1 database.js

Odpowiada za połączenie z odpowiednią bazą danych na podstawie zmiennej NODE_ENV , która znajduje się w pliku .env

1.1.2 passport.js

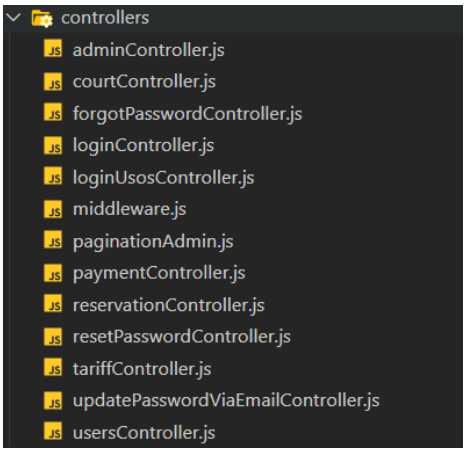
Zawiera implmentację autoryzacji użytkownika poprzec CAS oraz logowanie lokalne

1.1.3 axios-config.js

Konfiguracja axios'a wykorzystywana podczas testów

1.2 Katalog controllers

Katalog zawiera pliki które zawierają funkcje służące do działania całej aplikacji, są one eksportowane i podpinane do endpointów w folderze routes



Rysunek 3: Katalog controllers

1.2.1 adminController.js

Plik ten zawiera wszystkie funkcje, potrzebne to zarządzania aplikacją frontedową przez administratora systemu.

1.2.2 courtController.js

Funkcja odpowiadająca za informowanie frontedu o dostępnych sektorach

1.2.3 forgotPasswordController.js

Funkcje odpowiedzialne za zresetowanie hasła

1.2.4 loginController.js

Logowanie i wylogowywanie użytkownika z aplikacji (outh)

1.2.5 loginUsosController.js

Logowanie i wylogowywanie użytkownika z aplikacji (CAS)
Link do dokumentacji: `https://apps.usos.edu.pl/developers/api/`

1.2.6 middleware.js

Middleware wykorzystywane do zabezpieczenia projektu

1.2.7 paginationAdmin.js

Middleware dla aplikacji adminstratora ustalające liczbe rekordów po filtrowaniu oraz ustawia nagłówek Content-Range, który react-admin wymaga do wyświetlenia danych.

1.2.8 paymentController.js

Metody odpowiedzialne za odebranie tokenu z serwisu Pay'u oraz dokonanie płatności

1.2.9 reservationController.js

Kontroler odpowiedzialny za tworzenie i zwrócenie infomracji o rezerwacji-/rezerwacjach.

1.2.10 resetPasswordController.js

Funkcja sprawdzająca dostępność zmiany hasła

1.2.11 tariffController.js

Dodawanie, modyfikowanie i usuwanie cennika

1.2.12 updatePasswordViaEmailController.js

Kontrolerr do generowania nowego hasła

1.2.13 usersController.js

Tworzenie nowego użytkownika oraz modyfikacja jego danych personalnnych

1.3 Katalog lib

Katalog z konfiguracjami do bibliotek zewnętrznych

Rysunek 4: Katalog lib

1.3.1 password.js

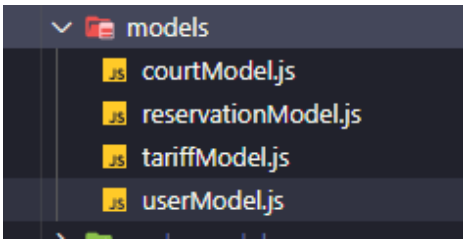
Generowanie hash z hasła oraz sprawdzenie czy hasło jest poprawne

1.3.2 cronEmail.js

Automatyzacja procesu przypominania o dokonanej rezerwacji

1.4 Katalog models

Katalog z modelami bazy danych



Rysunek 5: Katalog models

1.4.1 courtModel.js

Listing 1: courtModel

```
const courtModel = mongoose.Schema({
  nameObject: { type: String, required: true },
  nameCourt: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  date: [
    {
      nameOfDay: String,
      value: Boolean,
    },
  ],
  sessionTime: {
    type: String,
  },
  tariffId: {
    type: ObjectId,
    ref: 'courtsTariff',
    required: true,
  },
});
```

Model boiska składa się z:

