***SignIn.tsx***

import \* as React from "react";

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

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

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

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

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

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

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

import LockOutlinedIcon from "@mui/icons-material/LockOutlined";

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

import { Route } from "components/routing";

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

import { NotificationManager } from "react-notifications";

import { auth } from "firebase\_config";

import { useSignInWithEmailAndPassword } from "react-firebase-hooks/auth";

import { useEffect, useRef, useState } from "react";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { successfulAuth } from "helpers/auth.helpers";

import { isMobile } from "react-device-detect";

export const SignIn = () => {

const navigate = useNavigate();

const formRef = useRef<HTMLFormElement>();

const getFormField = (name: string): string => {

const data = new FormData(formRef.current);

return data.get(name)?.toString() || "";

};

const [signInWithEmailAndPassword, , loading, error] =

useSignInWithEmailAndPassword(auth);

const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {

event.preventDefault();

const email = getFormField("email");

const password = getFormField("password");

const userCredential = await signInWithEmailAndPassword(email, password);

if (userCredential) {

await successfulAuth(userCredential, navigate);

}

};

useEffect(() => {

if (error) {

NotificationManager.error(error?.name, error?.message);

}

}, [error]);

const [isSignInDisabled, setIsSignInDisabled] = useState<boolean>(false);

const updateIsSignInDisabled = () => {

setIsSignInDisabled(!getFormField("email") || !getFormField("password"));

};

return (

<Grid container component="main" sx={{ height: "100vh" }}>

<Grid

item

xs={false}

sm={4}

md={7}

sx={{

backgroundImage: "url(/images/sign-in-background-min.jpg)",

backgroundRepeat: "no-repeat",

backgroundColor: (t) =>

t.palette.mode === "light"

? t.palette.grey[50]

: t.palette.grey[900],

backgroundSize: "cover",

backgroundPosition: "center",

}}

/>

<Grid

item

xs={12}

sm={8}

md={5}

component={Paper}

elevation={6}

square

sx={{

display: "flex",

alignItems: isMobile ? "center" : undefined,

}}

>

<Box

sx={{

marginTop: "10px",

marginLeft: "20px",

marginRight: "20px",

width: "100%",

display: "flex",

flexDirection: "column",

alignItems: "center",

}}

>

<Avatar sx={{ m: 1, bgcolor: "secondary.main" }}>

<LockOutlinedIcon />

</Avatar>

<Typography component="h1" variant="h5">

Вход

</Typography>

<Box

component="form"

noValidate

onSubmit={handleSubmit}

sx={{ mt: 1 }}

ref={formRef}

>

<TextField

margin="normal"

required

fullWidth

id="email"

label="Электронная почта"

name="email"

autoComplete="email"

autoFocus

onChange={updateIsSignInDisabled}

/>

<TextField

margin="normal"

required

fullWidth

name="password"

label="Пароль"

type="password"

id="password"

autoComplete="current-password"

onChange={updateIsSignInDisabled}

/>

{loading ? (

<LoadingSpinner />

) : (

<Button

disabled={isSignInDisabled}

type="submit"

fullWidth

variant="contained"

sx={{ mt: 3, mb: 2 }}

>

Войти

</Button>

)}

<Grid container justifyContent="flex-end">

<Grid item>

<Link href={Route.signUp} variant="body2">

{"Нет аккаунта? Зарегистрируйтесь"}

</Link>

</Grid>

</Grid>

</Box>

</Box>

</Grid>

</Grid>

);

};

***SignOut.tsx***

import React, { useCallback, useEffect } from "react";

import { auth } from "firebase\_config";

import { ACTION\_NAMES } from "store/reducers/user/constants";

import { NotificationManager } from "react-notifications";

import { Route } from "components/routing";

import { useAppDispatch } from "store";

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

import { streamClient } from "containers/App/App";

export const SignOut = () => {

const navigate = useNavigate();

const dispatch = useAppDispatch();

const onSignOut = useCallback(async () => {

try {

if (!auth.currentUser) {

navigate(Route.home);

return;

}

await auth.signOut();

if (streamClient.user?.online) {

streamClient.disconnectUser().catch((e) => {

console.log(e);

NotificationManager.error("Выход из сервиса чата", "Авторизация");

});

}

dispatch({ type: ACTION\_NAMES.userSignOut });

NotificationManager.success("Успешный выход из системы!");

navigate(Route.home);

} catch (e) {

console.log(e);

NotificationManager.error("Выход из системы", "Авторизация");

navigate(-1);

}

}, [dispatch, navigate]);

useEffect(() => {

onSignOut();

}, [onSignOut]);

return <Outlet />;

};

***SignUp.tsx***

import \* as React from "react";

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

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

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

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

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

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

import LockOutlinedIcon from "@mui/icons-material/LockOutlined";

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

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

import { Route } from "components/routing";

import { useCreateUserWithEmailAndPassword } from "react-firebase-hooks/auth";

import { auth, firebaseConfig } from "firebase\_config";

import { useEffect, useRef, useState } from "react";

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

import { NotificationManager } from "react-notifications";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { successfulAuth, successfulSignUp } from "helpers/auth.helpers";

import { isMobile } from "react-device-detect";

import {

FormControl,

InputLabel,

MenuItem,

Select,

SelectChangeEvent,

} from "@mui/material";

import { Role } from "firestore/types/collections.types";

import { convertRoleToRussian } from "firestore/converters";

import { firebaseRepositories } from "firestore/data/repositories";

import { getAuth, signInWithEmailAndPassword } from "firebase/auth";

import { initializeApp } from "firebase/app";

const firebaseApp2 = initializeApp(firebaseConfig, "user\_creation");

const auth2 = getAuth(firebaseApp2);

export const SignUp = () => {

const formRef = useRef<HTMLFormElement>();

const getFormField = (name: string): string => {

const data = new FormData(formRef.current);

return data.get(name)?.toString() || "";

};

const [role, setRole] = useState<Role>(Role.PATIENT);

const [isLoading, setIsLoading] = useState<boolean>(false);

const [createUserWithEmailAndPassword, , loading, error] =

useCreateUserWithEmailAndPassword(auth2);

const navigate = useNavigate();

const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {

event.preventDefault();

try {

setIsLoading(true);

const firstName = getFormField("firstName");

const lastName = getFormField("lastName");

const email = getFormField("email");

const password = getFormField("password");

const relativePatientId = getFormField("relativePatientId");

const diabetType = +getFormField("diabetType");

if (role === Role.RELATIVE) {

if (!relativePatientId) {

NotificationManager.error(

"Код приглашения не может быть пустым",

"Авторизация"

);

return;

}

const patient = await firebaseRepositories.users.getDocById(

relativePatientId

);

if (!patient) {

NotificationManager.error(

"Неправильный код приглашения",

"Авторизация"

);

return;

}

}

const userCredential = await createUserWithEmailAndPassword(

email,

password

);

await auth2.signOut();

if (userCredential) {

await successfulSignUp({

firstName,

lastName,

email,

password,

role,

relativePatientId,

diabetType,

userCredential,

});

await signInWithEmailAndPassword(auth, email, password);

await successfulAuth(userCredential, navigate);

}

} catch (e) {

console.log(e);

NotificationManager.error(

"Создание нового пользователя",

"Ошибка при регистрации"

);

} finally {

setIsLoading(false);

}

};

useEffect(() => {

if (error) {

NotificationManager.error(error?.name, error?.message);

}

}, [error]);

const [isSignUpDisabled, setIsSignUpDisabled] = useState<boolean>(true);

const [arePasswordsEqual, setArePasswordsEqual] = useState<boolean>(true);

const updateUpSignInDisabled = (

currentRole: Role = getFormField("role") as Role

) => {

const email = getFormField("email");

const firstName = getFormField("firstName");

const lastName = getFormField("lastName");

const password = getFormField("password");

const password2 = getFormField("password2");

const relativePatientId = getFormField("relativePatientId");

setArePasswordsEqual(password === password2);

setIsSignUpDisabled(

[

email,

firstName,

lastName,

password,

password2,

(!!relativePatientId && currentRole === Role.RELATIVE) ||

currentRole === Role.PATIENT,

]

.map(Boolean)

.some((value) => !value) || password != password2

);

};

const handleRoleChange = (event: SelectChangeEvent) => {

const newRole = event.target.value as Role;

setRole(newRole);

updateUpSignInDisabled(newRole);

};

return (

<Grid container component="main" sx={{ height: "100vh" }}>

<Grid

item

xs={false}

sm={4}

md={7}

sx={{

backgroundImage: "url(/images/sign-up-background-min.jpg)",

backgroundRepeat: "no-repeat",

backgroundColor: (t) =>

t.palette.mode === "light"

? t.palette.grey[50]

: t.palette.grey[900],

backgroundSize: "cover",

backgroundPosition: "center",

}}

/>

<Grid

item

xs={12}

sm={8}

md={5}

component={Paper}

elevation={6}

square

sx={{

p: (\_theme) => \_theme.spacing(2),

display: "flex",

alignItems: isMobile ? "center" : undefined,

}}

>

<Box

sx={{

display: "flex",

flexDirection: "column",

alignItems: "center",

}}

>

<Avatar sx={{ m: 1, bgcolor: "secondary.main" }}>

<LockOutlinedIcon />

</Avatar>

<Typography component="h1" variant="h5">

Регистрация

</Typography>

<Box

ref={formRef}

component="form"

noValidate

onSubmit={handleSubmit}

sx={{ mt: 3 }}

>

<Grid container spacing={2}>

<Grid item xs={12} sm={6}>

<TextField

autoComplete="given-name"

name="firstName"

required

fullWidth

id="firstName"

disabled={loading || isLoading}

label="Имя"

autoFocus

onChange={() => updateUpSignInDisabled()}

/>

</Grid>

<Grid item xs={12} sm={6}>

<TextField

required

fullWidth

id="lastName"

label="Фамилия"

name="lastName"

disabled={loading || isLoading}

autoComplete="family-name"

onChange={() => updateUpSignInDisabled()}

/>

</Grid>

<Grid item xs={12}>

<TextField

required

fullWidth

id="email"

disabled={loading || isLoading}

label="Электронная почта"

name="email"

autoComplete="email"

onChange={() => updateUpSignInDisabled()}

/>

</Grid>

<Grid item xs={12}>

<TextField

required

fullWidth

name="password"

label="Пароль"

type="password"

id="password"

disabled={loading || isLoading}

autoComplete="new-password"

onChange={() => updateUpSignInDisabled()}

/>

</Grid>

<Grid item xs={12}>

<TextField

required

fullWidth

error={!arePasswordsEqual}

name="password2"

label="Повторённый пароль"

type="password"

id="password2"

disabled={loading || isLoading}

autoComplete="repeat-password"

onChange={() => updateUpSignInDisabled()}

helperText={!arePasswordsEqual && "Пароли должны совпадать!"}

/>

</Grid>

<Grid item xs={12}>

<FormControl fullWidth>

<InputLabel id="role-select-label">Роль</InputLabel>

<Select

labelId="role-select-label"

value={role}

name="role"

label="Роль"

disabled={loading || isLoading}

onChange={handleRoleChange}

>

{[Role.PATIENT, Role.RELATIVE].map((currentRole) => (

<MenuItem key={currentRole} value={currentRole}>

{convertRoleToRussian(currentRole)}

</MenuItem>

))}

</Select>

</FormControl>

</Grid>

{role === Role.PATIENT && (

<Grid item xs={12}>

<FormControl fullWidth>

<InputLabel id="diabet-type-select-label">

Тип диабета

</InputLabel>

<Select

labelId="diabet-type-select-label"

name="diabetType"

defaultValue={1}

label="Тип диабета"

disabled={loading || isLoading}

>

<MenuItem value={1}>1</MenuItem>

<MenuItem value={2}>2</MenuItem>

</Select>

</FormControl>

</Grid>

)}

{role === Role.RELATIVE && (

<Grid item xs={12}>

<TextField

required

fullWidth

name="relativePatientId"

label="Код приглашения"

disabled={loading || isLoading}

onChange={() => updateUpSignInDisabled()}

/>

</Grid>

)}

</Grid>

{loading || isLoading ? (

<LoadingSpinner />

) : (

<Button

disabled={isSignUpDisabled}

type="submit"

fullWidth

variant="contained"

sx={{ mt: 3, mb: 2 }}

>

Зарегистрироваться

</Button>

)}

<Grid container justifyContent="flex-end">

<Grid item>

<Link href={Route.signIn} variant="body2">

Уже есть аккаунт? Войдите

</Link>

</Grid>

</Grid>

</Box>

</Box>

</Grid>

</Grid>

);

};

***AppTitle.tsx***

import \* as React from "react";

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

import MuiAppBar, { AppBarProps as MuiAppBarProps } from "@mui/material/AppBar";

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

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

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

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

import { isMobile } from "react-device-detect";

import { useSelector } from "react-redux";

import { getUserInfoSelector } from "store/selectors";

import { getUserFullName } from "firestore/helpers";

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

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

import { Route } from "components/routing";

import { convertRoleToRussian } from "firestore/converters";

const AppBar = styled(MuiAppBar, {

shouldForwardProp: (prop) => prop !== "open",

})<MuiAppBarProps>(({ theme }) => ({

zIndex: theme.zIndex.drawer + 1,

display: "flex",

justifyContent: "center",

transition: theme.transitions.create(["width", "margin"], {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.leavingScreen,

}),

}));

export const AppTitle = ({

title,

isNavBarOpened,

setIsNavBarOpened,

}: {

title: string;

isNavBarOpened: boolean;

setIsNavBarOpened: React.Dispatch<boolean>;

}) => {

const toggleDrawer = () => {

setIsNavBarOpened(!isNavBarOpened);

};

const userInfo = useSelector(getUserInfoSelector);

const navigate = useNavigate();

const onProfileClick = () => {

setIsNavBarOpened(false);

navigate(Route.profile);

};

return (

<AppBar sx={{ width: "100%", height: "60px !important" }}>

<Toolbar>

{isMobile && (

<IconButton

edge="start"

color="inherit"

aria-label="open drawer"

onClick={toggleDrawer}

sx={{

marginRight: "20px",

}}

>

<MenuIcon />

</IconButton>

)}

<Typography

component="h1"

variant="h6"

color="inherit"

noWrap

sx={{ flexGrow: 1 }}

>

{title}

</Typography>

{!isMobile && (

<Typography

component="h1"

variant="h6"

color="inherit"

noWrap

onClick={onProfileClick}

sx={{

marginRight: "10px",

userSelect: "none",

":hover": { cursor: "pointer" },

}}

>

{getUserFullName(userInfo)} ({convertRoleToRussian(userInfo.role)})

</Typography>

)}

<IconButton onClick={onProfileClick}>

<Avatar src={userInfo.imageUrl} alt={getUserFullName(userInfo)} />

</IconButton>

</Toolbar>

</AppBar>

);

};

***Layout.tsx***

import React, { useState } from "react";

import { Grid } from "@mui/material";

import { Navbar } from "components/layout/Navbar";

import { matchPath, Outlet, useLocation } from "react-router-dom";

import { AppTitle } from "components/layout/AppTitle";

import { isMobile } from "react-device-detect";

import { Route, RouteTitles } from "components/routing/constants";

const findRoute = (path: string): Route =>

Object.values(Route)

.filter((route) => matchPath(route, path))

.at(-1) as Route;

export const Layout = () => {

const location = useLocation();

const currentRoute = findRoute(location.pathname);

const title = RouteTitles[currentRoute];

const [isNavBarOpened, setIsNavBarOpened] = useState<boolean>(!isMobile);

return (

<Grid container columnSpacing={10}>

<AppTitle

title={title}

isNavBarOpened={isNavBarOpened}

setIsNavBarOpened={setIsNavBarOpened}

/>

<Navbar

isNavBarOpened={isNavBarOpened}

setIsNavBarOpened={setIsNavBarOpened}

/>

<Outlet />

</Grid>

);

};

***constants.tsx***

import React from "react";

import { Role } from "firestore/types/collections.types";

import { Route } from "components/routing/constants";

import PersonIcon from "@mui/icons-material/Person";

import SummarizeIcon from "@mui/icons-material/Summarize";

import AlarmIcon from "@mui/icons-material/Alarm";

import TrendingUpIcon from "@mui/icons-material/TrendingUp";

import ChatIcon from "@mui/icons-material/Chat";

import FamilyRestroomIcon from "@mui/icons-material/FamilyRestroom";

import InsightsIcon from "@mui/icons-material/Insights";

import QueryStatsIcon from "@mui/icons-material/QueryStats";

import WysiwygIcon from "@mui/icons-material/Wysiwyg";

import PeopleIcon from "@mui/icons-material/People";

import LogoutIcon from "@mui/icons-material/Logout";

import LocalHospitalIcon from "@mui/icons-material/LocalHospital";

import ReviewsIcon from "@mui/icons-material/Reviews";

import RecommendIcon from "@mui/icons-material/Recommend";

import MonitorHeartIcon from "@mui/icons-material/MonitorHeart";

import FavoriteIcon from "@mui/icons-material/Favorite";

export type NavBarItem = {

id: number;

icon: React.ReactNode;

label: string;

route: Route;

roles: Role[];

};

export const topNavbarItems: NavBarItem[] = [

{

id: 0,

icon: <PersonIcon />,

label: "Профиль",

route: Route.profile,

roles: [...Object.values(Role)],

},

];

export const navbarItemsData: Omit<NavBarItem, "id">[] = [

{

icon: <SummarizeIcon />,

label: "Дневник",

route: Route.diary,

roles: [Role.PATIENT],

},

{

icon: <LocalHospitalIcon />,

label: "Доктор",

route: Route.doctor,

roles: [Role.PATIENT],

},

{

icon: <FavoriteIcon />,

label: "Рекомендации",

route: Route.recommendations,

roles: [Role.PATIENT],

},

{

icon: <MonitorHeartIcon />,

label: "Уведомления",

route: Route.notifications,

roles: [Role.DOCTOR],

},

{

icon: <RecommendIcon />,

label: "Назначения",

route: Route.healthStates,

roles: [Role.DOCTOR],

},

{

icon: <PeopleIcon />,

label: "Пациенты",

route: Route.patients,

roles: [Role.DOCTOR],

},

{

icon: <AlarmIcon />,

label: "Цели",

route: Route.goals,

roles: [],

},

{

icon: <TrendingUpIcon />,

label: "Статистика",

route: Route.statistics,

roles: [Role.PATIENT],

},

{

icon: <ChatIcon />,

label: "Диалоги",

route: Route.dialogs,

roles: [Role.PATIENT, Role.DOCTOR],

},

{

icon: <FamilyRestroomIcon />,

label: "Родственник",

route: Route.relative,

roles: [Role.PATIENT, Role.RELATIVE],

},

{

icon: <InsightsIcon />,

label: "Статистика родственника-пациента",

route: Route.relativeStatistics,

roles: [Role.RELATIVE],

},

{

icon: <QueryStatsIcon />,

label: "Статистика пациентов",

route: Route.patientsStatistics,

roles: [Role.DOCTOR],

},

{

icon: <WysiwygIcon />,

label: "Тематические материалы",

route: Route.thematicMaterials,

roles: [

Role.PATIENT,

Role.RELATIVE,

Role.DOCTOR,

Role.CONTENT\_MAKER,

Role.MODERATOR,

],

},

{

icon: <PeopleIcon />,

label: "Пользователи",

route: Route.accounts,

roles: [Role.ADMIN],

},

{

icon: <ReviewsIcon />,

label: "Отзывы",

route: Route.reviews,

roles: [Role.DOCTOR],

},

];

export const bottomNavBarItems: NavBarItem[] = [

{

id: 0,

icon: <LogoutIcon />,

label: "Выход",

route: Route.signOut,

roles: [...Object.values(Role)],

},

];

export const navBarItems: NavBarItem[] = [

...topNavbarItems,

...navbarItemsData,

...bottomNavBarItems,

].map((data, index) => ({

...data,

id: index,

}));

***Navbar.tsx***

import React from "react";

import {

Box,

Grid,

ListItemButton,

ListItemIcon,

ListItemText,

SwipeableDrawer,

} from "@mui/material";

import { isMobile } from "react-device-detect";

import { useWindowSize } from "hooks/useWindowSize";

import { NavBarItem, navBarItems } from "components/layout/Navbar/constants";

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

import { User } from "firestore/types/collections.types";

import { GlobalState } from "store";

import { useSelector } from "react-redux";

import "./navbar.scss";

export const Navbar = ({

isNavBarOpened,

setIsNavBarOpened,

}: {

isNavBarOpened: boolean;

setIsNavBarOpened: React.Dispatch<boolean>;

}) => {

const [width] = useWindowSize();

const navigate = useNavigate();

const user = useSelector<GlobalState, User>(

(state) => state.currentUser.user as User

);

const location = useLocation();

const navBarComponent = (navBarItem: NavBarItem) => (

<ListItemButton

key={navBarItem.id}

onClick={() => {

if (isMobile) {

setIsNavBarOpened(false);

}

navigate(navBarItem.route);

}}

className="item-container"

selected={location.pathname.startsWith(navBarItem.route)}

>

<ListItemIcon className="item-icon">{navBarItem.icon}</ListItemIcon>

<ListItemText primary={navBarItem.label} className="item-text" />

</ListItemButton>

);

const mapNavBarItems = (items: NavBarItem[]) =>

items

.filter((navBarItem) => navBarItem.roles.includes(user.role))

.map(navBarComponent);

const list = (

<Box

sx={{

width: isMobile ? 250 : (width / 12) \* 2,

minWidth: 170,

marginTop: "60px",

}}

role="presentation"

className="navbar-items-container"

>

{mapNavBarItems(navBarItems)}

</Box>

);

return (

<Grid item xs={isMobile ? undefined : 2}>

<SwipeableDrawer

variant={isMobile ? "temporary" : "permanent"}

anchor="left"

open={isNavBarOpened}

onClose={() => setIsNavBarOpened(false)}

onOpen={() => setIsNavBarOpened(true)}

>

{list}

</SwipeableDrawer>

</Grid>

);

};

***PageContainer.tsx***

import React, { CSSProperties } from "react";

import { Grid } from "@mui/material";

import { isMobile } from "react-device-detect";

export const PageContainer = ({

children,

className,

style,

}: {

children?: React.ReactNode;

className?: string;

style?: CSSProperties;

}) => {

return (

<Grid

item

xs={isMobile ? 12 : 10}

sx={{ marginTop: "60px", maxWidth: "initial" }}

>

<Grid item xs={12} className={className} style={style}>

{children}

</Grid>

</Grid>

);

};

***AccountsPage.jsx***

import React, { useState } from "react";

import { PageContainer } from "../../layout";

import { useGeneralDataHook } from "../../../hooks/useGeneralDataHook";

import { firebaseRepositories } from "../../../firestore/data/repositories";

import { LoadingSpinner } from "../../ui/LoadingSpinner";

import { Role } from "../../../firestore/types/collections.types";

import { AccountModal, UsersView } from "./elements";

import { useGeneralModalHandlers } from "../../../hooks/useGeneralModalHandlers";

import { initializeApp } from "firebase/app";

import { firebaseConfig } from "../../../firebase\_config";

import {

getAuth,

signInWithEmailAndPassword,

createUserWithEmailAndPassword,

updateEmail,

updatePassword,

deleteUser,

} from "firebase/auth";

const adminFirebaseApp = initializeApp(firebaseConfig, "admin-console");

const adminAuth = getAuth(adminFirebaseApp);

export const AccountsPage = () => {

const [isAccountUpdateLoading, setIsAccountUpdateLoading] = useState(false);

const [isUsersLoading, users, refreshUsers] = useGeneralDataHook(async () => {

return firebaseRepositories.users.getDocs();

}, []);

const [selectedAccount, setSelectedAccount] = useState(null);

const [selectedRole, setSelectedRole] = useState(null);

const [isAccountModalOpened, openAccountModal, closeAccountModal] =

useGeneralModalHandlers({

onOpen: (role, user = null) => {

setSelectedAccount(user);

setSelectedRole(role);

},

onClose: () => {

setSelectedAccount(null);

setSelectedRole(null);

},

});

const updateAccount = async (updatedAccount, existingAccount = null) => {

try {

setIsAccountUpdateLoading(true);

const userCredential = Boolean(existingAccount)

? await signInWithEmailAndPassword(

adminAuth,

existingAccount.email,

existingAccount.password

)

: await createUserWithEmailAndPassword(

adminAuth,

updatedAccount.email,

updatedAccount.password

);

if (existingAccount) {

await updateEmail(userCredential.user, updatedAccount.email);

await updatePassword(userCredential.user, updatedAccount.password);

}

await firebaseRepositories.users.updateDoc({

...updatedAccount,

docId: userCredential.user.uid,

});

await adminAuth.signOut();

await refreshUsers();

} catch (e) {

console.log(e);

} finally {

setIsAccountUpdateLoading(false);

}

};

const deleteAccount = async (account) => {

try {

setIsAccountUpdateLoading(true);

const userCredential = await signInWithEmailAndPassword(

adminAuth,

selectedAccount.email,

selectedAccount.password

);

// todo side effects, update connected users (reviews, relativePatient, relative, patient, doctor)

// await deleteUser(userCredential.user);

// await firebaseRepositories.users.deleteDocById(account.docId)

await adminAuth.signOut();

await refreshUsers();

} catch (e) {

console.log(e);

} finally {

setIsAccountUpdateLoading(false);

