МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г.Шухова)

Расчетно-графическое задание

дисциплина «Технологии web-программирования»

Выполнил: студент группы ВТ-41 Макаров Д.С.
Проверил: Картамышев С.В.

Белгород 2020

Содержание

Введение

Выбранная предметная область - система инвентаризации. Веб приложение должно обеспечивать функционал для добавления и управления хранилищами с секциями, хранилища могут принадлежать только одному пользователю. Так же должен присутствовать функционал по управлению хранимыми в хранилищах вещами.

Ход работы

2.1 Прототип

Перед созданием веб приложения был сверстан макет страниц в Figma. Далее на основе макета были сверстаны все html страницы будущего приложения. CSS фреймворки с готовыми компонентами не использовались.

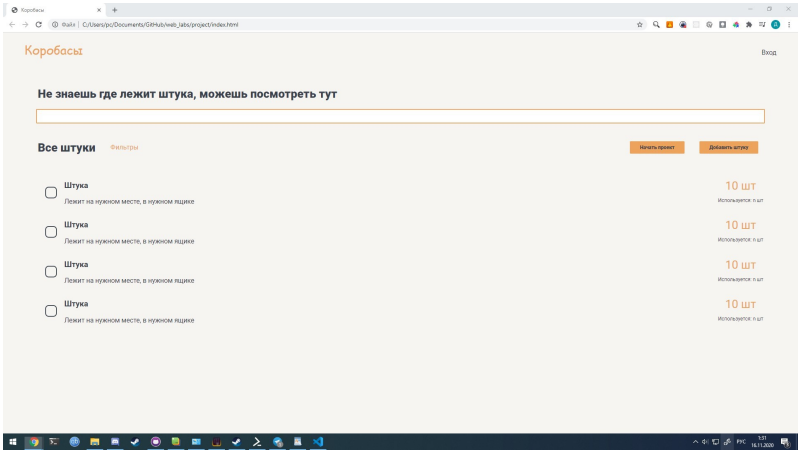

Рис. 1: Прототип главной страницы

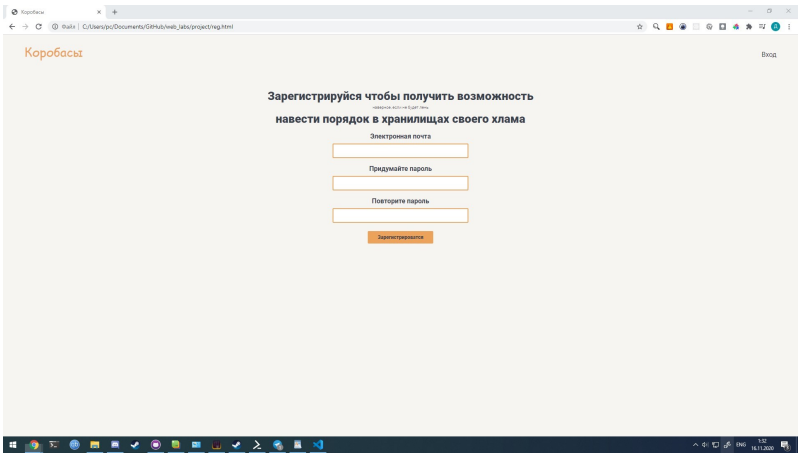

Рис. 2: Прототип страницы регистрации

2.2 Проектирование базы данных

В качестве СУБД использовался PostgreSQL, с ORM входящим в состав Python фреймворка Django.

