СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 8](#_Toc198892894)

[1 Постановка задачи 8](#_Toc198892895)

[1.1 Анализ предметной области 8](#_Toc198892896)

[1.2 Техническое задание на разработку ГОСТ 19.201-78. 9](#_Toc198892897)

[1.2.1 Введение 9](#_Toc198892898)

[1.2.1.1 Наименование программы 9](#_Toc198892899)

[1.2.1.2 Краткая характеристика области применения 9](#_Toc198892900)

[1.2.2 Основания для разработки 9](#_Toc198892901)

[1.2.2.1 Основание для проведения разработки 9](#_Toc198892902)

[1.2.2.2 Наименование темы разработки 9](#_Toc198892903)

[1.2.3 Назначение разработки 9](#_Toc198892904)

[1.2.3.1 Функциональное назначение 9](#_Toc198892905)

[1.2.3.2 Эксплуатационное назначение 9](#_Toc198892906)

[1.2.4 Требования к программе или программному изделию 10](#_Toc198892907)

[1.2.4.1 Требования к функциональным характеристикам 10](#_Toc198892908)

[1.2.4.2 Требования к надежности 10](#_Toc198892909)

[1.2.4.3 Время восстановления после отказа 10](#_Toc198892910)

[1.2.4.4 Отказы из-за некорректных действий оператора 10](#_Toc198892911)

[1.2.4.5 Условия эксплуатации 11](#_Toc198892912)

[1.2.4.6 Требования к видам обслуживания 11](#_Toc198892913)

[1.2.4.7 Требования к информационной и программной совместимости 11](#_Toc198892914)

[1.2.4.8 Требования к программной документации 12](#_Toc198892915)

[1.2.5.1 Стадии разработки 12](#_Toc198892916)

[1.2.5.2 Этапы разработки 12](#_Toc198892917)

[1.2.5.3 Содержание работ по этапам 13](#_Toc198892918)

[1.2.6 Порядок контроля и приёмки 13](#_Toc198892919)

[1.2.6.1 Виды испытаний 13](#_Toc198892920)

[2 Проектирование 15](#_Toc198892921)

[2.1 Функциональная структура программы 15](#_Toc198892922)

[2.2 Схема модулей программы 16](#_Toc198892923)

[2.3 Паспорта основных модулей 17](#_Toc198892924)

[2.3.1 Модуль Login 17](#_Toc198892925)

[2.3.2 Модуль Account 18](#_Toc198892926)

[2.3.3 Модуль WorkArea 19](#_Toc198892927)

[4.2 Листинг 19](#_Toc198892928)

[4.2.1 Листинг index.js 19](#_Toc198892929)

[4.2.2 Листинг WorkArea 20](#_Toc198892930)

[4.2.2 Листинг Toolbar 21](#_Toc198892931)

[4.2.4 Листинг PropertiesPanel 24](#_Toc198892932)

[4.2.5 Листинг Canvas 28](#_Toc198892933)

[4.2.2 Листинг Account 33](#_Toc198892934)

[4.2.2 Листинг WebSocketProvider 39](#_Toc198892935)

[4.2.2 Листинг Toaster 41](#_Toc198892936)

[4.2.2 Листинг Utils 42](#_Toc198892937)

[ЗАКЛЮЧЕНИЕ 44](#_Toc198892938)

# **ВВЕДЕНИЕ**

В современном мире цифровых технологий эффективная командная работа играет ключевую роль в успешном выполнении проектов, особенно в условиях удалённого взаимодействия и распределённых коллективов. Традиционные инструменты для совместной работы, такие как мессенджеры и таск-менеджеры, обеспечивают коммуникацию и управление задачами, но зачастую не предлагают интерактивных возможностей для творческого взаимодействия. В связи с этим актуальной становится разработка специализированных приложений, сочетающих в себе элементы визуального collaboration-инструментария и функционал для обмена файлами.

Данная работа посвящена созданию клиентской части приложения для командной работы, которое объединяет в себе функционал графического редактора (аналогичного работе с маркерной доской) с возможностью многопользовательского взаимодействия в реальном времени, а также инструменты для размещения и совместного использования файлов. Подобное решение позволит командам не только обсуждать задачи, но и визуализировать идеи, вносить правки в графические материалы и оперативно обмениваться документами в едином пространстве.

В процессе создания приложения будут использованы следующие инструменты и технологии:

* Язык программирования: JavaScript
* Библиотеки пользовательского интерфейса: React, Konva

# **1 Постановка задачи**

# **1.1 Анализ предметной области**

Цель разработки информационной системы (ИС) «Приложение для командной работы» заключается в создании универсального инструмента, который позволит эффективно управлять процессами разработки, внедрения и поддержки программного обеспечения. Это будет способствовать улучшению качества программных решений, оптимизации рабочего процесса и максимизации удовлетворенности пользователей.

# **1.2 Техническое задание на разработку ГОСТ 19.201-78.**

# **1.2.1 Введение**

# **1.2.1.1 Наименование программы**

Наименование программы «BrainStorm».

# **1.2.1.2 Краткая характеристика области применения**

Программа предназначена для ускорения процессов кооперативного принятия решений и планирования в бизнесе.

# **1.2.2 Основания для разработки**

# **1.2.2.1 Основание для проведения разработки**

Основанием для проведения разработки является задание на производственную преддипломную практику по модулю ПМ.01. Разработка модулей программного обеспечения для компьютерных систем.

# **1.2.2.2 Наименование темы разработки**

Наименование темы разработки: «Проектирование информационной системы «Приложение для командной работы».

# **1.2.3 Назначение разработки**

# **1.2.3.1 Функциональное назначение**

Функциональное назначение информационной системы (ИС) "Приложение для командной работы" представляет собой интерактивную платформу для совместной работы, сочетающую функции графического редактора с многопользовательским режимом и возможностью обмена файлами. Оно предназначено для команд, которым необходимо не только обсуждать задачи, но и визуализировать идеи, оперативно вносить правки в графические материалы и обмениваться документами в режиме реального времени.

# **1.2.3.2 Эксплуатационное назначение**

Требования эксплуатационного назначения не предъявляются.

# **1.2.4 Требования к программе или программному изделию**

# **1.2.4.1 Требования к функциональным характеристикам**

Программа должна обладать следующими функциональными характеристиками:

* функция входа в аккаунт,
* функция выхода из аккаунта,
* функция создания комнаты,
* функция выхода комнаты,
* функция рисования на холсте,
* функция изменения параметров компонента на холсте,
* функция удаления компонента на холсте,
* функция синхронизации компонентов,
* функция перемещения по холсту,
* функция загрузки изображения на холст.

# **1.2.4.2 Требования к надежности**

Требования к обеспечению надежного функционирования программы не предъявляются.

# **1.2.4.3 Время восстановления после отказа**

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать 1 часа для пользовательского интерфейса и 24 часов для бизнес-логики при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

# **1.2.4.4 Отказы из-за некорректных действий оператора**

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу пользователя без предоставления ему административных привилегий.

# **1.2.4.5 Условия эксплуатации**

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

# **1.2.4.6 Требования к видам обслуживания**

Программа не требует проведения каких-либо видов обслуживания.

# **1.2.4.7 Требования к информационной и программной совместимости**

**Требования к информационным структурам и методам решения**

Информационная структура приложения должна обеспечивать:

1. Отображение графического холста с инструментами рисования в режиме реального времени.
2. Поддержку загружаемых файлов (изображения, документы) с возможностью последующей выгрузкой файла на устройство пользователя.
3. Синхронизацию действий пользователей при совместном редактировании и исключения коллизий данных.