}

};

return (

<PageContainer

style={{

padding: "30px",

display: "flex",

flexDirection: "column",

alignItems: "center",

justifyContent: "center",

gap: "20px",

minHeight: "calc(100vh - 60px)",

width: "100%",

}}

>

{isUsersLoading || isAccountUpdateLoading ? (

<LoadingSpinner />

) : (

users && (

<div

style={{

display: "flex",

flexDirection: "column",

alignItems: "center",

gap: "30px",

width: "100%",

}}

>

{[Role.PATIENT, Role.DOCTOR, Role.RELATIVE, Role.ADMIN].map(

(role) => (

<UsersView

key={role}

role={role}

users={users.filter((user) => user.role === role)}

editAccount={openAccountModal}

deleteAccount={deleteAccount}

/>

)

)}

</div>

)

)}

{selectedRole && (

<AccountModal

isOpen={isAccountModalOpened}

onClose={closeAccountModal}

submitAccount={updateAccount}

selectedAccount={selectedAccount}

role={selectedRole}

/>

)}

</PageContainer>

);

};

***AccountModal.helpers.ts***

import dayjs from "dayjs";

import { Role, User } from "firestore/types/collections.types";

export const createUser = (role: Role): User => {

const baseUser = {

docId: "",

email: "",

password: "",

imageUrl: "",

name: {

first: "",

last: "",

},

address: "",

phone: "",

role,

};

const patientPart = {

diary: {

dailyLogs: [],

goals: [],

diabetType: 1,

},

doctor: "",

};

const relativePart = {

relativePatient: "",

};

const employeePart = {

employee: {

hiredAt: dayjs().toDate().toString(),

reviews: [],

salary: 0,

description: "",

},

};

return {

...baseUser,

...(role === Role.PATIENT ? patientPart : {}),

...(role === Role.RELATIVE ? relativePart : {}),

...([Role.DOCTOR, Role.CONTENT\_MAKER, Role.MODERATOR, Role.ADMIN].includes(

role

)

? employeePart

: {}),

};

};

***AccountModal.jsx***

import React, { useCallback, useEffect, useState } from "react";

import {

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

TextField,

} from "@mui/material";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { deepCopy } from "deep-copy-ts";

import { isMobile } from "react-device-detect";

import { convertRoleToRussian } from "firestore/converters";

import { createUser } from "./AccountModal.helpers";

import { Role } from "firestore/types/collections.types";

import { formatDate } from "../../../../../helpers/helpers";

import "./account-modal.scss";

export const AccountModal = ({

isOpen,

onClose,

submitAccount,

selectedAccount,

role,

}) => {

const getAccount = useCallback(

() => (selectedAccount ? deepCopy(selectedAccount) : createUser(role)),

[selectedAccount, role]

);

const [account, setAccount] = useState(getAccount());

const [isLoading, setIsLoading] = useState(false);

const resetAccount = useCallback(() => {

setAccount(getAccount());

}, [setAccount, getAccount]);

const handleFieldChange = (e) => {

const { name, value } = e.target;

setAccount((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleSubmitAccount = async (event) => {

event.preventDefault();

setIsLoading(true);

await submitAccount(account, selectedAccount);

setIsLoading(false);

onClose();

};

useEffect(() => {

resetAccount();

}, [selectedAccount, isOpen, resetAccount]);

return (

<Dialog open={isOpen} onClose={onClose} className="account-modal-dialog">

<DialogTitle>

{(selectedAccount ? "Редактирование" : "Добавление") +

" пользователя [" +

convertRoleToRussian(role) +

"]"}

</DialogTitle>

<DialogContent

className="account-modal-content"

sx={{

width: isMobile ? "300px" : "600px",

}}

>

<form className="account-form" onSubmit={handleSubmitAccount}>

<TextField

label="Роль"

name="role"

value={account.role}

disabled={true}

/>

{selectedAccount && (

<TextField

label="Идентификатор пользователя"

name="docId"

value={account.docId}

disabled={true}

/>

)}

<TextField

label="Эл. почта"

name="email"

value={account.email}

onChange={handleFieldChange}

disabled={isLoading}

/>

<TextField

label="Пароль"

name="password"

value={account.password}

onChange={handleFieldChange}

disabled={isLoading}

/>

<TextField

label="Ссылка на изображение профиля"

name="imageUrl"

value={account.imageUrl}

onChange={handleFieldChange}

disabled={isLoading}

/>

<TextField

label="Имя"

value={account.name.first}

onChange={({ target: { value } }) =>

setAccount((prevAccount) => ({

...prevAccount,

name: {

...prevAccount.name,

first: value,

},

}))

}

disabled={isLoading}

/>

<TextField

label="Фамилия"

value={account.name.last}

onChange={({ target: { value } }) =>

setAccount((prevAccount) => ({

...prevAccount,

name: {

...prevAccount.name,

last: value,

},

}))

}

disabled={isLoading}

/>

<TextField

label="Адрес"

name="address"

value={account.address}

onChange={handleFieldChange}

disabled={isLoading}

/>

<TextField

label="Телефон"

name="phone"

value={account.phone}

onChange={handleFieldChange}

disabled={isLoading}

/>

{/\* patient \*/}

{role === Role.PATIENT && (

<>

<TextField

label="Тип диабета"

value={account.diary.diabetType}

onChange={({ target: { value } }) =>

setAccount((prevAccount) => ({

...prevAccount,

diary: {

...prevAccount.diary,

diabetType: isNaN(+value) ? 0 : +value,

},

}))

}

disabled={isLoading}

/>

</>

)}

{/\* relative \*/}

{role === Role.RELATIVE && (

<>

<TextField

label="Код приглашения пациента родственника"

name="relativePatient"

value={account.relativePatient}

onChange={handleFieldChange}

disabled={isLoading}

/>

</>

)}

{/\* employee \*/}

{[

Role.DOCTOR,

Role.CONTENT\_MAKER,

Role.MODERATOR,

Role.ADMIN,

].includes(role) && (

<>

<TextField

label="Дата найма"

name="hiredAt"

value={formatDate(account.employee.hiredAt)}

disabled={true}

/>

<TextField

multiline

minRows={3}

maxRows={4}

label="Описание"

value={account.employee.description}

onChange={({ target: { value } }) =>

setAccount((prevAccount) => ({

...prevAccount,

employee: {

...prevAccount.employee,

description: value,

},

}))

}

disabled={isLoading}

fullWidth

/>

<TextField

label="Зарплата"

value={account.employee.salary}

onChange={({ target: { value } }) =>

setAccount((prevAccount) => ({

...prevAccount,

employee: {

...prevAccount.employee,

salary: isNaN(+value) ? 0 : +value,

},

}))

}

disabled={isLoading}

/>

</>

)}

<DialogActions className="controls">

<Button onClick={onClose} disabled={isLoading} variant="outlined">

Отмена

</Button>

{isLoading ? (

<LoadingSpinner style={{ margin: "0", width: "100px" }} />

) : (

<Button

type="submit"

variant="outlined"

disabled={false} // todo

autoFocus

>

{selectedAccount ? "Изменить" : "Добавить"}

</Button>

)}

</DialogActions>

</form>

</DialogContent>

</Dialog>

);

};

***UsersView.jsx***

import React from "react";

import {

Button,

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

} from "@mui/material";

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

import {

convertRoleToRussian,

convertRoleToRussianPlural,

} from "firestore/converters/role/role.converter";

import { getUserFullName } from "../../../../../firestore/helpers";

import { CardContainer } from "../../../../ui/CardContainer";

import { Role } from "../../../../../firestore/types/collections.types";

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

export const UsersView = ({ role, users, editAccount, deleteAccount }) => {

return (

<div style={{ display: "flex", width: "100%" }}>

<TableContainer component={CardContainer}>

<Typography variant="h5" sx={{ textAlign: "center" }}>

{convertRoleToRussianPlural(role)}

</Typography>

<div style={{ display: "flex", justifyContent: "center" }}>

<Button

color="success"

variant="contained"

sx={{ fontSize: "11px", fontWeight: "bold" }}

onClick={() => editAccount(role)}

>

Добавить

</Button>

</div>

<Table>

<TableHead>

<TableRow>

<TableCell>{convertRoleToRussian(role)}</TableCell>

<TableCell>Эл. почта</TableCell>

<TableCell>Телефон</TableCell>

<TableCell>Адрес</TableCell>

{role === Role.PATIENT && <TableCell>Тип диабета</TableCell>}

{[

Role.DOCTOR,

Role.CONTENT\_MAKER,

Role.MODERATOR,

Role.ADMIN,

].includes(role) && <TableCell>Зарплата</TableCell>}

<TableCell>Управление</TableCell>

</TableRow>

</TableHead>

<TableBody>

{users.map((user) => (

<TableRow key={user.docId}>

<TableCell>

<div

style={{

display: "flex",

gap: "10px",

alignItems: "center",

}}

>

<Avatar src={user.imageUrl} alt={getUserFullName(user)} />

<Typography>{getUserFullName(user)}</Typography>

</div>

</TableCell>

<TableCell>{user.email}</TableCell>

<TableCell>{user.phone}</TableCell>

<TableCell>{user.address}</TableCell>

{role === Role.PATIENT && (

<TableCell>{user.diary.diabetType}</TableCell>

)}

{[

Role.DOCTOR,

Role.CONTENT\_MAKER,

Role.MODERATOR,

Role.ADMIN,

].includes(role) && (

<TableCell>{user.employee.salary} $</TableCell>

)}

<TableCell>

<div style={{ display: "flex", gap: "5px" }}>

<Button

color="warning"

variant="contained"

sx={{ fontSize: "11px", fontWeight: "bold" }}

onClick={() => editAccount(role, user)}

>

Редактировать

</Button>

<Button

color="error"

variant="contained"

sx={{ fontSize: "11px", fontWeight: "bold" }}

onClick={() => deleteAccount(role, user)}

disabled={true} // todo side effects

>

Удалить

</Button>

</div>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

</div>

);

};

***DialogsPage.tsx***

import React, { SyntheticEvent, useCallback, useEffect, useState } from "react";

import { PageContainer } from "components/layout";

import { useDialogsData } from "components/pages/Dialogs/hooks/useDialogsData";

import { getUserSelector } from "store/selectors";

import { useSelector } from "react-redux";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import {

Channel,

ChannelHeader,

Chat,

MessageInput,

MessageList,

Thread,

Window,

} from "stream-chat-react";

import "stream-chat-react/dist/css/v2/index.css";

import { i18nInstance, streamClient } from "containers/App/App";

import useAsyncEffect from "use-async-effect";

import { Role, UserInfo } from "firestore/types/collections.types";

import { Autocomplete, TextField } from "@mui/material";

import { getUserFullName } from "firestore/helpers";

import { firebaseRepositories } from "firestore/data/repositories";

import { where } from "firebase/firestore";

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

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

import \* as emailjs from "@emailjs/browser";

import "./dialogs.scss";

export const DialogsPage = () => {

const currentUser = useSelector(getUserSelector);

// dialogs

const [dialogsUsers, setDialogsUsers] = useState<UserInfo[] | null>(null);

const [selectedUser, setSelectedUser] = useState<UserInfo | null>(null);

const { isLoading, channel } = useDialogsData({

currentUser,

dialogsUsers,

selectedUser,

});

// effects

// fetch dialogs on initial load

useAsyncEffect(async () => {

if (currentUser.role === Role.PATIENT && !currentUser.doctor) {

setDialogsUsers([]);

return;

}

const filter =

currentUser.role === Role.PATIENT

? where("docId", "==", currentUser.doctor)

: where("doctor", "==", currentUser.docId);

const newDialogsUsers = await firebaseRepositories.users.getDocs(filter);

setDialogsUsers(newDialogsUsers);

}, [currentUser]);

// set first user as selected if user is patient

useEffect(() => {

if (dialogsUsers && dialogsUsers.length > 0) {

setSelectedUser(dialogsUsers[0]);

}

}, [dialogsUsers, currentUser]);

// send emails when a new message occurs

useEffect(() => {

return streamClient.on("message.new", async (event) => {

if (!selectedUser || !dialogsUsers || !dialogsUsers.length) {

return;

}

if (event?.user?.id === currentUser.docId && event?.message?.text) {

const message = event.message.text;

const receiverId = selectedUser.docId;

const receiverUserInfo = dialogsUsers.find(

(user) => user.docId === receiverId

) as UserInfo;

await emailjs.send("service\_oxiflh8", "template\_4mhfpd6", {

to\_email: receiverUserInfo.email,

from\_name: getUserFullName(currentUser),

to\_name: getUserFullName(receiverUserInfo),

message,

});

}

}).unsubscribe;

}, [selectedUser, dialogsUsers, currentUser]);

// handlers

// handles dialog selection

const handleSelectedDialogChange = (

event: SyntheticEvent<Element, Event>,

option: unknown

) => {

if (!dialogsUsers) {

return;

}

const castedOption = option as { value: string; label: string } | null;

setSelectedUser(

dialogsUsers.find((user) => user.docId === castedOption?.value) ?? null

);

};

const getUserOption = useCallback((user: UserInfo) => {

return {

value: user.docId,

label: getUserFullName(user),

imageUrl: user.imageUrl,

};

}, []);

return (

<PageContainer className="dialogs-page-container">

<div className="dialogs-container">

{!dialogsUsers ? (

<LoadingSpinner />

) : (

<>

{!dialogsUsers.length && (

<div className="no-dialog-users">

{currentUser.role === Role.DOCTOR && "Нет пациентов"}

{currentUser.role === Role.PATIENT &&

"Доктор не выбран, перейдите на вкладку [Доктор]"}

</div>

)}

{dialogsUsers.length > 0 && currentUser.role === Role.DOCTOR && (

<Autocomplete

id="dialog-select"

className="dialogs-page-select"

value={selectedUser ? getUserOption(selectedUser) : null}

onChange={handleSelectedDialogChange}

options={dialogsUsers.map(getUserOption)}

renderInput={(params) => (

<TextField

{...params}

label="Выберите диалог"

inputProps={{

...params.inputProps,

}}

/>

)}

blurOnSelect

noOptionsText="Нет пациентов с таким именем."

renderOption={(props, option) => (

<Box

component="li"

sx={{ "& > img": { mr: 2, flexShrink: 0 } }}

{...props}

>

<Avatar

src={option.imageUrl}

sx={{

width: "30px",

height: "30px",

marginRight: "20px",

}}

/>

{option.label}

</Box>

)}

/>

)}

</>

)}

{selectedUser &&

dialogsUsers &&

dialogsUsers.length > 0 &&

// "!channel" check only because of typescript

(isLoading || !channel ? (

<LoadingSpinner />

) : (

<Chat

client={streamClient}

theme="str-chat\_\_theme-light"

i18nInstance={i18nInstance}

customClasses={{

chatContainer:

currentUser.role === Role.DOCTOR

? "str-chat\_\_container doctor-view"

: "str-chat\_\_container",

}}

>

<Channel channel={channel}>

<Window>

<ChannelHeader />

<MessageList />

<MessageInput />

</Window>

<Thread />

</Channel>

</Chat>

))}

</div>

</PageContainer>

);

};

***useDialogsData.ts***

import { Role, User, UserInfo } from "firestore/types/collections.types";

import { Channel } from "stream-chat";

import useAsyncEffect from "use-async-effect";

import { useState } from "react";

import { getUserFullName } from "firestore/helpers";

import { NotificationManager } from "react-notifications";

import { streamClient } from "containers/App/App";

const getStreamUserData = (userInfo: UserInfo) => ({

id: userInfo.docId,

name: `${userInfo.name.first} ${userInfo.name.last}`,

image: userInfo.imageUrl,

});

const getStreamUserToken = (uid: string) => streamClient.devToken(uid);

const connectStreamUser = async (userInfo: UserInfo) =>

streamClient.connectUser(

getStreamUserData(userInfo),

getStreamUserToken(userInfo.docId)

);

export const useDialogsData = ({

currentUser,

dialogsUsers,

selectedUser,

}: {

currentUser: User;

dialogsUsers: UserInfo[] | null;

selectedUser: UserInfo | null;

}) => {

const [channel, setChannel] = useState<Channel | null>(null);

const [isLoading, setIsLoading] = useState<boolean>(false);

useAsyncEffect(async () => {

if (!selectedUser || !dialogsUsers || !dialogsUsers.length) {

return;

}

try {

setIsLoading(true);

const patient =

currentUser.role === Role.PATIENT ? currentUser : selectedUser;

const doctor =

currentUser.role === Role.PATIENT ? selectedUser : currentUser;

const channelId = `${doctor.docId}\_${patient.docId}`;

const channelName =

currentUser.role === Role.PATIENT

? getUserFullName({ name: doctor.name })

: getUserFullName({ name: patient.name });

const channelImage = selectedUser.imageUrl;

await connectStreamUser(currentUser);

const newChannel = streamClient.channel("messaging", channelId);

await newChannel.watch();

await newChannel.update({

name: channelName,

image: channelImage,

members: [currentUser.docId, selectedUser.docId],

});

setChannel(newChannel);

} catch (e) {

console.log(e);

NotificationManager.error(

"Загрузке данных для диалогов",

"Ошибка данных диалогов"

);

} finally {

setIsLoading(false);

}

}, [currentUser, dialogsUsers, selectedUser]);

return {

isLoading,

channel,

};

};

***DailyLogDataView.tsx***

import React from "react";

import dayjs from "dayjs";

import { EditButton } from "components/ui/EditButton";

import { CardContainer } from "components/ui/CardContainer";

import { DailyLogData } from "firestore/converters";

import "./dailylog-data-view.scss";

export const DailyLogDataView = ({

dailyLogData,

handleEditButtonClick,

isEditable,

}: {

dailyLogData: DailyLogData;

handleEditButtonClick: () => void;

isEditable: boolean;

}) => {

const getDisplayValue = (propName: keyof DailyLogData): string | number =>

(+dailyLogData[propName] ? (+dailyLogData[propName]).toFixed(2) : "-") as

| string

| number;

return (

<CardContainer className="diary-page-content\_\_info-container">

<div className="diary-page-content\_\_info-container\_\_title">

<h2>{`${dayjs(dailyLogData.createdAt).format(

"HH:mm, DD MMMM YYYY"

)}`}</h2>

{isEditable && <EditButton onClick={handleEditButtonClick} />}

</div>

<p>Уровень сахара: {getDisplayValue("sugarLevel")} ммоль/л</p>

<p>Пульс: {getDisplayValue("pulse")} уд/мин</p>

<p>

Давление: {getDisplayValue("systolic")}/{getDisplayValue("diastolic")}{" "}

ммрт/ммрт

</p>

<p>Калории: {getDisplayValue("total")} ккал</p>

<p>Вес: {getDisplayValue("weight")} кг</p>

<p>Температура: {getDisplayValue("temperature")} °C</p>

</CardContainer>

);

};

***NoteModal.tsx***

import React, { useCallback, useEffect, useState } from "react";

import {

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

TextField,

} from "@mui/material";

import { Note } from "firestore/types/collections.types";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { deepCopy } from "deep-copy-ts";

import { formatDate } from "helpers/helpers";

import { isMobile } from "react-device-detect";

import dayjs from "dayjs";

import "./note-modal.scss";

export const NoteModal = ({

isOpen,

onClose,

submitNote,

selectedNote,

}: {

isOpen: boolean;

onClose: () => void;

submitNote: (note: Note) => Promise<void>;

selectedNote: Note | null;

}) => {

const getNote = useCallback(

() =>

selectedNote

? deepCopy(selectedNote)

: {

createdAt: dayjs().toDate().toString(),

content: "",

},

[selectedNote]

);

const [note, setNote] = useState<Note>(getNote());

const [isLoading, setIsLoading] = useState<boolean>(false);

const resetNote = useCallback(() => {

setNote(getNote());

}, [setNote, getNote]);

const handleFieldChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const { name, value } = e.target;

setNote((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleSubmitNote = async (event: React.FormEvent<HTMLFormElement>) => {

event.preventDefault();

setIsLoading(true);

await submitNote({

...note,

createdAt: dayjs().toDate().toString(),

});

setIsLoading(false);

onClose();

};

useEffect(() => {

resetNote();

}, [selectedNote, isOpen, resetNote]);

return (

<Dialog open={isOpen} onClose={onClose}>

<DialogTitle>

{(selectedNote ? "Редактирование" : "Добавление") + " заметки"}

</DialogTitle>

<DialogContent

className="modal-content"

sx={{

width: isMobile ? "300px" : "600px",

}}

>

<form className="note-form" onSubmit={handleSubmitNote}>

<TextField

label="Дата"

name="createdAt"

value={formatDate(note.createdAt)}

disabled={true}

/>

<TextField

multiline

minRows={3}

maxRows={3}

label="Текст"

name="content"

value={note.content}

onChange={handleFieldChange}

disabled={isLoading}

/>

<DialogActions className="controls">

<Button onClick={onClose} disabled={isLoading} variant="outlined">

Отмена

</Button>

{isLoading ? (

<LoadingSpinner style={{ margin: "0", width: "100px" }} />

) : (

<Button

type="submit"

variant="outlined"

disabled={selectedNote?.content === note.content}

autoFocus

>

{selectedNote ? "Изменить" : "Добавить"}

</Button>

)}

</DialogActions>

</form>

</DialogContent>

</Dialog>

);

};

***DiaryPage.tsx***

import React, { useEffect, useState } from "react";

import { PageContainer } from "components/layout";

import { DateCalendar } from "@mui/x-date-pickers";

import dayjs from "dayjs";

import { useDailyLog, useDailyLogData } from "components/pages/Diary/hooks/";

import { Button, TextField } from "@mui/material";

import {

DailyLog,

DailyLogPropName,

Diary,

Note,

PatientReport,

User,

} from "firestore/types/collections.types";

import { CardContainer } from "components/ui/CardContainer";

import { DailyLogDataView } from "components/pages/Diary/components";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { deepCopy } from "deep-copy-ts";

import { DiaryFormErrors } from "components/pages/Diary/types";

import { deepEqual } from "ts-deep-equal";

import { dailyLogDataValidationObj } from "components/pages/Diary/helpers";

import {

convertDailyLogDataToDailyLog,

convertDailyLogToDailyLogData,

DailyLogData,

} from "firestore/converters";

import { firebaseRepositories } from "firestore/data/repositories";

import { fetchUser } from "store/reducers/user/actions";

import { setDailyLog } from "store/reducers/user/userSlice";

import { NotificationManager } from "react-notifications";

import { GlobalState, useAppDispatch } from "store";

import { useSelector } from "react-redux";

import AddCircleIcon from "@mui/icons-material/AddCircle";

import { NoteModal } from "components/pages/Diary/components/NoteModal";

import { EditButton } from "components/ui/EditButton";

import { formatDate } from "helpers/helpers";

import DeleteIcon from "@mui/icons-material/Delete";

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

import {

calculatePatientReportDeviation,

generateDocId,

} from "firestore/helpers";

import "./diary.scss";

export const emptyErrors: DiaryFormErrors = {

sugarLevel: null,

pulse: null,

systolic: null,

diastolic: null,

total: null,

weight: null,

temperature: null,

};

const generateNotification = async ({

user,

dailyLogData,

}: {

user: User;

dailyLogData: DailyLogData;

}) => {

const patientId = user.docId;

const doctorId = user.doctor;

if (!doctorId) {

return;

}

const patientReports: PatientReport[] = Object.values(DailyLogPropName)

.filter((propName) => {

const currentValue = +dailyLogData[propName];

return (

!isNaN(currentValue) &&

currentValue &&

!!calculatePatientReportDeviation(propName, currentValue)

);

})

.map((propName) => ({

currentValue: +dailyLogData[propName],

propName,

}));

if (!patientReports.length) {

return;

}

await firebaseRepositories.notifications.updateDoc({

docId: generateDocId(),

createdAt: dayjs().toDate().toString(),

patient: patientId,

doctor: doctorId,

patientReports,

recommendation: null,

});

};

export const DiaryPage = () => {

// saving

const user = useSelector<GlobalState, User>(

(state) => state.currentUser.user as User

);

const dispatch = useAppDispatch();

// diary

const [currentDate, setCurrentDate] = useState<dayjs.Dayjs>(dayjs());

const originalDailyLog = useDailyLog(currentDate);

const { dailyLogData, setDailyLogData, resetDailyLogData } =

useDailyLogData(originalDailyLog);

const onDatePickerChange = (newValue: dayjs.Dayjs | null) => {

if (newValue) {

setCurrentDate(newValue);

}

};

const [isEditMode, setIsEditMode] = useState<boolean>(false);

const [errors, setErrors] = useState<DiaryFormErrors>(deepCopy(emptyErrors));

const [isSubmitDisabled, setIsSubmitDisabled] = useState<boolean>(true);

const [isFormSaving, setIsFormSaving] = useState<boolean>(false);

useEffect(() => {

if (!isEditMode) {

return;

}

const newErrors = Object.fromEntries(

Object.keys(dailyLogDataValidationObj).map((key) => [

key,

dailyLogDataValidationObj[key](

dailyLogData[key as keyof DailyLogData].toString()

),

])

) as unknown as DiaryFormErrors;

setErrors(newErrors);

setIsSubmitDisabled(

Object.values(newErrors).some((value) => !!value) ||

deepEqual(dailyLogData, convertDailyLogToDailyLogData(originalDailyLog))

);

}, [dailyLogData, originalDailyLog, isEditMode]);

const handleFieldChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const { name, value } = e.target;

setDailyLogData((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleEditButtonClick = () => {

setIsEditMode(true);

};

const handleCancelButtonClick = () => {

resetDailyLogData();

setIsEditMode(false);

};

const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {

e.preventDefault();

const diary: Diary = user.diary as Diary;

const dailyLogs: DailyLog[] = diary.dailyLogs;

try {

setIsFormSaving(true);

const newDailyLog: DailyLog = {

...convertDailyLogDataToDailyLog(dailyLogData),

createdAt: currentDate.toDate().toString(),

};

const newDiary: Diary = {

...diary,

dailyLogs: [

...dailyLogs.filter(

(dailyLog) =>

dayjs(dailyLog.createdAt).format("l") !== currentDate.format("l")

