**Приложение Д**

**Листинг кода**

Папка db (файл lines.js)

const sql = require("../models/db.js");

function isLineExists(line, line\_id) {

console.log("line", line)

console.log("line.line\_id", line.line\_id)

console.log("line\_id", line\_id)

return line.line\_id === line\_id;

}

const getAll = async (res) => {

sql.query(`SELECT \*

FROM metro\_line M

JOIN stations S ON M.line\_id = S.line\_id;

`, (err, data) => {

if (err) {

console.log("error: ", err);

res.status(400).json({

status: "failed",

message: err.message

});

return;

}

let sorted = []

data.map((row) => {

const foundLine = sorted.find(({ line\_id }) => line\_id === row.line\_id)

if (foundLine) {

console.log("foundLine", foundLine);

const station = {

station\_id: row.station\_id,

station\_name: row.station\_name,

station\_description: row.station\_description,

station\_picture: row.station\_picture

}

foundLine.stations.push(station)

} else {

const line = {

line\_id: row.line\_id,

line\_name: row.line\_name,

line\_description: row.line\_description,

line\_picture: row.line\_picture,

stations: [

{

station\_id: row.station\_id,

station\_name: row.station\_name,

station\_description: row.station\_description,

station\_picture: row.station\_picture

}

]

}

sorted.push(line)

}

})

res.status(200).json({

status: "success",

data: sorted

});

});

module.exports = {

getAll,

};

Папка db (файл localization.js)

const sql = require("../models/db.js");

const sendErr = (res, err) => {

res.status(400).json({

status: "failed",

message: err.message

});

return;

}

const getAll = async (res) => {

sql.query(`SELECT \* FROM translations;

`, (err, data) => {

if (err) {

sendErr(res, err)

}

const result = {

en: {

},

ru: {

},

bel: {

}

}

data.map((row) => {

console.log("--getAll localiz")

const category = row.category

const isCategoryExist = result.en.hasOwnProperty(category)

if(!isCategoryExist) {

result.en[category] = {}

result.ru[category] = {}

result.bel[category] = {}

addCategory(result, row)

}

// else {

// addCategory(result, row)

// }

}

res.status(200).json({

status: "success",

data: result

});

});

};

const addCategory = (result, row) => {

result.en[row.category] = row.en

result.ru[row.category] = row.ru

result.bel[row.category] = row.bel

}

const updateLocalisation = async (locale, value, category, res) => {

const queryRow = `UPDATE translations

SET ${locale} = '${value}'

WHERE category = '${category}'`

console.log("queryRow", queryRow)

sql.query(queryRow, (err, data) => {

if (err) {

sendErr(res, err)

}

console.log("data", data)

res.status(200).json({

status: "success",

});

});

}

module.exports = {

getAll,

updateLocalisation,

};

Папка db (файл station.js)

const sql = require("../models/db.js");

const uploadImage = async (res) => {

sql.query(`SELECT \*

FROM metro\_line M

JOIN stations S ON M.line\_id = S.line\_id;

`, (err, data) => {

if (err) {

console.log("error: ", err);

res.status(400).json({

status: "failed",

message: err.message

});

return;

}

res.status(200).json({

status: "success",

data: data

});

});

};

module.exports = {

uploadImage,

};

Папка db (файл users.js)

const bcrypt = require("bcrypt");

const jwt = require("jsonwebtoken");

const sql = require("../models/db.js");

const { generateAccessToken, generateRefreshToken } = require("../helpers/auth");

const getMultiple = async (res) => {

sql.query(`SELECT id, name, password FROM users`, (err, data) => {

if (err) {

console.log("error: ", err);

res.status(400).json({

status: "failed",

message: err.message

});

return;

}

res.status(200).json({

status: "success",

data: data

});

});

};

const createUser = async (user, res) => {

sql.query(

`INSERT INTO users (name, password) values ('${user.name}', '${user.password}')`,

(err, data) => {

if (err) {

console.log("error: ", err);

res.status(400).json({

status: "failed",

message: err.message

});

return;