**Требования к исходным кодам и языкам программирования**

Исходные коды программы должны быть реализованы на языках JavaScript. В качестве интегрированной среды разработки программы должна быть использована среда Visual Studio Code.

**Требования к программным средствам, используемым программой**

Системные программные средства, используемые конечным пользователям, должны быть представлены лицензионной локализованной версией любой операционной системы с браузером на базе Chrome версии 100 и новее.

Системные программные средства, используемые для обработки данных и размещения веб-сервера, должны быть представлены лицензионной локализованной версией Debian 9 и новее, веб-сервером nginx версии 1.28 и новее, брокером сообщений RabbitMQ версии 4.0.0 и новее.

**Требования к защите информации и программ**

Информация, передаваемая между клиентом и сервером должна быть передана с использованием протокола SSL и шифрования.

**Специальные требования**

Программа должна обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса, разработанного согласно рекомендациям и современным тенденциям к разработке веб-интерфейсов.

# **1.2.4.8 Требования к программной документации**

Предварительный состав программной документации

Состав программной документации должен включать в себя:

* техническое задание на разработку ГОСТ 19.2010-78,
* паспорта модулей.

**1.2.5 Стадии и этапы разработки**

# **1.2.5.1 Стадии разработки**

Разработка должна быть проведена в две стадии:

* техническое задание,
* технический и рабочий проекты.

# **1.2.5.2 Этапы разработки**

На стадии «Техническое задание» должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии «Технический и рабочий проекты» должны быть выполнены перечисленные ниже этапы работ:

* разработка программы,
* разработка программной документации,
* испытания программы.

# **1.2.5.3 Содержание работ по этапам**

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

* постановка задачи,
* определение и уточнение требований к техническим средствам,
* определение требований к программе,
* определение стадий, этапов и сроков разработки программы и документации на неё,
* выбор языка программирования,
* согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями ГОСТ.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

* разработка, согласование и утверждение программы и методики испытаний,
* проведение приёмосдаточных испытаний,
* корректировка программы и программной документации по результатам испытаний.

На этапе подготовки и передачи программы должна быть выполнена работа по сдаче программы и программной документации преподавателю.

# **1.2.6 Порядок контроля и приёмки**

# **1.2.6.1 Виды испытаний**

Приёмосдаточные испытания должны проводиться на объекте заказчика в сроки выполнения для курсового проекта.

Приёмосдаточные испытания программы должны проводиться согласно разработанной исполнителем и согласованно заказчиком «Программы и методики испытаний».

# **2 Проектирование**

# **2.1 Функциональная структура программы**

Таблица 1 Спецификация функций программы

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Действие** | **Объект** | **Вид** |
| **Авторизация** | | | | |
| Вход в аккаунт | Вход в существующий аккаунт пользователя | Пользователь | Основная |
| Выход из аккаунта | Выход из текущего аккаунта пользователя | Пользователь | Основная |
| **Аккаунт** | | | |
| Создание комнаты | Создание новой комнаты для коллабораций | Название комнаты | Основная |
| Удаление комнаты | Удаление существующей комнаты | Комната | Основная |
| **Холст** | | | |
| Перо | Рисование на холсте | Холст | Основная |
| Рука | Выбор элемента на холсте, трансформация, удаление | Объект | Основная |
| Табличка | Добавление текста на холст | Холст | Основная |
| Картинка | Добавление картинки на холст | Холст | Основная |
| Панель свойств | Редактирование свойств компонентов | Холст | Вспомогательная |
| Панель инструментов | Отображение доступных инструментов | Холст | Вспомогательная |
| **Глобальные** | | | |
| «Тостер» | Показ системных уведомлений | «Тост» | Вспомогательная |
| Сервис авторизации | Взаимодействие с API авторизации | Данные пользователя | Вспомогательная |
| Сервис синхронизации | Взаимодействие с WebSocket для синхронизации действий | Данные сессии | Вспомогательная |

# **2.2 Схема модулей программы**

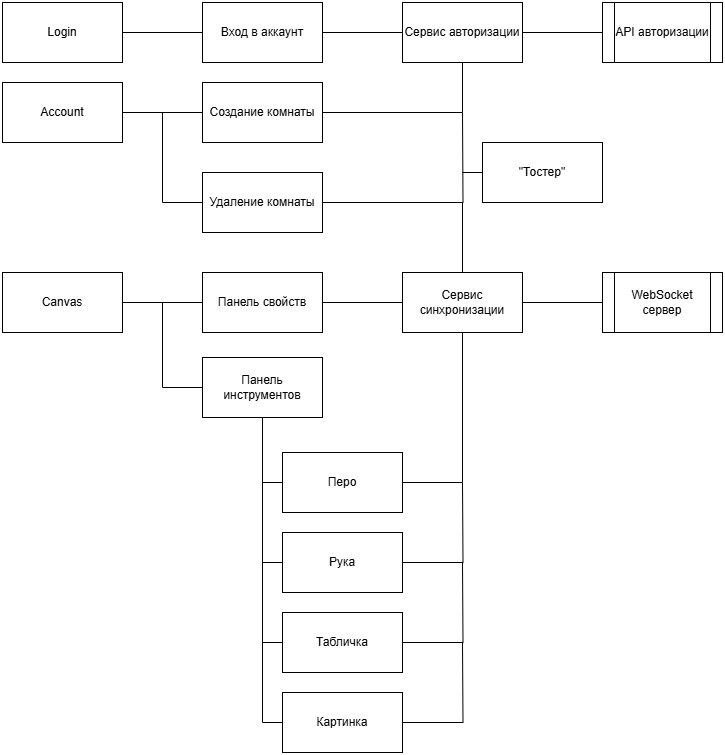


Рисунок 1 – Модульная структура программы

# **2.3 Паспорта основных модулей**

Примечание к паспортам модулей: в связи с особенностями разработки с использованием фреймворка React, блок схемы будут отражать общую суть логики действий программы.

Паспорт модулей был выполнен для основных представлений веб-приложений.

# **2.3.1 Модуль Login**

Таблица 2 Спецификация модуля Login

|  |  |  |
| --- | --- | --- |
| **Имя** | **Реализуемая функция** | **Параметры** |
| function Login() | Вывод страницы авторизации | нет |
| function onLoginClick(e) | Обработка нажатия на кнопку входа в аккаунт | e – событие входа в аккаунт |

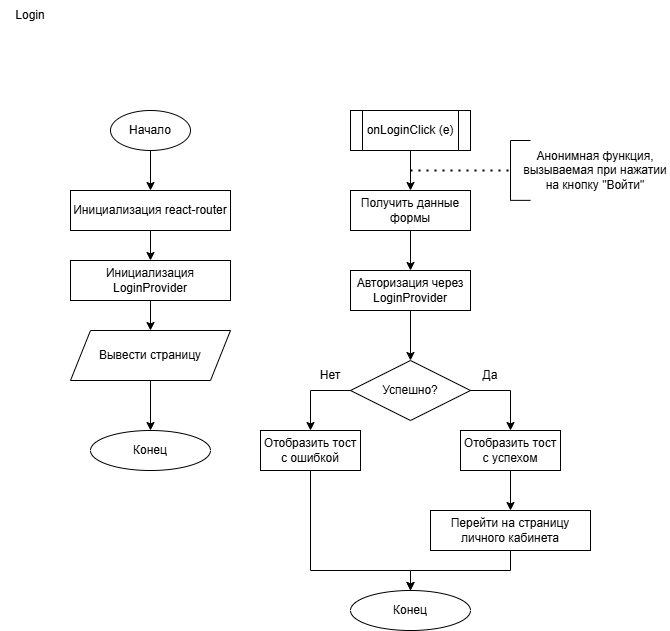


Рисунок 2 – Алгоритм страницы авторизации

# **2.3.2 Модуль Account**

Таблица 3 Спецификация модуля Account

|  |  |  |
| --- | --- | --- |
| **Имя** | **Реализуемая функция** | **Параметры** |
| UserProfile | Вывод информации о пользователе и его комнатах | Нет |

Блок-схема модуля находится в приложении А, так как её размеры не позволяют уместить на одной странице А4.