),

newDailyLog,

],

};

const newUser = {

...user,

diary: newDiary,

};

await firebaseRepositories.users.updateDoc(newUser);

await dispatch(fetchUser(newUser.docId));

await generateNotification({ user, dailyLogData });

} catch (err) {

dispatch(setDailyLog(originalDailyLog));

console.log(err);

NotificationManager.error(

"Сохранение дневника",

'[Ошибка на странице "дневник"]'

);

} finally {

setIsFormSaving(false);

setIsEditMode(false);

}

};

// notes

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

const [selectedNote, setSelectedNote] = useState<Note | null>(null);

const [deletedNote, setDeletedNote] = useState<Note | null>(null);

const handleSubmitNote = async (note: Note) => {

const diary: Diary = user.diary as Diary;

const dailyLogs: DailyLog[] = diary.dailyLogs;

try {

const notesWithoutSelected = dailyLogData.notes.filter(

(\_note) => \_note.createdAt !== selectedNote?.createdAt

);

const newNotes = [...notesWithoutSelected, note];

const newDailyLog: DailyLog = {

...originalDailyLog,

notes: newNotes,

};

const newDiary: Diary = {

...diary,

dailyLogs: [

...dailyLogs.filter(

(dailyLog) =>

dayjs(dailyLog.createdAt).format("l") !== currentDate.format("l")

),

newDailyLog,

],

};

const newUser = {

...user,

diary: newDiary,

};

await firebaseRepositories.users.updateDoc(newUser);

await dispatch(fetchUser(newUser.docId));

} catch (e) {

dispatch(setDailyLog(convertDailyLogToDailyLogData(originalDailyLog)));

console.log(e);

NotificationManager.error(

"Сохранение заметки",

'[Ошибка на странице "дневник"]'

);

}

};

const handleDeleteNote = async (note: Note) => {

const diary: Diary = user.diary as Diary;

const dailyLogs: DailyLog[] = diary.dailyLogs;

try {

setDeletedNote(note);

const newNotes = dailyLogData.notes.filter(

(\_note) => \_note.createdAt !== note.createdAt

);

const newDailyLog: DailyLog = {

...originalDailyLog,

notes: newNotes,

};

const newDiary: Diary = {

...diary,

dailyLogs: [

...dailyLogs.filter(

(dailyLog) =>

dayjs(dailyLog.createdAt).format("l") !== currentDate.format("l")

),

newDailyLog,

],

};

const newUser = {

...user,

diary: newDiary,

};

await firebaseRepositories.users.updateDoc(newUser);

await dispatch(fetchUser(newUser.docId));

} catch (e) {

dispatch(setDailyLog(convertDailyLogToDailyLogData(originalDailyLog)));

console.log(e);

NotificationManager.error(

"Сохранение заметки",

'[Ошибка на странице "дневник"]'

);

} finally {

setDeletedNote(null);

}

};

const onOpen = () => {

setIsOpen(true);

};

const handleOpenNoteModal = (note: Note) => {

setSelectedNote(note);

setIsOpen(true);

};

const handleCloseNoteModal = () => {

setIsOpen(false);

setTimeout(() => setSelectedNote(null), 300);

};

return (

<PageContainer className="diary-page-container">

<div className="diary-page-content">

<CardContainer className="calendar-container">

<DateCalendar

disabled={isEditMode}

value={currentDate}

onChange={onDatePickerChange}

/>

</CardContainer>

{!isEditMode ? (

<DailyLogDataView

dailyLogData={dailyLogData}

handleEditButtonClick={handleEditButtonClick}

isEditable

/>

) : (

<div className="edit-form-container">

<form className="edit-form" onSubmit={handleSubmit}>

<TextField

label="Уровень сахара (ммоль/л)"

name="sugarLevel"

value={

dailyLogData.sugarLevel === "0" ? "" : dailyLogData.sugarLevel

}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.sugarLevel}

helperText={errors.sugarLevel}

disabled={isFormSaving}

/>

<TextField

label="Пульс (уд/мин)"

name="pulse"

value={dailyLogData.pulse === "0" ? "" : dailyLogData.pulse}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.pulse}

helperText={errors.pulse}

disabled={isFormSaving}

/>

<TextField

label="Систолическое давление (ммрт)"

name="systolic"

value={

dailyLogData.systolic === "0" ? "" : dailyLogData.systolic

}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.systolic}

helperText={errors.systolic}

disabled={isFormSaving}

/>

<TextField

label="Диастолическое давление (ммрт)"

name="diastolic"

value={

dailyLogData.diastolic === "0" ? "" : dailyLogData.diastolic

}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.diastolic}

helperText={errors.diastolic}

disabled={isFormSaving}

/>

<TextField

label="Количество каллорий (ккал)"

name="total"

value={dailyLogData.total === "0" ? "" : dailyLogData.total}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.total}

helperText={errors.total}

disabled={isFormSaving}

/>

<TextField

label="Вес (кг)"

name="weight"

value={dailyLogData.weight === "0" ? "" : dailyLogData.weight}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.weight}

helperText={errors.weight}

disabled={isFormSaving}

/>

<TextField

label="Температура (°C)"

name="temperature"

value={

dailyLogData.temperature === "0"

? ""

: dailyLogData.temperature

}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.temperature}

helperText={errors.temperature}

disabled={isFormSaving}

/>

{isFormSaving ? (

<LoadingSpinner style={{ marginTop: "20px" }} />

) : (

<div className="form-controls-container">

<Button

className="cancel-button"

variant="contained"

onClick={handleCancelButtonClick}

>

Отмена

</Button>

<Button

className="save-button"

type="submit"

variant="contained"

disabled={isSubmitDisabled}

>

Сохранить

</Button>

</div>

)}

</form>

</div>

)}

<CardContainer className="notes" title="Заметки">

<div className="notes-content">

{!dailyLogData.notes.length ? (

<div className="notes-empty">Заметок нет</div>

) : (

[...dailyLogData.notes]

.sort((left, right) =>

dayjs(left.createdAt).diff(dayjs(right.createdAt), "hour")

)

.map((note) => (

<CardContainer className="note" key={note.createdAt}>

{deepEqual(note, deletedNote ?? {}) ? (

<LoadingSpinner />

) : (

<>

<div className="first-line">

<div className="date">

{formatDate(note.createdAt)}

</div>

<div className="controls">

<EditButton

onClick={() => handleOpenNoteModal(note)}

/>

<IconButton

className="delete-button"

onClick={() => handleDeleteNote(note)}

>

<DeleteIcon />

</IconButton>

</div>

</div>

<div className="content">{note.content}</div>

</>

)}

</CardContainer>

))

)}

<div className="page-controls">

<Button

className="add-note-button"

onClick={onOpen}

variant="outlined"

startIcon={<AddCircleIcon />}

>

Добавить заметку

</Button>

</div>

</div>

<NoteModal

isOpen={isOpen}

onClose={handleCloseNoteModal}

submitNote={handleSubmitNote}

selectedNote={selectedNote}

/>

</CardContainer>

</div>

</PageContainer>

);

};

***helpers.ts***

export class ValidationNumber {

value: string;

isAllowedToBeEmpty: boolean;

error: string | null;

constructor(value: string, isAllowedToBeEmpty = true) {

this.value = value;

this.isAllowedToBeEmpty = isAllowedToBeEmpty;

}

isValueEmptyOrNull = () => {

return this.value === "" || +this.value === 0;

};

isEmpty = (message = "") => {

if (this.isAllowedToBeEmpty) {

return this;

}

if (this.value === "") {

this.error = message;

}

return this;

};

isNumber = (message: string) => {

if (this.isAllowedToBeEmpty && this.isValueEmptyOrNull()) {

return this;

}

if (isNaN(+this.value)) {

this.error = message;

}

return this;

};

min = (value: number, message: string) => {

if (this.isAllowedToBeEmpty && this.isValueEmptyOrNull()) {

return this;

}

if (!this.error && +this.value < value) {

this.error = message;

}

return this;

};

max = (value: number, message: string) => {

if (this.isAllowedToBeEmpty && this.isValueEmptyOrNull()) {

return this;

}

if (!this.error && +this.value > value) {

this.error = message;

}

return this;

};

getError = () => this.error;

}

export const dailyLogDataValidationObj: {

[key: string]: (s: string) => string | null;

} = {

sugarLevel: (sugarLevel: string) =>

new ValidationNumber(sugarLevel)

.isEmpty("Уровень сахара не может быть пустым")

.isNumber("Уровень сахара должен быть числом")

.min(1.0, "Уровень сахара должен быть больше 1.0 ммоль/л")

.max(10.0, "Уровень сахара должен быть меньше 10.0 ммоль/л")

.getError(),

pulse: (pulse: string) =>

new ValidationNumber(pulse)

.isEmpty("Пульс не может быть пустым")

.isNumber("Пульс должен быть числом")

.min(50, "Пульс должен быть больше 50 уд/мин")

.max(110, "Пульс должен быть меньше 110 уд/мин")

.getError(),

systolic: (systolic: string) =>

new ValidationNumber(systolic)

.isEmpty("Систолическое давление не может быть пустым")

.isNumber("Систолическое давление должно быть числом")

.min(90, "Систолическое давление должно быть больше 90 ммрт")

.max(150, "Систолическое давление должно быть меньше 150 ммрт")

.getError(),

diastolic: (diastolic: string) =>

new ValidationNumber(diastolic)

.isEmpty("Диастолическое давление не может быть пустым")

.isNumber("Диастолическое давление должно быть числом")

.min(50, "Диастолическое давление должно быть больше 50 ммрт")

.max(110, "Диастолическое давление должно быть меньше 110 ммрт")

.getError(),

total: (total: string) =>

new ValidationNumber(total)

.isEmpty("Количество каллорий не может быть пустым")

.isNumber("Количество каллорий должно быть числом")

.min(200, "Каллорий должно быть больше 200")

.max(10000, "Каллорий должно быть меньше 10000")

.getError(),

weight: (weight: string) =>

new ValidationNumber(weight)

.isEmpty("Вес не может быть пустым")

.isNumber("Вес должен быть числом")

.min(40, "Вес должен быть больше 40 кг")

.max(200, "Вес должен быть меньше 200 кг")

.getError(),

temperature: (temperature: string) =>

new ValidationNumber(temperature)

.isEmpty("Температура не может быть пустой")

.isNumber("Температура должна быть числом")

.min(34, "Температура должна быть больше 34 °C")

.max(42, "Температура должна быть меньше 42 °C")

.getError(),

};

***useDailyLog.ts***

import { DailyLog, User } from "firestore/types/collections.types";

import { useSelector } from "react-redux";

import { GlobalState } from "store";

import dayjs from "dayjs";

import { useMemo } from "react";

export const createDailyLog = (date: dayjs.Dayjs): DailyLog => {

return {

createdAt: date.toString(),

takenMedications: [],

blood: {

sugarLevel: 0,

pulse: 0,

pressure: {

systolic: 0,

diastolic: 0,

},

},

calories: {

total: 0,

carbohydratesIntake: 0,

},

temperature: 0,

weight: 0,

notes: [],

};

};

export const useDailyLog = (date: dayjs.Dayjs) => {

const user = useSelector<GlobalState, User>(

(state) => state.currentUser.user as User

);

return useMemo(() => {

const dailyLogs: DailyLog[] = user.diary?.dailyLogs as DailyLog[];

const selectedLog = dailyLogs.find(

(log) => dayjs(log.createdAt).format("l") === date.format("l")

);

return selectedLog ?? createDailyLog(date);

}, [user, date]);

};

***useDailyLogData.ts***

import { DailyLog } from "firestore/types/collections.types";

import { useCallback, useEffect, useState } from "react";

import {

convertDailyLogToDailyLogData,

DailyLogData,

} from "firestore/converters";

export const useDailyLogData = (dailyLog: DailyLog) => {

const [dailyLogData, setDailyLogData] = useState<DailyLogData>(

convertDailyLogToDailyLogData(dailyLog)

);

const resetDailyLogData = useCallback(() => {

setDailyLogData(convertDailyLogToDailyLogData(dailyLog));

}, [setDailyLogData, dailyLog]);

// effects

useEffect(resetDailyLogData, [dailyLog, resetDailyLogData]);

return { dailyLogData, setDailyLogData, resetDailyLogData };

};

***types.ts***

export type DiaryFormErrors = {

sugarLevel: string | null;

pulse: string | null;

systolic: string | null;

diastolic: string | null;

total: string | null;

weight: string | null;

temperature: string | null;

};

***DoctorDescription.jsx***

import React, { useEffect, useState } from "react";

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

import { CardContainer } from "../../../ui/CardContainer";

import { isMobile } from "react-device-detect";

import { EditButton } from "../../../ui/EditButton";

import { Button, TextField } from "@mui/material";

export const DoctorDescription = ({ doctor, changeDescription }) => {

const [description, setDescription] = useState("");

const [isEditing, setIsEditing] = useState(false);

useEffect(() => {

setDescription(doctor.employee.description);

}, [doctor.employee.description]);

const onSave = () => {

changeDescription(description);

setIsEditing(false);

};

const onCancel = () => {

setDescription(doctor.employee.description);

setIsEditing(false);

};

return (

<CardContainer

sx={{ width: isMobile ? undefined : "900px", padding: "20px" }}

>

<Typography sx={{ textAlign: "center" }} variant="h6">

<div style={{ display: "flex", justifyContent: "center" }}>

<div>Описание</div>

{!isEditing && Boolean(changeDescription) && (

<EditButton onClick={() => setIsEditing(true)} />

)}

</div>

</Typography>

{!isEditing ? (

<Typography>{doctor.employee.description}</Typography>

) : (

<div style={{ display: "flex", flexDirection: "column", gap: "10px" }}>

<TextField

multiline

minRows={3}

maxRows={5}

label="Описание"

value={description}

onChange={({ target: { value } }) => setDescription(value)}

fullWidth

/>

<div

style={{ display: "flex", justifyContent: "center", gap: "10px" }}

>

<Button color="primary" variant="outlined" onClick={onCancel}>

Отмена

</Button>

<Button

color="primary"

variant="outlined"

onClick={onSave}

disabled={doctor.employee.description === description}

>

Сохранить

</Button>

</div>

</div>

)}

</CardContainer>

);

};

***DoctorDetails.jsx***

import React from "react";

import { DoctorDescription } from "../DoctorDescription";

import { DoctorReviews } from "../DoctorReviews";

import "./doctor-details.scss";

export const DoctorDetails = ({ doctor, editReview, deleteReview }) => {

return (

<div className="doctor-details">

<DoctorDescription doctor={doctor} />

<DoctorReviews

reviews={doctor.employee.reviews}

editReview={editReview}

deleteReview={deleteReview}

/>

</div>

);

};

***DoctorPage.jsx***

import React, { useCallback, useEffect, useState } from "react";

import { PageContainer } from "components/layout";

import { useSelector } from "react-redux";

import { NotificationManager } from "react-notifications";

import { getUserSelector } from "../../../store/selectors";

import { firebaseRepositories } from "../../../firestore/data/repositories";

import { where } from "firebase/firestore";

import { LoadingSpinner } from "../../ui/LoadingSpinner";

import { Role } from "../../../firestore/types/collections.types";

import { CardContainer } from "../../ui/CardContainer";

import { getUserFullName } from "../../../firestore/helpers";

import { convertRoleToRussian } from "../../../firestore/converters";

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

import { useAppDispatch } from "../../../store";

import { fetchUser } from "../../../store/reducers/user/actions";

import { setUser } from "../../../store/reducers/user/userSlice";

import { DoctorView } from "./DoctorView";

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

import { DoctorDetails } from "./DoctorDetails";

import { ReviewEditModal } from "./ReviewEditModal";

import { useGeneralModalHandlers } from "../../../hooks/useGeneralModalHandlers";

import "./doctor.scss";

export const DoctorPage = () => {

const user = useSelector(getUserSelector);

const dispatch = useAppDispatch();

const [isLoading, setIsLoading] = useState(false);

const [isRemoveLoading, setIsRemoveLoading] = useState(false);

const [isAcceptLoading, setIsAcceptLoading] = useState(false);

const [acceptedDoctor, setAcceptedDoctor] = useState(null);

const [doctors, setDoctors] = useState([]);

const loadDoctors = useCallback(async () => {

try {

setIsLoading(true);

const newDoctors = await firebaseRepositories.users.getDocs(

where("role", "==", Role.DOCTOR)

);

setDoctors(newDoctors);

} catch (e) {

console.log(e);

NotificationManager.error("Загрузка докторов", 'Страница "Доктор"');

} finally {

setIsLoading(false);

}

}, []);

useEffect(() => {

loadDoctors();

}, [loadDoctors]);

const [selectedDoctor, setSelectedDoctor] = useState(null);

const updateSelectedDoctor = useCallback(() => {

if (user.doctor) {

const newDoctor = doctors.find((doctor) => doctor.docId === user.doctor);

setSelectedDoctor(newDoctor);

} else {

setSelectedDoctor(null);

}

}, [user.doctor, doctors]);

useEffect(updateSelectedDoctor, [updateSelectedDoctor]);

const denyDoctor = async () => {

try {

setIsRemoveLoading(true);

await firebaseRepositories.users.updateDoc({

...user,

doctor: "",

});

await dispatch(fetchUser(user.docId));

} catch (e) {

console.log(e);

NotificationManager.error("Отказ от доктора", "Страница доктор");

dispatch(setUser(user));

} finally {

setIsRemoveLoading(false);

}

};

const acceptDoctor = async (uid) => {

try {

setIsAcceptLoading(true);

setAcceptedDoctor(uid);

await firebaseRepositories.users.updateDoc({

...user,

doctor: uid,

});

await dispatch(fetchUser(user.docId));

} catch (e) {

console.log(e);

NotificationManager.error("Выбор доктора", "Страница доктор");

dispatch(setUser(user));

} finally {

setIsAcceptLoading(false);

setAcceptedDoctor(null);

}

};

const editReview = async (updatedReview) => {

try {

setIsLoading(true);

let newReviews = selectedDoctor.employee.reviews.map((review) => {

if (review.reviewer !== updatedReview.reviewer) {

return review;

}

return updatedReview;

});

if (

!newReviews.find((review) => review.reviewer === updatedReview.reviewer)

) {

newReviews = [...newReviews, updatedReview];

}

const newDoctor = {

...selectedDoctor,

employee: {

...selectedDoctor.employee,

reviews: newReviews,

},

};

await firebaseRepositories.users.updateDoc(newDoctor);

await loadDoctors();

} catch (e) {

console.log(e);

NotificationManager.error("Изменение отзыва", "Страница доктор");

} finally {

setIsLoading(false);

}

};

const deleteReview = async (reviewToDelete) => {

try {

setIsLoading(true);

let newReviews = selectedDoctor.employee.reviews.filter(

(review) => review.reviewer !== reviewToDelete.reviewer

);

const newDoctor = {

...selectedDoctor,

employee: {

...selectedDoctor.employee,

reviews: newReviews,

},

};

await firebaseRepositories.users.updateDoc(newDoctor);

await loadDoctors();

} catch (e) {

console.log(e);

NotificationManager.error("Удаление отзыва", "Страница доктор");

} finally {

setIsLoading(false);

}

};

const [selectedReview, setSelectedReview] = useState(null);

const [isReviewModalOpened, openReviewModal, closeReviewModal] =

useGeneralModalHandlers({

onOpen: (review) => {

setSelectedReview(review);

},

onClose: () => {

setSelectedReview(null);

},

});

return (

<PageContainer className="doctor-page">

{isLoading ? (

<LoadingSpinner />

) : selectedDoctor ? (

<div className="selected-doctor-container">

<CardContainer className="avatar">

<img

className="avatar"

style={{ objectFit: "fill" }}

src={

selectedDoctor.imageUrl ||

"https://thumbs.dreamstime.com/b/default-avatar-profile-icon-vector-social-media-user-image-182145777.jpg"

}

alt={getUserFullName(selectedDoctor)}

/>

</CardContainer>

<CardContainer className="doctor-page-info" title="Выбранный доктор">

<div className="title">

<h2>{getUserFullName(selectedDoctor)}</h2>

</div>

<div>Эл. почта: {selectedDoctor.email}</div>

<div>Телефон: {selectedDoctor.phone}</div>

<div>Адрес: {selectedDoctor.address}</div>

<div>

Тип учётной записи: {convertRoleToRussian(selectedDoctor.role)}

</div>

<div className="role-logo-container">

<img

src={`/images/logos/${selectedDoctor.role}-logo.png`}

alt="Role logo"

/>

</div>

{isRemoveLoading ? (

<LoadingSpinner style={{ margin: "10px auto", width: "250px" }} />

) : (

<Button

variant="outlined"

sx={{

color: "red",

borderColor: "red",

width: "250px",

margin: "10px auto",

fontSize: "11pt",

}}

onClick={denyDoctor}

>

Отказаться от доктора

</Button>

)}

</CardContainer>

<DoctorDetails

doctor={selectedDoctor}

editReview={openReviewModal}

deleteReview={deleteReview}

/>

<ReviewEditModal

isOpen={isReviewModalOpened}

onClose={closeReviewModal}

selectedReview={selectedReview}

submitReview={editReview}

/>

</div>

) : (

<div className="doctors-list">

<Typography className="list-title" variant="h5">

Доктора

</Typography>

{doctors.map((doctor) => (

<CardContainer key={doctor.docId} className="doctor-row">

<div className="content">

<DoctorView doctor={doctor} />

<DoctorDetails doctor={doctor} />

</div>

<div className="controls">

<Button

className="select-doctor-button"

variant="outlined"

onClick={() => acceptDoctor(doctor.docId)}

>

{isAcceptLoading && acceptedDoctor === doctor.docId ? (

<LoadingSpinner />

) : (

"Выбрать доктора"

)}

</Button>

</div>

</CardContainer>

))}

</div>

)}

</PageContainer>

);

};

***DoctorReviews.jsx***

import React, { useMemo, useState } from "react";

import { CardContainer } from "../../../ui/CardContainer";

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

import {

Button,

IconButton,

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

} from "@mui/material";

import { formatDate } from "../../../../helpers/helpers";

import useAsyncEffect from "use-async-effect";

import { NotificationManager } from "react-notifications";

import { firebaseRepositories } from "../../../../firestore/data/repositories";

import { LoadingSpinner } from "../../../ui/LoadingSpinner";

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

import { getUserFullName } from "../../../../firestore/helpers";

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

import ReactStars from "react-stars/dist/react-stars";

import { EditButton } from "../../../ui/EditButton";

import { useSelector } from "react-redux";

import {

getUserIdSelector,

getUserSelector,

} from "../../../../store/selectors";

import DeleteIcon from "@mui/icons-material/Delete";

import dayjs from "dayjs";

import { Role } from "firestore/types/collections.types";

import "./doctor-reviews.scss";

export const DoctorReviews = ({ reviews, editReview, deleteReview }) => {

const userId = useSelector(getUserIdSelector);

const currentUser = useSelector(getUserSelector);

const [isLoading, setIsLoading] = useState(false);

const [users, setUsers] = useState([]);

useAsyncEffect(async () => {

try {

setIsLoading(true);

const newUsers = await firebaseRepositories.users.getDocs();

setUsers(newUsers);

} catch (e) {

console.log(e);

NotificationManager.error(

"Загрузка пользователей",

"Таблица отзывов доктора"

);

} finally {

setIsLoading(false);

}

}, []);

const sortedReviews = useMemo(() => {

const firstReview = [reviews.find((review) => review.reviewer === userId)];

const restReviews = reviews.filter((review) => review.reviewer !== userId);

return [

...firstReview.filter(Boolean),

...restReviews.sort((leftReview, rightReview) =>

dayjs(leftReview.createdAt).diff(dayjs(rightReview.createdAt), "second")

),

];

}, [reviews, userId]);

return (

<CardContainer className="doctor-reviews">

{isLoading || !users.length ? (

<LoadingSpinner />

) : (

<TableContainer component={Paper}>

<Typography sx={{ textAlign: "center" }} variant="h6">

Отзывы

</Typography>

{currentUser.role === Role.PATIENT &&

!sortedReviews.find((review) => review.reviewer === userId) &&

Boolean(editReview) && (

<div style={{ display: "flex", justifyContent: "center" }}>

<Button

color="success"

variant="contained"

onClick={() => editReview(null)}

>

Добавить отзыв

</Button>

</div>

)}

<Table sx={{ maxWidth: 900 }} aria-label="simple table">

<TableHead>

<TableRow>

<TableCell>Оценка</TableCell>

<TableCell>Дата</TableCell>

<TableCell>Пациент</TableCell>

<TableCell>Отзыв</TableCell>

<TableCell></TableCell>

</TableRow>

</TableHead>

<TableBody>

{sortedReviews.map((review) => {

const reviewer = users.find(

(user) => user.docId === review.reviewer

);

return (

<TableRow

key={formatDate(review.createdAt)}

sx={{ "&:last-child td, &:last-child th": { border: 0 } }}

>

<TableCell sx={{ minWidth: 100 }}>

<ReactStars value={review.score} edit={false} />

</TableCell>

<TableCell sx={{ minWidth: 200 }}>

{formatDate(review.createdAt)}

</TableCell>

<TableCell>

<div

style={{

display: "flex",

gap: "10px",

alignItems: "center",

}}

>

<Avatar

src={reviewer.imageUrl}

alt={getUserFullName(reviewer)}

/>

<Typography noWrap>

{getUserFullName(reviewer)}

</Typography>

</div>

</TableCell>

<TableCell>{review.content}</TableCell>

<TableCell>

{currentUser.role === Role.PATIENT &&

review.reviewer === userId &&

Boolean(editReview) &&

Boolean(deleteReview) && (

<div style={{ display: "flex", gap: "10px" }}>

<EditButton onClick={() => editReview(review)} />

<IconButton

color="error"

onClick={() => deleteReview(review)}

>

<DeleteIcon />

</IconButton>

</div>

)}

</TableCell>

</TableRow>

);

