**Полная техническая документация проекта "Интерактивная доска"**

**Содержание**

1. [Обзор проекта](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#1-%D0%BE%D0%B1%D0%B7%D0%BE%D1%80-%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0)
2. [Архитектура системы](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#2-%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B)
3. [Структура файлов](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#3-%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2)
4. [Модуль аутентификации (auth.js)](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#4-%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C-%D0%B0%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8-authjs)
5. [Модуль расчётов (engine.js)](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#5-%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C-%D1%80%D0%B0%D1%81%D1%87%D1%91%D1%82%D0%BE%D0%B2-enginejs)
6. [Модуль функционала карточек (card-features.js)](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#6-%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D0%B0-%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D1%87%D0%B5%D0%BA-card-featuresjs)
7. [Основной модуль приложения (script.js)](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#7-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D0%BE%D0%B9-%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-scriptjs)
8. [Система стилей (style.css)](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#8-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0-%D1%81%D1%82%D0%B8%D0%BB%D0%B5%D0%B9-stylecss)
9. [HTML-структура (index.html)](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#9-html-%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0-indexhtml)
10. [Форматы данных](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#10-%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D1%8B-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)
11. [Алгоритмы и бизнес-логика](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#11-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B-%D0%B8-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0)
12. [Взаимодействие компонентов](https://claude.ai/chat/304e6852-dde9-41c0-8c6c-ed1b3730051f#12-%D0%B2%D0%B7%D0%B0%D0%B8%D0%BC%D0%BE%D0%B4%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D0%B5-%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BE%D0%B2)

**1. Обзор проекта**

**1.1 Назначение**

Веб-приложение для визуализации и управления иерархической структурой лицензий в бизнес-модели. Позволяет:

* Создавать карточки лицензий с параметрами
* Соединять карточки линиями для отображения иерархии
* Автоматически рассчитывать баллы и циклы
* Управлять заметками с календарём
* Экспортировать в различные форматы (JSON, HTML, SVG, PNG, PDF)

**1.2 Технологический стек**

* **Frontend:** Vanilla JavaScript (ES6+), HTML5, CSS3
* **Библиотеки:**
  + html2canvas (v1.4.1) — рендеринг HTML в Canvas
  + jsPDF (v2.5.1) — генерация PDF
* **API браузера:** Web Crypto API, Pointer Events, Canvas API, SVG

**1.3 Ключевые возможности**

1. Управление карточками: создание, редактирование, удаление, перемещение
2. Система связей: линии с настройкой цвета, толщины, анимации
3. Автоматические расчёты: баллы в ногах, этапы, циклы
4. Заметки: многодневные заметки с календарём и цветовой маркировкой
5. Экспорт: JSON, HTML, SVG, PNG (с DPI), PDF
6. История: Undo/Redo с лимитом 50 операций
7. Режимы работы: обычный, выделение, иерархическое перемещение

**2. Архитектура системы**

**2.1 Общая схема**

┌─────────────────────────────────────────────────────────┐

│ index.html │

│ ┌──────────────────────────────────────────────────┐ │

│ │ UI Панели (левая/правая) │ │

│ ├──────────────────────────────────────────────────┤ │

│ │ Canvas (рабочая область) │ │

│ │ ┌────────────────────────────────────────────┐ │ │

│ │ │ SVG Layer (линии) │ │ │

│ │ ├────────────────────────────────────────────┤ │ │

│ │ │ Card Elements (карточки лицензий) │ │ │

│ │ └────────────────────────────────────────────┘ │ │

│ └──────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────┘

▲ ▲ ▲

│ │ │

┌─────┴─────┐ ┌────┴────┐ ┌─────┴───────┐

│ auth.js │ │engine.js│ │card-features│

│(защита) │ │(расчёты)│ │ .js(значки)│

└───────────┘ └─────────┘ └─────────────┘

▲

│

┌─────┴──────┐

│ script.js │

│ (основная │

│ логика) │

└────────────┘

**2.2 Слои приложения**

1. **Слой представления (UI Layer)**
   * HTML-разметка панелей управления
   * Canvas для размещения карточек
   * SVG для отрисовки линий
2. **Слой логики (Business Logic Layer)**
   * engine.js: математические расчёты
   * script.js: управление состоянием, события
   * card-features.js: работа со значками
3. **Слой данных (Data Layer)**
   * Состояние в памяти (cards, lines)
   * История операций (undoStack, redoStack)
   * Сессионное хранилище (аутентификация)
4. **Слой безопасности (Security Layer)**
   * auth.js: SHA-256 хэширование паролей
   * Модальное окно аутентификации