}

res.status(200).json({

status: "success",

});

}

);

};

const loginUser = async (user, res) => {

sql.query(`SELECT \* FROM users WHERE name = '${user.name}'`, (err, data) => {

if (err) {

console.log("error: ", err);

res.status(400).json({

status: "failed",

message: err.message

});

return;

}

if (data.length) {

const passwordIsValid = bcrypt.compareSync(user.password, data[0].password);

if (passwordIsValid) {

const userFromDB = {

id: data[0].id,

name: data[0].name,

role: data[0].role,

}

const accessToken = generateAccessToken(userFromDB)

const refreshToken = generateRefreshToken(userFromDB)

sql.query(

`INSERT INTO refreshtokens (refreshtoken) values ('${refreshToken}')`,

(err, data) => {

if (err) {

console.log("error: ", err);

res.status(400).json({

status: "failed",

message: "Error while writing refreshtoken to DB"

});

return;

}

res.status(200).json({ accessToken: accessToken, refreshToken: refreshToken })

}

);

} else {

res.status(400).json({

status: "failed",

message: "Invalid password"

});

}

}

});

};

const getRefreshToken = async (refreshtoken,res) => {

sql.query(`SELECT \* FROM refreshtokens WHERE refreshtoken = '${refreshtoken}'`, (err, data) => {

if (err) {

console.log("error: ", err);

res.status(403).json({

status: "failed",

message: err.message

});

return;

}

console.log("data", data[0].refreshtoken);

jwt.verify(data[0].refreshtoken, process.env.REFRESH\_TOKEN, (err, user) => {

if (err) return res.sendStatus(403);

const accessToken = generateAccessToken({ name: user.name })

res.status(200).json({ accessToken: accessToken })

})

});

};

const deleteRefreshToken = async (refreshtoken, res) => {

sql.query(`DELETE FROM refreshtokens WHERE refreshtoken = '${refreshtoken}'`, (err, data) => {

if (err) {

console.log("error: ", err);

res.status(403).json({

status: "failed",

message: err.message

});

return;

}

res.status(204).json({ status: "failed", })

});

};

module.exports = {

getMultiple,

createUser,

loginUser,

getRefreshToken,

deleteRefreshToken,

};

Папка helpers (файл auth.js)

const jwt = require("jsonwebtoken");

const generateAccessToken = (user) => {

return jwt.sign(user, process.env.ACCESS\_TOKEN, { expiresIn: process.env.ACCESS\_TOKEN\_LIFETIME });

}

const generateRefreshToken = (user) => {

return jwt.sign(user, process.env.REFRESH\_TOKEN);

}

module.exports = {

generateAccessToken,

generateRefreshToken,

};

Папка helpers (файл sendErr.js)

const sendErr = (res, err) => {

res.status(400).json({

status: "failed",

message: err.message

});

return;

}

module.exports = {

sendErr

};

Папка middlewares (файл authenticateToken.js)

const jwt = require("jsonwebtoken");

const authenticateToken = (req, res, next) => {

const authHeader = req.headers['authorization'];

const token = authHeader && authHeader.split(' ')[1]

if(token == undefined) return res.sendStatus(401)

jwt.verify(token, process.env.ACCESS\_TOKEN, (err, user) => {

if(err) return res.sendStatus(403)

req.user = user

next(

}

}

const requireJsonContent = (request, response, next) => {

if (request.headers['content-type'] !== 'application/json') {

response.status(400).send('Server requires application/json')

} else {

next()

}

}

module.exports = {

authenticateToken,

requireJsonContent

}

Папка middlewares (файл uploadImage.js)

const path = require("path");

const multer = require("multer");

const imageFilter = (req, file, cb) => {

if (file.mimetype.startsWith("image")) {

cb(null, true);

} else {

cb({message: "Please upload only images."}, false);

}

};

const storage = multer.diskStorage({

destination: (req, file, cb) => {

console.log("req", req)

console.log("file", file)

cb(null, path.join(\_\_dirname, "../public/images"));