})}

</TableBody>

</Table>

</TableContainer>

)}

</CardContainer>

);

};

***DoctorView.jsx***

import React from "react";

import { getUserFullName } from "../../../../firestore/helpers";

import ReactStars from "react-stars";

import { CardContainer } from "../../../ui/CardContainer";

export const DoctorView = ({ doctor }) => {

return (

<CardContainer className="doctor-view" sx={{ padding: "10px" }}>

<div className="avatar">

<img

src={doctor.imageUrl}

alt="doctor avatar"

width="200px"

height="200px"

/>

</div>

<div className="fullname">{getUserFullName(doctor)}</div>

<div className="score">{doctor.score}</div>

<ReactStars

edit={false}

size={24}

value={

doctor.employee.reviews

.map(({ score }) => score)

.reduce((a, b) => a + b, 0) / doctor.employee.reviews.length

}

/>

</CardContainer>

);

};

***ReviewEditModal.jsx***

import React, { useCallback, useEffect, useState } from "react";

import {

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

TextField,

} from "@mui/material";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { deepCopy } from "deep-copy-ts";

import { formatDate } from "helpers/helpers";

import { isMobile } from "react-device-detect";

import dayjs from "dayjs";

import "./review-modal.scss";

import { useSelector } from "react-redux";

import { getUserInfoSelector } from "../../../../store/selectors";

import ReactStars from "react-stars";

export const ReviewEditModal = ({

isOpen,

onClose,

submitReview,

selectedReview,

}) => {

const user = useSelector(getUserInfoSelector);

const getReview = useCallback(

() =>

selectedReview

? deepCopy(selectedReview)

: {

createdAt: dayjs().toDate().toString(),

score: 0,

reviewer: user.docId,

content: "",

},

[selectedReview, user.docId]

);

const [review, setReview] = useState(getReview());

const [isLoading, setIsLoading] = useState(false);

const resetReview = useCallback(() => {

setReview(getReview());

}, [setReview, getReview]);

const handleFieldChange = (e) => {

const { name, value } = e.target;

setReview((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleSubmitReview = async (event) => {

event.preventDefault();

setIsLoading(true);

await submitReview({

...review,

createdAt: dayjs().toDate().toString(),

});

setIsLoading(false);

onClose();

};

useEffect(() => {

resetReview();

}, [selectedReview, isOpen, resetReview]);

return (

<Dialog open={isOpen} onClose={onClose}>

<DialogTitle>

{(selectedReview ? "Редактирование" : "Добавление") + " отзыва"}

</DialogTitle>

<DialogContent

className="modal-content"

sx={{

width: isMobile ? "300px" : "600px",

}}

>

<form className="review-form" onSubmit={handleSubmitReview}>

<div style={{ display: "flex", gap: "10px", alignItems: "center" }}>

Оценка:

<ReactStars

value={review.score}

size={32}

onChange={(value) =>

handleFieldChange({ target: { name: "score", value } })

}

/>

</div>

<TextField

label="Дата"

name="createdAt"

value={formatDate(review.createdAt)}

disabled={true}

/>

<TextField

multiline

minRows={3}

maxRows={3}

label="Отзыв"

name="content"

value={review.content}

onChange={handleFieldChange}

disabled={isLoading}

/>

<DialogActions className="controls">

<Button onClick={onClose} disabled={isLoading} variant="outlined">

Отмена

</Button>

{isLoading ? (

<LoadingSpinner style={{ margin: "0", width: "100px" }} />

) : (

<Button

type="submit"

variant="outlined"

disabled={

selectedReview?.content === review.content &&

selectedReview?.score === review.score

}

autoFocus

>

{selectedReview ? "Изменить" : "Добавить"}

</Button>

)}

</DialogActions>

</form>

</DialogContent>

</Dialog>

);

};

***HealthStateModal.jsx***

import React, { useCallback, useEffect, useState } from "react";

import {

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

TextField,

} from "@mui/material";

import { isMobile } from "react-device-detect";

import { LoadingSpinner } from "../../../ui/LoadingSpinner";

import { deepCopy } from "deep-copy-ts";

import { generateDocId } from "firestore/helpers";

import { convertDailyLogPropNameToRussian } from "../../../../firestore/converters";

export const HealthStateModal = ({

isOpen,

onClose,

submitHealthState,

healthStateToEdit,

propName,

}) => {

const getHealthState = useCallback(

() =>

healthStateToEdit

? deepCopy(healthStateToEdit)

: {

title: "",

min: "",

max: "",

warning: "",

recommendation: "",

},

[healthStateToEdit]

);

const [healthState, setHealthState] = useState(getHealthState());

const [isLoading, setIsLoading] = useState(false);

const resetHealthState = useCallback(() => {

setHealthState(getHealthState());

}, [setHealthState, getHealthState]);

const handleFieldChange = (e) => {

const { name, value } = e.target;

setHealthState((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleSubmitHealthState = async (event) => {

event.preventDefault();

setIsLoading(true);

await submitHealthState({

docId: healthStateToEdit?.docId ?? generateDocId(),

...healthState,

min: +healthState.min ?? 0,

max: +healthState.max ?? 0,

propName,

});

setIsLoading(false);

onClose();

};

useEffect(resetHealthState, [healthStateToEdit, isOpen, resetHealthState]);

return (

<Dialog open={isOpen} onClose={onClose}>

<DialogTitle>

{(healthStateToEdit ? "Редактирование" : "Добавление") + " назначения"}

</DialogTitle>

<DialogContent

className="modal-content"

sx={{

width: isMobile ? "300px" : "600px",

}}

>

<form

className="health-state-form"

style={{

padding: "10px",

display: "flex",

flexDirection: "column",

flexWrap: "wrap",

gap: "20px",

alignItems: "center",

}}

onSubmit={handleSubmitHealthState}

>

<TextField

label="Название"

name="title"

value={healthState.title}

onChange={handleFieldChange}

disabled={isLoading}

fullWidth

/>

<TextField

label="Показатель"

name="propName"

value={convertDailyLogPropNameToRussian(propName)}

disabled={true}

fullWidth

/>

<TextField

label="Минимум"

name="min"

value={healthState.min}

onChange={handleFieldChange}

disabled={isLoading}

fullWidth

/>

<TextField

label="Минимум"

name="max"

value={healthState.max}

onChange={handleFieldChange}

disabled={isLoading}

fullWidth

/>

<TextField

multiline

minRows={3}

maxRows={4}

label="Предупреждение"

name="warning"

value={healthState.warning}

onChange={handleFieldChange}

disabled={isLoading}

fullWidth

/>

<TextField

multiline

minRows={3}

maxRows={4}

label="Рекоммендация"

name="recommendation"

value={healthState.recommendation}

onChange={handleFieldChange}

disabled={isLoading}

fullWidth

/>

<DialogActions

className="controls"

sx={{ width: "100%", display: "flex", alignItems: "flexEnd" }}

>

<Button onClick={onClose} disabled={isLoading} variant="outlined">

Отмена

</Button>

{isLoading ? (

<LoadingSpinner style={{ margin: "0", width: "100px" }} />

) : (

<Button

type="submit"

variant="outlined"

disabled={

healthStateToEdit?.warning === healthState.warning &&

healthStateToEdit?.recommendation ===

healthState.recommendation &&

healthStateToEdit?.min === healthState.min &&

healthStateToEdit?.max === healthState.max &&

healthStateToEdit?.title === healthState.title

}

autoFocus

>

{healthStateToEdit ? "Изменить" : "Добавить"}

</Button>

)}

</DialogActions>

</form>

</DialogContent>

</Dialog>

);

};

***HealthStatesPage.jsx***

import React, { useMemo, useState } from "react";

import { PageContainer } from "../../layout";

import { firebaseRepositories } from "../../../firestore/data/repositories";

import { useGeneralDataHook } from "../../../hooks/useGeneralDataHook";

import { LoadingSpinner } from "../../ui/LoadingSpinner";

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

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

import {

Button,

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

} from "@mui/material";

import {

convertDailyLogPropNameToMeasurement,

convertDailyLogPropNameToRussian,

} from "../../../firestore/converters";

import { CardContainer } from "../../ui/CardContainer";

import "./health-states.scss";

import { HealthStateModal } from "./components/HealthStateModal";

export const HealthStatesPage = () => {

const [isLoading, healthStates, reset] = useGeneralDataHook(

async () => firebaseRepositories.healthStates.getDocs(),

[]

);

const groupedHealthStates = useMemo(() => {

if (!healthStates) {

return [];

}

return healthStates.reduce((acc, healthState) => {

if (!acc[healthState.propName]) {

acc[healthState.propName] = [];

}

acc[healthState.propName] = [...acc[healthState.propName], healthState];

return acc;

}, {});

}, [healthStates]);

// modal

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

const [healthStateToEdit, setHealthStateToEdit] = useState(null);

const [propName, setPropName] = useState(null);

const submitHealthState = async (healthState) => {

await firebaseRepositories.healthStates.updateDoc(healthState);

await reset();

};

const deleteHealthState = async (healthState) => {

await firebaseRepositories.healthStates.deleteDocById(healthState.docId);

await reset();

};

const onOpenAdd = (\_propName) => {

setHealthStateToEdit(null);

setPropName(\_propName);

setIsOpen(true);

};

const onOpenEdit = (healthState) => {

setHealthStateToEdit(healthState);

setPropName(healthState.propName);

setIsOpen(true);

};

const onClose = () => {

setIsOpen(false);

setHealthStateToEdit(null);

setPropName(null);

};

return (

<PageContainer className="health-states-page">

{isLoading && <LoadingSpinner />}

{!isLoading && Object.entries(groupedHealthStates).length && (

<div className="health-states-container">

{Object.entries(groupedHealthStates).map(

([\_propName, \_healthStates]) => (

<CardContainer key={\_propName}>

<TableContainer component={Paper}>

<Typography sx={{ textAlign: "center" }} variant="h6">

{convertDailyLogPropNameToRussian(\_propName)}

</Typography>

<div className="add-button">

<Button

variant="contained"

color="success"

onClick={() => onOpenAdd(\_propName)}

>

Добавить

</Button>

</div>

<Table>

<TableHead>

<TableRow>

<TableCell>Название</TableCell>

<TableCell sx={{ minWidth: "120px" }}>

Минимум

</TableCell>

<TableCell sx={{ minWidth: "120px" }}>

Максимум

</TableCell>

<TableCell>Предупреждение</TableCell>

<TableCell>Назначение</TableCell>

<TableCell>Управление</TableCell>

</TableRow>

</TableHead>

<TableBody>

{\_healthStates.map((healthState) => (

<TableRow

key={healthState.docId}

sx={{

"&:last-child td, &:last-child th": { border: 0 },

}}

>

<TableCell>{healthState.title}</TableCell>

<TableCell>

{healthState.min}{" "}

{convertDailyLogPropNameToMeasurement(

healthState.propName

)}

</TableCell>

<TableCell>

{healthState.max}{" "}

{convertDailyLogPropNameToMeasurement(

healthState.propName

)}

</TableCell>

<TableCell>{healthState.warning}</TableCell>

<TableCell>{healthState.recommendation}</TableCell>

<TableCell>

<div className="controls">

<Button

variant="contained"

color="warning"

onClick={() => onOpenEdit(healthState)}

>

Редактировать

</Button>

<Button

variant="contained"

color="error"

onClick={() => deleteHealthState(healthState)}

>

Удалить

</Button>

</div>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

</CardContainer>

)

)}

</div>

)}

<HealthStateModal

isOpen={isOpen}

onClose={onClose}

submitHealthState={submitHealthState}

healthStateToEdit={healthStateToEdit}

propName={propName}

/>

</PageContainer>

);

};

***HomePage.tsx***

import React from "react";

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

import { Route } from "components/routing";

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

import "./home.scss";

export const HomePage = () => {

const navigate = useNavigate();

return (

<div className="home-page">

<h2 className="title">Добро пожаловать в дневник диабетика</h2>

<div className="logo">

<img

src="/logo512.png"

alt="home-page-logo"

width="300px"

height="300px"

/>

</div>

<div className="controls">

<Button variant="outlined" onClick={() => navigate(Route.signIn)}>

Войти

</Button>

<Button variant="outlined" onClick={() => navigate(Route.signUp)}>

Зарегистрироваться

</Button>

</div>

</div>

);

};

***NotFoundPage.tsx***

import React from "react";

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

import { Route } from "components/routing";

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

import { getUserSelector } from "store/selectors";

import { useSelector } from "react-redux";

import "./not-found.scss";

export const NotFoundPage = () => {

const navigate = useNavigate();

const user = useSelector(getUserSelector);

return (

<div className="not-found-page">

<h2 className="title">Страница не найдена :(</h2>

<div className="image">

<img

src="https://cdn1.specialist.ru/Content/Image/News/Small/404-error-s.jpg"

alt="not-found-image"

width="300px"

height="300px"

/>

</div>

<div className="controls">

{user ? (

<Button variant="outlined" onClick={() => navigate(Route.profile)}>

Вернуться в профиль

</Button>

) : (

<Button variant="outlined" onClick={() => navigate(Route.home)}>

Вернуться на главную

</Button>

)}

</div>

</div>

);

};

***NotificationDetails.jsx***

import React from "react";

import { CardContainer } from "../../../../ui/CardContainer";

import {

convertDailyLogPropNameToMeasurement,

convertDailyLogPropNameToRussian,

} from "../../../../../firestore/converters";

import { calculatePatientReportDeviation } from "../../../../../firestore/helpers";

import "./notification-details.scss";

export const NotificationDetails = ({ notification }) => {

return (

<div className="notification-details">

{notification.patientReports.map((patientReport) => (

<>

<CardContainer

className="patient-report"

key={notification.docId + patientReport.propName}

>

<div className="prop-name">

{convertDailyLogPropNameToRussian(patientReport.propName)}:

</div>

<div className="current-value">

{patientReport.currentValue}{" "}

{convertDailyLogPropNameToMeasurement(patientReport.propName)}

</div>

</CardContainer>

<div

style={{

fontSize: "13px",

textAlign: "center",

fontStyle: "italic",

}}

>

Отклонение от нормы:{" "}

{calculatePatientReportDeviation(

patientReport.propName,

patientReport.currentValue

)}

{" " + convertDailyLogPropNameToMeasurement(patientReport.propName)}

</div>

</>

))}

</div>

);

};

***RecommendationDetailsModal.jsx***

import React, { useMemo } from "react";

import {

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

} from "@mui/material";

import { CardContainer } from "../../../../ui/CardContainer";

import { convertDailyLogPropNameToMeasurement } from "../../../../../firestore/converters";

import "./recommendation-details-modal.scss";

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

import { formatDate } from "../../../../../helpers/helpers";

export const RecommendationDetailsModal = ({

isOpen,

onClose,

recommendation,

}) => {

const healthStates = useMemo(() => {

if (recommendation?.healthStates) {

return recommendation.healthStates;

}

return [];

}, [recommendation]);

return (

<Dialog

open={isOpen}

onClose={onClose}

className="recommendation-details-dialog"

>

<DialogTitle sx={{ textAlign: "center", marginBottom: "-10px" }}>

Подробности рекомендации

</DialogTitle>

<DialogContent

className="modal-content"

sx={{

paddingBottom: "20px",

minWidth: "400px",

maxWidth: "1400px",

overflowY: "auto",

}}

>

<div className="recommendation-details-modal-content">

<div

style={{

margin: "0 auto",

p: 1,

width: "300px",

display: "flex",

flexDirection: "column",

alignItems: "center",

}}

>

<Typography

sx={{ textAlign: "center" }}

variant="h6"

component="div"

>

Дата

</Typography>

<CardContainer

style={{ padding: "5px", width: "250px", textAlign: "center" }}

>

{Boolean(recommendation?.createdAt) &&

formatDate(recommendation?.createdAt)}

</CardContainer>

</div>

<Typography

sx={{ textAlign: "center", marginTop: "10px" }}

variant="h6"

component="div"

>

Назначения

</Typography>

{!Boolean(healthStates.length) && (

<CardContainer

style={{

padding: "10px",

textAlign: "center",

}}

>

Назначений нет.

</CardContainer>

)}

{Boolean(healthStates.length) && (

<TableContainer component={CardContainer}>

<Table>

<TableHead>

<TableRow>

<TableCell>Название</TableCell>

<TableCell sx={{ minWidth: "120px" }}>Минимум</TableCell>

<TableCell sx={{ minWidth: "120px" }}>Максимум</TableCell>

<TableCell>Предупреждение</TableCell>

<TableCell>Назначение</TableCell>

</TableRow>

</TableHead>

<TableBody>

{healthStates.map((healthState) => (

<TableRow

key={healthState.docId}

sx={{

"&:last-child td, &:last-child th": { border: 0 },

}}

>

<TableCell>{healthState.title}</TableCell>

<TableCell>

{healthState.min}{" "}

{convertDailyLogPropNameToMeasurement(

healthState.propName

)}

</TableCell>

<TableCell>

{healthState.max}{" "}

{convertDailyLogPropNameToMeasurement(

healthState.propName

)}

</TableCell>

<TableCell>{healthState.warning}</TableCell>

<TableCell>{healthState.recommendation}</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

)}

<div

style={{

margin: "20px auto",

p: 1,

width: "300px",

display: "flex",

flexDirection: "column",

alignItems: "center",

}}

>

<Typography

sx={{ textAlign: "center" }}

variant="h6"

component="div"

>

Комментарий

</Typography>

<CardContainer

style={{ padding: "5px", width: "250px", textAlign: "center" }}

>

{Boolean(recommendation?.comment)

? recommendation?.comment

: "Комментарий пуст."}

</CardContainer>

</div>

</div>

</DialogContent>

<DialogActions className="controls">

<Button onClick={onClose} variant="outlined">

Отмена

</Button>

</DialogActions>

</Dialog>

);

};

***HealthStatesChip.jsx***

import \* as React from "react";

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

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

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

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

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

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

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

const ITEM\_HEIGHT = 48;

const ITEM\_PADDING\_TOP = 8;

const MenuProps = {

PaperProps: {

style: {

maxHeight: ITEM\_HEIGHT \* 4.5 + ITEM\_PADDING\_TOP,

width: 250,

},

},

};

export const HealthStatesChip = ({

healthStatesOptions,

healthStates,

setHealthStates,

}) => {

const handleChange = (event) => {

const {

target: { value },

} = event;

setHealthStates(

// On autofill we get a stringified value.

typeof value === "string" ? value.split(",") : value

);

};

return (

<FormControl fullWidth>

<InputLabel id="demo-multiple-chip-label">Назначения</InputLabel>

<Select

labelId="demo-multiple-chip-label"

id="demo-multiple-chip"

multiple

value={healthStates}

onChange={handleChange}

input={<OutlinedInput id="select-multiple-chip" label="Назначения" />}

renderValue={(selected) => (

<Box sx={{ display: "flex", flexWrap: "wrap", gap: 0.5 }}>

{selected.map((value) => {

const selectedHealthState = healthStatesOptions.find(

({ docId }) => docId === value

);

if (!selectedHealthState) {

return null;

}

return <Chip key={value} label={selectedHealthState.title} />;

})}

</Box>

)}

MenuProps={MenuProps}

>

{healthStatesOptions.map((healthState) => (

<MenuItem key={healthState.docId} value={healthState.docId}>

{healthState.title}

</MenuItem>

))}

</Select>

</FormControl>

);

};

***RecommendationModal.jsx***

import React, { useCallback, useEffect, useState } from "react";

import {

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

TextField,

} from "@mui/material";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { deepCopy } from "deep-copy-ts";

import { formatDate } from "helpers/helpers";

import { isMobile } from "react-device-detect";

import dayjs from "dayjs";

import "./recommendation-modal.scss";

import { HealthStatesChip } from "./HealthStatesChip";

const INITIAL\_RECOMMENDATION = {

createdAt: "",

healthStates: [],

comment: "",

patientReply: null,

};

const copyRecommendation = (recommendation) => {

return {

...deepCopy(recommendation),

healthStates: recommendation.healthStates.map(({ docId }) => docId),

};

};

export const RecommendationModal = ({

isOpen,

onClose,

submitRecommendation,

selectedRecommendation,

healthStates,

}) => {

const getRecommendation = useCallback(

() =>

selectedRecommendation

? copyRecommendation(selectedRecommendation)

: { ...INITIAL\_RECOMMENDATION, createdAt: new Date().toString() },

[selectedRecommendation]

);

const [recommendation, setRecommendation] = useState(getRecommendation());

const [isLoading, setIsLoading] = useState(false);

const resetRecommendation = useCallback(() => {

setRecommendation(getRecommendation());

}, [setRecommendation, getRecommendation]);

const handleFieldChange = (e) => {

const { name, value } = e.target;

setRecommendation((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleSubmitRecommendation = async (event) => {

event.preventDefault();

setIsLoading(true);

await submitRecommendation({

...recommendation,

healthStates: recommendation.healthStates.map((id) =>

healthStates.find(({ docId }) => docId === id)

),

createdAt: dayjs().toDate().toString(),

});

setIsLoading(false);

onClose();

};

useEffect(() => {

resetRecommendation();

}, [selectedRecommendation, isOpen, resetRecommendation]);

const areEqual = (left, right) => {

const withoutCreatedAtLeft = { ...left, createdAt: undefined };

const withoutCreatedAtRight = { ...right, createdAt: undefined };

return (

JSON.stringify(withoutCreatedAtLeft) ===

JSON.stringify(withoutCreatedAtRight)

);

};

return (

<Dialog open={isOpen} onClose={onClose}>

<DialogTitle>

{(selectedRecommendation ? "Редактирование" : "Добавление") +

" рекомендации"}

</DialogTitle>

<DialogContent

className="modal-content"

sx={{

width: isMobile ? "300px" : "600px",

}}

>

<form

className="recommendation-form"

onSubmit={handleSubmitRecommendation}

>

<TextField

label="Дата"

name="createdAt"

value={formatDate(recommendation.createdAt)}

disabled={true}

fullWidth

/>

<HealthStatesChip

healthStatesOptions={healthStates}

healthStates={recommendation.healthStates}

setHealthStates={(newHealthStates) =>

setRecommendation({

...recommendation,

healthStates: newHealthStates,

})

}

/>

<TextField

multiline

minRows={3}

maxRows={3}

label="Комментарий"

name="comment"

value={recommendation.comment}

onChange={handleFieldChange}

disabled={isLoading}

fullWidth

/>

<DialogActions className="controls">

<Button onClick={onClose} disabled={isLoading} variant="outlined">

Отмена

</Button>

{isLoading ? (

<LoadingSpinner style={{ margin: "0", width: "100px" }} />

) : (

<Button

type="submit"

variant="outlined"

disabled={

(selectedRecommendation &&

areEqual(

recommendation,

copyRecommendation(selectedRecommendation)

)) ||

areEqual(recommendation, INITIAL\_RECOMMENDATION)

}

autoFocus

>

{selectedRecommendation ? "Изменить" : "Добавить"}

</Button>

)}

</DialogActions>

</form>

</DialogContent>

</Dialog>

);

};

***NotificationsPage.jsx***

import React, { useMemo, useState } from "react";

import { PageContainer } from "../../layout";

import { useGeneralDataHook } from "../../../hooks/useGeneralDataHook";

import { firebaseRepositories } from "../../../firestore/data/repositories";

import { LoadingSpinner } from "../../ui/LoadingSpinner";

import "./notifications.scss";

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

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

import { convertUserToUserInfo } from "../../../firestore/converters";

import {

Button,

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

} from "@mui/material";

import { formatDate } from "../../../helpers/helpers";

import { CardContainer } from "../../ui/CardContainer";

import { and, where } from "firebase/firestore";

import { Role } from "../../../firestore/types/collections.types";

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

import { getUserFullName } from "../../../firestore/helpers";

import { RecommendationModal } from "./components/RecommendationModal";

import { useSelector } from "react-redux";

import { getUserSelector } from "../../../store/selectors";

import dayjs from "dayjs";

import { Status } from "../Recommendations/components/Status/Status";

import { NotificationDetails } from "./components/NotificationDetails/NotificationDetails";

import { useGeneralModalHandlers } from "../../../hooks/useGeneralModalHandlers";

import { RecommendationDetailsModal } from "./components/RecommendationDetailsModal/RecommendationDetailsModal";

const getHealthStatesByPatientReports = (patientReports, healthStates) =>

patientReports

.map(({ currentValue, propName: reportPropName }) =>

healthStates.filter(

({ min, max, propName }) =>

propName === reportPropName &&

min <= currentValue &&

currentValue <= max

)

)

.flat();

export const NotificationsPage = () => {

const user = useSelector(getUserSelector);

const [isNotificationsLoading, notifications, resetNotifications] =

useGeneralDataHook(

async () =>

(

await firebaseRepositories.notifications.getDocs(

where("doctor", "==", user.docId)

)

).sort(

(a, b) => -dayjs(a.createdAt).diff(dayjs(b.createdAt), "second")

),

[]

);

const [isPatientsLoading, patients] = useGeneralDataHook(async () => {

if (!notifications || !notifications.length) {

return;

}

return (

await firebaseRepositories.users.getDocs(

and(

where("role", "==", Role.PATIENT),

where(

"docId",

"in",

notifications.map(({ patient }) => patient)

)

)

)

).map(convertUserToUserInfo);

}, [notifications]);

const [isHealthStatesLoading, healthStates] = useGeneralDataHook(

async () => firebaseRepositories.healthStates.getDocs(),

[]

);

const isLoading = useMemo(

() => isNotificationsLoading || isPatientsLoading || isHealthStatesLoading,

[isNotificationsLoading, isPatientsLoading, isHealthStatesLoading]