**3. Структура файлов**

project/

├── index.html # Главная страница

├── style.css # Все стили приложения

├── auth.js # Модуль аутентификации

├── engine.js # Модуль расчётов

├── card-features.js # Модуль значков и контекстного меню

├── script.js # Основная логика приложения

├── package.json # Конфигурация npm

├── .gitignore # Исключения для Git

├── LICENSE # Лицензия MIT

├── README.md # Описание проекта

├── password-hasher.html # Утилита генерации хэшей

└── Описание.docx # Инструкция пользователя

**4. Модуль аутентификации (auth.js)**

**4.1 Назначение**

Защита доступа к приложению с помощью паролей, хэшированных по SHA-256.

**4.2 Класс PasswordProtector**

**4.2.1 Свойства класса**

{

validHashes: Array<string>, // Массив допустимых SHA-256 хэшей

sessionKey: string, // Ключ для sessionStorage ('fohowAuth')

modalElement: HTMLElement, // Ссылка на модальное окно

formElement: HTMLFormElement, // Форма ввода пароля

passwordInput: HTMLInputElement, // Поле ввода пароля

errorElement: HTMLElement, // Элемент для ошибок

submitButton: HTMLButtonElement, // Кнопка отправки

defaultSubmitText: string, // Текст кнопки по умолчанию

focusableElements: Array, // Фокусируемые элементы модалки

isProcessing: boolean // Флаг обработки запроса

}

**4.2.2 Конструктор**

constructor() {

// 1. Инициализация validHashes (массив SHA-256 хэшей)

// 2. Получение ссылок на DOM-элементы

// 3. Привязка обработчиков событий (bind)

// 4. Проверка наличия всех необходимых элементов

// 5. Установка слушателей:

// - submit на форму

// - input на поле пароля (очистка ошибок)

// - keydown на модалку (Tab-навигация)

// - focusin на document (возврат фокуса в модалку)

// 6. Вызов checkAuthentication()

}

**4.2.3 Методы**

**checkAuthentication()**

Проверяет sessionStorage на наличие ключа 'fohowAuth'. Если authenticated — вызывает hideModalInstantly(), иначе — openModal().

**openModal()**

1. Снимает атрибут hidden с модального окна
2. Добавляет класс 'auth-modal-open' к body
3. Очищает ошибки (clearError)
4. Сбрасывает форму (reset)
5. Через requestAnimationFrame:
   * Добавляет класс 'auth-modal--visible' для анимации
   * Обновляет фокусируемые элементы
   * Устанавливает фокус на поле ввода

**hideModalInstantly()**

Скрывает модалку без анимации:

1. Устанавливает атрибут hidden
2. Удаляет класс 'auth-modal--visible'
3. Удаляет класс 'auth-modal-open' у body

**closeModal()**

Закрывает с анимацией:

1. Удаляет класс 'auth-modal--visible'
2. Удаляет класс 'auth-modal-open' у body
3. Через 400ms устанавливает атрибут hidden

**updateFocusableElements()**

Собирает все фокусируемые элементы в модалке: a[href], button:not([disabled]), input:not([disabled]), select:not([disabled]), textarea:not([disabled]), [tabindex]:not([tabindex="-1"]). Фильтрует только видимые (offsetParent !== null).

**focusFirstElement()**

Устанавливает фокус на passwordInput. Если passwordInput отсутствует, фокусирует первый элемент из focusableElements. Вызывает select() для выделения текста.

**handleKeyDown(event)**

Обработка Tab-навигации в модалке:

1. Проверка event.key === 'Tab'
2. Обновление focusableElements при необходимости
3. Определение первого и последнего элементов
4. При Shift+Tab на первом → фокус на последний
5. При Tab на последнем → фокус на первый

**handleDocumentFocus(event)**

Ловушка фокуса (focus trap): если модалка видима и фокус вне её — возврат фокуса в модалку.

**handleSubmit(event)** *(async)*

1. event.preventDefault()
2. Проверка isProcessing (защита от повторной отправки)
3. Проверка поддержки crypto.subtle
4. Получение пароля из поля ввода (trim)
5. Валидация (пустой пароль → ошибка)
6. togglePendingState(true) — установка состояния обработки
7. Хэширование пароля через sha256()
8. Сравнение с validHashes
9. Если совпадает: сохранение в sessionStorage, clearError(), closeModal()
10. Если не совпадает → displayError()
11. finally: togglePendingState(false)

**togglePendingState(pending)**

Управление состоянием кнопки:

1. isProcessing = pending
2. submitButton.disabled = pending
3. submitButton.textContent = pending ? 'Проверяем…' : defaultSubmitText

**displayError(message)**

Отображение/скрытие ошибки: если message пустое — скрыть errorElement, иначе — установить textContent и убрать hidden.