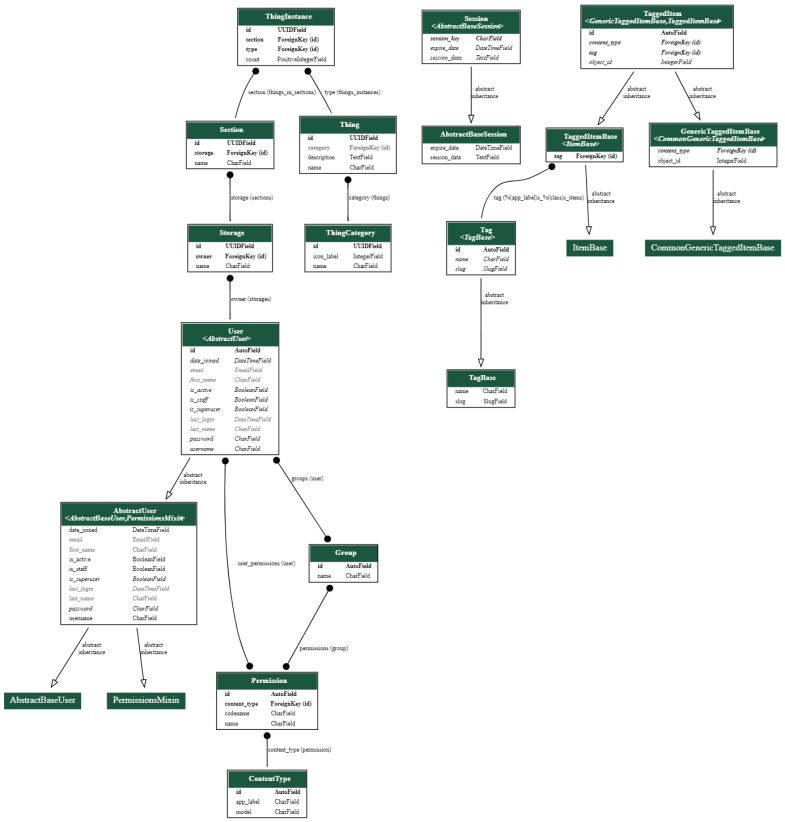

Рис. 3: Прототип страницы регистрации

Таблицы Section, Storage, Thing, ThingInstance, ThingCategory был описаны в соответствующих моделях (см. в приложении).

Все модели связанные с пользователями и их разрешениями, предоставлены фреймворком.

Таблицы связанные с тегами реализованы в сторонней библиотеке, которая реализует систему тегов для Django - django-taggit.

2.3 Разработка клиентского приложения

В качестве библиотеки для UI использовался - ReactJS, и так как эта библиотека предоставляет лишь функционал по отрисовке в DOM. Были использованы следующие дополнительные библиотеки:

- react-router: библиотека для работы с History API.
- react-router-dom: библиотека для декларативного роутинга в приложении.
- react-form-hook: библиотека реализующая формы ввода информации.
- react-js: библиотека предоставляющая UI компонент “всплывающее окно”
- axios: http клиент для взаимодействия с API.

Для переноса html прототипа на язык JSX, требовалось разделить пользовательский интерфейс на базовые компоненты. После разбиения было описано базовое содержимое страниц и получен макет приложения реализованный на JS.

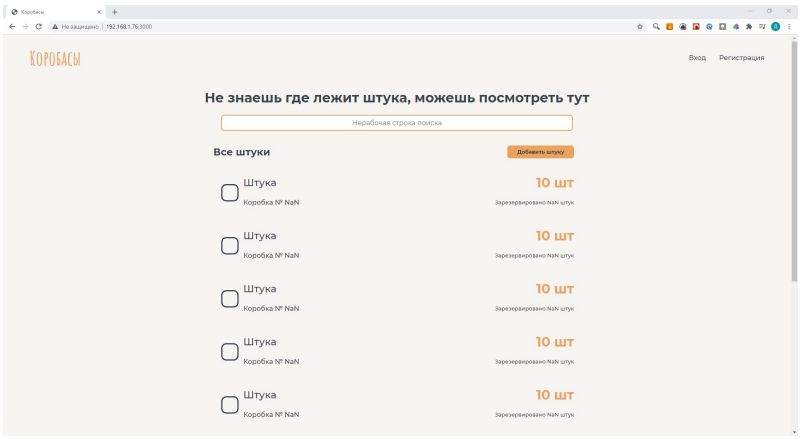

Рис. 4: Главная страница

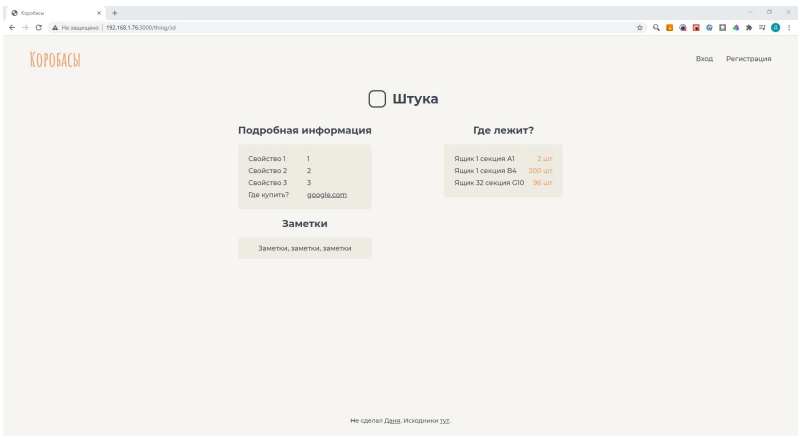

Рис. 5: Страница с описание хранимой вещи

2.4 Разработка серверной части приложения

Серверная часть проекта была реализована на языке Python с использованием фреймворка Django, а так же библиотеки django-rest-framework для реализации REST API.