);

// recommendation modal

const [selectedNotification, setSelectedNotification] = useState(null);

const [selectedHealthStates, setSelectedHealthStates] = useState([]);

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

const submitRecommendation = async (recommendation) => {

await firebaseRepositories.notifications.updateDoc({

...selectedNotification,

recommendation,

});

await resetNotifications();

};

const onDeleteRecommendation = async (notification) => {

await firebaseRepositories.notifications.updateDoc({

...notification,

recommendation: null,

});

await resetNotifications();

};

const onOpen = (notification) => {

setSelectedHealthStates(

getHealthStatesByPatientReports(notification.patientReports, healthStates)

);

setSelectedNotification(notification);

setIsOpen(true);

};

const onClose = () => {

setSelectedHealthStates([]);

setSelectedNotification(null);

setIsOpen(false);

};

const [selectedRecommendation, setSelectedRecommendation] = useState(null);

const [isRecModalOpened, openRecModal, closeRecModal] =

useGeneralModalHandlers({

onOpen: (recommendation) => {

setSelectedRecommendation(recommendation);

},

onClose: () => {

setSelectedRecommendation(null);

},

});

return (

<PageContainer className="notifications-page">

{isLoading && <LoadingSpinner />}

{!isLoading && notifications && !notifications.length && (

<div className="no-notifications">Нет уведомлений</div>

)}

{!isLoading &&

notifications != null &&

patients != null &&

healthStates != null &&

!!notifications.length &&

!!patients.length &&

!!healthStates.length && (

<CardContainer className="notifications-container">

<TableContainer component={Paper}>

<Typography sx={{ textAlign: "center" }} variant="h6">

Уведомления

</Typography>

<Table>

<TableHead>

<TableRow>

<TableCell>Дата</TableCell>

<TableCell>Пациент</TableCell>

<TableCell>Отклонения по показателям</TableCell>

<TableCell>Рекомендация</TableCell>

<TableCell>Ответ пациента</TableCell>

</TableRow>

</TableHead>

<TableBody>

{notifications.map((notification) => {

const patient = patients.find(

(currentPatient) =>

notification.patient === currentPatient.docId

);

return (

<TableRow

key={notification.docId}

sx={{

"&:last-child td, &:last-child th": { border: 0 },

}}

>

<TableCell>

{formatDate(notification.createdAt)}

</TableCell>

<TableCell>

<div

style={{

display: "flex",

gap: "10px",

alignItems: "center",

}}

>

<Avatar

src={patient.imageUrl}

alt={getUserFullName(patient)}

/>

<Typography>{getUserFullName(patient)}</Typography>

</div>

</TableCell>

<TableCell sx={{ minWidth: "300px" }}>

<NotificationDetails notification={notification} />

</TableCell>

<TableCell>

{notification?.recommendation && (

<CardContainer

sx={{

padding: "5px",

margin: "10px auto",

fontSize: "12px",

}}

>

Дата рекомендации:{" "}

{formatDate(

notification.recommendation.createdAt

)}

</CardContainer>

)}

{notification.recommendation && (

<div className="controls">

<div style={{ display: "flex", gap: "5px" }}>

<Button

variant="contained"

color="info"

sx={{ fontSize: "11px", fontWeight: "bold" }}

onClick={() =>

openRecModal(notification?.recommendation)

}

>

Подробнее

</Button>

{!notification?.recommendation

?.patientReply && (

<>

<Button

variant="contained"

color="warning"

sx={{

fontSize: "11px",

fontWeight: "bold",

}}

onClick={() => onOpen(notification)}

>

Редактировать

</Button>

<Button

variant="contained"

color="error"

sx={{

fontSize: "11px",

fontWeight: "bold",

}}

onClick={() =>

onDeleteRecommendation(notification)

}

>

Удалить

</Button>

</>

)}

</div>

{notification?.recommendation?.patientReply && (

<div

style={{

fontSize: "12px",

textAlign: "center",

}}

>

Изменение рекомендации, после ответа

пользователя запрещено.

</div>

)}

</div>

)}

{!notification.recommendation && (

<div

style={{

display: "flex",

justifyContent: "center",

}}

>

<Button

variant="contained"

color="info"

sx={{ fontSize: "11px", fontWeight: "bold" }}

onClick={() => onOpen(notification)}

>

Добавить рекомендацию

</Button>

</div>

)}

</TableCell>

<TableCell sx={{ minWidth: "270px" }}>

<Status notification={notification} />

</TableCell>

</TableRow>

);

})}

</TableBody>

</Table>

</TableContainer>

</CardContainer>

)}

<RecommendationModal

isOpen={isOpen}

onClose={onClose}

submitRecommendation={submitRecommendation}

selectedRecommendation={selectedNotification?.recommendation ?? null}

healthStates={selectedHealthStates}

/>

<RecommendationDetailsModal

isOpen={isRecModalOpened}

onClose={closeRecModal}

recommendation={selectedRecommendation}

/>

</PageContainer>

);

};

***PatientsPage.jsx***

import React from "react";

import { PageContainer } from "../../layout";

import { LoadingSpinner } from "../../ui/LoadingSpinner";

import { Role } from "../../../firestore/types/collections.types";

import { useGeneralDataHook } from "../../../hooks/useGeneralDataHook";

import { firebaseRepositories } from "../../../firestore/data/repositories";

import { and, where } from "firebase/firestore";

import {

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

} from "@mui/material";

import { CardContainer } from "../../ui/CardContainer";

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

import {

convertRoleToRussian,

convertRoleToRussianPlural,

} from "../../../firestore/converters/role/role.converter";

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

import { getUserFullName } from "../../../firestore/helpers";

import { getUserSelector } from "../../../store/selectors";

import { useSelector } from "react-redux";

export const PatientsPage = () => {

const currentUser = useSelector(getUserSelector);

const [isPatientsLoading, patients] = useGeneralDataHook(async () => {

return firebaseRepositories.users.getDocs(

and(

where("role", "==", Role.PATIENT),

where("doctor", "==", currentUser.docId)

)

);

}, [currentUser]);

return (

<PageContainer

style={{

padding: "30px",

display: "flex",

flexDirection: "column",

alignItems: "center",

justifyContent: "center",

gap: "20px",

minHeight: "calc(100vh - 60px)",

width: "100%",

}}

>

{isPatientsLoading ? (

<LoadingSpinner />

) : (

patients && (

<div

style={{

display: "flex",

flexDirection: "column",

alignItems: "center",

gap: "30px",

width: "100%",

}}

>

{!patients.length && "Пациентов нет."}

{Boolean(patients.length) && (

<div style={{ display: "flex", width: "100%" }}>

<TableContainer component={CardContainer}>

<Typography variant="h5" sx={{ textAlign: "center" }}>

{convertRoleToRussianPlural(Role.PATIENT)}

</Typography>

<Table>

<TableHead>

<TableRow>

<TableCell>

{convertRoleToRussian(Role.PATIENT)}

</TableCell>

<TableCell>Эл. почта</TableCell>

<TableCell>Телефон</TableCell>

<TableCell>Адрес</TableCell>

<TableCell>Тип диабета</TableCell>

</TableRow>

</TableHead>

<TableBody>

{patients.map((patient) => (

<TableRow key={patient.docId}>

<TableCell>

<div

style={{

display: "flex",

gap: "10px",

alignItems: "center",

}}

>

<Avatar

src={patient.imageUrl}

alt={getUserFullName(patient)}

/>

<Typography>

{getUserFullName(patient)}

</Typography>

</div>

</TableCell>

<TableCell>{patient.email}</TableCell>

<TableCell>{patient.phone}</TableCell>

<TableCell>{patient.address}</TableCell>

<TableCell>{patient.diary.diabetType}</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

</div>

)}

</div>

)

)}

</PageContainer>

);

};

***ImageUploader.jsx***

import React from "react";

import { default as ImageUploaderLib } from "react-images-upload";

export const ImageUploader = ({ onDrop }) => (

<ImageUploaderLib

withIcon

buttonText="Загрузите картинку профиля"

onChange={onDrop}

imgExtension={[".jpg", ".gif", ".png", ".gif", ".jpeg", ".jfif"]}

maxFileSize={5242880}

singleImage

/>

);

***ProfilePage.tsx***

import React, { useState, useEffect, useMemo } from "react";

import { TextField, Button, Card, CardMedia } from "@mui/material";

import { useSelector } from "react-redux";

import { GlobalState, useAppDispatch } from "store";

import { Role, User } from "firestore/types/collections.types";

import {

convertRoleToRussian,

convertUserDataToUserInfo,

convertUserInfoToUserData,

convertUserToUserInfo,

UserData,

} from "firestore/converters";

import { PageContainer } from "components/layout";

import { isMobile } from "react-device-detect";

import { ProfileFormErrors } from "firestore/types/client.types";

import { deepCopy } from "deep-copy-ts";

import { ImageUploader } from "./ImageUploader";

import { getDownloadURL, ref, uploadBytes } from "firebase/storage";

import { storage } from "firebase\_config";

import { firebaseRepositories } from "firestore/data/repositories";

import { fetchUser } from "store/reducers/user/actions";

import { NotificationManager } from "react-notifications";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { setUser } from "store/reducers/user/userSlice";

import { deepEqual } from "ts-deep-equal";

import { EditButton } from "components/ui/EditButton";

import { DoctorDescription } from "components/pages/Doctor/DoctorDescription";

import "./profile.scss";

const validationObj: {

[key: string]: (s: string) => string | null;

} = {

address: (address: string) => {

const addressRegex = /[A-Za-zА-Яа-я]{5,}/;

const errorMessage = "Адрес должен содержать хотя бы 5 символов";

return addressRegex.test(address) ? null : errorMessage;

},

firstName: (firstName: string) => {

const nameRegex = /\S+/;

const errorMessage = "Имя должно содержать хотя бы 1 символ";

return nameRegex.test(firstName) ? null : errorMessage;

},

lastName: (lastName: string) => {

const nameRegex = /\S+/;

const errorMessage = "Фамилия должна содержать хотя бы 1 символ";

return nameRegex.test(lastName) ? null : errorMessage;

},

phone: (phone: string) => {

const phoneRegex = /^\+375-\d{2}-\d{7}$/;

const errorMessage = "Номер должен быть формата +375-XX-XXXXXXX";

return phoneRegex.test(phone) ? null : errorMessage;

},

};

const emptyErrors: ProfileFormErrors = {

firstName: null,

lastName: null,

address: null,

phone: null,

};

export const ProfilePage: React.FC = () => {

const dispatch = useAppDispatch();

// redux state

const user = useSelector<GlobalState, User>(

(state) => state.currentUser.user as User

);

const originalUserInfo = useMemo(() => convertUserToUserInfo(user), [user]);

// local state

const [isEditMode, setIsEditMode] = useState(false);

const [userData, setUserData] = useState<UserData>(

convertUserInfoToUserData(originalUserInfo)

);

const [errors, setErrors] = useState<ProfileFormErrors>(

deepCopy(emptyErrors)

);

const [isSubmitDisabled, setIsSubmitDisabled] = useState<boolean>(true);

const [isImageLoading, setIsImageLoading] = useState<boolean>(false);

const [isFormSaving, setIsFormSaving] = useState<boolean>(false);

const [isDoctorDescriptionSaving, setIsDoctorDescriptionSaving] =

useState<boolean>(false);

// effects

useEffect(

() => setUserData(convertUserInfoToUserData(originalUserInfo)),

[originalUserInfo]

);

useEffect(() => {

const newErrors = Object.fromEntries(

Object.keys(validationObj).map((key) => [

key,

validationObj[key](userData[key as keyof UserData]),

])

) as unknown as ProfileFormErrors;

setErrors(newErrors);

setIsSubmitDisabled(

Object.values(newErrors).some((value) => !!value) ||

deepEqual(userData, convertUserInfoToUserData(originalUserInfo))

);

}, [userData, originalUserInfo]);

const handleFieldChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const { name, value } = e.target;

setUserData((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleEditButtonClick = () => {

setIsEditMode(true);

};

const handleCancelButtonClick = () => {

setUserData(convertUserInfoToUserData(originalUserInfo));

setIsEditMode(false);

};

const onImageUpload = async (pictureFiles: File[]) => {

if (!pictureFiles.length) {

return;

}

try {

setIsImageLoading(true);

const [image] = pictureFiles;

const fileRef = ref(storage, image.name);

const snapshot = await uploadBytes(fileRef, image);

const imageUrl = await getDownloadURL(snapshot.ref);

const newUser = deepCopy(user);

newUser.imageUrl = imageUrl;

await firebaseRepositories.users.updateDoc(newUser);

await dispatch(fetchUser(newUser.docId));

} catch (e) {

console.log(e);

NotificationManager.error(

"Сохранение изображения",

'[Ошибка на странице "профиль"]'

);

} finally {

setIsImageLoading(false);

}

};

const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {

e.preventDefault();

try {

setIsFormSaving(true);

const newUser = {

...user,

...convertUserDataToUserInfo(userData),

};

await firebaseRepositories.users.updateDoc(newUser);

await dispatch(fetchUser(newUser.docId));

} catch (err) {

dispatch(

setUser({

...user,

...originalUserInfo,

})

);

console.log(err);

NotificationManager.error(

"Сохранение данных профиля",

'[Ошибка на странице "профиль"]'

);

} finally {

setIsFormSaving(false);

setIsEditMode(false);

}

};

const changeDescription = async (newDescription: string) => {

if (!user.employee || user.role !== Role.DOCTOR) {

return;

}

try {

setIsDoctorDescriptionSaving(true);

const newEmployee: User["employee"] = {

...user.employee,

description: newDescription,

};

const newDoctor: User = {

...user,

employee: newEmployee,

};

await firebaseRepositories.users.updateDoc(newDoctor);

await dispatch(fetchUser(newDoctor.docId));

} catch (e) {

console.log(e);

} finally {

setIsDoctorDescriptionSaving(false);

}

};

return (

<PageContainer>

<div className="profile-page">

<div

className="profile-page-content"

style={{ flexWrap: isMobile ? "wrap" : undefined }}

>

<div className="profile-image-container">

<Card

className="profile-image-card"

style={{ width: "300px", height: "250px" }}

>

<CardMedia

component="img"

height="100%"

width="100%"

sx={{ objectFit: "fill" }}

image={

userData.imageUrl ||

"https://thumbs.dreamstime.com/b/default-avatar-profile-icon-vector-social-media-user-image-182145777.jpg"

}

alt="Profile Image"

/>

</Card>

{isImageLoading ? (

<LoadingSpinner style={{ marginTop: "20px" }} />

) : (

<ImageUploader onDrop={onImageUpload} />

)}

</div>

{!isEditMode ? (

<Card className="profile-info-container">

<div className="profile-info-container\_\_title">

<h2>{`${userData.firstName} ${userData.lastName}`}</h2>

<EditButton onClick={handleEditButtonClick} />

</div>

<p>Эл. почта: {userData.email}</p>

<p>Телефон: {userData.phone}</p>

<p>Адрес: {userData.address}</p>

<p>Тип учётной записи: {convertRoleToRussian(userData.role)}</p>

{userData.role === Role.PATIENT && (

<p>Тип диабета: {user.diary?.diabetType}</p>

)}

<p className="role-logo-container">

<img src={`/images/logos/${userData.role}-logo.png`} alt="" />

</p>

</Card>

) : (

<Card className="edit-form-container">

<form className="edit-form" onSubmit={handleSubmit}>

<TextField

label="Имя"

name="firstName"

value={userData.firstName}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.firstName}

helperText={errors.firstName}

disabled={isFormSaving}

/>

<TextField

label="Фамилия"

name="lastName"

value={userData.lastName}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.lastName}

helperText={errors.lastName}

disabled={isFormSaving}

/>

<TextField

label="Телефон"

name="phone"

value={userData.phone}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.phone}

helperText={errors.phone}

disabled={isFormSaving}

/>

<TextField

label="Эл. почта"

name="email"

value={userData.email}

onChange={handleFieldChange}

fullWidth

margin="normal"

disabled={true}

/>

<TextField

label="Адрес"

name="address"

value={userData.address}

onChange={handleFieldChange}

fullWidth

margin="normal"

error={!!errors.address}

helperText={errors.address}

disabled={isFormSaving}

/>

{isFormSaving ? (

<LoadingSpinner style={{ marginTop: "20px" }} />

) : (

<div className="form-controls-container">

<Button

className="cancel-button"

variant="contained"

onClick={handleCancelButtonClick}

>

Отмена

</Button>

<Button

className="save-button"

type="submit"

variant="contained"

disabled={isSubmitDisabled}

>

Сохранить

</Button>

</div>

)}

</form>

</Card>

)}

</div>

{user.role === Role.DOCTOR && Boolean(changeDescription) && (

<div className="doctor-info" style={{ marginBottom: "5rem" }}>

{isDoctorDescriptionSaving ? (

<LoadingSpinner />

) : (

<DoctorDescription

doctor={user}

changeDescription={changeDescription}

/>

)}

</div>

)}

</div>

</PageContainer>

);

};

***HealthStateModal.jsx***

import React from "react";

import {

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

} from "@mui/material";

import { CardContainer } from "../../../../ui/CardContainer";

import "./health-state-modal.scss";

import { convertDailyLogPropNameToMeasurement } from "../../../../../firestore/converters";

export const HealthStateModal = ({ isOpen, onClose, healthStates }) => {

return (

<Dialog open={isOpen} onClose={onClose} className="health-state-dialog">

<DialogTitle sx={{ textAlign: "center" }}>Назначения</DialogTitle>

<DialogContent

className="health-state-modal-content"

sx={{ width: "1400px" }}

>

<CardContainer className="health-state-container">

<TableContainer>

<Table>

<TableHead>

<TableRow>

<TableCell>Название</TableCell>

<TableCell sx={{ minWidth: "120px" }}>Минимум</TableCell>

<TableCell sx={{ minWidth: "120px" }}>Максимум</TableCell>

<TableCell>Предупреждение</TableCell>

<TableCell>Назначение</TableCell>

</TableRow>

</TableHead>

<TableBody>

{healthStates.map((healthState) => (

<TableRow

key={healthState.docId}

sx={{

"&:last-child td, &:last-child th": { border: 0 },

}}

>

<TableCell>{healthState.title}</TableCell>

<TableCell>

{healthState.min}{" "}

{convertDailyLogPropNameToMeasurement(

healthState.propName

)}

</TableCell>

<TableCell>

{healthState.max}{" "}

{convertDailyLogPropNameToMeasurement(

healthState.propName

)}

</TableCell>

<TableCell>{healthState.warning}</TableCell>

<TableCell>{healthState.recommendation}</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

</CardContainer>

</DialogContent>

<DialogActions className="controls">

<Button onClick={onClose} variant="outlined">

Отмена

</Button>

</DialogActions>

</Dialog>

);

};

***Status.jsx***

import React, { useCallback, useMemo } from "react";

import { CardContainer } from "../../../../ui/CardContainer";

import { formatDate } from "../../../../../helpers/helpers";

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

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

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

import { PatientReplyStatus } from "../../../../../firestore/types/collections.types";

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

export const Status = ({ notification, onStatusChange }) => {

const recommendation = notification.recommendation ?? null;

const selectedStatus = useMemo(() => {

if (!recommendation) {

return PatientReplyStatus.DOESNT\_EXIST;

}

if (!recommendation.patientReply) {

return PatientReplyStatus.NOT\_SEEN;

}

return recommendation.patientReply.status;

}, [recommendation]);

const onChange = useCallback(

({ target: { value } }) => {

if (onStatusChange) {

return onStatusChange(notification, value);

}

return null;

},

[notification, onStatusChange]

);

return (

<>

{recommendation?.patientReply && (

<CardContainer sx={{ mb: 1, p: 1 }}>

Дата ответа: {formatDate(recommendation?.patientReply.createdAt)}

</CardContainer>

)}

{!onStatusChange && (

<CardContainer sx={{ mb: 1, p: 1 }}>

<div style={{ textAlign: "center" }}>Статус рекомендации</div>

<div

style={{

textAlign: "center",

fontSize: "10pt",

fontWeight: "bold",

margin: "1px",

}}

>

{selectedStatus}

</div>

</CardContainer>

)}

{onStatusChange && (

<FormControl fullWidth>

<InputLabel id="patient-reply-status-select-label-id">

Статус рекомендации

</InputLabel>

<Select

labelId="patient-reply-status-select-label-id"

value={selectedStatus}

label="Статус рекомендации"

onChange={onChange}

disabled={!Boolean(onStatusChange)}

>

{Object.values(PatientReplyStatus).map((status) => (

<MenuItem

key={notification.docId + status}

value={status}

style={{

display: [

PatientReplyStatus.NOT\_SEEN,

PatientReplyStatus.DOESNT\_EXIST,

].includes(status)

? "none"

: "block",

}}

>

{status}

</MenuItem>

))}

</Select>

</FormControl>

)}

</>

);

};

***RecommendationsPage.jsx***

import React, { useMemo, useState } from "react";

import "./recommendations.scss";

import { firebaseRepositories } from "../../../firestore/data/repositories";

import { useGeneralDataHook } from "../../../hooks/useGeneralDataHook";

import { useSelector } from "react-redux";

import { getUserSelector } from "../../../store/selectors";

import { where } from "firebase/firestore";

import { PageContainer } from "../../layout";

import { LoadingSpinner } from "../../ui/LoadingSpinner";

import {

Button,

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

} from "@mui/material";

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

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

import { formatDate } from "../../../helpers/helpers";

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

import { getUserFullName } from "../../../firestore/helpers";

import { CardContainer } from "../../ui/CardContainer";

import { HealthStateModal } from "./components/HealthStateModal";

import dayjs from "dayjs";

import { Status } from "./components/Status/Status";

import { NotificationDetails } from "../Notifications/components/NotificationDetails/NotificationDetails";

export const RecommendationsPage = () => {

const user = useSelector(getUserSelector);

const [isNotificationsLoading, notifications, resetNotifications] =

useGeneralDataHook(async () => {

const newNotifications = await firebaseRepositories.notifications.getDocs(

where("patient", "==", user.docId)

);

return newNotifications

.filter((notification) => notification.recommendation)

.sort(

(a, b) =>

-dayjs(a.recommendation.createdAt).diff(

dayjs(b.recommendation.createdAt),

"second"

)

);

}, []);

const [isDoctorLoading, doctors] = useGeneralDataHook(async () => {

if (notifications && notifications.length) {

return firebaseRepositories.users.getDocs(

where(

"docId",

"in",

notifications.map(({ doctor }) => doctor)

)

);

}

return null;

}, [notifications]);

const isLoading = useMemo(

() => isNotificationsLoading || isDoctorLoading,

[isNotificationsLoading, isDoctorLoading]

);

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

const [healthStates, setHealthStates] = useState([]);

const onOpen = (notification) => {

setHealthStates(notification.recommendation.healthStates);

setIsOpen(true);

};

const onClose = () => {

setHealthStates([]);

setIsOpen(false);

};

const onStatusChange = async (notification, newStatus) => {

const { patientReply } = notification.recommendation;

const newPatientReply = {

createdAt: dayjs().toDate().toString(),

status: newStatus,

comment: patientReply?.comment ?? "",

};

await firebaseRepositories.notifications.updateDoc({

...notification,

recommendation: {

...notification.recommendation,

patientReply: newPatientReply,

},

});

await resetNotifications();

};

return (

<PageContainer className="recommendations-page">

{isLoading && <LoadingSpinner />}

{!isLoading && !!notifications && !notifications.length && (

<div className="no-recommendations">Нет рекомендаций</div>

)}

{!isLoading && !!notifications && !!notifications.length && !!doctors && (

<CardContainer className="recommendations-container">

<TableContainer component={Paper}>

<Typography sx={{ textAlign: "center" }} variant="h6">

Рекоммендации

</Typography>

<Table>

<TableHead>

<TableRow>

<TableCell>Дата</TableCell>

<TableCell>Доктор</TableCell>

<TableCell>Отклонения по показателям</TableCell>

<TableCell>Назначения</TableCell>

<TableCell>Комментарий</TableCell>

<TableCell>Статус</TableCell>

</TableRow>

</TableHead>

<TableBody>

{notifications.map((notification) => {

const doctor = doctors.find(

({ docId }) => docId === notification.doctor

);

return (

<TableRow

key={notification.docId}

sx={{

"&:last-child td, &:last-child th": { border: 0 },

}}

>

<TableCell>

{formatDate(notification.recommendation.createdAt)}

</TableCell>

<TableCell>

<div

style={{

display: "flex",

gap: "10px",

alignItems: "center",

}}

>

<Avatar

src={doctor.imageUrl}

alt={getUserFullName(doctor)}

/>

<Typography>{getUserFullName(doctor)}</Typography>

</div>

</TableCell>

<TableCell sx={{ minWidth: "300px" }}>

<CardContainer className="report-date">

Дата уведомления: {formatDate(notification.createdAt)}

</CardContainer>

<NotificationDetails notification={notification} />

</TableCell>

<TableCell>

<div className="recommendation-details">

{!!notification.recommendation.healthStates

.length && (

<div className="advices">

{notification.recommendation.healthStates.map(

(healthState) => (

<CardContainer

key={notification.docId + healthState.docId}

className="advice"

>

{healthState.title}

</CardContainer>

)

)}

<Button

variant="contained"

color="info"

sx={{ fontSize: "11px", fontWeight: "bold" }}

onClick={() => onOpen(notification)}

>

Подробнее

</Button>

</div>

)}

{!notification.recommendation.healthStates.length && (

<div>Назначений нет.</div>

)}

</div>

</TableCell>

<TableCell>

{notification.recommendation.comment}

</TableCell>

<TableCell>

<Status

notification={notification}

onStatusChange={onStatusChange}

/>

</TableCell>

</TableRow>

);