# **2.3.3 Модуль WorkArea**

Таблица 4 Спецификация модуля WorkArea

|  |  |  |
| --- | --- | --- |
| **Имя** | **Реализуемая функция** | **Параметры** |
| function WorkArea | Рабочая область (рисование, синхронизация, инструменты) | ID комнаты через адресную строку. |

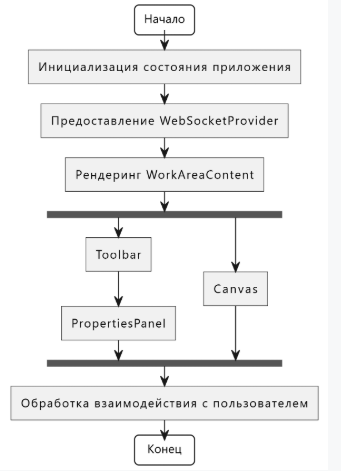


Рисунок 3 – Алгоритм удаления записи

# **4.2 Листинг**

Актуальный листинг программы доступен по адресу https://github.com/cd-con/brainstorm

# **4.2.1 Листинг index.js**

import ReactDOM from 'react-dom/client';

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

import './index.css';

//import reportWebVitals from './reportWebVitals';

import 'bootstrap/dist/css/bootstrap.min.css';

import Toaster from './utility/toaster'

import Login from './pages/login';

import UserProfile from './pages/account';

import WorkArea from './pages/canvas/WorkArea';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <BrowserRouter>

    <Toaster/>

    <Routes>

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

      <Route path="you" element={<UserProfile />} />

      <Route path="/canvas/:canvasId" element={<WorkArea/>} />

    </Routes>

  </BrowserRouter>

);

//reportWebVitals();

# **4.2.2 Листинг WorkArea**

import React, { useState } from 'react';

import Toolbar from './Toolbar';

import Canvas from './Canvas';

import PropertiesPanel from './PropertiesPanel';

import WebSocketProvider from '../../providers/room/WebSocketProvider.js';

import { iAmBusy } from '../../utility/utils';

const WorkArea = () => {

  const [tool, setTool] = useState('pen');

  const [lines, setLines] = useState([]);

  const [texts, setTexts] = useState([]);

  const [images, setImages] = useState([]);

  const [selectedId, setSelectedId] = useState(null);

  const [color, setColor] = useState('#000000');

  const [scale, setScale] = useState(1);

  const [stagePos, setStagePos] = useState({ x: 0, y: 0 });

  // TODO Move to toolbar

  const addText = async (syncUpdate) => {

    const newText = {

      id: Date.now(),

      text: 'New Text',

      x: 50,

      y: 50,

      fontSize: 20,

      fill: color,

      zIndex: 1

    };

    setTexts((prev) => [...prev, newText]);

    const response = await syncUpdate(newText.id, 'text', newText, () => {

      setTexts((prev) => prev.filter((t) => t.id !== newText.id));

    });

    if (!response.success) {

      iAmBusy(`Failed to add text: ${response.error}`);

    }

  };

  return (

    <WebSocketProvider setLines={setLines} setTexts={setTexts} setImages={setImages} setSelectedId={setSelectedId}>

      <div style={{ position: 'relative', height: '100vh', width: '100vw' }}>

        <div

          className="d-flex flex-column position-absolute top-50 start-0 translate-middle-y rounded bg-light p-2 ms-2"

          style={{ zIndex: 10, width: '250px' }}

        >

          <Toolbar

            tool={tool}

            setTool={setTool}

            addText={addText}

            setImages={setImages}

            images={images}

            color={color}

            setColor={setColor}

            selectedId={selectedId}

            setLines={setLines}

            setTexts={setTexts}

            scale={scale}

            setScale={setScale}

            setStagePos={setStagePos}

          />

          <PropertiesPanel

            selectedId={selectedId}

            lines={lines}

            texts={texts}

            images={images}

            setLines={setLines}

            setTexts={setTexts}

            setImages={setImages}

            color={color}

            setColor={setColor}

          />

        </div>

        <Canvas

          tool={tool}

          lines={lines}

          setLines={setLines}

          texts={texts}

          setTexts={setTexts}

          images={images}

          setImages={setImages}

          selectedId={selectedId}

          setSelectedId={setSelectedId}

          color={color}

          scale={scale}

          setScale={setScale}

          stagePos={stagePos}

          setStagePos={setStagePos}

        />

      </div>

    </WebSocketProvider>

  );

};

export default WorkArea;

# **4.2.2 Листинг Toolbar**

import React, { useContext } from 'react';

import { Button, ButtonGroup, Form, InputGroup } from 'react-bootstrap';

import { WebSocketContext } from '../../providers/room/WebSocketProvider.js';

import { iAmBusy } from '../../utility/utils';

import { useNavigate } from 'react-router';

const Toolbar = ({

  tool,

  setTool,

  addText,

  setImages,

  images,

  color,

  setColor,

  selectedId,

  setLines,

  setTexts,

  scale,

  setScale,

  setStagePos

}) => {

  const navigate = useNavigate();

  const { syncUpdate, uploadImage } = useContext(WebSocketContext);

  // Добавляем недостающую функцию

  const addImage = async (e) => {

    const file = e.target.files[0];

    if (!file) return;

    const reader = new FileReader();

    reader.onload = () => {

      const img = new window.Image();

      img.src = reader.result;

      img.onload = async () => {

        const newImage = {

          id: Date.now(),

          image: img,

          x: 50,

          y: 50,

          width: 100,

          height: 100,

          zIndex: 1,

        };

        setImages((prev) => [...prev, newImage]);

        const response = await syncUpdate(newImage.id, 'image', newImage, () => {

          setImages((prev) => prev.filter((i) => i.id !== newImage.id));

        });

        if (response.success) {

          uploadImage(newImage.id, reader.result);

        } else {

          iAmBusy(`Failed to add image: ${response.error}`);

          setImages((prev) => prev.filter((i) => i.id !== newImage.id));

        }

      };

    };

    reader.readAsDataURL(file);

  };

  const handleZoomIn = () => setScale(prev => Math.min(prev + 0.1, 4));

  const handleZoomOut = () => setScale(prev => Math.max(prev - 0.1, 0.1));

  const handleZoomReset = () => {

    setScale(1);

    setStagePos({ x: 0, y: 0 });

  };

  return (

    <div className="d-flex flex-wrap align-items-center gap-3 p-2 bg-light border-bottom">

      <Button variant="outline-secondary" onClick={() => navigate("/you")}>🚪</Button>

      <ButtonGroup>

        <Button

          variant={tool === 'select' ? 'primary' : 'outline-primary'}

          onClick={() => setTool('select')}

          title="Выделение"

        >

          ✥

        </Button>

        <Button

          variant={tool === 'pen' ? 'primary' : 'outline-primary'}

          onClick={() => setTool('pen')}

          title="Карандаш"