},

filename: (req, file, cb) => {

console.log("file", file);

cb(null, Date.now() + file.originalname);

}

})

const uploadImageMiddleware = multer({storage: storage, fileFilter: imageFilter})

module.exports = {

uploadImageMiddleware,

storage: storage,

}

Папка models (файл config.js)

module.exports = {

HOST: "localhost",

USER: "root",

PASSWORD: "805@gavrosh",

DB: "metro",

};

Папка models (файл db.js)

const mysql = require("mysql");

const dbConfig = require("./config.js");

const connection = mysql.createConnection({

host: dbConfig.HOST,

user: dbConfig.USER,

password: dbConfig.PASSWORD,

database: dbConfig.DB,

})

connection.connect((error) => {

if (error) throw error;

console.log("Успешное подключение к базе данных");

});

module.exports = connection;

Папка routes (файл index.js)

var express = require("express");

var router = express.Router();

router.get("/", function (req, res, next) {

res.render("index", { title: "Express" });

});

router.post("/image", async (req, res) => {

try {

uploadSingleImage(req, res, function (err) {

console.log("req", req);

console.log(req.file);

const { entiyName, entiy } = req.body;

console.log("entiyName", entiyName)

if (!entiyName) {

return res.send({ error: `You must provide line name.` });

}

if (err) {

console.log("error", err);

return res.status(400).send({ error: err.message })

}

if (req.file == undefined) {

return res.send({ error: `You must select a file.` });

}

const file = req.file;

let queryRow

if (entiy === "line") {

queryRow = `UPDATE metro\_line

SET line\_picture = '${file.filename}'

WHERE line\_name = '${entiyName}'`

} else {

queryRow = `UPDATE stations

SET station\_picture = '${file.filename}'

WHERE station\_name = '${entiyName}'`

}

console.log("queryRow", queryRow)

sql.query(queryRow, (err, data) => {

console.log("err", err)

if (err) {

sendErr(res, err)

}

res.status(200).send({

filename: file.filename,

mimetype: file.mimetype,

originalname: file.originalname,

size: file.size,

fieldname: file.fieldname

})

});

})

} catch (err) {

return res.status(400).send({ error: err.message })

}

});

module.exports = router;

Папка routes (файл line.js)

const express = require("express");

const router = express.Router();

const multipart = require("connect-multiparty");

const { getAll } = require("../db/lines.js");

const multipartMiddleware = multipart();

router.get("/all", async (req, res) => {

try {

await getAll(res);

} catch (err) {

console.log("err", err);

}

});

module.exports = router;

Папка routes (файл localization.js)

const express = require("express");

const router = express.Router();

const { authenticateToken } = require("../middlewares/authenticateToken");

const bcrypt = require("bcrypt");

const multipart = require("connect-multiparty");

const { getAll, updateLocalisation } = require("../db/localization.js");

const { signupValidation } = require("../helpers/validation.js");

const { getRefreshToken, deleteRefreshToken } = require("../db/users");

const multipartMiddleware = multipart();

router.get("/all", async (req, res) => {

try {

console.log("user", req.user);

await getAll(res);

} catch (err) {

console.log("err", err);

}

});

router.put("/", multipartMiddleware, async (req, res) => {

try {

if (!req.body) return res.sendStatus(400);

console.log("req.body", req.body)

const { locale, value, category } = req.body

await updateLocalisation(locale, value, category, res);

} catch (err) {

console.log("err", err);

}

});

module.exports = router;

Папка routes (файл station.js)

const express = require("express");

const router = express.Router();

const path = require("path");

const sql = require("../models/db.js");

const multipart = require("connect-multiparty");

const { uploadImageMiddleware } = require("../middlewares/uploadImage.js");

const { sendErr } = require("../helpers/sendErr");

const multipartMiddleware = multipart();