})}

</TableBody>

</Table>

</TableContainer>

</CardContainer>

)}

<HealthStateModal

isOpen={isOpen}

onClose={onClose}

healthStates={healthStates}

/>

</PageContainer>

);

};

***useRelatives.ts***

import { Role, User } from "firestore/types/collections.types";

import useAsyncEffect from "use-async-effect";

import { useCallback, useState } from "react";

import { NotificationManager } from "react-notifications";

import { firebaseRepositories } from "firestore/data/repositories";

import { and, where } from "firebase/firestore";

export const useRelatives = (user: User) => {

const [patient, setPatient] = useState<User | null>(null);

const [relative, setRelative] = useState<User | null>(null);

const [isLoading, setIsLoading] = useState<boolean>(false);

const updateRelatives = useCallback(async () => {

try {

setIsLoading(true);

if (user.role === Role.PATIENT) {

setPatient(user);

const [newRelative] = await firebaseRepositories.users.getDocs(

and(

where("relativePatient", "==", user.docId),

where("role", "==", Role.RELATIVE)

)

);

setRelative(newRelative ?? null);

}

if (user.role === Role.RELATIVE) {

setRelative(user);

if (!user.relativePatient) {

throw new Error("У родственника должен быть добавлен пациент!");

}

const newPatient = await firebaseRepositories.users.getDocById(

user.relativePatient

);

setPatient(newPatient);

}

} catch (e) {

console.log(e);

NotificationManager.error(

"Ошибка загрузки пары родственников",

'Страница "родственник"'

);

} finally {

setIsLoading(false);

}

}, [user, setIsLoading, setPatient, setRelative]);

useAsyncEffect(updateRelatives, [user]);

return {

isLoading,

patient,

relative,

updateRelatives,

};

};

***RelativePage.tsx***

import React, { useCallback, useMemo, useState } from "react";

import { PageContainer } from "components/layout";

import { useSelector } from "react-redux";

import { getUserSelector } from "store/selectors";

import { useRelatives } from "components/pages/Relative/hooks/useRelatives";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { CardContainer } from "components/ui/CardContainer";

import { Role, User, UserInfo } from "firestore/types/collections.types";

import {

convertRoleToRussian,

convertUserToUserInfo,

} from "firestore/converters";

import { getUserFullName } from "firestore/helpers";

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

import { NotificationManager } from "react-notifications";

import { firebaseRepositories } from "firestore/data/repositories";

import { isMobile } from "react-device-detect";

import { deleteUser, getAuth, signInWithEmailAndPassword } from "firebase/auth";

import { firebaseConfig } from "firebase\_config";

import { initializeApp } from "firebase/app";

import "./relative.scss";

export const RelativePage = () => {

const currentUser = useSelector(getUserSelector);

const [isRemoveLoading, setIsRemoveLoading] = useState<boolean>(false);

const { isLoading, patient, relative, updateRelatives } =

useRelatives(currentUser);

const userInfo: UserInfo | null = useMemo(() => {

const user: User | null =

currentUser.role === Role.PATIENT ? relative : patient;

if (!user) {

return null;

}

return convertUserToUserInfo(user);

}, [currentUser, relative, patient]);

const removeRelative = async () => {

if (!relative) {

return;

}

try {

setIsRemoveLoading(true);

const firebaseApp2 = initializeApp(firebaseConfig, "relativeApp");

const auth2 = getAuth(firebaseApp2);

const relativeUserCredential = await signInWithEmailAndPassword(

auth2,

relative.email,

relative.password as string

);

await deleteUser(relativeUserCredential.user);

await firebaseRepositories.users.deleteDocById(relative.docId);

await updateRelatives();

await auth2.signOut();

} catch (e) {

console.log(e);

NotificationManager.error(

"Удаление родственника",

'Страница "родственник"'

);

} finally {

setIsRemoveLoading(false);

}

};

const handlePatientIdCopy = useCallback(async () => {

const type = "text/plain";

const blob = new Blob([currentUser.docId], { type });

const data = [new ClipboardItem({ [type]: blob })];

await navigator.clipboard.write(data);

}, [currentUser]);

return (

<PageContainer className="relative-page">

{isLoading ? (

<LoadingSpinner />

) : !userInfo ? (

<CardContainer className="no-relative-container">

<div className="title">

<h2>Родственник не приглашён</h2>

</div>

<img src="/images/missing-relative.png" alt="relative not found" />

<div

className="code"

style={{ justifyContent: isMobile ? "center" : "space-between" }}

>

<div className="code-title">Код приглашения</div>

<div className="code-content">

<button onClick={handlePatientIdCopy}>{currentUser.docId}</button>

<div className="description">

Код выше нужно скопировать (клик по тексту) и передать

родственнику для регистрации

</div>

</div>

</div>

</CardContainer>

) : (

<div className="relative-page-content">

<CardContainer className="avatar">

<img

className="avatar"

style={{ objectFit: "fill" }}

src={

userInfo.imageUrl ||

"https://thumbs.dreamstime.com/b/default-avatar-profile-icon-vector-social-media-user-image-182145777.jpg"

}

alt={getUserFullName(userInfo)}

/>

</CardContainer>

<CardContainer

className="relative-page-info"

title={

currentUser.role === Role.PATIENT

? "Родственник"

: "Родственник (пациент)"

}

>

<div className="title">

<h2>{getUserFullName(userInfo)}</h2>

</div>

<div>Эл. почта: {userInfo.email}</div>

<div>Телефон: {userInfo.phone}</div>

<div>Адрес: {userInfo.address}</div>

{userInfo.role === Role.PATIENT && (

<div>Тип диабета: {patient?.diary?.diabetType}</div>

)}

<div>Тип учётной записи: {convertRoleToRussian(userInfo.role)}</div>

<div className="role-logo-container">

<img

src={`/images/logos/${userInfo.role}-logo.png`}

alt="Role logo"

/>

</div>

{currentUser.role === Role.PATIENT &&

(isRemoveLoading ? (

<LoadingSpinner

style={{ margin: "10px auto", width: "250px" }}

/>

) : (

<Button

variant="outlined"

sx={{

color: "red",

borderColor: "red",

width: "250px",

margin: "10px auto",

fontSize: "11pt",

}}

onClick={removeRelative}

>

Удалить родственника

</Button>

))}

</CardContainer>

</div>

)}

</PageContainer>

);

};

***ReviewsPage.jsx***

import React from "react";

import { PageContainer } from "../../layout";

import { DoctorReviews } from "../Doctor/DoctorReviews";

import { getUserSelector } from "../../../store/selectors";

import { useSelector } from "react-redux";

import "./reviews.scss";

import { CardContainer } from "../../ui/CardContainer";

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

import ReactStars from "react-stars/dist/react-stars";

export const ReviewsPage = () => {

const user = useSelector(getUserSelector);

return (

<PageContainer className="reviews-page">

<CardContainer className="score">

<Typography variant="h5">Средняя оценка</Typography>

<Typography>

<ReactStars

edit={false}

size={50}

value={

user.employee.reviews

.map(({ score }) => score)

.reduce((a, b) => a + b, 0) / user.employee.reviews.length

}

/>

</Typography>

</CardContainer>

<DoctorReviews reviews={user.employee.reviews} />

</PageContainer>

);

};

***LineChart.tsx***

import React, { useMemo } from "react";

import { ResizableBox } from "components/ui/ResizableBox";

import { DailyLogData } from "firestore/converters";

import { isMobile } from "react-device-detect";

import {

DailyLogdataKey,

lineChartOptions,

} from "components/pages/Statistics/constants";

import { Line } from "react-chartjs-2";

import {

Chart as ChartJS,

// @ts-ignore

CategoryScale,

LinearScale,

// @ts-ignore

PointElement,

// @ts-ignore

LineElement,

// @ts-ignore

Title,

Tooltip,

// @ts-ignore

Legend,

} from "chart.js";

import dayjs from "dayjs";

// @ts-ignore

ChartJS.register(

CategoryScale,

// @ts-ignore

LinearScale,

PointElement,

LineElement,

Title,

Tooltip,

Legend

);

export const LineChart = ({

dailyLogsData,

propName,

title,

}: {

dailyLogsData: DailyLogData[];

propName: DailyLogdataKey;

title: string;

}) => {

const data = useMemo(() => {

const filteredDailyLogs = dailyLogsData.filter(

(dailyLogData) => +dailyLogData[propName]

);

const mappedDailyLogs = filteredDailyLogs.map((dailyLogData) => ({

label: dayjs(dailyLogData.createdAt).format("DD MMMM YYYY"),

data: dailyLogData[propName],

}));

return {

labels: mappedDailyLogs.map(({ label }) => label),

datasets: [

{

label: title,

data: mappedDailyLogs.map((dailyLog) => dailyLog.data),

borderColor: "rgb(53, 162, 235)",

backgroundColor: "rgba(53, 162, 235, 0.5)",

},

],

};

}, [dailyLogsData, propName, title]);

const boxSize = isMobile

? {

width: 300,

height: 400,

}

: {

width: 600,

};

return (

<ResizableBox {...boxSize}>

<Line data={data} options={lineChartOptions} />

</ResizableBox>

);

};

***constants.ts***

import { DailyLogData } from "firestore/converters";

export type DailyLogdataKey = keyof Omit<

DailyLogData,

"createdAt" | "takenMedications" | "notes" | "carbohydratesIntake"

>;

export const chartLabels: {

[key in DailyLogdataKey]: string;

} = {

weight: "Вес",

sugarLevel: "Уровень сахара",

pulse: "Пульс",

systolic: "Систолическое давление",

diastolic: "Диастолическое давление",

total: "Колиество каллорий",

temperature: "Температура",

};

export const chartsInfo = Object.entries(chartLabels).map(([key, value]) => ({

propName: key as DailyLogdataKey,

title: value,

}));

export const lineChartOptions = {

responsive: true,

plugins: {

legend: {

position: "top" as const,

},

title: {

display: false,

text: "",

},

},

maintainAspectRatio: false,

};

***StatisticsPage.tsx***

import React, { SyntheticEvent, useEffect, useMemo, useState } from "react";

import { PageContainer } from "components/layout";

import { useSelector } from "react-redux";

import { getUserSelector } from "store/selectors";

import { CardContainer } from "components/ui/CardContainer";

import { chartsInfo } from "components/pages/Statistics/constants";

import { LineChart } from "components/pages/Statistics/charts/LineChart";

import dayjs from "dayjs";

import {

convertDailyLogToDailyLogData,

DailyLogData,

} from "firestore/converters";

import { usePatientsStatistics } from "components/pages/Statistics/usePatientsStatistics";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { User, Role } from "firestore/types/collections.types";

import { DesktopDatePicker, MobileDatePicker } from "@mui/x-date-pickers";

import { isMobile } from "react-device-detect";

import { getUserFullName } from "firestore/helpers";

import { Autocomplete, TextField, Typography } from "@mui/material";

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

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

import "./statistics.scss";

const getMinDate = (dailyLogsData: DailyLogData[]) =>

new Date(

Math.min(...dailyLogsData.map(({ createdAt }) => +new Date(createdAt)))

).toString();

const getMaxDate = (dailyLogsData: DailyLogData[]) =>

new Date(

Math.max(...dailyLogsData.map(({ createdAt }) => +new Date(createdAt)))

).toString();

const getDateWithDayOnly = (date: string | null) =>

dayjs(dayjs(date ?? undefined).format("YYYY-MM-DD"));

const getDateWithDayOnlyDayjs = (date: dayjs.Dayjs | null) =>

getDateWithDayOnly(date ? date.toDate().toString() : null);

export const StatisticsPage = () => {

const user = useSelector(getUserSelector);

const { isLoading, patients } = usePatientsStatistics(user);

const [selectedPatient, setSelectedPatient] = useState<User | null>(null);

useEffect(() => {

if (!patients.length) {

return;

}

if ([Role.PATIENT, Role.RELATIVE, Role.DOCTOR].includes(user.role)) {

setSelectedPatient(patients[0]);

}

}, [user, patients, setSelectedPatient]);

const dailyLogsData = useMemo(() => {

if (selectedPatient && selectedPatient.diary?.dailyLogs) {

return selectedPatient.diary.dailyLogs

.map(convertDailyLogToDailyLogData)

.sort(

(left, right) =>

+new Date(left.createdAt) - +new Date(right.createdAt)

);

}

return [];

}, [selectedPatient]);

const [startDate, setStartDate] = useState<dayjs.Dayjs | null>(null);

const [endDate, setEndDate] = useState<dayjs.Dayjs | null>(dayjs());

const filteredDailyLogsData: DailyLogData[] = useMemo(

() =>

dailyLogsData

.filter(

({ createdAt }) =>

getDateWithDayOnly(createdAt).diff(

!startDate

? getDateWithDayOnly(getMinDate(dailyLogsData) ?? null)

: getDateWithDayOnlyDayjs(startDate),

"day"

) >= 0

)

.filter(

({ createdAt }) =>

getDateWithDayOnlyDayjs(endDate).diff(

getDateWithDayOnly(createdAt),

"day"

) >= 0

),

[dailyLogsData, startDate, endDate]

);

const DatePickerComponent = isMobile ? MobileDatePicker : DesktopDatePicker;

const getUserOption = (patient: User) => ({

value: patient.docId,

label: getUserFullName(patient),

imageUrl: patient.imageUrl,

});

const handleSelectedUserChange = (event: SyntheticEvent, option: unknown) => {

if (!patients.length) {

return;

}

setStartDate(null);

setEndDate(dayjs());

const castedOption = option as { value: string; label: string } | null;

setSelectedPatient(

patients.find((patient) => patient.docId === castedOption?.value) ?? null

);

};

return (

<PageContainer className="statistics-page-container">

{isLoading ? (

<LoadingSpinner />

) : (

<>

{!patients.length && (

<div className="no-patients" style={{ fontSize: "18pt" }}>

{user.role === Role.DOCTOR && "Нет пациентов"}

</div>

)}

{patients.length > 0 && (

<div className="statistics-page-content">

{user.role === Role.RELATIVE && selectedPatient && (

<Typography component="h1" variant="h5">

{getUserFullName(selectedPatient)}

</Typography>

)}

{user.role === Role.DOCTOR && (

<Autocomplete

id="patients-select"

className="select"

value={

selectedPatient ? getUserOption(selectedPatient) : null

}

onChange={handleSelectedUserChange}

options={patients.map(getUserOption)}

renderInput={(params) => (

<TextField

{...params}

label="Выберите пациента"

inputProps={{

...params.inputProps,

}}

/>

)}

blurOnSelect

noOptionsText="Нет пациентов с таким именем."

renderOption={(props, option) => (

<Box

component="li"

sx={{ "& > img": { mr: 2, flexShrink: 0 } }}

{...props}

>

<Avatar

src={option.imageUrl}

sx={{

width: "30px",

height: "30px",

marginRight: "20px",

}}

/>

{option.label}

</Box>

)}

/>

)}

{selectedPatient && (

<>

<div className="diabet-type">

Тип диабета: {selectedPatient.diary?.diabetType}

</div>

<div className="controls">

<DatePickerComponent

className={isMobile ? "mobile-date-picker" : ""}

label="Дата начала периода"

value={startDate}

maxDate={endDate}

onChange={(newValue) => setStartDate(newValue)}

/>

<DatePickerComponent

className={isMobile ? "mobile-date-picker" : ""}

label="Дата конца периода"

value={endDate}

minDate={startDate}

onChange={(newValue) => setEndDate(newValue)}

/>

</div>

<div className="charts">

{chartsInfo.map((chartInfo, index) => (

<CardContainer

title={chartInfo.title}

className="chart-card-container"

key={index}

>

<LineChart

dailyLogsData={filteredDailyLogsData}

title={chartInfo.title}

propName={chartInfo.propName}

/>

</CardContainer>

))}

</div>

</>

)}

<div style={{ height: "5rem" }} />

</div>

)}

</>

)}

</PageContainer>

);

};

***usePatientsStatistics.ts***

import { Role, User } from "firestore/types/collections.types";

import { useState } from "react";

import useAsyncEffect from "use-async-effect";

import { NotificationManager } from "react-notifications";

import { firebaseRepositories } from "firestore/data/repositories";

import { where } from "firebase/firestore";

export const usePatientsStatistics = (user: User) => {

const [patients, setPatients] = useState<User[]>([]);

const [relative, setRelative] = useState<User | null>(null);

const [isLoading, setIsLoading] = useState<boolean>(false);

useAsyncEffect(async () => {

try {

setIsLoading(true);

if (user.role === Role.PATIENT) {

setPatients([user]);

const relatives = await firebaseRepositories.users.getDocs(

where("relativePatient", "==", user.docId)

);

if (relatives && relatives.length === 1) {

setRelative(relatives[0]);

}

}

if (user.role === Role.RELATIVE) {

if (!user.relativePatient) {

throw new Error("There's no relativePatient for this user[relative]");

}

setRelative(relative);

const relativePatient = await firebaseRepositories.users.getDocById(

user.relativePatient

);

if (!relativePatient) {

throw new Error(`Patient with this id doesn't exist.`);

}

setPatients([relativePatient]);

}

if (user.role === Role.DOCTOR) {

const newPatients = await firebaseRepositories.users.getDocs(

where("doctor", "==", user.docId)

);

setPatients(newPatients);

}

} catch (e) {

console.log(e);

NotificationManager.error(

"useRelativeDailyLogsData",

"Data loading error"

);

} finally {

setIsLoading(false);

}

}, [user]);

return {

isLoading,

relative,

patients,

};

};

***ThematicMaterialPage.tsx***

import React, { useCallback, useEffect, useState } from "react";

import {

Role,

ThematicMaterial,

User,

} from "firestore/types/collections.types";

import { firebaseRepositories } from "firestore/data/repositories";

import { NotificationManager } from "react-notifications";

import { PageContainer } from "components/layout";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

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

import { CardContainer } from "components/ui/CardContainer";

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

import { getUserFullName } from "firestore/helpers";

import { Button, TextField } from "@mui/material";

import ArrowBackIcon from "@mui/icons-material/ArrowBack";

import { Route } from "components/routing";

import ReactHtmlParser from "react-html-parser";

import { formatDate } from "helpers/helpers";

import { EditButton } from "components/ui/EditButton";

import { useSelector } from "react-redux";

import { getUserInfoSelector } from "store/selectors";

import pretty from "pretty";

import { deepCopy } from "deep-copy-ts";

import DeleteIcon from "@mui/icons-material/Delete";

import EditIcon from "@mui/icons-material/Edit";

import "./thematic-material.scss";

import { useGeneralModalHandlers } from "../../../hooks/useGeneralModalHandlers";

import { ThematicMaterialEditModal } from "../ThematicMaterials";

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

const formatContent = (content: string) => pretty(content, { ocd: true });

export const ThematicMaterialPage = () => {

const navigate = useNavigate();

const userInfo = useSelector(getUserInfoSelector);

const [isLoading, setIsLoading] = useState<boolean>(false);

const [thematicMaterial, setThematicMaterial] =

useState<ThematicMaterial | null>(null);

const [author, setAuthor] = useState<User | null>(null);

const params = useParams();

const thematicMaterialId = params.id as string;

const loadThematicMaterial = useCallback(async () => {

try {

setIsLoading(true);

const newMaterial =

await firebaseRepositories.thematicMaterials.getDocById(

thematicMaterialId

);

if (!newMaterial) {

throw new Error("Материал не найден");

}

const newAuthor = await firebaseRepositories.users.getDocById(

newMaterial.author

);

if (!newAuthor) {

throw new Error("Автор материала не найден");

}

setThematicMaterial(newMaterial);

setAuthor(newAuthor);

} catch (e) {

console.log(e);

NotificationManager.error(

"Загрузка тематического материала",

"Ошибка Firestore"

);

} finally {

setIsLoading(false);

}

}, [thematicMaterialId]);

useEffect(() => {

loadThematicMaterial();

}, [loadThematicMaterial]);

const [isEditMode, setIsEditMode] = useState<boolean>(false);

const [newContent, setNewContent] = useState<string>("");

const [isFormSaving, setIsFormSaving] = useState<boolean>(false);

const handleThematicMaterialEdit = () => {

setNewContent(formatContent(thematicMaterial?.content as string));

setIsEditMode((prev) => !prev);

};

const handleCancel = () => {

setNewContent("");

setIsEditMode(false);

};

const handleFieldChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const { value } = e.target;

setNewContent(value);

};

const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {

event.preventDefault();

try {

setIsFormSaving(true);

const newThematicMaterial: ThematicMaterial = {

...deepCopy(thematicMaterial),

content: newContent,

} as ThematicMaterial;

await firebaseRepositories.thematicMaterials.updateDoc(

newThematicMaterial

);

const updatedMaterial =

await firebaseRepositories.thematicMaterials.getDocById(

thematicMaterialId

);

if (!updatedMaterial) {

throw new Error("Материал не найден");

}

const newAuthor = await firebaseRepositories.users.getDocById(

updatedMaterial.author

);

if (!newAuthor) {

throw new Error("Автор материала не найден");

}

setThematicMaterial(updatedMaterial);

setAuthor(newAuthor);

} catch (e) {

console.log(e);

NotificationManager.error("Сохранение статьи", "Страница статьи");

} finally {

setNewContent("");

setIsEditMode(false);

setIsFormSaving(false);

}

};

const [

isThematicMaterialModalOpened,

openThematicMaterialModal,

closeThematicMaterialModal,

] = useGeneralModalHandlers();

const submitThematicMaterial = async (

updatedThematicMaterial: ThematicMaterial

) => {

try {

setIsLoading(true);

await firebaseRepositories.thematicMaterials.updateDoc(

updatedThematicMaterial

);

await loadThematicMaterial();

} catch (e) {

console.log(e);

} finally {

setIsLoading(false);

}

};

const deleteThematicMaterial = async () => {

if (!thematicMaterial?.docId) {

return;

}

try {

setIsLoading(true);

await firebaseRepositories.thematicMaterials.deleteDocById(

thematicMaterial.docId

);

navigate(Route.thematicMaterials);

} catch (e) {

console.log(e);

} finally {

setIsLoading(false);

}

};

return (

<PageContainer className="thematic-material-page">

{!isLoading && (!thematicMaterial || !author) && (

<Typography variant="h6" component="h2" sx={{ mt: 2 }}>

Тематический материал не найден.