        >

          ✏️

        </Button>

      </ButtonGroup>

      <ButtonGroup>

        <Button

          variant="outline-success"

          onClick={() => { setTool('select'); addText(syncUpdate); }}

          title="Добавить текст"

        >

          🔤

        </Button>

        <Form.Control

          type="file"

          accept="image/\*"

          onChange={addImage}

          style={{ display: 'none' }}

          id="image-upload"

        />

        <Button

          variant="outline-success"

          onClick={() => { setTool('select'); document.getElementById('image-upload').click(); }}

          title="Добавить изображение"

        >

          🖼️

        </Button>

      </ButtonGroup>

      <InputGroup style={{ width: 'auto' }}>

        <Button variant="outline-secondary" onClick={handleZoomOut} title="Уменьшить">➖</Button>

        <Button variant="outline-secondary" onClick={handleZoomReset} title="Сбросить масштаб">

          {Math.round(scale \* 100)}%

        </Button>

        <Button variant="outline-secondary" onClick={handleZoomIn} title="Увеличить">➕</Button>

      </InputGroup>

      <Form.Control

        type="color"

        value={color}

        onChange={(e) => setColor(e.target.value)}

        style={{ width: '40px' }}

        title="Выберите цвет"

      />

    </div>

  );

};

export default Toolbar;

# **4.2.4 Листинг PropertiesPanel**

import React, { useContext, useState } from 'react';

import { Form, FormControl, InputGroup, Button } from 'react-bootstrap';

import { WebSocketContext } from '../../providers/room/WebSocketProvider.js';

import { iAmBusy } from '../../utility/utils';

const PropertiesPanel = ({ selectedId, lines, texts, images, setLines, setTexts, setImages, color, setColor }) => {

  const { syncUpdate, uploadImage } = useContext(WebSocketContext);

  const [imageUrl, setImageUrl] = useState('');

  if (!selectedId) {

    return <div className="p-3" style={{ background: '#f8f9fa' }}>Выберите объект для редактирования</div>;

  }

  const [type, id] = selectedId.split('-');

  const selectedObject =

  type === 'line' ? lines.find((line) => line.id === parseInt(id)) :

  type === 'text' ? texts.find((text) => text.id === parseInt(id)) :

  type === 'image' ? images.find((img) => img.id === parseInt(id)) :

  null;

  // Дополнительная проверка

  // DEPRECATED - возможно стоит убрать

  if (!selectedObject) {

    return <div className="p-3" style={{ background: '#f8f9fa' }}>Объект не выбран</div>;

  }

  const handleTextChange = async (e) => {

    if (type === 'text') {

      const newText = e.target.value;

      setTexts((prev) =>

        prev.map((t) => (t.id === parseInt(id) ? { ...t, text: newText } : t))

      );

      const response = await syncUpdate(id, 'text', { text: newText }, () => {

        setTexts((prev) =>

          prev.map((t) => (t.id === parseInt(id) ? { ...t, text: selectedObject.text } : t))

        );

      });

      if (!response.success) {

        iAmBusy(`Не удалось обновить текст: ${response.error}`);

      }

    }

  };

  const handleColorChange = async (e) => {

    const newColor = e.target.value;

    setColor(newColor);

    if (type === 'text') {

      setTexts((prev) =>

        prev.map((t) => (t.id === parseInt(id) ? { ...t, fill: newColor } : t))

      );

      const response = await syncUpdate(id, 'text', { fill: newColor }, () => {

        setTexts((prev) =>

          prev.map((t) => (t.id === parseInt(id) ? { ...t, fill: selectedObject.fill } : t))

        );

      });

      if (!response.success) {

        iAmBusy(`Не удалось изменить цвет: ${response.error}`);

      }

    } else if (type === 'line') {

      setLines((prev) =>

        prev.map((l) => (l.id === parseInt(id) ? { ...l, color: newColor } : l))

      );

      const response = await syncUpdate(id, 'line', { color: newColor }, () => {

        setLines((prev) =>

          prev.map((l) => (l.id === parseInt(id) ? { ...l, color: selectedObject.color } : l))

        );

      });

      if (!response.success) {

        iAmBusy(`Не удалось изменить цвет: ${response.error}`);

      }

    }

  };

  const handleImageUpload = async (e) => {

    const file = e.target.files[0];

    if (!file) return;

    const reader = new FileReader();

    reader.onload = async () => {

      const img = new window.Image();

      img.src = reader.result;

      img.onload = async () => {

        const newImage = {

          image: img,

          width: img.width,

          height: img.height,

          url: null

        };

        setImages((prev) =>

          prev.map((i) => (i.id === parseInt(id) ? { ...i, ...newImage } : i))

        );

        const response = await syncUpdate(id, 'image', newImage, () => {

          setImages((prev) =>

            prev.map((i) => (i.id === parseInt(id) ? { ...i, ...selectedObject } : i))

          );

        });

        if (response.success) {

          await uploadImage(id, reader.result);

        } else {

          iAmBusy(`Не удалось обновить изображение: ${response.error}`);

        }

      };

    };

    reader.readAsDataURL(file);

  };

  const handleImageUrlChange = async () => {

    if (!imageUrl) return;

    const img = new window.Image();

    img.src = imageUrl;

    img.onload = async () => {

      const newImage = {

        url: imageUrl,

        width: img.width,

        height: img.height,

        image: null

      };

      setImages((prev) =>

        prev.map((i) => (i.id === parseInt(id) ? { ...i, ...newImage } : i))

      );

      const response = await syncUpdate(id, 'image', newImage, () => {

        setImages((prev) =>

          prev.map((i) => (i.id === parseInt(id) ? { ...i, ...selectedObject } : i))

        );

      });

      if (!response.success) {

        iAmBusy(`Не удалось обновить изображение: ${response.error}`);

      }

    };

    img.onerror = () => {

      iAmBusy('Не удалось загрузить изображение по указанной ссылке');

    };

  };

  const handleZIndexChange = async (e) => {

    const newZIndex = parseInt(e.target.value) || 0;

    const updateStateAndSync = async (state, setState, objType) => {

      setState((prev) =>

        prev.map((item) => (item.id === parseInt(id) ? { ...item, zIndex: newZIndex } : item))

      );

      const response = await syncUpdate(id, objType, { zIndex: newZIndex }, () => {

        setState((prev) =>

          prev.map((item) => (item.id === parseInt(id) ? { ...item, zIndex: selectedObject.zIndex } : item))

        );

      });

      if (!response.success) {

        alert(`Не удалось обновить Z-индекс: ${response.error}`);

      }

    };

    if (type === 'line') {

      await updateStateAndSync(lines, setLines, 'line');

    } else if (type === 'text') {

      await updateStateAndSync(texts, setTexts, 'text');

    } else if (type === 'image') {

      await updateStateAndSync(images, setImages, 'image');

    }

  };

  return (

    <div className="p-3" style={{ background: '#f8f9fa', zIndex: 10 }}>

      <h5>Свойства объекта</h5>

      <Form onSubmit={(e) => e.preventDefault()}>

        <InputGroup className="mb-3">

          <InputGroup.Text>Z-индекс</InputGroup.Text>

          <FormControl

            value={selectedObject.zIndex || 0}

            type="number"

            onChange={handleZIndexChange}

            placeholder="Позиция по оси Z"

          />

        </InputGroup>

        {type === 'text' && (

          <>

            <InputGroup className="mb-3">

              <InputGroup.Text>Текст</InputGroup.Text>

              <FormControl

                value={selectedObject.text}

                onChange={handleTextChange}

                placeholder="Введите текст"