// upload.array('images', 12)

const uploadSingleImage = uploadImageMiddleware.single('image');

router.post("/image", async (req, res) => {

try {

uploadSingleImage(req, res, function (err) {

const { lineName } = req.body;

if (!lineName) {

return res.send({ error: `You must provide line name.` });

}

if (err) {

console.log("error", err);

return res.status(400).send({ error: err.message })

}

if (req.file == undefined) {

return res.send({ error: `You must select a file.` });

}

const file = req.file;

const queryRow = `UPDATE metro\_line

SET line\_picture = '${file.filename}'

WHERE line\_name = '${lineName}'`

console.log("queryRow", queryRow)

sql.query(queryRow, (err, data) => {

console.log("err", err)

if (err) {

sendErr(res, err)

}

console.log("data", data)

res.status(200).send({

filename: file.filename,

mimetype: file.mimetype,

originalname: file.originalname,

size: file.size,

fieldname: file.fieldname

})

});

})

} catch (err) {

return res.status(400).send({ error: err.message })

}

});

router.get("/image/:imageName", async (req, res) => {

const { imageName } = req.params

console.log("imageName", imageName)

const imagePath = path.join(\_\_dirname, `../public/images/${imageName}`)

res.sendFile(imagePath);

});

router.post("/", async (req, res) => {

try {

if (!req.body) return res.sendStatus(400);

console.log("req.body0", req.body)

uploadSingleImage(req, res, function (err) {

console.log("req", req);

const {

stationNameDB,

stationDescriptionDB,

stationNameEN,

stationNameBY,

stationNameRU,

stationDescriptionEN,

stationDescriptionBY,

stationDescriptionRU,

lineId,

} = req.body;

if (!stationNameDB) {

return res.status(400).send({ error: `You must provide line name.` });

}

if (err) {

console.log("error", err);

return res.status(400).send({ error: err.message })

}

if (req.file == undefined) {

return res.send({ error: `You must select a file.` });

}

const file = req.file;

console.log("file", file)

const queryRow = `

INSERT INTO stations (line\_id, station\_name, station\_description, station\_picture)

VALUES (${lineId}, '${stationNameDB}', '${stationDescriptionDB}', '${file.filename}');

`

console.log("queryRow", queryRow)

const queryRow2 = `

INSERT INTO translations (category, en, ru, bel)

VALUES ('${stationNameDB}', '${stationNameEN}', '${stationNameRU}', '${stationNameBY}');

`

console.log("queryRow2", queryRow2)

const queryRow3 = `

INSERT INTO translations (category, en, ru, bel)

VALUES ('${stationDescriptionDB}', '${stationDescriptionEN}', '${stationDescriptionRU}', '${stationDescriptionBY}');

`

console.log("querqueryRow3yRow", queryRow3)

let error = null

sql.query(queryRow, (err, data) => {

if (err) {

console.log("err", err)

error = err

}

});

sql.query(queryRow2, (err, data) => {

if (err) {

error = err

}

});

sql.query(queryRow3, (err, data) => {

if (err) {

error = err

}

});

res.status(200).send({

stationNameDB,

stationDescriptionDB,

})

}

} catch (err) {

return res.status(400).send({ error: err.message })

}

});

router.delete("/:stationName/:stationDescription", multipartMiddleware, async (req, res) => {

try {

if (!req.params) return res.sendStatus(400);

const { stationName, stationDescription } = req.params

const deleteFromStations = `

DELETE FROM stations WHERE station\_name = '${stationName}';

`

const deleteFromTranslation = `

DELETE FROM translations WHERE category IN ('${stationName}', '${stationDescription}');

sql.query(deleteFromStations, (err, data) => {

console.log("err", err)

if (err) {

sendErr(res, err)

}

console.log("data", data)

sql.query(deleteFromTranslation, (err, data) => {

console.log("err", err)

if (err) {

sendErr(res, err)

}

console.log("data", data)

res.status(200).send({

status: "success"

})

});

});

} catch (err) {

return res.status(400).send({ error: err.message })

}

});

module.exports = router;