</Typography>

)}

{isLoading ? (

<LoadingSpinner />

) : (

thematicMaterial &&

author && (

<div className="thematic-material-container">

<div

style={{

display: "flex",

justifyContent: "space-between",

gap: "20px",

}}

>

<Button

onClick={() => navigate(Route.thematicMaterials)}

variant="contained"

startIcon={<ArrowBackIcon />}

sx={{ width: "min-content" }}

>

Вернуться

</Button>

{[Role.DOCTOR, Role.CONTENT\_MAKER].includes(userInfo.role) && (

<div style={{ display: "flex", gap: "20px" }}>

<Button

onClick={openThematicMaterialModal as () => void}

variant="contained"

color="warning"

startIcon={<EditIcon />}

sx={{ width: "min-content" }}

>

Редактировать

</Button>

<Button

onClick={() => deleteThematicMaterial()}

variant="contained"

color="error"

startIcon={<DeleteIcon />}

sx={{ width: "min-content" }}

>

Удалить

</Button>

</div>

)}

</div>

<CardContainer className="thematic-material-header-container">

<div className="image">

<img

src={thematicMaterial.imageUrl}

alt={thematicMaterial.title}

/>

</div>

<div className="short-info">

<div className="author">

<div className="author-badge">

<div className="author-badge\_\_avatar">

<Avatar src={author.imageUrl} />

</div>

<div className="author-badge\_\_fullname">

{getUserFullName(author)}

</div>

</div>

<div className="light-text">Автор</div>

</div>

<div className="title">{thematicMaterial.title}</div>

<div className="light-text">

{formatDate(thematicMaterial.createdAt)}

</div>

<div className="description">

{thematicMaterial.description}

</div>

</div>

</CardContainer>

<CardContainer className="thematic-material-body">

{!isEditMode &&

[Role.DOCTOR, Role.CONTENT\_MAKER].includes(userInfo.role) && (

<EditButton onClick={handleThematicMaterialEdit} />

)}

{isEditMode ? (

<form className="edit-form" onSubmit={handleSubmit}>

<TextField

multiline

label="Текст статьи"

name="content"

value={newContent}

onChange={handleFieldChange}

fullWidth

margin="normal"

disabled={isFormSaving}

/>

<div className="controls">

<Button

variant="outlined"

onClick={handleCancel}

disabled={isFormSaving}

>

Отмена

</Button>

{isFormSaving ? (

<LoadingSpinner style={{ margin: "0", width: "100px" }} />

) : (

<Button

variant="outlined"

type="submit"

disabled={

formatContent(thematicMaterial.content) ===

formatContent(newContent) || isFormSaving

}

>

Подтвердить

</Button>

)}

</div>

</form>

) : (

ReactHtmlParser(thematicMaterial.content)

)}

</CardContainer>

{/\* todo comments for thematic material page \*/}

{/\*<CardContainer className="comments" title="Комментарии">\*/}

{/\*{thematicMaterial.comments.map((comment) => (\*/}

{/\* <div key={comment.message.createdAt}>{JSON.stringify(comment, null, ' ')}</div>\*/}

{/\*))}\*/}

{/\*</CardContainer>\*/}

<ThematicMaterialEditModal

isOpen={isThematicMaterialModalOpened}

onClose={closeThematicMaterialModal as () => void}

submitThematicMaterial={submitThematicMaterial}

selectedThematicMaterial={thematicMaterial}

/>

</div>

)

)}

</PageContainer>

);

};

***ThematicMaterialEditModal.jsx***

import React, { useCallback, useEffect, useState } from "react";

import {

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

TextField,

} from "@mui/material";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { deepCopy } from "deep-copy-ts";

import { formatDate } from "helpers/helpers";

import { isMobile } from "react-device-detect";

import dayjs from "dayjs";

import { useSelector } from "react-redux";

import { getUserInfoSelector } from "../../../../../store/selectors";

import "./thematic-material-modal.scss";

import {

generateDocId,

getUserFullName,

} from "../../../../../firestore/helpers";

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

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

export const ThematicMaterialEditModal = ({

isOpen,

onClose,

submitThematicMaterial,

selectedThematicMaterial,

}) => {

const user = useSelector(getUserInfoSelector);

const getThematicMaterial = useCallback(

() =>

selectedThematicMaterial

? deepCopy(selectedThematicMaterial)

: {

createdAt: dayjs().toDate().toString(),

docId: generateDocId(),

imageUrl: "",

title: "",

description: "",

content: "",

author: user.docId,

comments: [],

},

[selectedThematicMaterial, user.docId]

);

const [thematicMaterial, setThematicMaterial] = useState(

getThematicMaterial()

);

const [isLoading, setIsLoading] = useState(false);

const resetThematicMaterial = useCallback(() => {

setThematicMaterial(getThematicMaterial());

}, [setThematicMaterial, getThematicMaterial]);

const handleFieldChange = (e) => {

const { name, value } = e.target;

setThematicMaterial((prevData) => ({

...prevData,

[name]: value,

}));

};

const handleSubmitThematicMaterial = async (event) => {

event.preventDefault();

setIsLoading(true);

await submitThematicMaterial({

...thematicMaterial,

createdAt: dayjs().toDate().toString(),

});

setIsLoading(false);

onClose();

};

useEffect(() => {

resetThematicMaterial();

}, [selectedThematicMaterial, isOpen, resetThematicMaterial]);

return (

<Dialog open={isOpen} onClose={onClose}>

<DialogTitle>

{(selectedThematicMaterial ? "Редактирование" : "Добавление") +

" тематического материала"}

</DialogTitle>

<DialogContent

className="modal-content"

sx={{

width: isMobile ? "300px" : "600px",

}}

>

<form

className="thematic-material-form"

onSubmit={handleSubmitThematicMaterial}

>

<TextField

label="Дата"

name="createdAt"

value={formatDate(thematicMaterial.createdAt)}

disabled={true}

/>

<div style={{ display: "flex", gap: "10px", alignItems: "center" }}>

Автор:

<div

style={{

display: "flex",

gap: "10px",

alignItems: "center",

}}

>

<Avatar src={user.imageUrl} alt={getUserFullName(user)} />

<Typography>{getUserFullName(user)}</Typography>

</div>

</div>

<TextField

label="Ссылка на изображение"

name="imageUrl"

value={thematicMaterial.imageUrl}

onChange={handleFieldChange}

/>

<TextField

label="Название"

name="title"

value={thematicMaterial.title}

onChange={handleFieldChange}

/>

<TextField

label="Описание"

name="description"

value={thematicMaterial.description}

onChange={handleFieldChange}

/>

<TextField

multiline

minRows={3}

maxRows={5}

label="Содержание"

name="content"

value={thematicMaterial.content}

onChange={handleFieldChange}

disabled={isLoading}

/>

<DialogActions className="controls">

<Button onClick={onClose} disabled={isLoading} variant="outlined">

Отмена

</Button>

{isLoading ? (

<LoadingSpinner style={{ margin: "0", width: "100px" }} />

) : (

<Button

type="submit"

variant="outlined"

disabled={

selectedThematicMaterial?.imageUrl ===

thematicMaterial.imageUrl &&

selectedThematicMaterial?.title === thematicMaterial.title &&

selectedThematicMaterial?.description ===

thematicMaterial.description &&

selectedThematicMaterial?.content === thematicMaterial.content

}

autoFocus

>

{selectedThematicMaterial ? "Изменить" : "Добавить"}

</Button>

)}

</DialogActions>

</form>

</DialogContent>

</Dialog>

);

};

***ThematicMaterialView.tsx***

import React from "react";

import { ThematicMaterial } from "firestore/types/collections.types";

import { CardContainer } from "components/ui/CardContainer";

import { isMobile } from "react-device-detect";

import "./thematic-material-view.scss";

export const ThematicMaterialView = ({

imageUrl,

title,

description,

onClick,

}: ThematicMaterial & {

onClick: () => void;

}) => {

return (

<CardContainer

className="thematic-material-view"

onClick={onClick}

sx={{ width: isMobile ? "170px" : "200px" }}

>

<CardContainer className="image-container">

<img src={imageUrl} alt={title} />

</CardContainer>

<div className="title">{title}</div>

<div className="description">{description}</div>

</CardContainer>

);

};

***ThematicMaterialsPage.tsx***

import React, { useCallback, useEffect, useMemo, useState } from "react";

import { PageContainer } from "components/layout";

import { Role, ThematicMaterial } from "firestore/types/collections.types";

import { firebaseRepositories } from "firestore/data/repositories";

import { NotificationManager } from "react-notifications";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { ThematicMaterialView } from "components/pages/ThematicMaterials/components/ThematicMaterialView";

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

import { Route } from "components/routing";

import { isMobile } from "react-device-detect";

import { useGeneralModalHandlers } from "../../../hooks/useGeneralModalHandlers";

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

import { ThematicMaterialEditModal } from "./components";

import dayjs from "dayjs";

import { useSelector } from "react-redux";

import { getUserSelector } from "../../../store/selectors";

import "./thematic-materials.scss";

export const ThematicMaterialsPage = () => {

const user = useSelector(getUserSelector);

const navigate = useNavigate();

const [isLoading, setIsLoading] = useState<boolean>(false);

const [thematicMaterials, setThematicMaterials] = useState<

ThematicMaterial[]

>([]);

const loadThematicMaterials = useCallback(async () => {

try {

setIsLoading(true);

const newMaterials =

await firebaseRepositories.thematicMaterials.getDocs();

setThematicMaterials(newMaterials);

} catch (e) {

console.log(e);

NotificationManager.error(

"Загрузка тематических материалов",

"Ошибка Firestore"

);

} finally {

setIsLoading(false);

}

}, []);

useEffect(() => {

loadThematicMaterials();

}, [loadThematicMaterials]);

const [

isThematicMaterialModalOpened,

openThematicMaterialModal,

closeThematicMaterialModal,

] = useGeneralModalHandlers();

const submitThematicMaterial = async (thematicMaterial: ThematicMaterial) => {

try {

setIsLoading(true);

await firebaseRepositories.thematicMaterials.updateDoc(thematicMaterial);

await loadThematicMaterials();

} catch (e) {

console.log(e);

} finally {

setIsLoading(false);

}

};

const sortedThematicMaterials = useMemo(() => {

return thematicMaterials.sort((left, right) =>

dayjs(left.createdAt).diff(dayjs(right.createdAt))

);

}, [thematicMaterials]);

return (

<PageContainer

className="thematic-materials-page"

style={{

padding: isMobile ? "20px 10px" : "20px",

}}

>

{isLoading ? (

<LoadingSpinner />

) : (

<div style={{ display: "flex", flexDirection: "column", gap: "20px" }}>

{[Role.DOCTOR, Role.CONTENT\_MAKER].includes(user.role) && (

<div style={{ display: "flex", justifyContent: "center" }}>

<Button

color="success"

variant="contained"

sx={{ fontSize: "11px", fontWeight: "bold" }}

onClick={openThematicMaterialModal as () => void}

>

Добавить тематический материал

</Button>

</div>

)}

<div

className="thematic-materials-container"

style={{

gap: isMobile ? "10px" : "20px",

width: isMobile ? "350px" : "1080px",

}}

>

{sortedThematicMaterials.map((thematicMaterial) => (

<ThematicMaterialView

key={thematicMaterial.docId}

{...thematicMaterial}

onClick={() =>

navigate(

`${Route.thematicMaterials}/${thematicMaterial.docId}`

)

}

/>

))}

</div>

<ThematicMaterialEditModal

isOpen={isThematicMaterialModalOpened}

onClose={closeThematicMaterialModal as () => void}

submitThematicMaterial={submitThematicMaterial}

selectedThematicMaterial={null}

/>

</div>

)}

</PageContainer>

);

};

***AppRoutes.tsx***

import React, { Suspense } from "react";

import {

Navigate,

Route as DomRoute,

Routes as RoutesContainer,

} from "react-router-dom";

import { ProtectedRoutes } from "components/routing/ProtectedRoutes";

import { Route } from "components/routing/constants";

import { PublicRoutes } from "components/routing/PublicRoutes";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import {

DialogsPage,

DiaryPage,

DoctorPage,

HealthStatesPage,

Home,

NotFound,

Profile,

RelativePage,

SignIn,

SignOut,

SignUp,

StatisticsPage,

ThematicMaterialPage,

ThematicMaterialsPage,

ReviewsPage,

NotificationsPage,

RecommendationsPage,

PatientsPage,

} from "components/pages";

import { RolesCheckRoutes } from "components/routing/RoleCheckRoutes";

import { Role } from "firestore/types/collections.types";

import { AccountsPage } from "../../pages/Accounts";

// const HomePage = React.lazy(() => import("components/pages/HomePage"));

export const AppRoutes = () => {

return (

<Suspense fallback={<LoadingSpinner />}>

<RoutesContainer>

{/\* Protected routes \*/}

<DomRoute element={<ProtectedRoutes />}>

{/\* Profile \*/}

<DomRoute element={<RolesCheckRoutes roles={Object.values(Role)} />}>

<DomRoute path={Route.profile} element={<Profile />} />

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.PATIENT]} />}>

<DomRoute path={Route.diary} element={<DiaryPage />} />

</DomRoute>

<DomRoute

element={<RolesCheckRoutes roles={[Role.PATIENT, Role.DOCTOR]} />}

>

<DomRoute path={Route.dialogs} element={<DialogsPage />} />

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.PATIENT]} />}>

<DomRoute path={Route.statistics} element={<StatisticsPage />} />

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.RELATIVE]} />}>

<DomRoute

path={Route.relativeStatistics}

element={<StatisticsPage />}

/>

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.DOCTOR]} />}>

<DomRoute

path={Route.patientsStatistics}

element={<StatisticsPage />}

/>

</DomRoute>

<DomRoute

element={

<RolesCheckRoutes

roles={[

Role.PATIENT,

Role.RELATIVE,

Role.DOCTOR,

Role.CONTENT\_MAKER,

Role.MODERATOR,

]}

/>

}

>

<DomRoute

path={Route.thematicMaterials}

element={<ThematicMaterialsPage />}

/>

<DomRoute

path={Route.thematicMaterial}

element={<ThematicMaterialPage />}

/>

</DomRoute>

<DomRoute

element={<RolesCheckRoutes roles={[Role.PATIENT, Role.RELATIVE]} />}

>

<DomRoute path={Route.relative} element={<RelativePage />} />

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.PATIENT]} />}>

<DomRoute path={Route.doctor} element={<DoctorPage />} />

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.DOCTOR]} />}>

<DomRoute path={Route.reviews} element={<ReviewsPage />} />

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.DOCTOR]} />}>

<DomRoute

path={Route.healthStates}

element={<HealthStatesPage />}

/>

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.DOCTOR]} />}>

<DomRoute

path={Route.notifications}

element={<NotificationsPage />}

/>

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.DOCTOR]} />}>

<DomRoute path={Route.patients} element={<PatientsPage />} />

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.PATIENT]} />}>

<DomRoute

path={Route.recommendations}

element={<RecommendationsPage />}

/>

</DomRoute>

<DomRoute element={<RolesCheckRoutes roles={[Role.ADMIN]} />}>

<DomRoute path={Route.accounts} element={<AccountsPage />} />

</DomRoute>

{/\* Sign out \*/}

<DomRoute path={Route.signOut} element={<SignOut />} />

</DomRoute>

{/\* Default "/" \*/}

<DomRoute

path={Route.default}

element={<Navigate replace to={Route.home} />}

/>

{/\* HomePage \*/}

<DomRoute path={Route.home} element={<Home />} />

<DomRoute element={<PublicRoutes />}>

{/\* auth: sign-in, sign-up \*/}

<DomRoute

path={Route.auth}

element={<Navigate replace to={Route.signIn} />}

/>

<DomRoute path={Route.signIn} element={<SignIn />} />

<DomRoute path={Route.signUp} element={<SignUp />} />

</DomRoute>

{/\* not found \*/}

<DomRoute path={Route.notFound} element={<NotFound />} />

{/\* any \*/}

<DomRoute path={Route.any} element={<NotFound />} />

</RoutesContainer>

</Suspense>

);

};

***constants.ts***

export enum Route {

any = "\*",

default = "/",

notFound = "/not-found",

home = "/home",

auth = "/auth",

signIn = "/auth/sign-in",

signUp = "/auth/sign-up",

signOut = "/auth/sign-out",

profile = "/profile",

diary = "/patient/diary",

goals = "/patient/goals",

statistics = "/statistics",

relativeStatistics = "/statistics/relative",

dialogs = "/dialogs",

thematicMaterials = "/thematic-materials",

thematicMaterial = "/thematic-materials/:id",

patientsStatistics = "/doctor/patients-statistics",

accounts = "/admin/accounts",

relative = "/relative",

doctor = "/doctor",

reviews = "/reviews",

healthStates = "/health-states",

notifications = "/notifications",

recommendations = "/recommendations",

patients = "/patients",

}

export const RouteTitles: {

[key in Route]: string;

} = {

[Route.any]: "",

[Route.default]: "",

[Route.notFound]: "Не найдено",

[Route.home]: "Главная",

[Route.auth]: "Авторизация",

[Route.signIn]: "Вход",

[Route.signUp]: "Регистрация",

[Route.signOut]: "Выход",

[Route.profile]: "Профиль",

[Route.diary]: "Дневник",

[Route.goals]: "Цели",

[Route.statistics]: "Статистика",

[Route.relativeStatistics]: "Статистика родственника",

[Route.dialogs]: "Диалоги",

[Route.thematicMaterials]: "Тематические материалы",

[Route.thematicMaterial]: "Тематический материал",

[Route.patientsStatistics]: "Статистика пациентов",

[Route.accounts]: "Пользователи",

[Route.relative]: "Родственник",

[Route.doctor]: "Доктор",

[Route.reviews]: "Отзывы",

[Route.healthStates]: "Назначения",

[Route.notifications]: "Уведомления",

[Route.recommendations]: "Рекомендации",

[Route.patients]: "Пациенты",

};

***ProtectedRoutes.tsx***

import React, { useEffect } from "react";

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

import { Route } from "components/routing/constants";

import { useAuthState } from "react-firebase-hooks/auth";

import { auth } from "firebase\_config";

import { GlobalState, useAppDispatch } from "store";

import { setUser, UserSliceState } from "store/reducers/user/userSlice";

import { useSelector } from "react-redux";

import { Layout } from "components/layout/Layout";

import { LoadingSpinner } from "components/ui/LoadingSpinner";

import { fetchUser } from "store/reducers/user/actions";

export const ProtectedRoutes = () => {

const [authUser, loading] = useAuthState(auth);

const currentUser = useSelector<GlobalState, UserSliceState>(

(state) => state.currentUser

);

const dispatch = useAppDispatch();

useEffect(() => {

return auth.onAuthStateChanged(async (userCredential) => {

if (userCredential) {

await dispatch(fetchUser(userCredential.uid));

} else {

dispatch(setUser(null));

}

});

}, [dispatch]);

if (loading || (authUser && !currentUser.user)) {

return <LoadingSpinner />;

}

return authUser && currentUser.user ? (

<Layout />

) : (

<Navigate replace to={Route.notFound} />

);

};

***PublicRoutes.tsx***

import React from "react";

import { Navigate, Outlet } from "react-router-dom";

import { Route } from "components/routing/constants";

import { useAuthState } from "react-firebase-hooks/auth";

import { auth } from "firebase\_config";

export const PublicRoutes = () => {

const [authUser] = useAuthState(auth);

return authUser ? <Navigate replace to={Route.profile} /> : <Outlet />;

};

***RolesCheckRoutes.tsx***

import React from "react";

import { Navigate, Outlet } from "react-router-dom";

import { Route } from "components/routing/constants";

import { useSelector } from "react-redux";

import { GlobalState } from "store";

import { Role, User } from "firestore/types/collections.types";

export const RolesCheckRoutes = ({ roles }: { roles: Role[] }) => {

const user = useSelector<GlobalState, User>(

(state) => state.currentUser.user as User

);

return !roles.includes(user.role) ? (

<Navigate replace to={Route.notFound} />

) : (

<Outlet />

);

};

***CardContainer.jsx***

import React from "react";

import { Card, Typography } from "@mui/material";

import "./card-container.scss";

export const CardContainer = (props) => {

return (

<Card

{...props}

className={

"card-container" + (props.className ? " " + props.className : "")

}

onClick={props.onClick}

>

{props.title && (

<Typography variant="h6" className="card-container-title">

{props.title}

</Typography>

)}

{props.children}

</Card>

);

};

***EditButton.tsx***

import React from "react";

import { IconButton } from "@mui/material";

import { ReactComponent as EditIcon } from "assets/icons/edit-icon.svg";

import "./edit-button.scss";

export const EditButton = ({ onClick }: { onClick: () => void }) => {

return (

<IconButton className="edit-button-container" onClick={onClick}>

<EditIcon className="edit-button-icon" />

</IconButton>

);

};

***LoadingSpinner.tsx***

import \* as React from "react";

import { CircularProgress } from "@mui/material";

import "./loading-spinner.scss";

export const LoadingSpinner = ({ style }: { style?: React.CSSProperties }) => {

return (

<div className="loading-spinner-container" style={style}>

<CircularProgress sx={{ margin: "auto" }} />

</div>

);

};

***ResizableBox.tsx***

import React, { CSSProperties } from "react";

import { ResizableBox as ReactResizableBox } from "react-resizable";

import "react-resizable/css/styles.css";

import "./react-resizable.scss";

export const ResizableBox = ({

children,

width,

height = 300,

resizable = true,

style,

className,

}: {

children?: React.ReactNode;

width: number;

height?: number;

resizable?: boolean;

style?: CSSProperties;

className?: string;

}) => {

return (

<div

style={{

display: "inline-block",

width: "100%",

background: "white",

padding: ".5rem",

borderRadius: "0.5rem",

boxShadow: "0 30px 40px rgba(0,0,0,.1)",

...style,

}}

>

{resizable ? (

<ReactResizableBox width={width} height={height}>

<div

style={{

width: "100%",

height: "100%",

}}

className={className}

>

{children}

</div>

</ReactResizableBox>

) : (

<div

style={{

width: `${width}px`,

height: `${height}px`,

}}

className={className}

>

{children}

</div>

)}

</div>

);

};

***App.tsx***

import React from "react";

import { NotificationContainer } from "react-notifications";

import { AppRoutes } from "components/routing";

import { StreamChat } from "stream-chat";

import \* as emailjs from "@emailjs/browser";

import { Streami18n } from "stream-chat-react";

import ruUpdated from "assets/i18/ru.json";

import "containers/App/app.scss";

export const streamClient = StreamChat.getInstance("bq8h3h82pvd8");

export const i18nInstance = new Streami18n({

language: "ru",

translationsForLanguage: ruUpdated,

});

emailjs.init("y-xzk78YvyzBfbFUe");

export const App = () => {

return (

<div className="App">

<AppRoutes />

<NotificationContainer />

</div>

);

};

***notifications.d.ts***

declare module "react-notifications" {

import { ReactNode } from "react";

import { EventEmitter } from "events";

enum NotificationType {

INFO = "info",

SUCCESS = "success",

WARNING = "warning",

ERROR = "error",

}

enum EventType {

CHANGE = "change",

INFO = "info",

SUCCESS = "success",

WARNING = "warning",

ERROR = "error",

}

interface NotificationProps {

type: NotificationType;

title?: ReactNode;

message: ReactNode;

timeOut?: number;

onClick: () => void;

onRequestHide: () => void;

}

interface NotificationsProps {

notifications: Notification[];

onRequestHide?: (notification: Notification) => void;

enterTimeout?: number;

leaveTimeout?: number;

}

interface NotificationContainerProps {

enterTimeout?: number;

leaveTimeout?: number;

}

interface INotificationManagerCreate {

type: EventType;

title?: NotificationProps["title"];

message?: NotificationProps["message"];

timeout?: number;

onClick?: () => void;

priority?: boolean;

}

class Notification extends React.Component<NotificationProps, unknown> {}

class Notifications extends React.Component<NotificationsProps, unknown> {}

class NotificationContainer extends React.Component<

NotificationContainerProps,

unknown

> {}

class NotificationManager extends EventEmitter {

static create(INotificationManagerCreate): void;

static info(

message?: INotificationManagerCreate["message"],

title?: INotificationManagerCreate["title"],

timeOut?: INotificationManagerCreate["timeout"],

onClick?: INotificationManagerCreate["onClick"],

priority?: INotificationManagerCreate["priority"]

): void;

static success(

message?: INotificationManagerCreate["message"],

title?: INotificationManagerCreate["title"],

timeOut?: INotificationManagerCreate["timeout"],

onClick?: INotificationManagerCreate["onClick"],

priority?: INotificationManagerCreate["priority"]

): void;

static warning(

message?: INotificationManagerCreate["message"],

title?: INotificationManagerCreate["title"],

timeOut?: INotificationManagerCreate["timeout"],

onClick?: INotificationManagerCreate["onClick"],

priority?: INotificationManagerCreate["priority"]

): void;

static error(

message?: INotificationManagerCreate["message"],

title?: INotificationManagerCreate["title"],

timeOut?: INotificationManagerCreate["timeout"],

onClick?: INotificationManagerCreate["onClick"],

priority?: INotificationManagerCreate["priority"]

): void;

static remove(notification: Notification): void;

}

}

***svg.d.ts***

declare module "\*.svg" {

const content: React.FunctionComponent<React.SVGAttributes<SVGElement>>;

export const ReactComponent = content;

}

***firebase\_config.ts***

// Import the functions you need from the SDKs you need

import { initializeApp } from "firebase/app";

import {

browserSessionPersistence,

getAuth,

setPersistence,

} from "firebase/auth";

import { getStorage } from "firebase/storage";

// TODO: Add SDKs for Firebase products that you want to use

// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration

// For Firebase JS SDK v7.20.0 and later, measurementId is optional

export const firebaseConfig = {

apiKey: "AIzaSyDJ4qhMADEo2Hb-Xdl0SCC5bW26ti0qFeQ",

authDomain: "diabeticdiary-fedb9.firebaseapp.com",

projectId: "diabeticdiary-fedb9",

storageBucket: "diabeticdiary-fedb9.appspot.com",

messagingSenderId: "1018876156693",

appId: "1:1018876156693:web:7042656da3b1132063a71c",

measurementId: "G-RDTQC98HRR",

};

// Initialize Firebase

export const firebaseApp = initializeApp(firebaseConfig);

export const auth = getAuth(firebaseApp);

export const storage = getStorage(firebaseApp);

(async () => {

await setPersistence(auth, browserSessionPersistence);

})();

***constants.ts***

import { DailyLogPropName } from "./types/collections.types";

export enum CollectionName {

USERS = "users",

MEDICATIONS = "medications",

THEMATIC\_MATERIALS = "thematicMaterials",

DIALOGS = "dialogs",

HEALTH\_STATES = "healthStates",

NOTIFICATIONS = "notifications",

}

export const PATIENT\_REPORT\_RANGES = {

[DailyLogPropName.SUGAR\_LEVEL]: {

min: 3.5,

max: 7,

},

[DailyLogPropName.PULSE]: {

min: 80,

max: 100,

},

[DailyLogPropName.SYSTOLIC]: {

min: 110,

max: 130,

},

[DailyLogPropName.DIASTOLIC]: {

min: 70,

max: 90,

},

[DailyLogPropName.WEIGHT]: {

min: 50,

max: 95,

},

[DailyLogPropName.TOTAL]: {

min: 2000,

max: 3500,

},

[DailyLogPropName.TEMPERATURE]: {

min: 36.1,

max: 37.2,

},

};

***dailyLog.converter.ts***

import { DailyLog, DailyLogPropName } from "firestore/types/collections.types";

import { DailyLogData } from "firestore/converters/dailyLog/types";

import { PartialBy } from "firestore/types/client.types";

import { deepCopy } from "deep-copy-ts";

import {

MEASUREMENTS,

RUSSIAN\_DAILY\_LOG\_PROP\_NAMES,

} from "firestore/converters/role/constants";

export const convertDailyLogToDailyLogData = (

dailyLog: DailyLog

): DailyLogData => {

const copiedDailyLog: PartialBy<DailyLog, "blood" | "calories"> =

deepCopy(dailyLog);

delete copiedDailyLog.blood;

delete copiedDailyLog.calories;

return {

...copiedDailyLog,

sugarLevel: dailyLog.blood.sugarLevel.toString(),

pulse: dailyLog.blood.pulse?.toString() ?? "",

systolic: dailyLog.blood.pressure?.systolic?.toString() ?? "",

diastolic: dailyLog.blood.pressure?.diastolic?.toString() ?? "",

total: dailyLog.calories?.total?.toString() ?? "",

carbohydratesIntake:

dailyLog.calories?.carbohydratesIntake?.toString() ?? "",

temperature: dailyLog.temperature?.toString() ?? "",

weight: dailyLog.weight?.toString() ?? "",

};

};

export const convertDailyLogDataToDailyLog = (

dailyLogData: DailyLogData

): DailyLog => {

const copiedDailyLogData: PartialBy<

DailyLogData,

| "sugarLevel"

| "pulse"

| "systolic"

| "diastolic"

| "total"

| "carbohydratesIntake"

| "temperature"

| "weight"