Для каждой модели, был описан сериализатор и набор представлений для каждого необходимого метода HTTP запроса, который был связан с URL.

Все наборы представлений были ограничены в доступе только для владельцев сущностей (хранилище может просматривать только владелец хранилища).

Для реализации авторизации с серверной части была использована библиотека django-simple-jwt, которая предоставляет уже настроенное API для JWT авторизации при помощи REST API и встроенных в Django моделей User.

Так же для упрощения разработки и документирования API был использован стандарт OpenAPI, сгенерировав схему API встроенными в DRF функциями и используя генератор документации Redoc, была получена документация на API.

2.5 Контейнеризация

Для упрощения процесса разворачивания проекта, был использован Docker и утилита docker-compose.

Была описана сеть содержащая 3 контейнера:

- postgresql: контейнер с СУБД
- backend: контейнер с серверной частью приложения
- frontend: контейнер для сборки и отладки клиентской части приложения

2.6 Связывание клиентской и серверной части

С клиентской части для обращения к серверному API использовался HTTP клиент axios, для сохранения refresh и access токенов использовался localStorage, так же была реализована обертка для получения access токена и его обновлении при истечении для непрерывной сессии пока refresh токен валиден.

Все функции для работы с API и авторизацией находятся в контексте всего приложения, что позволяет использовать текущее состояние авторизации во всех компонентах без проброса состояний через все дерево компонентов.

Тестирование API производилось при помощи утилиты Postman.