- nameObject (Nazwa obiektu)
- nameCourt (nazwa sektora)
- description (opis boiska)
- date (Lista dni tygodnia, w którym boisko jest dostępne)
- sessionTime (czas trwania rezerwacji)
- tariffId (id cennika)

1.4.2 reservationModel.js

Listing 2: reservationModel

```
const reservationModel = mongoose.Schema({
  hour: {
    type: String,
  },
  start: {
    type: Date,
    required: true,
  },
  dayString: {
    type: String,
  },
  end: {
    type: Date,
  },
  courtId: {
    type: ObjectId,
    ref: 'courtModel',
    required: true,
  },
  userId: {
    type: ObjectId,
    ref: 'userModel',
    required: true,
  },
  vat: {
    type: Boolean,
    default: false,
  },
  isServedVat: {
    type: Boolean,
    default: false,
  },
  paid: {
    type: Boolean,
    default: false,
  },
  orderId: {
    type: String,
  },
  emailSent: {
    type: Boolean,
    default: false,
```

```
    required: true,
  },
  price: {
    type: Number,
  },
  referId: String,
});
```

Model rezerwacji składa się z:

- start (data początkowa rezerwacji)
- end (data końcowa rezerwacji)
- dayString (data rezerwacji)
- hour (godzina rezerwacji)
- courtId (id boiska)
- userId (id usera)
- vat (czy użytkownik chce dostać fakture vat)
- isServedVat (czy dane do faktury zostały wygenerowane do pdf)
- paid (czy rezerwacja została opłacona)
- orderId (id płatności PayU)
- emailSent (czy przypomnienie o rezerwacji zostało wysłane)
- price (cena rezerwacji)
- referId (id rezerwacji z koszyka)

1.4.3 tariffModel.js

Listing 3: tariffModel

```
const courtsTariff = mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  classes_and_sports_training: {
    type: String,
    required: true,
  },
  tournament_matches: {
    type: String,
    required: true,
  },
  university_club: {
    type: String,
    required: true,
  },
});
```

Model cennika składa się z:

- name (nazwa obiektu)
- classes_and_sports_training (jeden z typów cennika)
- tournament_matches (jeden z typów cennika)
- university_club (jeden z typów cennika)

1.4.4 userModel.js

Listing 4: userModel

```
const userModel = mongoose.Schema({
  email: String,
  hash: String,
  salt: String,
  idUsos: String,
  studentStatus: String,
  studentNumber: String,
  name: String,
  surname: String,
  age: Number,
  phone_number: String,
  sex: String,
  role: {
    type: String,
    default: 'user',
  },
  adressStreet: String,
  adressCity: String,
  adressPostalCode: String,
  isStudent: {
    type: Boolean,
    default: false,
  },
  isActive: {
    type: Boolean,
    default: true,
  },
  nip: String,
  createDate: Date,
  resetPasswordToken: String,
  resetPasswordExpires: Date,
  firstLogin: {
    type: Boolean,
    default: true,
  },
  sumPrice: {
    type: Number,
  },
  reservations: [
    {
      title: String,
```

```
      start: Date,
      dayString: String,
      end: Date,
      courtId: {
        type: ObjectId,
        ref: 'courtModel',
        required: true,
      },
      nameCourt: String,
      userId: String,
      isPaid: Boolean,
      price: Number,
      vat: {
        type: Boolean,
        default: false,
      },
    },
  ],
});
```

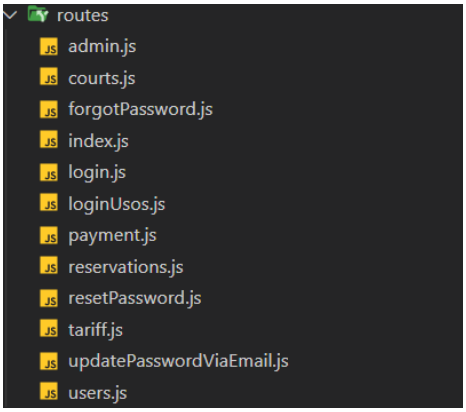
Model użytkownika składa się z:

- email (email użytkownika)
- hash (hash hasła)
- salt (sól do hasła)
- idUsos (id użytkownika zwrócone z systemu CAS)
- studentStatus (czy jest aktywnym studentem)
- name (imie użytkownika)
- surname (nazwisko użytkownika)
- age (wiek użytkownika)
- phone_number (numer telefonu użytkownika)
- sex (płeć użytkownika)
- role (admin lub użytkownik)
- adressStreet (miejsce zamieszkania użytkownika - ulica)
- adressCcity (miejsce zamieszkania użytkownika - miasto)
- adressPostalCode (miejsce zamieszkania użytkownika - kod pocztowy)
- isStudent (czy jest studentem, ustalane przy logowaniu poprzez CAS)

- isActive (czy użytkownik ma aktywne konto)
- nip (NIP użytkownika)
- resetPasswordToken (token do resetowania hasła)
- resetPasswordExpires (czas w którym możemy zresetować hasło)
- firstLogin (czy jest pierwsze logowanie użytkownika)
- sumPrice (cena do koszyka łączna)
- reservations (koszyk z rezerwacjami użytkownika)

1.5 Katalog routes

Katalog ze ścieżkami



Rysunek 6: Katalog routes

1.5.1 admin.js

Metoda	Endpoint	Opis
GET	/api/admin/users	Zwraca wszystkich użytkowników
GET	/api/admin/users/:userId	Zwraca użytkownika o podanym ID
DELETE	/api/admin/users/:userId	Usuwa użytkownika o podanym ID
GET	/api/admin/reservations	Zwraca wszystkie rezerwacje
POST	/api/admin/reservations	Blokuje termin rezerwacji
GET	/api/admin/reservations/:id	Zwraca rezerwajcę o podanym ID
PATCH	/api/admin/reservations/:id	Modyfikuje rezerwajcę o podanym ID
DELETE	/api/admin/reservations/:id	Usuwa rezerwajce o podanym ID
GET	/api/admin/courts	Zwraca cennik
POST	/api/admin/courts	Tworzenie sektora oraz jego cennika w panelu administratora
DELETE	/api/admin/courts/:courtId	Usuwanie boiska o danym ID
PATCH	/api/admin/courts/:courtId	Modyfikacja boiska
GET	/api/admin/courts/:courtId	Zwraca boisko/strefę o podanym ID
GET	/api/admin/priceLists	Zwraca podział cen na wszystkie obiekty
GET	/api/admin/priceLists/:id	Cena o podanym ID
PUT	/api/admin/priceLists/:id	Modyfikacja cennika

1.5.2 authMiddleware.js

Middleware odpowiedzialne za zabezpieczenia ścieżek przed nieautoryzowanym odpytywaniem serwera

1.5.3 courts.js

Metoda	Endpoint	Opis
GET	/api/courts	Zwraca podział boiska

1.5.4 forgotPassword.js

Metoda	Endpoint	Opis
POST	/api/forgotPassword	Zapomniane hasło

1.5.5 login.js

Metoda	Endpoint	Opis
POST	/api/login/:role	Logowanie do panelu administratora
POST	/api/checkUser	Sprawdza czy użytkownik jest zalogowan
POST	/api/login/	Logowanie do aplikacji
GET	/api/logout	Wylogowanie

1.5.6 loginUsos.js

Metoda	Endpoint	Opis
GET	/api/loginUsos/connect	Logowanie przez CAS
GET	/api/loginUsos/callback	Callback logowania przez CAS
GET	/api/loginUsos/logout	Wylogowanie użytkownika przez CAS

1.5.7 payment.js

Metoda	Endpoint	Opis
POST	/api/getToken	Odbiór tokenu z PAY'U
POST	/api/createPayment	Tworzenie płatności
POST	/api/notify	Powiadomienia z serwisu PayU o sukcesie lub anulowaniu płatności

1.5.8 reservations.js

Metoda	Endpoint	Opis
GET	/api/reservationsDate	Zwraca wszystkie dostępne godziny dla odpowiedniego sekto
GET	/api/reservations/:userId	Rezerwacje użytkownika o danym ID
POST	/api/reservation/addToBasket	Dodaje rezerwacje do koszyka

1.5.9 resetPassword.js

Metoda	Endpoint	Opis
GET	/api/reset	Resetowanie hasła

1.5.10 tariff.js

Metoda	Endpoint	Opis
GET	/api/priceLists	Zwraca podział cen na wszystkie obiekty dla aplikacji frontendowej