> = deepCopy(dailyLogData);

delete copiedDailyLogData.sugarLevel;

delete copiedDailyLogData.pulse;

delete copiedDailyLogData.systolic;

delete copiedDailyLogData.diastolic;

delete copiedDailyLogData.total;

delete copiedDailyLogData.carbohydratesIntake;

delete copiedDailyLogData.temperature;

delete copiedDailyLogData.weight;

return {

...copiedDailyLogData,

blood: {

sugarLevel: +dailyLogData.sugarLevel,

pulse: +dailyLogData.pulse,

pressure: {

systolic: +dailyLogData.systolic,

diastolic: +dailyLogData.diastolic,

},

},

calories: {

total: +dailyLogData.total,

carbohydratesIntake:

dailyLogData.carbohydratesIntake === ""

? 0

: +dailyLogData.carbohydratesIntake,

},

temperature: +dailyLogData.temperature,

weight: +dailyLogData.weight,

};

};

export const convertDailyLogPropNameToRussian = (propName: DailyLogPropName) =>

RUSSIAN\_DAILY\_LOG\_PROP\_NAMES[propName];

export const convertDailyLogPropNameToMeasurement = (

propName: DailyLogPropName

) => MEASUREMENTS[propName];

***types.ts***

import { DailyLog } from "firestore/types/collections.types";

import { Omit } from "firestore/types/client.types";

export type DailyLogData = Required<

Omit<DailyLog, "blood" | "calories" | "temperature" | "weight"> & {

sugarLevel: string;

pulse: string;

systolic: string;

diastolic: string;

total: string;

carbohydratesIntake: string;

temperature: string;

weight: string;

}

>;

***constants.ts***

import { DailyLogPropName, Role } from "firestore/types/collections.types";

export const RUSSIAN\_ROLES: {

[key in Role]: string;

} = {

[Role.PATIENT]: "пациент",

[Role.RELATIVE]: "родственник",

[Role.DOCTOR]: "доктор",

[Role.CONTENT\_MAKER]: "контент-мейкер",

[Role.MODERATOR]: "модератор",

[Role.ADMIN]: "админ",

};

export const RUSSIAN\_ROLES\_PLURAL: {

[key in Role]: string;

} = {

[Role.PATIENT]: "Пациенты",

[Role.RELATIVE]: "Родственники",

[Role.DOCTOR]: "Доктора",

[Role.CONTENT\_MAKER]: "Контент-мейкеры",

[Role.MODERATOR]: "Модераторы",

[Role.ADMIN]: "Админы",

};

export const RUSSIAN\_DAILY\_LOG\_PROP\_NAMES: {

[key in DailyLogPropName]: string;

} = {

[DailyLogPropName.SUGAR\_LEVEL]: "Уровень сахара",

[DailyLogPropName.PULSE]: "Пульс",

[DailyLogPropName.SYSTOLIC]: "Систолическое давление",

[DailyLogPropName.DIASTOLIC]: "Диастолическое давление",

[DailyLogPropName.WEIGHT]: "Вес",

[DailyLogPropName.TOTAL]: "Количество калорий",

[DailyLogPropName.TEMPERATURE]: "Температура",

};

export const MEASUREMENTS: {

[key in DailyLogPropName]: string;

} = {

[DailyLogPropName.SUGAR\_LEVEL]: "ммоль/л",

[DailyLogPropName.PULSE]: "уд/мин",

[DailyLogPropName.SYSTOLIC]: "ммрт",

[DailyLogPropName.DIASTOLIC]: "ммрт",

[DailyLogPropName.WEIGHT]: "кг",

[DailyLogPropName.TOTAL]: "ккал",

[DailyLogPropName.TEMPERATURE]: "°C",

};

***role.converter.ts***

import { Role } from "firestore/types/collections.types";

import {

RUSSIAN\_ROLES,

RUSSIAN\_ROLES\_PLURAL,

} from "firestore/converters/role/constants";

export const convertRoleToRussian = (role: Role): string => RUSSIAN\_ROLES[role];

export const convertRoleToRussianPlural = (role: Role): string =>

RUSSIAN\_ROLES\_PLURAL[role];

***user.converter.ts***

import { User, UserInfo } from "firestore/types/collections.types";

export const convertUserToUserInfo = (user: User): UserInfo => {

const clonedUser: User = {

...user,

};

delete clonedUser.diary;

delete clonedUser.employee;

delete clonedUser.relativePatient;

return JSON.parse(JSON.stringify(clonedUser));

};

***types.ts***

import { UserInfo } from "firestore/types/collections.types";

export type UserData = Pick<

UserInfo,

"docId" | "address" | "phone" | "imageUrl" | "email" | "role"

> & {

firstName: string;

lastName: string;

};

***userInfo.converter.ts***

import { UserInfo } from "firestore/types/collections.types";

import { PartialBy } from "firestore/types/client.types";

import { deepCopy } from "deep-copy-ts";

import { UserData } from "firestore/converters";

export const convertUserInfoToUserData = (userInfo: UserInfo): UserData => {

const copiedUserInfo: PartialBy<UserInfo, "name"> = deepCopy(userInfo);

delete copiedUserInfo.name;

return {

...copiedUserInfo,

firstName: userInfo.name.first,

lastName: userInfo.name.last,

};

};

export const convertUserDataToUserInfo = (userData: UserData): UserInfo => {

const copiedUserInfo: PartialBy<UserData, "firstName" | "lastName"> =

deepCopy(userData);

delete copiedUserInfo.firstName;

delete copiedUserInfo.lastName;

return {

...copiedUserInfo,

name: {

first: userData.firstName,

last: userData.lastName,

},

};

};

***helpers.ts***

import { ZodError } from "zod";

import { NotificationManager } from "react-notifications";

export const tryToExecute = <Out, ErrOut>({

callback,

error: { result, title, message },

}: {

callback: () => Out;

error: {

result: ErrOut;

title: string;

message: string;

};

}): Out | ErrOut => {

try {

return callback();

} catch (e) {

const err = e as ZodError;

console.log(err);

NotificationManager.error(message, title);

return result;

}

};

***repositories.ts***

import { FirebaseDocId } from "firestore/types/collections.types";

import { CollectionName } from "firestore/constants";

import {

collection,

CollectionReference,

doc,

Firestore,

getDoc,

getDocs,

getFirestore,

DocumentReference,

setDoc,

deleteDoc,

QueryFieldFilterConstraint,

query,

Query,

QueryCompositeFilterConstraint,

or,

} from "firebase/firestore";

import { firebaseApp } from "firebase\_config";

import { z } from "zod";

import { tryToExecute } from "firestore/data/helpers";

export type QueryFilter =

| QueryFieldFilterConstraint

| QueryCompositeFilterConstraint

| null;

export interface IRepository<T extends FirebaseDocId> {

getDocs: (queryFilter: QueryFilter) => Promise<T[]>;

getDocById: (docId: string) => Promise<T | null>;

updateDoc: (document: T) => Promise<void>;

deleteDocById: (docId: string) => Promise<void>;

}

export class Repository<T extends FirebaseDocId> implements IRepository<T> {

private validator: z.ZodSchema;

private firestore: Firestore;

private collectionName: CollectionName;

private collectionRef: CollectionReference<T>;

constructor(collectionName: CollectionName, validator: z.ZodSchema) {

this.validator = validator;

this.firestore = getFirestore(firebaseApp);

this.collectionName = collectionName;

this.collectionRef = collection(

this.firestore,

this.collectionName

) as CollectionReference<T>;

}

getDocs = async (queryFilter: QueryFilter = null): Promise<T[]> => {

const docsQuery: Query<T> =

queryFilter !== null

? query(this.collectionRef, or(queryFilter))

: this.collectionRef;

const docsSnapshot = await getDocs<T>(docsQuery);

const docs = docsSnapshot.docs.map((docSnapshot) => docSnapshot.data());

return tryToExecute<T[], []>({

callback: () => {

docs.forEach((document) => this.validator.parse(document));

return docs;

},

error: {

result: [],

title: `Репозиторий<${this.collectionName}> - [Получение документов]`,

message: "Неправильный формат документа",

},

});

};

getDocById = async (docId: string): Promise<T | null> => {

const docSnapshot = await getDoc<T>(

doc(this.firestore, this.collectionName, docId) as DocumentReference<T>

);

const document = docSnapshot.data();

if (!document) {

return null;

}

return tryToExecute<T, null>({

callback: () => {

this.validator.parse(document);

return document;

},

error: {

result: null,

title: `Репозиторий<${this.collectionName}> - [Получение документа по id]`,

message: "Неправильный формат документа",

},

});

};

updateDoc = async (document: T): Promise<void> => {

const isValid = tryToExecute<true, false>({

callback: () => {

this.validator.parse(document);

return true;

},

error: {

result: false,

title: `Репозиторий<${this.collectionName}> - [Обновление документа]`,

message: "Неправильный формат документа",

},

});

if (!isValid) {

return;

}

await setDoc(

doc(

this.firestore,

this.collectionName,

document.docId

) as DocumentReference<T>,

document

);

};

deleteDocById = async (docId: string): Promise<void> => {

await deleteDoc(

doc(this.firestore, this.collectionName, docId) as DocumentReference<T>

);

};

}

***helpers.ts***

import { faker } from "@faker-js/faker";

import { DailyLogPropName, UserInfo } from "./types/collections.types";

import { PATIENT\_REPORT\_RANGES } from "./constants";

export const generateDocId = () =>

faker.datatype.uuid().replaceAll("-", "").slice(0, 20);

export const getUserFullName = ({

name: { first, last },

}: Pick<UserInfo, "name">) => `${first} ${last}`;

export const calculatePatientReportDeviation = (

propName: DailyLogPropName,

value: number

) => {

const { min, max } = PATIENT\_REPORT\_RANGES[propName];

if (value < min) {

return +(value - min).toFixed(2);

}

if (value > max) {

return +(value - max).toFixed(2);

}

return 0;

};

***client.types.ts***

import { UserData } from "firestore/converters";

export type Omit<T, K extends keyof T> = Pick<T, Exclude<keyof T, K>>;

export type PartialBy<T, K extends keyof T> = Omit<T, K> & Partial<Pick<T, K>>;

export type ProfileFormErrors = {

[key in keyof Pick<

UserData,

"firstName" | "lastName" | "address" | "phone"

>]: string | null;

};

***collections.types.ts***

export type Timestamp = string;

export type AuthUserId = string;

export type FirebaseDocId = {

docId: AuthUserId;

};

export type User = FirebaseDocId & {

email: string;

password: string;

imageUrl: string;

name: {

first: string;

last: string;

};

address: string;

phone: string;

role: Role;

diary?: Diary;

employee?: {

hiredAt: Timestamp;

reviews: EmployeeReview[];

salary: number;

description: string;

};

relativePatient?: AuthUserId;

doctor?: string;

};

export type UserInfo = Omit<

User,

"diary" | "password" | "employee" | "relativePatient"

>;

export type Medication = FirebaseDocId & {

imageUrl: string;

title: string;

description: string;

instruction: string;

medicationRoute: MedicationRoute;

};

export type ThematicMaterial = FirebaseDocId & {

imageUrl: string;

content: string;

createdAt: Timestamp;

title: string;

description: string;

author: AuthUserId;

comments: Comment[];

};

export enum Role {

PATIENT = "patient",

RELATIVE = "relative",

DOCTOR = "doctor",

CONTENT\_MAKER = "content maker",

MODERATOR = "moderator",

ADMIN = "admin",

}

export enum GoalCategory {

EDUCATION = "education",

FITNESS = "fitness",

CAREER = "career",

OTHER = "other",

}

export enum GoalStatus {

COMPLETED = "completed",

IN\_PROGRESS = "in progress",

PENDING = "pending",

CANCELLED = "cancelled",

}

export enum MedicationRoute {

ORAL = "oral",

TOPICAL = "topical",

INJECTION = "injection",

}

export enum EmployeeReviewRate {

GREAT = "great",

ACCEPTABLE = "acceptable",

BAD = "bad",

TERRIBLE = "terrible",

}

export type Diary = {

diabetType: number;

dailyLogs: DailyLog[];

goals: Goal[];

};

export type DailyLog = {

createdAt: Timestamp;

takenMedications: TakenMedication[];

blood: {

sugarLevel: number;

pulse?: number;

pressure?: {

systolic: number;

diastolic: number;

};

};

calories?: {

total: number;

carbohydratesIntake?: number;

};

temperature?: number;

weight?: number;

notes: Note[];

};

export type TakenMedication = {

medication: Medication;

time: Timestamp;

dosage: number;

};

export type Note = {

createdAt: Timestamp;

content: string;

};

export type Goal = {

title: string;

description: string;

notes: Note[];

createdAt: Timestamp;

deadline: Timestamp;

category: GoalCategory;

status: GoalStatus;

progress: number;

};

export type Comment = {

message: Message;

score: number;

};

export type Message = {

createdAt: Timestamp;

content: string;

sender: AuthUserId;

};

export type EmployeeReview = {

createdAt: Timestamp;

score: number;

content: string;

reviewer: AuthUserId;

};

export enum DailyLogPropName {

SUGAR\_LEVEL = "sugarLevel",

PULSE = "pulse",

SYSTOLIC = "systolic",

DIASTOLIC = "diastolic",

WEIGHT = "weight",

TOTAL = "total",

TEMPERATURE = "temperature",

}

export type HealthState = FirebaseDocId & {

title: string;

propName: DailyLogPropName;

min: number;

max: number;

warning: string;

recommendation: string;

};

export enum PatientReplyStatus {

DONE = "Выполнена, спасибо!",

IGNORED = "Проигнорирована",

DENIED = "Отклонена",

NOT\_SEEN = "Не просмотрена",

DOESNT\_EXIST = "Не сформирована",

}

export type PatientReply = {

createdAt: Timestamp;

status: PatientReplyStatus;

comment: string;

};

export type Recommendation = {

createdAt: Timestamp;

healthStates: HealthState[];

comment: string;

patientReply: PatientReply | null;

};

export type PatientReport = {

currentValue: number;

propName: DailyLogPropName;

};

export type Notification = FirebaseDocId & {

createdAt: Timestamp;

patient: AuthUserId;

doctor: AuthUserId;

patientReports: PatientReport[];

recommendation: Recommendation | null;

};

***schemas.ts***

import { z } from "zod";

import {

DailyLogPropName,

EmployeeReviewRate,

GoalCategory,

GoalStatus,

MedicationRoute,

PatientReplyStatus,

Role,

} from "./collections.types";

export const authUserIdSchema = z.string();

export const firebaseDocIdSchema = z

.object({

docId: authUserIdSchema,

})

.strict();

export const timestampSchema = z.string();

export const roleSchema = z.nativeEnum(Role);

export const goalCategorySchema = z.nativeEnum(GoalCategory);

export const goalStatusSchema = z.nativeEnum(GoalStatus);

export const medicationRouteSchema = z.nativeEnum(MedicationRoute);

export const employeeReviewRateSchema = z.nativeEnum(EmployeeReviewRate);

export const noteSchema = z

.object({

createdAt: timestampSchema,

content: z.string(),

})

.strict();

export const goalSchema = z

.object({

title: z.string(),

description: z.string(),

notes: z.array(noteSchema),

createdAt: timestampSchema,

deadline: timestampSchema,

category: goalCategorySchema,

status: goalStatusSchema,

progress: z.number(),

})

.strict();

export const medicationSchema = firebaseDocIdSchema

.extend({

imageUrl: z.string(),

title: z.string(),

description: z.string(),

instruction: z.string(),

medicationRoute: medicationRouteSchema,

})

.strict();

export const takenMedicationSchema = z

.object({

medication: medicationSchema,

time: timestampSchema,

dosage: z.number(),

})

.strict();

export const messageSchema = z

.object({

createdAt: timestampSchema,

content: z.string(),

sender: authUserIdSchema,

})

.strict();

export const employeeReviewSchema = z

.object({

createdAt: timestampSchema,

score: z.number(),

content: z.string(),

reviewer: authUserIdSchema,

})

.strict();

export const dailyLogSchema = z

.object({

createdAt: timestampSchema,

takenMedications: z.array(takenMedicationSchema),

blood: z

.object({

sugarLevel: z.number(),

pulse: z.number().optional(),

pressure: z

.object({

systolic: z.number(),

diastolic: z.number(),

})

.strict()

.optional(),

})

.strict(),

calories: z

.object({

total: z.number(),

carbohydratesIntake: z.number().optional(),

})

.strict()

.optional(),

temperature: z.number().optional(),

weight: z.number().optional(),

notes: z.array(noteSchema),

})

.strict();

export const commentSchema = z

.object({

message: messageSchema,

score: z.number(),

})

.strict();

export const thematicMaterialSchema = firebaseDocIdSchema

.extend({

imageUrl: z.string(),

content: z.string(),

createdAt: timestampSchema,

title: z.string(),

description: z.string(),

author: authUserIdSchema,

comments: z.array(commentSchema),

})

.strict();

export const diarySchema = z

.object({

diabetType: z.number(),

dailyLogs: z.array(dailyLogSchema),

goals: z.array(goalSchema),

})

.strict();

export const userSchema = firebaseDocIdSchema

.extend({

email: z.string(),

password: z.string(),

imageUrl: z.string(),

name: z

.object({

first: z.string(),

last: z.string(),

})

.strict(),

address: z.string(),

phone: z.string(),

role: roleSchema,

diary: diarySchema.optional(),

employee: z

.object({

hiredAt: timestampSchema,

reviews: z.array(employeeReviewSchema),

salary: z.number(),

description: z.string(),

})

.strict()

.optional(),

relativePatient: authUserIdSchema.optional(),

doctor: z.string().optional(),

})

.strict();

export const dailyLogPropNameSchema = z.nativeEnum(DailyLogPropName);

export const healthStateSchema = firebaseDocIdSchema

.extend({

title: z.string(),

propName: dailyLogPropNameSchema,

min: z.number(),

max: z.number(),

warning: z.string(),

recommendation: z.string(),

})

.strict();

export const patientReplyStatusSchema = z.nativeEnum(PatientReplyStatus);

export const patientReplySchema = z

.object({

createdAt: timestampSchema,

status: patientReplyStatusSchema,

comment: z.string(),

})

.strict();

export const recommendationSchema = z

.object({

createdAt: timestampSchema,

healthStates: z.array(healthStateSchema),

comment: z.string(),

patientReply: patientReplySchema.or(z.null()),

})

.strict();

export const patientReportSchema = z

.object({

currentValue: z.number(),

propName: dailyLogPropNameSchema,

})

.strict();

export const notificationSchema = firebaseDocIdSchema

.extend({

createdAt: timestampSchema,

patient: authUserIdSchema,

doctor: authUserIdSchema,

patientReports: z.array(patientReportSchema),

recommendation: recommendationSchema.or(z.null()),

})

.strict();

***auth.helpers.ts***

import { UserCredential } from "firebase/auth";

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

import { NotificationManager } from "react-notifications";

import { Route } from "components/routing";

import { Role } from "firestore/types/collections.types";

import { firebaseRepositories } from "firestore/data/repositories";

import { faker } from "@faker-js/faker";

export const successfulAuth = async (

userCredential: UserCredential,

navigate: NavigateFunction

) => {

NotificationManager.success("Успешная авторизация!");

navigate(Route.profile);

};

export const successfulSignUp = async ({

firstName,

lastName,

email,

password,

role,

relativePatientId,

diabetType,

userCredential,

}: {

firstName: string;

lastName: string;

email: string;

password: string;

role: Role;

relativePatientId?: string;

diabetType?: number;

userCredential: UserCredential;

}) => {

const { uid } = userCredential.user;

const doctors = (await firebaseRepositories.users.getDocs()).filter(

(user) => user.role === Role.DOCTOR

);

const assignedDoctor = faker.helpers.arrayElement(doctors);

const patientData =

role === Role.PATIENT && diabetType

? {

diary: {

diabetType,

dailyLogs: [],

goals: [],

},

doctor: assignedDoctor.docId,

}

: undefined;

const relativeData =

role === Role.PATIENT

? undefined

: {

relativePatient: relativePatientId,

};

await firebaseRepositories.users.updateDoc({

docId: uid,

address: "",

email,

password,

imageUrl: "",

name: { first: firstName, last: lastName },

phone: "",

role,

...patientData,

...relativeData,

});

};

***helpers.ts***

import dayjs from "dayjs";

export const formatDate = (date: string) =>

dayjs(date).format("HH:mm, D MMMM YYYY");

***useGeneralDataHook.ts***

import { useState } from "react";

import { NotificationManager } from "react-notifications";

import useAsyncEffect from "use-async-effect";

export const useGeneralDataHook = <T>(

callback: () => Promise<T>,

deps: unknown[] = []

) => {

const [isLoading, setIsLoading] = useState<boolean>(false);

const [data, setData] = useState<T | null>(null);

const reset = async () => {

try {

setIsLoading(true);

const newData = await callback();

setData(newData);

} catch (e) {

console.log(e);

NotificationManager.error("Получение данных с сервера", "Ошибка сервера");

} finally {

setIsLoading(false);

}

};

useAsyncEffect(reset, deps);

return [isLoading, data, reset];

};

***useGeneralModalHandlers.ts***

import { useCallback, useState } from "react";

export const useGeneralModalHandlers = ({

onOpen,

onClose,

}: {

onOpen?: (...args: unknown[]) => void;

onClose?: () => void;

} = {}) => {

const [isModalOpened, setIsModalOpened] = useState(false);

const openModal = useCallback(

(...args: unknown[]) => {

onOpen?.(...args);

setIsModalOpened(true);

},

[setIsModalOpened, onOpen]

);

const closeModal = useCallback(() => {

onClose?.();

setIsModalOpened(false);

}, [setIsModalOpened, onClose]);

return [isModalOpened, openModal, closeModal];

};

***useWindowSize.ts***

import { useLayoutEffect, useState } from "react";

export const useWindowSize = (): [number, number] => {

const [size, setSize] = useState<[number, number]>([0, 0]);

useLayoutEffect(() => {

function updateSize() {

setSize([document.body.clientWidth, document.body.clientHeight]);

}

window.addEventListener("resize", updateSize);

updateSize();

return () => window.removeEventListener("resize", updateSize);

}, []);

return size;

};

***actions.ts***

import { setIsPending, setUser } from "store/reducers/user/userSlice";

import { firebaseRepositories } from "firestore/data/repositories";

import { NotificationManager } from "react-notifications";

import { AppDispatch } from "store/store";

export const fetchUser = (uid: string) => async (dispatch: AppDispatch) => {

try {

dispatch(setIsPending(true));

const user = await firebaseRepositories.users.getDocById(uid);

dispatch(setUser(user));

} catch (e) {

console.log(e);

NotificationManager.error(

"Загрузка пользователя из firestore!",

"Ошибка Firestore"

);

} finally {

dispatch(setIsPending(false));

}

};

***constants.ts***

export const ACTION\_NAMES = {

userSignIn: "users/sign-in",

userSignOut: "users/sign-out",

};

***userSlice.ts***

import { createSlice } from "@reduxjs/toolkit";

import dayjs from "dayjs";

import { DailyLog, User } from "firestore/types/collections.types";

export const MODULE\_NAME\_USER = "currentUser";

export type UserSliceState = {

user: User | null;

isPending: boolean;

};

const initialState: UserSliceState = {

user: null,

isPending: false,

};

const userSlice = createSlice({

name: MODULE\_NAME\_USER,

initialState,

reducers: {

setIsPending: (state, action) => {

state.isPending = action.payload;

},

setUser: (state, action) => {

state.user = action.payload;

},

setDailyLog: (state, action) => {

if (state.user?.diary) {

const dailyLog: DailyLog = action.payload;

state.user.diary.dailyLogs = [

...state.user.diary.dailyLogs.filter(

(log) =>

dayjs(log.createdAt).format("l") !==

dayjs(dailyLog.createdAt).format("l")

),

dailyLog,

];

}

},

},

});

export const userReducer = userSlice.reducer;

export const { setIsPending, setUser, setDailyLog } = userSlice.actions;

***selectors.ts***

import { GlobalState } from "store/store";

import { User } from "firestore/types/collections.types";

import {

convertDailyLogToDailyLogData,

convertUserToUserInfo,

} from "firestore/converters";

import { createSelector } from "reselect";

export const getUserSelector = (state: GlobalState) => {

return state.currentUser.user as User;

};

export const getUserIdSelector = createSelector(

[getUserSelector],

(user) => user.docId

);

export const getUserInfoSelector = createSelector([getUserSelector], (user) =>

convertUserToUserInfo(user)

);

export const getDailyLogsSelector = createSelector(

[getUserSelector],

(user) => user.diary?.dailyLogs ?? []

);

export const getDailyLogsDataSelector = createSelector(

[getDailyLogsSelector],

(dailyLogs) => dailyLogs.map(convertDailyLogToDailyLogData)

);

***store.ts***

import { combineReducers, configureStore, Reducer } from "@reduxjs/toolkit";

import {

MODULE\_NAME\_USER,

userReducer,

UserSliceState,

} from "store/reducers/user/userSlice";

import { useDispatch } from "react-redux";

import { ACTION\_NAMES } from "store/reducers/user/constants";

export type GlobalState = {

[MODULE\_NAME\_USER]: UserSliceState;

};

const appReducer: Reducer = combineReducers<GlobalState>({

[MODULE\_NAME\_USER]: userReducer,

});

const rootReducer: Reducer<GlobalState> = (state, action) => {

if (action.type === ACTION\_NAMES.userSignOut) {

return appReducer(undefined, action);

}

return appReducer(state, action);

};

export const store = configureStore({

reducer: rootReducer,

middleware: (getDefaultMiddleware) =>

getDefaultMiddleware({

serializableCheck: false,

}),

});

export type AppDispatch = typeof store.dispatch;

export const useAppDispatch = () => useDispatch<AppDispatch>();