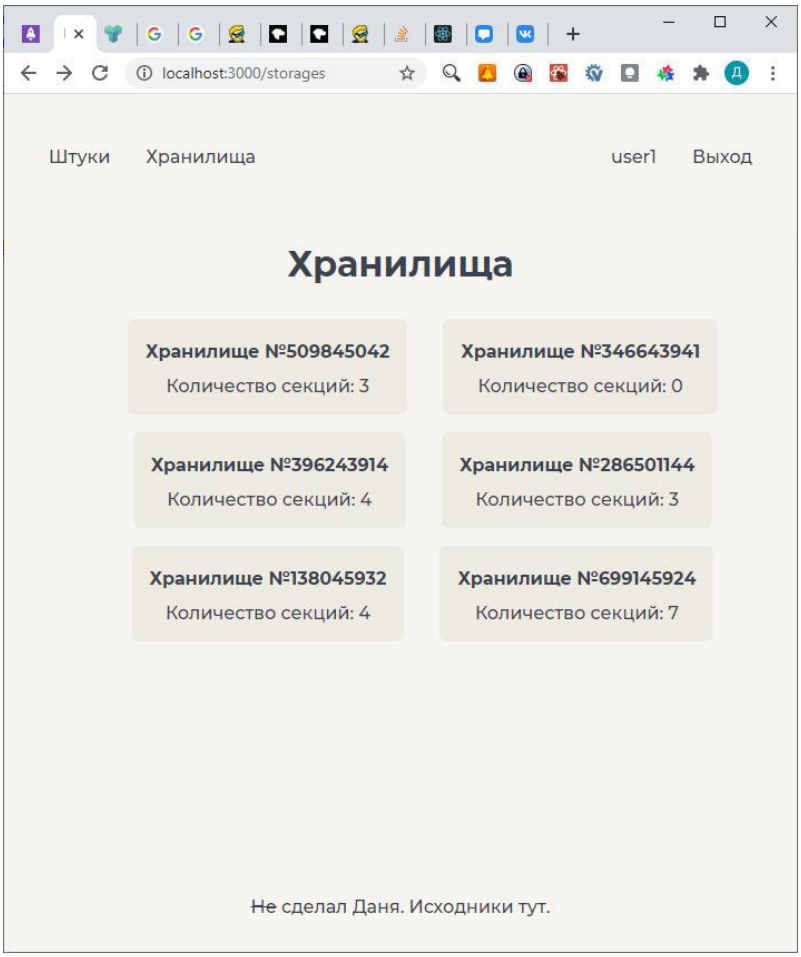

Рис. 6: Полученные клиентом данные с сервера

Вывод

В результате данной работы была реализована система по управлению хранилищами. В ходе выполнения работы были изучены библиотеки для создания веб приложений на языке JS, язык CSS, фреймворк Django и библиотека DRF для создания REST API, для взаимодействия с БД по средствам HTTP запросов. Также получены навыки работы с контейнерами Docker и утилитой docker-compose для одновременного развертывания сети контейнеров.

Приложение

Содержимое файла dockerCompose.yml

```
version: "3.8"

services:
  dev_backend:
    container_name: korobasy_dev_backend
    build:
      dockerfile: dev_backend.dockerfile
      context: .
    environment:
      - DEBUG=1
      - DJANGO_ALLOWED_HOSTS=localhost 192.168.1.76
      - PYTHONUNBUFFERED=1
      - SECRET_KEY=secret
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=korobasy
      - POSTGRES_PORT=5432
    command: sh ./run_django.sh
    restart: always
    depends_on:
      - dev_frontend
      - database
    volumes:
      - .:/app
    ports:
      - "8000:8000"

  dev_frontend:
    container_name: korobasy_dev_frontend
    build:
      dockerfile: dev_frontend.dockerfile
      context: .
    environment:
      - CI=true
    command: npm start
    restart: always
    volumes:
      - ./frontend/src:/frontend/src
      - ./frontend/public:/frontend/public
    ports:
      - "3000:3000"

  database:
    container_name: korobasy_dev_postgres
    image: postgres:13
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=korobasy
    ports:
      - "5432:5432"
```

Содержимое файла LoginForm.js

```
import React from 'react';
import {Redirect} from 'react-router-dom'
```

```
import {useForm} from "react-hook-form";
import {useAuth} from "../../useAuth";
import {Spinner} from "../Spinner/Spinner";

export default function LoginForm() {
    const {register, errors, handleSubmit} = useForm();
    const auth = useAuth();

    const onSubmit = data => {
        auth.setLoginStatus('loginNone');
        auth.login(data.username, data.password);
    }

    const formBody = <form onSubmit={handleSubmit(onSubmit)}>
        {errors.username && <p className='form-error'>{errors.username.message}</p>}
        {errors.password && <p className='form-error'>{errors.password.message}</p>}
        <div className="form-input">
            <p>Имя пользователя</p>
            <input
                name='username'
                autoComplete='username'
                ref={register({
                    required: "Без имени не получится."
                })}
            />
            <p>Пароль</p>
            <input
                name='password'
                type="password"
                autoComplete='current-password'
                ref={register({
                    required: "Пароль нужно ввести."
                })}
            />
        </div>
        <button type="submit">Войти</button>
    </form>;

    switch (auth.loginStatus) {
        case('loginWait'): {
            return (
                <Spinner spinnerSize='medium'/>
            );
        }
        case('loginError'): {
            return (
                <>
                    <p className='form-error'>Неверный логин или пароль</p>
                    {formBody}
                </>
            );
        }
        case('loginNone'): {
            return <>{formBody}</>;
        }
        case('loginSuccess'): {
            return <Redirect to='/'/>
        }
    }
}
```

Содержимое файла serializersStorages.py

```
from rest_framework import serializers
from storages.models import Storage,Section
from things.models import ThingInstance
from taggit_serializer.serializers import (TagListSerializerField,
                                           TaggitSerializer)

class StorageSerializer(TaggitSerializer,serializers.ModelSerializer,):
    tags = TagListSerializerField()
    sections = serializers.PrimaryKeyRelatedField(many=True, queryset=Section.objects.all())
    owner = serializers.ReadOnlyField(source='owner.username')
    class Meta:
        model = Storage
        fields = ('id', 'name','owner','sections','tags')

class SectionSerializer(serializers.ModelSerializer):
    things_in_sections =
    ↪  serializers.PrimaryKeyRelatedField(many=True,queryset=ThingInstance.objects.all())
    class Meta:
        model = Section
        fields = ('id', 'name','storage','things_in_sections')
```