1.5.11 updatePasswordViaEmail.js

Metoda	Endpoint	Opis
PATCH	/api/updatePasswordViaEmail	Ustawia nowe hasło

1.5.12 user.js

Metoda	Endpoint	Opis
POST	/api/user/create	Tworzenie nowego użytkownika
PATCH	/api/user/update	Aktualizacja danych użytkownika
GET	/api/getUser/:userId	Zwraca użytkownika o podanym ID

1.6 Katalog swagger

Katalog z plikami do tworzenia dokumentacji online projektu

Rysunek 7: swagger katalog

1.7 Plik .env

W tym pliku umieszczamy potrzebne zmienne, klucze, ustawienia których nie chcemy udostępniać w kodzie. Plik musi być utworzony przed pierwszym uruchomieniem projektu.

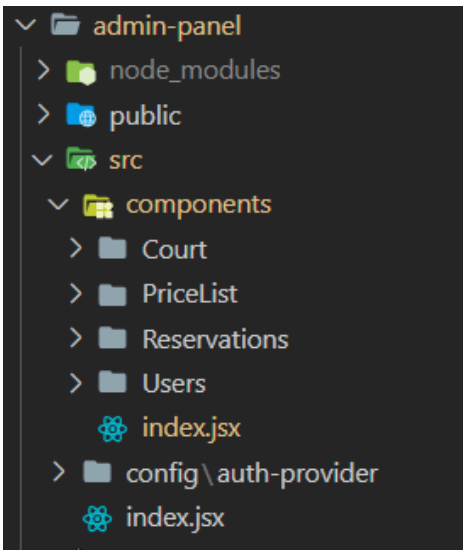
Listing 5: .env

```
DB_CONNECTION = ''
DB_CONNECTION_PROD = ''
DB_CONNECTION_ATLAS = ''
DB_CONNECTION_ATLAS_TEST = ''
NODE_ENV = ''
USOS_CONSUMER_KEY = ''
USOS_CONSUMER_SECRET = ''
OAUTH_SECRET = ''
EMAIL_ADDRESS = ''
EMAIL_PASSWORD = ''
PAYU_CLIENT_ID = ''
PAYU_CLIENT_SECRET = ''
REACT_APP_SECRET = ''
```

1.8 Plik app.js

Główny plik projektu, w którym umieszczona jest niezbędna konfiguracja

2 React-admin



Rysunek 8: Struktura admina

Część admina w aplikacji do zarządzania boiskiem szkolnym jest napisana z wykorzystaniem biblioteki react-admin w react.

2.1 Boiska

W katalogu Court mieszczą się pliki

Rysunek 9: Katalog Court

2.1.1 CourtCreate

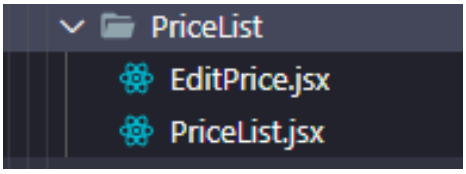
Komponent odpowiedzialny za tworzenie boiska
Wykorzystuje endpoint POST /api/admin/courts

2.1.2 CourtList

Komponent odpowiedzialny za wyświetlanie listy boisk
Wykorzystuje endpoint GET /api/admin/courts

2.2 Cennik

W katalogu PriceList mieszczą się pliki



Rysunek 10: Katalog PriceList

2.2.1 EditPrice

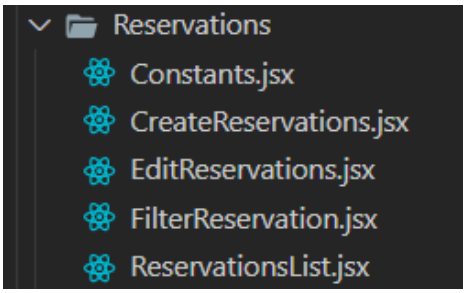
Komponent odpowiedzialny za edycję cennika
Wykorzystuje endpoint PATCH /api/admin/priceLists/:id

2.2.2 PriceList

Komponent odpowiedzialny za wyświetlanie cennika
Wykorzystuje endpoint GET /api/admin/priceLists

2.3 Rezerwacje

W katalogu Reservations mieszczą się pliki



Rysunek 11: Katalog Reservations

2.3.1 Constants

Stałe zmienne do formularzy

2.3.2 CreateReservations

Komponent odpowiedzialny za tworzenie rezerwacji
Wykorzystuje endpoint POST /api/admin/reservations