Папка routes (файл user.js)

const express = require("express");

const bcrypt = require("bcrypt");

const multipart = require("connect-multiparty");

const { getMultiple, createUser, loginUser } = require("../db/users.js");

const { signupValidation } = require("../helpers/validation.js");

const { authenticateToken } = require("../middlewares/authenticateToken");

const { getRefreshToken, deleteRefreshToken } = require("../db/users");

const router = express.Router();

const multipartMiddleware = multipart();

/\* GET users listing. \*/

router.get("/all", authenticateToken, async (req, res) => {

try {

console.log("user", req.user);

await getMultiple(res);

} catch (err) {

console.log("err", err);

}

});

//sing up

router.post("/", multipartMiddleware, signupValidation, async (req, res) => {

try {

if (!req.body) return res.sendStatus(400);

const salt = await bcrypt.genSalt(10);

const hashedPassword = await bcrypt.hash(req.body.password, salt);

const user = {

name: req.body.name,

password: hashedPassword,

};

await createUser(user, res);

} catch (err) {

console.log("err", err);

}

});

// логин

router.post("/login", multipartMiddleware, async (req, res) => {

try {

if (!req.body) return res.sendStatus(400);

const user = {

name: req.body.name,

password: req.body.password,

};

await loginUser(user, res);

} catch (err) {

console.log("err", err);

}

});

//Refresh token in request

router.post("/token", async (req, res) => {

const refreshToken = req.body.token

await getRefreshToken(refreshToken, res)

});

router.delete("/logout/:refreshToken", async (req, res) => {

await deleteRefreshToken(req.params.refreshToken, res)

});

module.exports = router;

Файл app.js

// подключение пакетов

const createError = require("http-errors");

const express = require("express");

const path = require("path");

const cookieParser = require("cookie-parser");

const logger = require("morgan");

const bodyParser = require("body-parser");

const indexRouter = require("./routes/index");

const userRouter = require("./routes/user");

const localizationRouter = require("./routes/localization");

const lineRouter = require("./routes/line");

const stationRouter = require("./routes/station");

const app = express();

const cors = require("cors");

const whitelist = ["http://localhost:3001", "http://localhost:3002"]

const corsOptions = {

origin: function (origin, callback) {

if (!origin || whitelist.indexOf(origin) !== -1) {

callback(null, true)

} else {

callback(new Error("Not allowed by CORS"))

}

},

credentials: true,

}

app.use(cors(corsOptions))

require("dotenv").config();

// view engine setup

app.set("views", path.join(\_\_dirname, "views"));

app.set("view engine", "pug");

app.use(bodyParser.json()); // to support JSON-encoded bodies

app.use(

bodyParser.urlencoded({

// to support URL-encoded bodies

extended: true,

})

);

app.use(logger("dev"));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, "public")));

app.use("/", indexRouter);

app.use("/user", userRouter);

app.use("/localization", localizationRouter);

app.use("/line", lineRouter);

app.use("/station", stationRouter);

// вызвать 404 оишбку в случае ошибки (catch 404 and forward to error handler)

app.use(function (req, res, next) {

next(createError(404));

});

// обработчик ошибок

app.use(function (err, req, res, next) {

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get("env") === "development" ? err : {};

// визуализировать страницу ошибки

res.status(err.status || 500);

res.render("error");

});

module.exports = app;

Папка store (файл lineSlice.js)

import { createSlice } from '@reduxjs/toolkit'

export const linesSlice = createSlice({

name: 'lines',

initialState: {

data: [],

currentLine: "home\_moscowTitle"

},

reducers: {

update: (state, action) => {

state.data = action.payload

},

updateCurrentLineImage: (state, action) => {

const { filename, lineName } = action.payload

const index = state.data.findIndex(line => line.line\_name == lineName);

const newArray = [...state.data];

newArray[index].line\_picture = filename

state.data = newArray

},

decrement: (state) => {

state.value -= 1

},

incrementByAmount: (state, action) => {

state.value += action.payload

},

},

})

function selectedLine(elem, currentLineName) {

return elem.line\_name === currentLineName;