Содержимое файла serializersThings.py

```
from rest_framework import serializers
from things.models import Thing,ThingInstance,ThingCategory
from taggit_serializer.serializers import (TagListSerializerField,
                                           TaggitSerializer)

class ThingSerializer(TaggitSerializer, serializers.ModelSerializer):
    tags = TagListSerializerField()
    things_instances =
    ↪  serializers.PrimaryKeyRelatedField(many=True,queryset=ThingInstance.objects.all())
    class Meta:
        model = Thing
        fields = ('id', 'name','description','things_instances','category','tags')

class ThingInstanceSerializer(serializers.ModelSerializer):
    class Meta:
        model = ThingInstance
        fields = ('id', 'section','type','count')

class ThingCategorySerializer(serializers.ModelSerializer):
    things = serializers.PrimaryKeyRelatedField(many=True,queryset=Thing.objects.all())
    class Meta:
        model = ThingCategory
        fields = ('id', 'name','icon_label','things')
```

Содержимое файла storagesModels.py

```
from uuid import uuid4
from django.db import models
from taggit.managers import TaggableManager

class Storage(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid4, editable=False)
    name = models.CharField(max_length=128)
    owner = models.ForeignKey('auth.User',related_name='storages',on_delete=models.CASCADE)
    tags = TaggableManager()
```

```
    def __str__(self):
        return str(id)

class Section(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid4, editable=False)
    name = models.CharField(max_length=128)
    storage = models.ForeignKey(
        'Storage',
        related_name='sections',
        on_delete=models.CASCADE
    )

    def __str__(self):
        return str(id)
```

Содержимое файла thingsModels.py

```
from uuid import uuid4
from django.db import models
from taggit.managers import TaggableManager

class Thing(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid4, editable=False)
    name = models.CharField(max_length=100)
    description = models.TextField()
    category = models.ForeignKey(
        'ThingCategory',
        related_name='things',
        blank=True,
        null=True,
        on_delete=models.CASCADE
    )

    tags = TaggableManager()
    def __str__(self):
        return str(id)

class ThingInstance(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid4, editable=False)
    section = models.ForeignKey(
        'storages.Section',
        related_name='things_in_sections',
        on_delete=models.CASCADE
    )
    type = models.ForeignKey(
        'Thing',
        related_name='things_instances',
        on_delete=models.CASCADE
    )
    count = models.PositiveIntegerField()

    def __str__(self):
        return str(id)

class ThingCategory(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid4, editable=False)
    name = models.CharField(max_length=100)
    icon_label = models.IntegerField()

    def __str__(self):
        return str(id)
```

Содержимое файла urls.py

```
from django.conf.urls import url, include
from django.urls import path
from storages import views as storages_views
from users import views as users_views
from things import views as things_views
from rest_framework.routers import DefaultRouter
from rest_framework.urlpatterns import format_suffix_patterns
from drf_spectacular.views import SpectacularAPIView, SpectacularRedocView,
↪  SpectacularSwaggerView

router = DefaultRouter()
router.register(r'storages', storages_views.StorageViewset)
router.register(r'sections', storages_views.SectionViewset)
router.register(r'things', things_views.ThingViewset)
router.register(r'instances', things_views.ThingInstanceViewset)
router.register(r'category', things_views.ThingCategoryViewset)
router.register(r'users', users_views.UserViewset)

urlpatterns = [
    url(r'^api/', include(router.urls)),
    url(r'^auth/',include('djoser.urls')),
    url(r'^auth/',include('djoser.urls.jwt')),
    path('api/schema/', SpectacularAPIView.as_view(), name='schema'),
    path('api/schema/redoc/', SpectacularRedocView.as_view(url_name='schema'), name='redoc'),
]
```