2.3.3 EditReservations

Komponent odpowiedzialny za edycję rezerwacji
Wykorzystuje endpoint PATCH /api/admin/reservations/:id

2.3.4 FilterReservations

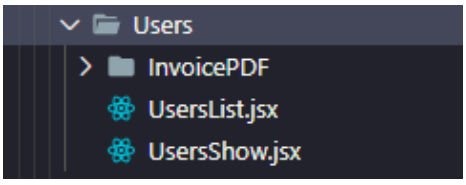
Komponent odpowiedzialny za filtrowanie rezerwacji

2.3.5 ReservationsList

Komponent odpowiedzialny za wyświetlenie listy rezerwacji
Wykorzystuje endpoint GET /api/admin/reservations

2.4 Użytkownicy

W katalogu Users mieszczą się pliki



Rysunek 12: Katalog Users

2.4.1 Katalog invoicePDF

W tym katalogu znajduje się komponent odpowiedzialny za generowanie danych użytkownika do PDF z rezerwacji

2.4.2 UsersList

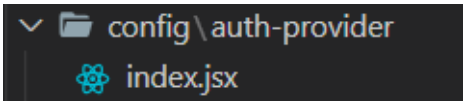
Komponent odpowiedzialny za wyświetlenie użytkowników
Wykorzystuje endpoint GET /api/admin/users

2.4.3 UsersShow

Komponent odpowiedzialny za wyświetlenie szczegółowych danych użytkownika. Wykorzystuje endpoint GET /api/admin/users/:id

2.5 Config

W katalogu Config mieści się plik

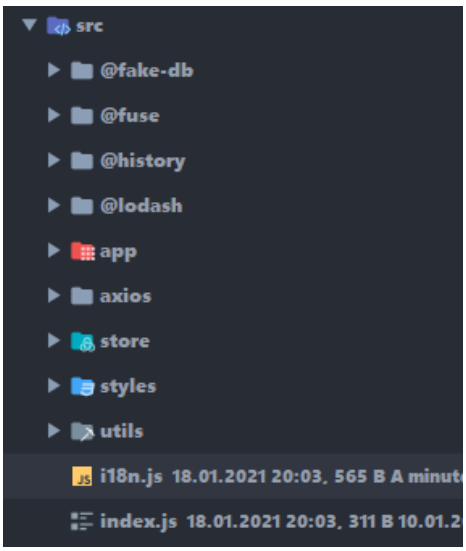


Rysunek 13: Katalog Config

2.5.1 auth-prodiver

W katalogu auth-provider znajduje się komponenet odpowiedzialny za logowanie. Wykorzystuje endpoint GET /api/login/admin

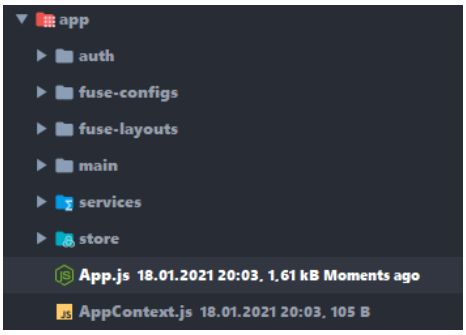
3 DevCourt



Rysunek 14: Struktura DevCourt

Aplikacja do zarządzania boiskiem szkolnym jest napisana z wykorzystaniem biblioteki React.js

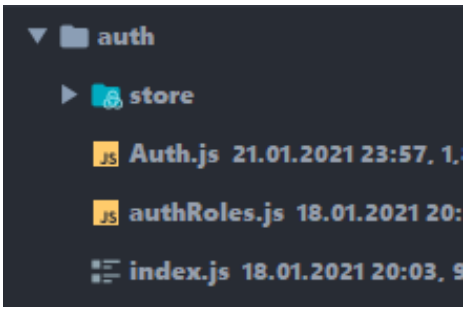
3.1 App



Rysunek 15: Katalog app

Folder zawierające komponenty, strony, style oraz store i serwisy autoryzacji.

3.1.1 Auth



Rysunek 16: Katalog Auth

Folder zawierający metody do autoryzacji.