              />

            </InputGroup>

            <InputGroup className="mb-3">

              <InputGroup.Text>Цвет текста</InputGroup.Text>

              <FormControl

                type="color"

                value={selectedObject.fill}

                onChange={handleColorChange}

                style={{ width: '50px' }}

              />

            </InputGroup>

          </>

        )}

        {type === 'line' && (

          <InputGroup className="mb-3">

            <InputGroup.Text>Цвет линии</InputGroup.Text>

            <FormControl

              type="color"

              value={selectedObject.color}

              onChange={handleColorChange}

              style={{ width: '50px' }}

            />

          </InputGroup>

        )}

        {type === 'image' && (

          <>

            <InputGroup className="mb-3">

              <FormControl

                type="file"

                accept="image/\*"

                onChange={handleImageUpload}

                placeholder="Загрузить изображение"

              />

            </InputGroup>

            <InputGroup className="mb-3">

              <FormControl

                value={imageUrl}

                onChange={(e) => setImageUrl(e.target.value)}

                placeholder="Ссылка на изображение"

              />

              <Button

                variant="outline-secondary"

                onClick={handleImageUrlChange}

              >

                Обновить

              </Button>

            </InputGroup>

          </>

        )}

      </Form>

    </div>

  );

};

export default PropertiesPanel;

# **4.2.5 Листинг Canvas**

import { useRef, useEffect, useState, useContext } from 'react';

import { Stage, Layer, Line, Text, Image, Transformer } from 'react-konva';

import { handleMouseDown, handleMouseMove, handleMouseUp, handleSelect, iAmBusy } from '../../utility/utils.js';

import { WebSocketContext } from '../../providers/room/WebSocketProvider.js';

const GRID\_SIZE = 50;

const GRID\_COLOR = '#e0e0e0';

const MIN\_SCALE = 0.1;

const MAX\_SCALE = 4;

const Canvas = ({ tool, lines, setLines, texts, setTexts, images, setImages, selectedId, setSelectedId, color, scale, setScale, stagePos, setStagePos }) => {

  const { selectObject, deselectObject, syncUpdate } = useContext(WebSocketContext);

  const isDrawing = useRef(false);

  const stageRef = useRef(null);

  const transformerRef = useRef(null);

  const [dimensions, setDimensions] = useState({

    width: window.innerWidth,

    height: window.innerHeight,

  });

  if (stagePos === null || stagePos === undefined){

    stagePos = {x: 0, y: 0}

  }

  const drawGrid = () => {

  console.warn("DrawGrid feature temporary disabled due to incorrect drawing state. This is not an error.")

  return [];

  }

  // Горизонтальные линии

  for (let y = startY; y < endY; y += GRID\_SIZE) {

    gridLines.push(

      <Line

        key={`hline-${y}`}

        points={[-stage.x(), y \* scale + stage.y(), width - stage.x(), y \* scale + stage.y()]}

        stroke={GRID\_COLOR}

        strokeWidth={1}

        listening={false}

      />

    );

  }

  return gridLines;

};

  useEffect(() => {

    const handleResize = () => {

      setDimensions({

        width: window.innerWidth,

        height: window.innerHeight,

      });

    };

    window.addEventListener('resize', handleResize);

    return () => window.removeEventListener('resize', handleResize);

  }, []);

  useEffect(() => {

    if (!transformerRef.current || !selectedId) {

      if (transformerRef.current) {

        transformerRef.current.nodes([]);

      }

      return;

    }

    const node = stageRef.current?.findOne(`#${selectedId}`);

    if (node) {

      transformerRef.current.nodes([node]);

      //node.moveToTop();

      transformerRef.current.getLayer()?.batchDraw();

    } else {

      transformerRef.current.nodes([]);

    }

  }, [selectedId]);

  const handleWheel = (e) => {

    //fix

    e.evt.preventDefault();

    const stage = stageRef.current;

    if (!stage) return;

    const oldScale = scale;

    const pointer = stage.getPointerPosition();

    if (!pointer) return;

    const newScale = e.evt.deltaY > 0

      ? Math.max(oldScale \* 0.9, MIN\_SCALE)

      : Math.min(oldScale / 0.9, MAX\_SCALE);

    const mousePointTo = {

      x: (pointer.x - stage.x()) / oldScale,

      y: (pointer.y - stage.y()) / oldScale,

    };

    setScale(newScale);

    setStagePos({

      x: pointer.x - mousePointTo.x \* newScale,

      y: pointer.y - mousePointTo.y \* newScale,

    });

  };

  const handleStageClick = (e) => {

    if (e.target === e.currentTarget && tool === 'select' && selectedId) {

      deselectObject(selectedId.split('-')[1], selectedId.split('-')[0]);

      setSelectedId(null);

    }

  };

  const handleStageDragMove = (e) => {

    if (tool === 'select' && !selectedId) {

      setStagePos({

        x: e.target.x(),

        y: e.target.y(),

      });

    }

  };

  const handleStageDragEnd = (e) => {

    if (tool === 'select' && !selectedId) {

      setStagePos({

        x: e.target.x(),

        y: e.target.y(),

      });

    }

  };

  const handleDelete = async (e) => {

    if (e.key === 'Delete' && selectedId) {

      const [type, id] = selectedId.split('-');

      const idNum = parseInt(id);

      let prevState;

      if (type === 'line') {

        prevState = [...lines];

        setLines(prev => prev.filter(l => l.id !== idNum));

      } else if (type === 'text') {

        prevState = [...texts];

        setTexts(prev => prev.filter(t => t.id !== idNum));

      } else if (type === 'image') {

        prevState = [...images];

        setImages(prev => prev.filter(i => i.id !== idNum));

      }

      setSelectedId(null); // УДАЛИТЬ!

      const response = await syncUpdate(id, type, null, () => {

        if (type === 'line') setLines(prevState);

        else if (type === 'text') setTexts(prevState);

        else if (type === 'image') setImages(prevState);

      });

      // ИСПОЛЬЗОВАТЬ КОГДА БУДЕТ ГОТОВ БЭК

      /\*

      if (response.success) {

        setSelectedId(null);

      } else {

        alert(`Failed to delete: ${response.error}`);

      }\*/

    }

  };

  useEffect(() => {

    window.addEventListener('keydown', handleDelete);

    return () => window.removeEventListener('keydown', handleDelete);

  }, [selectedId, lines, texts, images]);

  const handleDragEnd = async (e, obj) => {

    if (!selectedId) return;

    const newPos = { x: e.target.x(), y: e.target.y() };

    let prevState;

    if (obj.type === 'line') {

      prevState = [...lines];

      setLines(prev => prev.map(l => l.id === obj.id ? { ...l, ...newPos } : l));

    } else if (obj.type === 'text') {

      prevState = [...texts];

      setTexts(prev => prev.map(t => t.id === obj.id ? { ...t, ...newPos } : t));

    } else if (obj.type === 'image') {

      prevState = [...images];

      setImages(prev => prev.map(i => i.id === obj.id ? { ...i, ...newPos } : i));

    }

    const response = await syncUpdate(obj.id, obj.type, newPos, () => {

      if (obj.type === 'line') setLines(prevState);

      else if (obj.type === 'text') setTexts(prevState);

      else if (obj.type === 'image') setImages(prevState);

    });

    if (!response.success) {

      alert(`Failed to update position: ${response.error}`);