}

export const selectLines = (state) => state.lines.data

export const selectLine = (state, currentLineName) => {

return state.lines.data.find((elem) => {

return elem.line\_name === currentLineName

})

}

export const selectCurrentLineName = (state) => state.lines.currentLine

// export const isUserLoggedIn = (state) => state.user.id !== ""

// Action creators are generated for each case reducer function

export const { update, updateLocalisationResouces, updateCurrentLineImage } = linesSlice.actions

export default linesSlice.reducer

Папка store (файл store.js)

import { configureStore } from '@reduxjs/toolkit'

import userReducer from './userSlice'

import localizationReducer from './localizationSlice'

import linesReducer from './linesSlice'

export default configureStore({

reducer: {

user: userReducer,

localization: localizationReducer,

lines: linesReducer

},

})

Папка store (файл useSlice.js)

import { createSlice } from '@reduxjs/toolkit'

export const userSlice = createSlice({

name: 'users',

initialState: {

id: '',

name: '',

role: ''

},

reducers: {

update: (state, action) => {

state.id = action.payload.id

state.name = action.payload.name

state.role = action.payload.role

},

decrement: (state) => {

state.value -= 1

},

incrementByAmount: (state, action) => {

state.value += action.payload

},

logout: (state) => {

state.id = ""

state.name = ""

state.role = ""

}

},

})

export const selectUserState = (state) => state.user

export const isUserLoggedIn = (state) => state.user.id !== ""

// Action creators are generated for each case reducer function

export const { increment, decrement, incrementByAmount, update, logout } = userSlice.actions

export default userSlice.reducer

Файл App js

import React, { Suspense } from "react";

import "./App.css";

import { BrowserRouter, Route, Routes } from "react-router-dom";

import { useSelector } from "react-redux";

import Sidebar from "./components/Sidebar";

import Home from "./pages/Home/Home.jsx";

import Map from "./pages/Map/Map.jsx";

import Media from "./pages/Media/Media.jsx";

import Resources from "./pages/Resources/Resources.jsx";

import Contacts from "./pages/Contacts/Contacts.jsx";

import Search from "./pages/Search/Search.jsx";

import Auth from "./pages/Auth/Auth.jsx";

import FisrtBranch from "./pages/FirstBranch/HomeFirstBranch/HomeFirstBranch.jsx";

import SecondBranch from "./pages/SecondBranch/HomeSecondBranch/HomeSecondBranch.jsx";

import ThirdBranch from "./pages/ThirdBranch/HomeThirdBranch/HomeThirdBranch.jsx";

import Admin from "./pages/Admin/Admin.jsx";

import Error from "./pages/Error/Error.jsx";

import Frunzenskaya from "./pages/SecondBranch/Frunzenskaya";

import Partizanskaya from "./pages/SecondBranch/Frunzenskaya";

import { TokenManager } from "./common/TokenManager.jsx";

import { LocalizationProvider } from "./common/LocalizationProvider.jsx";

import Line from "./pages/Line";

import { isUserLoggedIn, logout } from "./store/userSlice";

const App = () => {

const isLoggedIn = useSelector(isUserLoggedIn);

return (

<Suspense fallback="Loading...">

<BrowserRouter>

<TokenManager>

<LocalizationProvider>

<Sidebar>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/map" element={<Map />} />

<Route path="/resources" element={<Resources />} />

<Route path="/media" element={<Media />} />

<Route path="/contacts" element={<Contacts />} />

<Route path="/search" element={<Search />} />

<Route path="/auth" element={<Auth />} />

<Route path="/bluebranch" element={<FisrtBranch />} />

<Route path="/redbranch" element={<SecondBranch />} />

<Route path="/greenbranch" element={<ThirdBranch />} />

{isLoggedIn && <Route path="/admin" element={<Admin />} />

}

<Route path="/line/:name" element={<Line />} />

<Route path="\*" element={<Error />} />

</Routes>

</Sidebar>

</LocalizationProvider>

</TokenManager>

</BrowserRouter>

</Suspense>

);