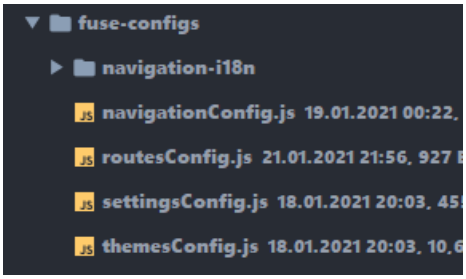
Store Pliki zawarte w folderze to metody do wywołania przy logowaniu, wylogowaniu, zmiane danych dla użytkownika, rejestracji, pobraniu danych użytkownika.

Auth.js Serwis zawierający logowanie i wylogowanie.

authRoles.js Opis roli użytkowników.

index.js Plik zawierający import obu plików Auth.js i authRoles.js.

3.1.2 Fuse-Configs



Rysunek 17: Katalog Fuse-Configs

Folder zawierający konfigurację nawigacji strony, ustawień, wyglądu oraz ścieżek strony.

navigation-i18n Folder zawierający tłumaczenia z języka angielskiego na język polski.

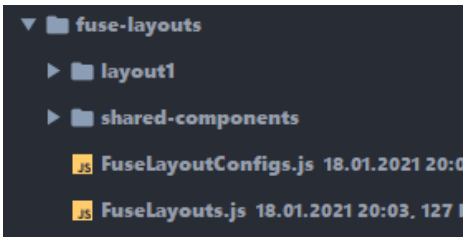
navigationConfig.js Plik zawierający konfigurację nawigacji po aplikacji.

routesConfig.js Plik zawierający konfigurację ścieżek aplikacji.

settingsConfig.js Plik zawierający konfigurację ustawień aplikacji.

themesConfig.js Plik zawierający konfigurację wyglądu aplikacji.

3.1.3 Fuse-Layouts



Rysunek 18: Katalog Fuse-Layout

Folder zawierający komponenty odpowiedzialne za układ strony oraz konfigurację.

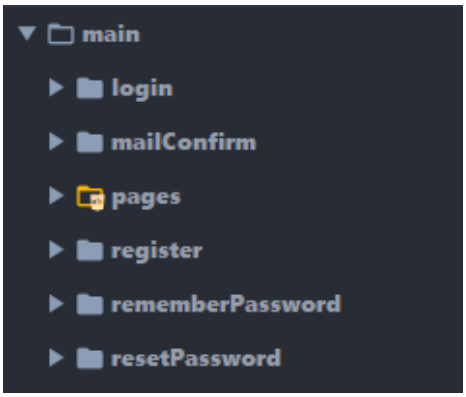
layout1 Folder zawierający układu strony,czyli toolbaru, navbaru, footera, oraz leftside strony.

shared-components Folder zawierający reużywalne komponenty na stronie.

FuseLayoutConfig.js Plik zawierający konfigurację layoutu.

FuseLayout,js Plik zawierający import layoutu.

3.1.4 Main



Rysunek 19: Katalog Main

Folder zawierający komponent oraz strony aplikacji.

Login Folder zawierający komponenty i konfigurację związane z logowaniem.

Tabs Folder zawierający komponenty do logowania USOS i zwykłego logowania.

Login.js Plik tworzący cały komponent logowania.

LoginConfig.js Plik konfiguracyjny logowania.

MailConfirm Folder zawierający komponent wyświetlający się po pomyślnej rejestracji.

MailConfirm.js Komponent odpowiedzialny za wyświetlenie pomyślnej rejestracji.

MailConfirmConfig.js Plik konfiguracyjny potwierdzającego pomyślną rejestracje.

Pages Folder zawierający strony naszej aplikacji - Calendar, Home, InstructionLoader, KnowledgeBase, NotFound, UserProfile, UserReservation oraz plik pagesConfig.js odpowiadający za przekazanie konfiguracji poszczególnej strony do serwisu, który następnie je wykorzystuje.

Calendar Folder zawierający główną funkcjonalność - kalendarz. Podzielony jest on na cztery moduły. Nagłówek, środek kalendarza, okna pomocnicze służące do wyboru godzin rezerwacji, podglądu dokonanej rezerwacji oraz tryb pokazujący harmonogram całego dnia. Dodatkowo posiada on swój własny system zarządzania danych przy użyciu reduxa, który umieszczony jest w folderze store oraz plik z pomocniczymi funkcjami - utils.js.