    }

  };

  return (

    <Stage

      width={dimensions.width}

      height={dimensions.height}

      scaleX={scale}

      scaleY={scale}

      x={stagePos.x}

      y={stagePos.y}

      onMouseDown={(e) => handleMouseDown(

  e,

  tool,

  isDrawing,

  lines,

  setLines,

  color,

  syncUpdate,

  scale,

  stagePos

)}

      onMouseMove={(e) => handleMouseMove(

  e,

  tool,

  isDrawing,

  lines,

  setLines,

  syncUpdate,

  scale,

  stagePos

)}

      onMouseUp={() => handleMouseUp(isDrawing)}

      onWheel={handleWheel}

      onClick={handleStageClick}

      onDragMove={handleStageDragMove}

      onDragEnd={handleStageDragEnd}

      ref={stageRef}

      style={{ position: 'absolute', top: 0, left: 0 }}

      draggable={tool === 'select' && !selectedId}

    >

      <Layer>

        {drawGrid()}

        {[...lines.map(line => ({ ...line, type: 'line' })),

         ...texts.map(text => ({ ...text, type: 'text' })),

         ...images.map(img => ({ ...img, type: 'image' }))]

          .sort((a, b) => (a.zIndex || 0) - (b.zIndex || 0))

          .map((obj) => {

            const isSelected = true;//selectedId === `${obj.type}-${obj.id}`;

            if (obj.type === 'line') {

              return (

                <Line

                  key={`line-${obj.id}`}

                  id={`line-${obj.id}`}

                  points={obj.points}

                  stroke={obj.color}

                  strokeWidth={5 / scale}

                  lineCap="round"

                  lineJoin="round"

                  draggable={tool === 'select'}

                  onClick={(e) => {

                    if (tool === 'select') {

                      handleSelect(obj.id, 'line', selectObject);

                      setSelectedId(`line-${obj.id}`);

                      e.cancelBubble = true;

                    }

                  }}

                  onDragEnd={(e) => handleDragEnd(e, obj)}

                />

              );

            } else if (obj.type === 'text') {

              return (

                <Text

                  key={`text-${obj.id}`}

                  id={`text-${obj.id}`}

                  text={obj.text}

                  x={obj.x}

                  y={obj.y}

                  fontSize={obj.fontSize / scale}

                  fill={obj.fill}

                  draggable={tool === 'select'}

                  onClick={(e) => {

                    if (tool === 'select') {

                      handleSelect(obj.id, 'text', selectObject);

                      setSelectedId(`text-${obj.id}`);

                      e.cancelBubble = true;

                    }

                  }}

                  onDragEnd={(e) => handleDragEnd(e, obj)}

                />

              );

            } else if (obj.type === 'image') {

              const imageObj = obj.image || (obj.url ? new window.Image() : null);

              if (obj.url && imageObj) {

                imageObj.src = obj.url;

              }

              return (

                <Image

                  key={`image-${obj.id}`}

                  id={`image-${obj.id}`}

                  image={imageObj}

                  x={obj.x}

                  y={obj.y}

                  width={obj.width / scale}

                  height={obj.height / scale}

                  draggable={tool === 'select'}

                  onClick={(e) => {

                    if (tool === 'select') {

                      handleSelect(obj.id, 'image', selectObject);

                      setSelectedId(`image-${obj.id}`);

                      e.cancelBubble = true;

                    }

                  }}

                  onDragEnd={(e) => handleDragEnd(e, obj)}

                />

              );

            }

            return null;

          })}

        <Transformer

          ref={transformerRef}

          rotateEnabled={true}

          visible={selectObject != null}

          boundBoxFunc={(oldBox, newBox) => {

            if (newBox.width < 5 || newBox.height < 5) {

              return oldBox;

            }

            return newBox;

          }}

        />

      </Layer>

    </Stage>

  );

};

export default Canvas;

# **4.2.2 Листинг Account**

import { useState, useEffect } from 'react';

import { useNavigate } from 'react-router';

import { Container, Row, Col, Card, Button, ListGroup, Modal, Form, Spinner, Alert } from 'react-bootstrap';

import 'bootstrap/dist/css/bootstrap.min.css';

import LoginProvider from '../providers/auth/Debug.js'; // Adjust path as needed

const UserProfile = () => {

  const navigate = useNavigate();

  const [userData, setUserData] = useState(null);

  const [rooms, setRooms] = useState([]);

  const [showCreateModal, setShowCreateModal] = useState(false);

  const [newRoomName, setNewRoomName] = useState('');

  const [showDeleteModal, setShowDeleteModal] = useState(false);

  const [roomToDelete, setRoomToDelete] = useState(null);

  const [loading, setLoading] = useState(true);

  const [error, setError] = useState(null);

  useEffect(() => {

    const provider = new LoginProvider();

    provider

      .getUserProfile()

      .then((profileData) => {

        setUserData(profileData);

        setRooms(profileData.rooms || []);

        setLoading(false);

      })

      .catch((err) => {

        setError('Failed to load user profile');

        setLoading(false);

      });

  }, []);

  const handleJoinRoom = (roomId) => {

    navigate(`/canvas/${roomId}`);

  };

  const handleCreateRoom = () => {

    if (newRoomName.trim()) {

      const newRoom = {

        id: `room-uuid-${Date.now()}`,

        name: newRoomName,

        isOwner: true,

      };

      setRooms([...rooms, newRoom]);

      setNewRoomName('');

      setShowCreateModal(false);

    }

  };

  const handleDeleteRoom = (roomId) => {

    setRooms(rooms.filter((room) => room.id !== roomId));

    setShowDeleteModal(false);

    setRoomToDelete(null);

  };

  const handleShowDeleteModal = (room) => {

    setRoomToDelete(room);

    setShowDeleteModal(true);

  };

  const handleLogout = async () => {

    try {

      const provider = new LoginProvider();

      await provider.logout();

      setUserData(null);

      setRooms([]);

      navigate('/'); // Redirect to login page after logout

    } catch (err) {

      window.addNotification({

        title: 'Выход',

        message: 'Не удалось выйти из аккаунта.\n\nВы теперь здесь навсегда.',

        variant: 'warning',

        delay: 3000

      });

    }

  };

  if (loading) {

    return (

      <Container className="my-4 text-center">

        <Spinner animation="border" variant="primary" />

      </Container>

    );

  }

  if (error) {

    return (

      <Container className="my-4">

        <Alert variant="danger">{error}</Alert>

      </Container>

    );

  }

  return (

    <Container className="my-4">

      <Row className="justify-content-center">

        <Col md={6}>

          <Card>

            <Card.Body className="text-center">

              <img

                src={userData?.ppic}

                alt=""

                className="rounded-circle mb-3 bg-secondary"

                style={{ width: '128px', height: '128px' }}

                onError={(e) => (e.target.src = '')}

              />

              <Card.Title>{userData?.username}</Card.Title>

              <Button

                variant="primary"

                onClick={() => setShowCreateModal(true)}

                className="mb-3 me-2"

              >

                Create Room

              </Button>

              <Button variant="outline-danger" onClick={handleLogout} className="mb-3">

                Logout

              </Button>

              <ListGroup variant="flush">

                {rooms.map((room) => (

                  <ListGroup.Item key={room.id} className="d-flex justify-content-between align-items-center">

                    <span>{room.name}</span>

                    <div>

                      <Button

                        variant="outline-primary"

                        size="sm"

                        onClick={() => handleJoinRoom(room.id)}

                        className="me-2"

                      >

                        Join

                      </Button>

                      {room.isOwner && (

                        <Button

                          variant="outline-danger"

                          size="sm"