Содержимое файла useApi.js

```
import React, {createContext, useContext} from "react";
import baseUrl from './constants'
import {useAuth} from "./useAuth";
import axios from 'axios'

const apiContext = createContext();

export function ProvideApi({children}) {
    const auth = useProvideApi();
    return <apiContext.Provider value={auth}>{children}</apiContext.Provider>;
}

export const useApi = () => {
    return useContext(apiContext);
};

function useProvideApi() {
    const auth = useAuth();

    const getThing = async (uuid) => {
        let accessToken = await auth.getAccessToken();
        try {
            const response = await axios({
                method: 'get',
                url: baseUrl + `/api/things/${uuid.toString()}`,
                headers: {
                    Authorization: `Bearer ${accessToken}`
                }
            });
            const data = await response.data;
```

```
        return data;
    } catch (e) {
        return null
    }
}

const getStorage = async (uuid) => {
    let accessToken = await auth.getAccessToken();
    try {
        const response = await axios({
            method: 'get',
            url: baseUrl + `/api/storages/${uuid.toString()}`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            }
        });
        const data = await response.data;
        return data;
    } catch (e) {
        return null
    }
}

const postStorage = async (name) => {
    let accessToken = await auth.getAccessToken();
    const username = auth.user.username
    try {
        const response = await axios({
            method: 'post',
            url: baseUrl + `/api/storages/`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            },
            data: {
               name: name,
               owner: username,
               sections: [],
               tags: []
            }
        });
    }catch (e){}
}

const patchStorageName = async (uuid,name) => {
    let accessToken = await auth.getAccessToken();
    try {
        const response = await axios({
            method: 'patch',
            url: baseUrl + `/api/storages/${uuid}/`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            },
            data: {
                name: name
            }
        });
    }catch (e){}
}

const deleteStorage = async (uuid) => {
```

```
    let accessToken = await auth.getAccessToken();
    try {
        const response = await axios({
            method: 'delete',
            url: baseUrl + `/api/storages/${uuid}/`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            },
        }).then(
            async ()=>{
                await auth.updateUser()
            }
        );
    }catch (e){}
}

const getSection = async (uuid) => {
    let accessToken = await auth.getAccessToken();
    try {
        const response = await axios({
            method: 'get',
            url: baseUrl + `/api/sections/${uuid.toString()}`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            }
        });
        const data = await response.data;
        return data;
    } catch (e) {
        return null
    }
}

const getInstance = async (uuid) => {
    let accessToken = await auth.getAccessToken();
    try {
        const response = await axios({
            method: 'get',
            url: baseUrl + `/api/instances/${uuid.toString()}`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            }
        });
        const data = await response.data;
        return data;
    } catch (e) {
        return null
    }
}

const getCategory = async (uuid) => {
    try {
        const response = await axios({
            method: 'get',
            url: baseUrl + `/api/category/${uuid.toString()}`,
            headers: {
                Authorization: `Bearer ${auth.accessToken.toString()}`
            }
        });
        return response.data;
```

```
    } catch (e) {
        return null
    }
}

const postThing = async (name,description) => {
    let accessToken = await auth.getAccessToken();
    try {
        const response = await axios({
            method: 'post',
            url: baseUrl + `/api/things/`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            },
            data: {
                category: '',
                name: name,
                description: description,
                tags: [],
                things_instances: []
            }
        });
    } catch (e) {
        return null
    }
}

const postInstance = async (count,thing,section) => {
    let accessToken = await auth.getAccessToken();
    try {
        const response = await axios({
            method: 'post',
            url: baseUrl + `/api/instance/`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            },
            data: {
                count: count,
                type: thing,
                section: section
            }
        });
    } catch (e) {
        return null
    }
}

const getThings = async () => {
    let accessToken = await auth.getAccessToken();
    try {
        const response = await axios({
            method: 'get',
            url: baseUrl + `/api/things/`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            }
        });
        const data = await response.data
        return data
    } catch (e) {
```

```
            return null
        }
    }


    return {
        getStorage,
        postStorage,
        patchStorageName,
        deleteStorage,
        getSection,
        getInstance,
        getThing,
        postThing,
        postInstance,
        getThings
    };
}
```