Home Folder zawierający widok strony głównej. Jest to pierwsza strona pokazywana po zalogowaniu użytkownika. Sam katalog zawiera w sobie jeszcze pomniejsze komponenty sektory. Każdy komponent sektoru jest odpowiedzialny za pobranie z bazy danych informacji nt. pojedynczego obszaru boiska i wyświetleniu jej na stronie. Zawiera on w sobie nazwa sektoru, długość rezerwacji, dostępność w poszczególne dni oraz przycisk przekierowujący na kalendarz.

InstructionLoader Folder zawierający okienko informacyjne, które pokazuje się jedynie przy pierwszym logowaniu każdego użytkownika do naszego systemu. Zawiera on w sobie przyciski przekierowujące na stronę główną lub instrukcję.

KnowledgeBase Folder zawierający stronę z instrukacji. Można do niego wejść przy pomocy przycisku na dolnym toolbarze lub poprzez przekierowanie z okienka informacyjnego opisanego powyżej. Sama instrukcja zawiera wysuwane komponenty, które opisują każdy szczegół naszej aplikacji klienckiej.

Not Found Folder zawierający pomocniczą stronę ukazującą się gdy użytkownik znajdzie się na ścieżce nie obsługującej przez nasz serwer. Posiada on specjalną konfigurację rutera dzięki czemu przy każdym przejściu na nie właściwą ścieżke się nam pojawi.

UserProfile Folder zawierający stronę z panelem użytkownika. Ten komponent pobiera z bazy wszystkie potrzebne informację nt. zalogowanego użytkownika i wyświetla ja w poszczególnych kontrolkach. Mamy tutaj podział na użytkowników zalogowanych przez USOS lub poprzedzonych rejestracją na naszej aplikacji. Różni się to tym, że Ci z USOS nie mają możliwości zmiany swojego imienia, nazwiska oraz maila. Natomiast zwykli użytkownicy mają pełnię możliwiści. Zachowując oczywiście opracowane przez nas walidacje, które również są opisane w tym folderze.

UserReservation Folder zawierający stronę z historią rezerwacji. Pobiera ona z bazy wszystko zrealizowane oraz opłacone już rezerwacje. Są one wyświetlane przy pomocy tabeli.

Register Folder zawierający komponenty używane do rejestracji.

Tabs Folder zawierający komponenty rejestracji.

Register.js Plik zawierający komponent, który składa komponent rejestracji.

RegisterConfig.js Plik konfiguracyjny rejestracji.

RememberPassword Folder zawierający komponent do przypominania hasła.

Tabs Folder zawierający widok przypominania hasła

RememberPassword.js Komponent składający przypominania hasła.

RememberPasswordConfig.js Plik konfiguracyjny przypominania hasła.

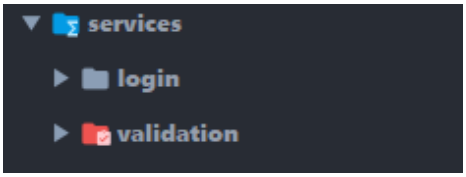
ResetPassword Folder zawierający komponent do resetowania hasła.

Tabs Folder zawierający widok resetu hasła.

ResetPassword.js Komponent składający resetowanie hasła.

ResetPasswordConfig.js Plik konfiguracyjny resetowania hasła.

3.2 Services



Rysunek 20: Katalog Services

Folder zawierający serwisy związane z logowaniem oraz walidacją.

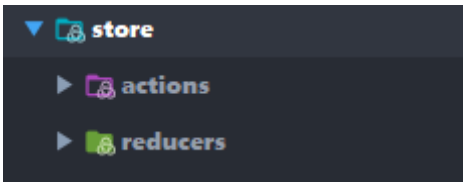
3.2.1 Login

Folder zawiera plik authService.js, w którym znajdują się wszystkie metody potrzebne do logowania oraz plik index.js, który importuje wszystkie metody z authService.js.

3.2.2 Validation

Folder zawierający, pliki które posiadają domyślne dane do walidacji oraz schematy walidacji.

3.3 Store



Rysunek 21: Katalog Store

Folder zawierający podstawowy reducer aplikacji.

3.3.1 Fuse

Folder zawierający metody do zarządzania powiadomieniami, nawigacją, ustawieniami, modalami lub wiadomościami.

3.3.2 i18nSlice.js

Metody zmieniające słowa z języka na język.

3.3.3 index.js

Plik konfiguracyjny store.