                          onClick={() => handleShowDeleteModal(room)}

                        >

                          Delete

                        </Button>

                      )}

                    </div>

                  </ListGroup.Item>

                ))}

              </ListGroup>

            </Card.Body>

          </Card>

        </Col>

      </Row>

      {/\* Create Room Modal \*/}

      <Modal show={showCreateModal} onHide={() => setShowCreateModal(false)} centered>

        <Modal.Header closeButton>

          <Modal.Title>Create New Room</Modal.Title>

        </Modal.Header>

        <Modal.Body>

          <Form>

            <Form.Group controlId="roomName">

              <Form.Label>Room Name</Form.Label>

              <Form.Control

                type="text"

                value={newRoomName}

                onChange={(e) => setNewRoomName(e.target.value)}

                placeholder="Enter room name"

              />

            </Form.Group>

          </Form>

        </Modal.Body>

        <Modal.Footer>

          <Button variant="secondary" onClick={() => setShowCreateModal(false)}>

            Cancel

          </Button>

          <Button variant="primary" onClick={handleCreateRoom} disabled={!newRoomName.trim()}>

            Create

          </Button>

        </Modal.Footer>

      </Modal>

      {/\* Delete Room Confirmation Modal \*/}

      <Modal show={showDeleteModal} onHide={() => setShowDeleteModal(false)} centered>

        <Modal.Header closeButton>

          <Modal.Title>Confirm Delete</Modal.Title>

        </Modal.Header>

        <Modal.Body>

          Are you sure you want to delete the room "{roomToDelete?.name}"?

        </Modal.Body>

        <Modal.Footer>

          <Button variant="secondary" onClick={() => setShowDeleteModal(false)}>

            Cancel

          </Button>

          <Button

            variant="danger"

            onClick={() => handleDeleteRoom(roomToDelete?.id)}

          >

            Delete

          </Button>

        </Modal.Footer>

      </Modal>

    </Container>

  );

};

export default UserProfile;

**4.2.2 Листинг Login**import { Button, Col, Form } from 'react-bootstrap';

import { useNavigate } from "react-router";

import LoginProvider from '../providers/auth/Debug';

function Login() {

  const navigate = useNavigate();

  const provider = new LoginProvider();

  const onLoginClick = async (e) => {

    e.preventDefault();

    const email = e.target.form[0].value;

    const password = e.target.form[1].value;

    if (await provider.login({ email, password })) {

      window.addNotification({

        title: 'Вход',

        message: 'Вы успешно вошли в аккаунт!',

        variant: 'info',

        delay: 10000

      });

      navigate('/you'); // Redirect to dashboard on success

    } else {

      window.addNotification({

        title: 'Вход',

        message: 'Неверный E-mail или пароль!',

        variant: 'warning',

        delay: 3000

      });

    }

  };

  return (

    <div className='container d-flex align-items-center justify-content-center min-vh-100 w-75'>

      <Form className='container p-5 bg-light rounded'>

        <Form.Group className='col p-0 m-0' controlId="formTitle">

          <p className='text-muted p-0 m-0'>Добро пожаловать в</p>

          <h2 className='bold p-0'>BrainStorm</h2>

        </Form.Group>

        <Form.Group className="mb-3" controlId="formEmail">

          <Form.Label column sm="2">E-mail</Form.Label>

          <Col sm="10">

            <Form.Control type='text' placeholder='Электронная почта' />

          </Col>

        </Form.Group>

        <Form.Group className="mb-3" controlId="formPassword">

          <Form.Label column sm="2">Пароль</Form.Label>

          <Col sm="10">

            <Form.Control type="password" placeholder="Пароль" />

          </Col>

        </Form.Group>

        <Button type='button' className='me-2' onClick={onLoginClick}>Войти</Button>

        <Button type='button' className='ms-2 bg-secondary'>Зарегистрироваться</Button>

      </Form>

    </div>

  );

}

export default Login;

**4.2.2 Листинг Debug (тестовая авторизация)**

class LoginProvider {

  constructor(config = {}) {

    this.token = localStorage.getItem('authToken') || null;

  }

  // Установка токена и сохранение в localStorage

  setToken(token) {

    this.token = token;

    if (token) {

      localStorage.setItem('authToken', token);

    } else {

      localStorage.removeItem('authToken');

    }

  }

  // Получение токена

  getToken() {

    return this.token;

  }

  // Проверка, авторизован ли пользователь

  isAuthenticated() {

    return !!this.token;

  }

  // Вход в систему

  async login(credentials) {

    try {

        console.log(credentials)

      if (credentials.email === "test" && credentials.password === "test"){

        this.setToken("test");

        return true;

      }

      return false;

    } catch (error) {

      throw new Error(`Login error: ${error.message}`);

    }

  }

  // Регистрация нового пользователя

  async register(userData) {

    try {

      this.setToken("test");

      return { success: true };

    } catch (error) {

      throw new Error(`Registration error: ${error.message}`);

    }

  }

  // Выход из системы

  async logout() {

    try {

      this.setToken(null);

      return { success: true };

    } catch (error) {

      throw new Error(`Logout error: ${error.message}`);

    }

  }

  // Обновление токена

  async refreshToken() {

    try {

      this.setToken("test");

      return { success: true };

    } catch (error) {

      this.setToken(null);

      throw new Error(`Token refresh error: ${error.message}`);

    }

  }

  // Получение профиля пользователя

  async getUserProfile() {

    try {

      return {

        username: "Test user | NO AUTH SERVER",

        ppic: "cdn.myapp.com/user/88696988",

        rooms: [

            {

                id: "room-uuid-for-sync",

                name: "Test room | NO SYNC",

                isOwner: true,

            },

            {

                id: "room-uuid-for-sync-noowner",

                name: "Test room (Not owner) | NO SYNC",

                isOwner: false,

            }

        ]

      }

    } catch (error) {

      throw new Error(`Profile fetch error: ${error.message}`);

    }

  }

}

export default LoginProvider;

# **4.2.2 Листинг WebSocketProvider**

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

import { iAmBusy } from '../../utility/utils';

export const WebSocketContext = createContext();

const WebSocketProvider = ({ children, setLines, setTexts, setImages, setSelectedId }) => {

  const wsRef = useRef(null);

  const [pendingSelect, setPendingSelect] = useState(null); // Track pending selection

  useEffect(() => {

    wsRef.current = new WebSocket('ws://localhost:6942/canvas'); // Replace with your WebSocket URL

    wsRef.current.addEventListener("error",  (e) => { window.addNotification({

        title: 'Ошибка синхронизации',

        message: "Произошла ошибка при попытке синхронизации данных!\n\nПожалуйста, обновите страницу и повторите попытку!",

        variant: 'warning',

        delay: 10000

    }); console.log(e)});

    wsRef.current.onopen = () => {

        wsRef.onError = (e) =>  window.addNotification({

          title: 'Подключено!',

          message: e,

          variant: 'info',

          delay: 5000

      });

      wsRef.current.send(JSON.stringify({ type: 'join' }));

    };

    wsRef.current.onmessage = (event) => {

      const message = JSON.parse(event.data);

      switch (message.type) {

        case 'init':

          setLines(message.data.lines || []);

          setTexts(message.data.texts || []);

          setImages(message.data.images || []);

          break;

        case 'select':

          if (message.canEdit) {

            setSelectedId(`${message.objectType}-${message.id}`);

            setPendingSelect(null);

          } else {

            iAmBusy(`Object ${message.objectType}-${message.id} is locked by another user`);

            setPendingSelect(null);

          }

          break;

        case 'update':

          if (!message.success) {

            iAmBusy(`Update failed: ${message.error}`);

            // Revert handled in individual handlers

          }

          break;

        case 'image':

          if (message.success) {

            setImages((prev) =>

              prev.map((img) =>

                img.id === message.id ? { ...img, image: null, url: message.url } : img

              )

            );