Содержимое файла useAuth.js

```
import React, {createContext, useContext, useState} from "react";
import baseUrl from './constants'
import axios from 'axios'

const authContext = createContext();

export function ProvideAuth({children}) {
    const auth = useProvideAuth();
    return <authContext.Provider value={auth}>{children}</authContext.Provider>;
}

export const useAuth = () => {
    return useContext(authContext);
};

function useProvideAuth() {
    const [user, setUser] = useState(null)

    const getAccessToken = async () => {
        const accessToken = window.localStorage.getItem('accessToken');
        if (verifyToken(accessToken)) return accessToken;
        else {
            const refreshToken = getRefreshToken();
            try {
                const response = await axios({
                    method: 'post',
                    url: baseUrl + '/auth/jwt/refresh',
                    data: {
                        refresh: refreshToken.toString()
                    }
                })
                const accessToken = await response.data.access;
                window.localStorage.setItem('accessToken', accessToken);
                return accessToken;
            } catch (e) {
                window.localStorage.removeItem('accessToken');
                setLoginStatus('loginNone');
                return null
            }
```

```
    }
}

const getRefreshToken = () => {
    let refreshToken = window.localStorage.getItem('refreshToken');
    if (verifyToken(refreshToken)) {
        return refreshToken;
    } else {
        window.localStorage.removeItem('refreshToken');
        return null
    }
}

const verifyToken = (token) => {
    try {
        const jwt_token = JSON.parse(atob(token.split('.')[1]))
        return ((Date.now() / 1000 | 0) < jwt_token.exp)
    } catch (e) {
        return false
    }
}

const login = async (username, password) => {
    try {
        setLoginStatus('loginWait');
        const response = await axios({
            method: 'post',
            url: baseUrl + '/auth/jwt/create',
            data: {
                username: username,
                password: password
            }
        })
        const data = await response.data;
        window.localStorage.setItem('accessToken', data.access);
        window.localStorage.setItem('refreshToken', data.refresh);
        setLoginStatus('loginSuccess');
        const result = await getUser();
        setUser(result);
    } catch (e) {
        console.log('loginError');
        setLoginStatus('loginError');

    }
};

const logout = () => {
    window.localStorage.removeItem('accessToken');
    window.localStorage.removeItem('refreshToken');
    setLoginStatus('loginNone');
};

const [loginStatus, setLoginStatus] = useState(
    verifyToken(getRefreshToken()) ? 'loginSuccess' : 'loginNone'
)


const getUserInfo = async (userId) => {
    const accessToken = await getAccessToken();
    try {
```

```
        const response = await axios({
            method: 'get',
            url: baseUrl + `/api/users/${userId.toString()}`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            }
        });
        return response.data;
    } catch (e) {
        return {}
    }
}

const getUser = async () => {
    let accessToken = await getAccessToken();
    try {
        const response = await axios({
            method: 'get',
            url: baseUrl + `/auth/users/me`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            }
        })
        const userId = await response.data.id;
        const userInfo = await getUserInfo(userId);
        return userInfo
    } catch (e) {
        return null
    }
    ;
};

const updateUser = async () => {
    let accessToken = await getAccessToken();
    try {
        const response = await axios({
            method: 'get',
            url: baseUrl + `/auth/users/me`,
            headers: {
                Authorization: `Bearer ${accessToken}`
            }
        })
        const userId = await response.data.id;
        const userInfo = await getUserInfo(userId);
        setUser(userInfo)
    } catch (e) {
        return null
    }
    ;
};


if (loginStatus === 'loginSuccess' && user == null) {
    getUser().then(
        (result) => {
            setUser(result)
        }
    )
}
;
```

```
    return {
        loginStatus,
        getAccessToken,
        setLoginStatus,
        updateUser,
        user,
        login,
        logout
    };
}
```

Содержимое файла viewsStorages.py

```
from storages.models import Storage,Section
from storages.serializers import StorageSerializer,SectionSerializer
from storages.permissions import IsOwner
from rest_framework.viewsets import ModelViewSet
from rest_framework.permissions import IsAuthenticated

class StorageViewset(ModelViewSet):
    queryset = Storage.objects.all()
    serializer_class = StorageSerializer

    permission_classes = (IsAuthenticated,IsOwner)

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)

class SectionViewset(ModelViewSet):
    queryset = Section.objects.all()
    serializer_class = SectionSerializer

    permission_classes = (IsAuthenticated,)
```

Содержимое файла viewsThings.py

```
from things.models import Thing,ThingInstance,ThingCategory
from things.serializers import ThingSerializer,ThingInstanceSerializer,ThingCategorySerializer
from rest_framework.viewsets import ModelViewSet
from rest_framework.permissions import IsAuthenticated

class ThingViewset(ModelViewSet):
    queryset = Thing.objects.all()
    serializer_class = ThingSerializer

    permission_classes = (IsAuthenticated,)


class ThingInstanceViewset(ModelViewSet):
    queryset = ThingInstance.objects.all()
    serializer_class = ThingInstanceSerializer

    permission_classes = (IsAuthenticated,)

class ThingCategoryViewset(ModelViewSet):
    queryset = ThingCategory.objects.all()
    serializer_class = ThingCategorySerializer

    permission_classes = (IsAuthenticated,)
```