3.3.4 rootReducer.js

Plik zestawiający wszystkie reducery w jeden obiekt.

3.3.5 withReducer.js

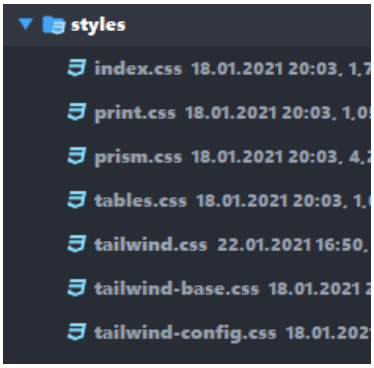
Reducer, który możesz wykorzystać w komponencie.

3.4 App.js

Plik zawierający konfigurację aplikacji.

3.5 Styles

W katalogu Styles mieszczą się pliki, z danymi stylami



Rysunek 22: Katalog styles

3.5.1 index.css

Style odpowiedzialne za początkowe ustawienie wartości dla poszczególnych znaczników HTML.

3.5.2 print.css

Style odpowiedzialne za ustawienie niektórych wartości CSS w komponentach.

3.5.3 tables.css

Style odpowiedzialne za ostylowanie tabeli na stronie.

3.5.4 tailwind.css

Wszystkie style z tailwinda.

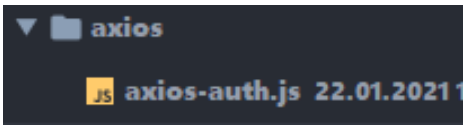
3.5.5 tailwind-base.css

Style odpowiedzialne za podstawowe wartości tailwinda.

3.5.6 tailwind-config.css

Podstawowa konfiguracja tailwinda.

3.6 Axios



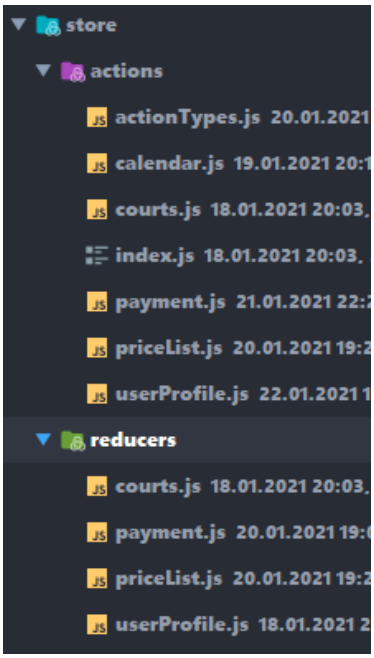
Rysunek 23: Katalog axios

Folder zawierający konfigurację Axiosa

3.6.1 axios-auth.js

Konfiguracja podstawowa instancji wysyłań zapytań.

3.7 Store



Rysunek 24: Katalog store

3.7.1 Actions

Folder zawierający akcję aplikacji.

actionTypes.js Plik zawierający nazwy akcji.

calendar.js Plik zawierający wszystkie akcje zawarte w kalendarzu.

courts.js Plik zawierający wszystkie akcje związane z pobieraniem danych dla boiska.

index.js Plik zawierający wszystkie akcję importowane w jednym pliku.

payment.js Plik zawierający wszystkie akcję związane z płatnością.

priceList.js Plik zawierający wszystkie akcję związane z cennikiem.

userProfile.js Plik zawierający wszystkie akcję związane z profilem użytkownika.

3.7.2 Reducers

Folder zawierający metody zmieniające state zależnie od wywołanej akcji.

courts.js Metoda zmieniająca state zależnie od akcji, związana z boiskiem.

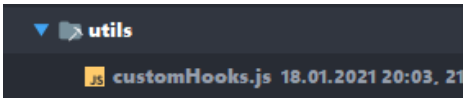
payment.js Metoda zmieniająca state zależnie od akcji, związanej z płatnością.

priceList.js Metoda zmieniająca state zależnie od akcji, związanej z cennikiem.

userProfile.js Metoda zmieniająca state zależnie od akcji, związanej z profilem użytkownika.

3.8 Utils

W katalogu Utils mieszczą się pliki z pomocniczymi funkcjami.



Rysunek 25: Katalog Utils

3.8.1 customHooks.js

Funkcja odpowiedzialna za wywołanie danej funkcji jeden raz.