          } else {

            iAmBusy(`Image upload failed: ${message.error}`);

            setImages((prev) => prev.filter((img) => img.id !== message.id));

          }

          break;

        default:

          console.log('Unknown message:', message);

      }

    };

    wsRef.current.onclose = () => {

      console.log('WebSocket disconnected');

    };

    return () => {

      if (wsRef.current) {

        wsRef.current.close();

      }

    };

  }, [setLines, setTexts, setImages, setSelectedId]);

  const selectObject = (id, objectType, callback) => {

    setPendingSelect(`${objectType}-${id}`);

    wsRef.current.send(JSON.stringify({ type: 'select', id, objectType }));

    // Callback to proceed with selection if needed

    if (callback) callback();

  };

  const deselectObject = (id, objectType) => {

    wsRef.current.send(JSON.stringify({ type: 'deselect', id, objectType }));

    setSelectedId(null);

    setPendingSelect(null);

  };

  const syncUpdate = async (id, objectType, data, revertCallback) => {

    const response = await new Promise((resolve) => {

      wsRef.current.send(JSON.stringify({ type: 'update', id, objectType, data }));

      // Wait for server response (handled in onmessage)

      const handler = (event) => {

        const message = JSON.parse(event.data);

        if (message.type === 'update' && message.id === id && message.objectType === objectType) {

          resolve(message);

          wsRef.current.removeEventListener('message', handler);

        }

      };

      wsRef.current.addEventListener('message', handler);

    });

    if (!response.success) {

      revertCallback();

    }

    return response;

  };

  const uploadImage = async (id, base64) => {

    wsRef.current.send(JSON.stringify({ type: 'image', id, base64 }));

    // Response handled in onmessage

  };

  return (

    <WebSocketContext.Provider value={{ selectObject, deselectObject, syncUpdate, uploadImage }}>

      {children}

    </WebSocketContext.Provider>

  );

};

export default WebSocketProvider;

# **4.2.2 Листинг Toaster**

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

import { Toast, ToastContainer, Row, Col, Button } from 'react-bootstrap';

import 'bootstrap/dist/css/bootstrap.min.css';

let toastId = 0;

const timeSince = (timestamp) => {

  const now = Date.now();

  const secondsPast = Math.floor((now - timestamp) / 1000);

  if (secondsPast < 60) {

    return `${secondsPast}с назад`;

  }

  if (secondsPast < 3600) {

    const minutes = Math.floor(secondsPast / 60);

    return `${minutes}мин ${minutes === 1 ? '' : 'с'} назад`;

  }

  const hours = Math.floor(secondsPast / 3600);

  return `${hours}ч ${hours === 1 ? '' : 'с'} назад`;

};

export default function Toaster() {

   const [toasts, setToasts] = useState([]);

  const addToast = ({ title, message, variant = 'info', delay = 5000 }) => {

    setToasts((prev) => [

      ...prev,

      { id: toastId++, title, message, variant, delay, show: true, timestamp: Date.now() },

    ]);

    //toasts[toasts.length - 1].show = true; // Add intro animation

  };

  const closeToast = (id) => {

    setToasts((prev) =>

      prev.map((toast) =>

        toast.id === id ? { ...toast, show: false } : toast

      )

    );

    setTimeout(() => {

      setToasts((prev) => prev.filter((toast) => toast.id !== id));

    }, 400);

  };

  // Update toast time

  useEffect(() => {

    const interval = setInterval(() => {

      setToasts((prev) =>

        prev.map((toast) =>

          toast.show ? { ...toast, timestamp: toast.timestamp } : toast

        )

      );

    }, 1000); // Обновляем каждую секунду

    // Очистка интервала при размонтировании

    return () => clearInterval(interval);

  }, []);

  // Экспортируем функцию добавления уведомления через window для глобального доступа

  useEffect(() => {

    window.addNotification = addToast;

    return () => {

      delete window.addNotification;

    };

  }, []);

  return (

    <Row>

      <Col xs={12}>

        <ToastContainer position="bottom-end" className="p-3">

          {toasts.map((toast) => (

            <Toast

              key={toast.id}

              onClose={() => closeToast(toast.id)}

              show={toast.show}

              delay={toast.delay}

              autohide

              bg={toast.variant}

            >

              <Toast.Header>

                <img

                  src="holder.js/20x20?text=%20"

                  className="rounded me-2"

                  alt=""

                />

                <strong className="me-auto">{toast.title}</strong>

                <small>{timeSince(toast.timestamp)}</small>

              </Toast.Header>

              <Toast.Body>{toast.message}</Toast.Body>

            </Toast>

          ))}

        </ToastContainer>

      </Col>

    </Row>

  );

};

# **4.2.2 Листинг Utils**

export const handleMouseDown = (e, tool, isDrawing, lines, setLines, color, syncUpdate, scale = 1, stagePos = {x: 0, y: 0}) => {

  if (tool !== 'pen') return;

  const stage = e.target.getStage();

  if (!stage) return;

  const pos = stage.getPointerPosition();

  if (!pos) return;

  const x = (pos.x - stagePos.x) / scale;

  const y = (pos.y - stagePos.y) / scale;

  isDrawing.current = true;

  const newLine = {

    id: Date.now(),

    points: [x, y],

    color: color,

    zIndex: 0,

  };

  setLines([...lines, newLine]);

  syncUpdate(newLine.id, 'line', newLine, () => {

    setLines(prev => prev.filter(l => l.id !== newLine.id));

  });

};

export const handleMouseMove = (e, tool, isDrawing, lines, setLines, syncUpdate, scale = 1, stagePos = {x: 0, y: 0}) => {

  if (!isDrawing.current || tool !== 'pen') return;

  const stage = e.target.getStage();

  if (!stage) return;

  const pos = stage.getPointerPosition();

  if (!pos) return;

  const x = (pos.x - stagePos.x) / scale;

  const y = (pos.y - stagePos.y) / scale;

  const lastLine = lines[lines.length - 1];

  if (!lastLine) return;

  lastLine.points = lastLine.points.concat([x, y]);

  setLines([...lines.slice(0, -1), lastLine]);

  syncUpdate(lastLine.id, 'line', lastLine, () => {

    setLines(prev => prev.filter(l => l.id !== lastLine.id));

  });

};

export const handleMouseUp = (isDrawing) => {

  isDrawing.current = false;

};

export const handleSelect = (id, type, selectObject) => {

  selectObject(id, type);

};

export const iAmBusy = (text) => {

    window.addNotification({

        title: 'Занято!',

        message: text,

        variant: 'warning',

        delay: 3000

    });

}

# **ЗАКЛЮЧЕНИЕ**

В условиях возрастающей потребности в эффективных инструментах для удалённой командной работы разработка специализированного приложения, объединяющего графический редактор, многопользовательское взаимодействие в реальном времени и обмен файлами, представляется актуальной и востребованной задачей.

Предлагаемое решение на базе JavaScript с использованием библиотек React и Konva позволит создать интуитивно понятную и функциональную платформу для совместной работы. Интеграция возможностей визуального редактирования (аналог маркерной доски) с инструментами обмена файлами обеспечит командам удобное пространство для генерации идей, обсуждения проектов и оперативного внесения правок.

Таким образом, реализация данного приложения не только упростит процесс удалённого взаимодействия, но и повысит продуктивность команд за счёт объединения ключевых функций в единой среде. Дальнейшее развитие проекта может включать расширение функционала, интеграцию с популярными сервисами (Jira, Slack, Google Drive) и оптимизацию производительности для работы с большими распределёнными коллективами.

Приложение А

Блок-схема модуля Account