};

export default App;

Файл YoutubeEmbed.js

import React from "react";

import PropTypes from "prop-types";

const YoutubeEmbed = ({ embedId }) => (

<div className="video-responsive">

<iframe

width="650"

height="350"

src={`https://www.youtube.com/embed/${embedId}`}

frameBorder="0"

allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture"

allowFullScreen

title="Как строят метро в Минске"

/>

</div>

);

YoutubeEmbed.propTypes = {

embedId: PropTypes.string.isRequired,

};

export default YoutubeEmbed;

Файл Sidebar.jsx

import React, { useState } from "react";

import { useSelector } from "react-redux";

import HomeIcon from "@mui/icons-material/HomeOutlined";

import ContactIcon from "@mui/icons-material/Info";

import SearchIcon from "@mui/icons-material/AccountTree";

import MediaIcon from "@mui/icons-material/MenuBook";

import MapIcon from "@mui/icons-material/MapOutlined";

import UsefulLink from "@mui/icons-material/AttachFileOutlined";

import UserIcon from "@mui/icons-material/AccountCircleOutlined";

import MenuIcon from "@mui/icons-material/Menu";

import { NavLink } from "react-router-dom";

import { useTranslation } from "react-i18next";

import { isUserLoggedIn } from "../store/userSlice";

const Sidebar = ({ children }) => {

const [t] = useTranslation();

const isLoggedIn = useSelector(isUserLoggedIn);

const routes = [

{

path: "/",

name: t("nav.home"),

icon: <HomeIcon />,

},

{

path: "/search",

name: t("nav.projects"),

icon: <SearchIcon />,

},

{

path: "/media",

name: t("nav.stage"),

icon: <MediaIcon />,

},

{

path: "/map",

name: t("nav.map"),

icon: <MapIcon />,

},

{

path: "/contacts",

name: t("nav.kompozition"),

icon: <ContactIcon />,

},

{

path: "/resources",

name: t("nav.info"),

icon: <UsefulLink />,

}

];

const privateRoutes = [

{

path: "/admin",

name: t("nav.panel"),

icon: <UserIcon />,

},

]

let resultRoutes = [];

console.log("isLoggedIn0", isLoggedIn)

if (isLoggedIn) {

resultRoutes = routes.concat(privateRoutes)

} else {

resultRoutes = routes

}

console.log("resultRoutes", resultRoutes)

const [isOpen, setIsOpen] = useState(false);

const toggle = () => setIsOpen(!isOpen);

return (

<div className="container">

<div className="sidebar" style={{ width: isOpen ? "200px" : "50px" }}>

<div style={{ marginLeft: isOpen ? "45%" : "33%" }} className="bars">

<MenuIcon onClick={toggle} />

</div>

{resultRoutes.map((item, index) => (

<NavLink

to={item.path}

key={index}

className="link"

activeclassName="active"

>

<div className="icon">{item.icon}</div>

<div

className="link\_text"

style={{ display: isOpen ? "block" : "none" }}

>

{item.name}

</div>

</NavLink>

))}

</div>

<main style={{ marginLeft: isOpen ? "200px" : "50px" }}>{children}</main>

</div>

);

};

export default Sidebar;

Файл HeaderOptions.jsx

import React from "react";

import Stack from "@mui/material/Stack";

import Button from "@mui/material/Button";

import { useNavigate } from "react-router-dom";

import i18n from "../i18n";

import { useSelector, useDispatch } from "react-redux";

import Axios from "axios";

import "./HeaderOptions.css";

import { useTranslation } from "react-i18next";

import { isUserLoggedIn, logout } from "../store/userSlice";

import { grey } from "@mui/material/colors";

import { styled } from "@mui/material/styles";

import { update } from "../store/localizationSlice";

const HeaderOptions = () => {

const [t] = useTranslation();

const WhiteBtn = styled(Button)(({ theme }) => ({

color: theme.palette.getContrastText(grey[50]),

backgroundColor: grey[50],

"&:hover": {

backgroundColor: grey[300],

},

}));

const dispatch = useDispatch();

const navigate = useNavigate();

const changeLanguage = (lng) => {

i18n.changeLanguage(lng);

dispatch(update(lng));

