Фронтенд часть веб-приложения реализована с использованием React. Вся кодовая база разбита на компоненты, которые являются наименьшими функциональными единицами и монтируются в браузере благодаря библиотеке React. Эти компоненты обмениваются информацией друг с другом, используя пропсы и контекст приложения. В пример можно привести два компонента: DoctorSelect (листинг 4.1) и AddExamination (листинг 4.2). Компонент DoctorSelect отображает выпадающий список с врачами и позволяет пользователю выбрать одного из них. При изменении выбора вызывается переданная через пропсы функция onChange, которая передает идентификатор выбранного врача в родительский компонент. Компонент AddExamination отвечает за добавление нового осмотра. Он содержит несколько состояний: newExamination для хранения данных осмотра, doctorId для хранения идентификатора выбранного врача и notification для отображения сообщений. Компонент включает в себя DoctorSelect и передает ему функцию handleChangeDoctor через пропсы, которая обновляет состояние doctorId при изменении выбора врача. Когда пользователь выбирает врача в DoctorSelect, вызывается функция handleChange, которая передает значение идентификатора выбранного врача через пропс onChange в компонент AddExamination. В результате функция handleChangeDoctor в AddExamination обновляет состояние doctorId, сохраняя выбранного врача. Таким образом, два компонента взаимодействуют и обмениваются данными через пропсы, что позволяет компоненту AddExamination использовать идентификатор выбранного врача при добавлении нового осмотра. Этот подход обеспечивает модульность и гибкость кода, делая его более организованным и легким для поддержки. Кроме того, использование контекста позволяет управлять глобальным состоянием приложения и передавать данные между компонентами без необходимости передачи пропсов через несколько уровней компонентов.

Листинг 4.1 – DoctorSelect.jsx

import { useEffect, useState } from "react";

import { getDoctorInfoRequest } from "../api/userApi.jsx";

import "./DoctorSelect.css";

export default function DoctorSelect({ onChange, selectedDoctorId }) {

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

const handleChange = (e) => {

onChange(e.target.value);

};

useEffect(() => {

const fetchDoctorInfo = async () => {

const data = await getDoctorInfoRequest();

setDoctors(data);

};

fetchDoctorInfo();

}, []);

return (

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

<select className="doctor-select" value={selectedDoctorId} onChange={handleChange}>

<option value="">Выберите врача</option>

{doctors.map(doctor => (

<option key={doctor.id} value={doctor.id}>{doctor.fio}</option>

))}

</select>

</div>

);

}

Листинг 4.2 – AddExamination.jsx

import Arrow from "../../Arrow/Arrow.jsx";

import DoctorSelect from "../DoctorSelect.jsx";

import { examinationAddRequest } from "../../api/patiensApi.jsx";

import { useState } from "react";

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

import "./EditExamination.css";

export default function AddExamination() {

const { historyId } = useParams();

const [newExamination, setNewExamination] = useState({

date: "",

conclusion: ""

});

const [doctorId, setDoctorId] = useState("");

const [notification, setNotification] = useState(" ");

const [notificationType, setNotificationType] = useState("");

const handleChangeDoctor = (childDoctorId) => {

setDoctorId(childDoctorId);

};

const handleChange = (e) => {

setNewExamination({ ...newExamination, [e.target.name]: e.target.value });

};

const handleSubmit = async (e) => {

e.preventDefault();

const requestData = {

date: newExamination.date,

conclusion: newExamination.conclusion,

userId: doctorId

};

const response = await examinationAddRequest(historyId, requestData);

if (response === 201) {

setNotification("Осмотр успешно добавлен");

setNotificationType("success");

} else {

setNotification("Произошла ошибка при добавлении");

setNotificationType("error");

}

};

return (

<div className="add-examination-container">

<Arrow />

<div className="add-examination-form-container">

<form className="add-examination-form">

<input type="date" name="date" placeholder="Дата осмотра" required onChange={handleChange} />

<textarea name="conclusion" placeholder="Заключение" required onChange={handleChange} />

<DoctorSelect onChange={handleChangeDoctor} />

<button type="submit" onClick={handleSubmit}>Добавить осмотр</button>

</form>

</div>

<pre className={`notification ${notificationType}`}>{notification}</pre>

</div>

);

}

Контекст используется для управления глобальным состоянием приложения, что позволяет избежать передачи пропсов через несколько уровней компонентов. Файл AppContext.jsx (листинг 4.3) предоставляет провайдер контекста и методы для управления состоянием, такие как установка роли пользователя и состояния авторизации. AppContext используется для хранения и управления глобальным состоянием приложения, таким как роль пользователя (role) и статус, является ли пользователь врачом (isDoctor). В AppProvider состояние и методы управления этим состоянием передаются через провайдер контекста. Это позволяет любому компоненту внутри провайдера получить доступ к этим данным и функциям, избегая передачи пропсов через несколько уровней компонентов. Когда приложение загружается, useEffect выполняет асинхронный запрос для получения роли пользователя и обновляет состояние role и isDoctor соответственно. Эти значения сохраняются в sessionStorage, чтобы их можно было легко восстановить при повторной загрузке приложения.

Листинг 4.3 – AppContext.jsx

import { createContext, useEffect, useState } from 'react';

import PropTypes from "prop-types";

import { getUserRole } from "../api/userApi.jsx";

export const AppContext = createContext(null);

export const AppProvider = ({ children }) => {

const [role, setRole] = useState(() => sessionStorage.getItem('role') || "");

const [isRoleSet, setIsRoleSet] = useState(false);

const [isDoctor, setIsDoctor] = useState(() => {

const savedIsDoctor = sessionStorage.getItem('isDoctor');

return savedIsDoctor === 'true';

});

useEffect(() => {

const fetchUserRole = async () => {

const role = await getUserRole();

setRole(role);

sessionStorage.setItem('role', role);

if (role === "Doctor") {

setIsDoctor(true);

sessionStorage.setItem('isDoctor', 'true');

} else {

setIsDoctor(false);

sessionStorage.setItem('isDoctor', 'false');

}

};

if (isRoleSet) {

fetchUserRole();

}

}, [isRoleSet]);

useEffect(() => {

sessionStorage.setItem('isDoctor', isDoctor);

}, [isDoctor]);

return (

<AppContext.Provider value={{ role, setRole, setIsRoleSet, isDoctor, setIsDoctor }}>

{children}

</AppContext.Provider>

);

};

AppProvider.propTypes = {

children: PropTypes.node.isRequired,

};

Часто используемые данные сохраняются в хранилище сессии, это дополнительно необходимо для сохранности информации, если кто-то из медицинского персонала забудет выйди из учетной записи. Также это ускоряет доступ к необходимой информации, избегая дополнительных запросов к серверу.