};

const isLoggedIn = useSelector(isUserLoggedIn);

console.log("isLoggedIn", isLoggedIn)

const onLogOut = () => {

const refreshToken = localStorage.getItem("refreshToken");

Axios.delete(`http://localhost:3000/user/logout/${refreshToken}`)

.then((res) => {

if (res.status === 204) {

localStorage.removeItem("token");

localStorage.removeItem("refreshToken");

navigate("/auth");

} else {

console.log("Something went wrong during logout", res);

}

})

.catch((err) => {

console.error("Something went wrong during logout", err);

});

console.log("dispatch logout")

dispatch(logout());

};

return (

<Stack className="input-lang" spacing={2} direction="row">

<WhiteBtn onClick={() => changeLanguage("en")} variant="contained">

ENG

</WhiteBtn>

<WhiteBtn onClick={() => changeLanguage("bel")} variant="contained">

{t("bel")}

</WhiteBtn>

<WhiteBtn

color="primary"

onClick={() => changeLanguage("ru")}

variant="contained"

>

{t("ru")}

</WhiteBtn>

{isLoggedIn ? (

<Button onClick={() => onLogOut()} variant="contained">

{t("logout")}

</Button>

) : (

<Button onClick={() => navigate("/auth")} variant="contained">

{t("login")}

</Button>

)}

</Stack>

);

};

export default HeaderOptions;

Файл ImageList

import React from "react";

import Stack from "@mui/material/Stack";

import Button from "@mui/material/Button";

import { useNavigate } from "react-router-dom";

import i18n from "../i18n";

import { useSelector, useDispatch } from "react-redux";

import Axios from "axios";

import "./HeaderOptions.css";

import { useTranslation } from "react-i18next";

import { isUserLoggedIn, logout } from "../store/userSlice";

import { grey } from "@mui/material/colors";

import { styled } from "@mui/material/styles";

import { update } from "../store/localizationSlice";

const HeaderOptions = () => {

const [t] = useTranslation();

const WhiteBtn = styled(Button)(({ theme }) => ({

color: theme.palette.getContrastText(grey[50]),

backgroundColor: grey[50],

"&:hover": {

backgroundColor: grey[300],

},

}));

const dispatch = useDispatch();

const navigate = useNavigate();

const changeLanguage = (lng) => {

i18n.changeLanguage(lng);

dispatch(update(lng));

};

const isLoggedIn = useSelector(isUserLoggedIn);

console.log("isLoggedIn", isLoggedIn)

const onLogOut = () => {

const refreshToken = localStorage.getItem("refreshToken");

Axios.delete(`http://localhost:3000/user/logout/${refreshToken}`)

.then((res) => {

if (res.status === 204) {

localStorage.removeItem("token");

localStorage.removeItem("refreshToken");

navigate("/auth");

} else {

console.log("Something went wrong during logout", res);

}

})

.catch((err) => {

console.error("Something went wrong during logout", err);

});

console.log("dispatch logout")

dispatch(logout());

};

return (

<Stack className="input-lang" spacing={2} direction="row">

<WhiteBtn onClick={() => changeLanguage("en")} variant="contained">

ENG

</WhiteBtn>

<WhiteBtn onClick={() => changeLanguage("bel")} variant="contained">

{t("bel")}

</WhiteBtn>

<WhiteBtn

color="primary"

onClick={() => changeLanguage("ru")}

variant="contained"

>

{t("ru")}

</WhiteBtn>

{isLoggedIn ? (

<Button onClick={() => onLogOut()} variant="contained">

{t("logout")}

</Button>

) : (

<Button onClick={() => navigate("/auth")} variant="contained">

{t("login")}

</Button>

)}

</Stack>

);

};

export default HeaderOptions;