**clearError()**

Вызывает displayError('') для очистки.

**sha256(message)** *(async)*

SHA-256 хэширование:

1. Создание TextEncoder
2. Кодирование строки в Uint8Array
3. crypto.subtle.digest('SHA-256', data)
4. Преобразование ArrayBuffer в массив байтов
5. Конвертация в hex-строку (toString(16).padStart(2, '0'))
6. Возврат строки хэша

**4.3 Инициализация**

document.addEventListener('DOMContentLoaded', function() {

new PasswordProtector();

});

**4.4 HTML-структура модалки**

<div id="auth-modal" class="auth-modal" hidden>

<div class="auth-modal\_\_overlay"></div>

<div class="auth-modal\_\_dialog" role="dialog" aria-modal="true">

<form class="auth-modal\_\_form" data-auth-form>

<h2 class="auth-modal\_\_title">Доступ к проекту</h2>

<p class="auth-modal\_\_description">...</p>

<div class="auth-modal\_\_field">

<label class="auth-modal\_\_label">Пароль</label>

<input class="auth-modal\_\_input" type="password" />

</div>

<p class="auth-modal\_\_error" data-auth-error hidden></p>

<ul class="auth-modal\_\_hints">...</ul>

<button type="submit" class="auth-modal\_\_submit">Войти</button>

</form>

</div>

</div>

**4.5 Генерация хэшей (password-hasher.html)**

Вспомогательная утилита для создания SHA-256 хэшей:

async function hashPassword() {

const password = document.getElementById('password').value;

const encoder = new TextEncoder();

const data = encoder.encode(password);

const hashBuffer = await crypto.subtle.digest('SHA-256', data);

const hashArray = Array.from(new Uint8Array(hashBuffer));

const hashHex = hashArray.map(b => b.toString(16).padStart(2, '0')).join('');

// Отображение результата

}

**5. Модуль расчётов (engine.js)**

javascript

**5.1 Назначение**

Выполняет математические расчёты баллов, этапов и циклов на основе структуры карточек и линий без манипуляций с DOM.

1. Модуль расчётов (engine.js)

5.1 Назначение

Выполняет математические расчёты баллов, этапов и циклов на основе структуры карточек и линий без манипуляций с DOM.

5.2 Структура модуля

(function (global) {

// Приватные функции

function normalizeSide(side) { ... }

function parsePv330(html) { ... }

function buildIndex(state) { ... }

function pickParent(byId, line) { ... }

function calcStagesAndCycles(totalBalls) { ... }

// Публичный API

const Engine = {

recalc(state) { ... }

};

global.Engine = Engine;

})(window);

5.3 Приватные функции

5.3.1 normalizeSide(side)

/\*\*

\* Нормализует сторону соединения к 'left' или 'right'

\* @param {string} side - 'left', 'right', 'bottom', 'top', или пустая строка

\* @returns {string} - 'right' или 'left'

\*/

function normalizeSide(side) {

const s = String(side || '').toLowerCase();

if (s === 'right') return 'right';

// left / bottom / top / unknown -> считаем левой

return 'left';

}

Логика:

* Явное 'right' → 'right'
* Всё остальное → 'left' (согласно ТЗ "снизу — влево")

5.3.2 parsePv330(html)

/\*\*

\* Парсит значение PV из HTML (формат X/330pv)

\* @param {string} html - HTML-строка карточки

\* @returns {{value: number, isFull: boolean}}

\*/

function parsePv330(html) {

// Регулярное выражение: (\d+)\s\*\/\s\*330\s\*pv

const m = /(\d+)\s\*\/\s\*330\s\*pv/i.exec(html || '');

const v = m ? parseInt(m[1], 10) : 0;

return { value: v, isFull: v >= 330 };

}

Логика:

* Ищет паттерн "число/330pv" (регистронезависимо)
* Возвращает числовое значение и флаг полноты (≥330)

5.3.3 buildIndex(state)

/\*\*

\* Создаёт Map для быстрого поиска карточек по id

\* @param {Object} state - Сериализованное состояние

\* @returns {Map<string, Object>} - Map(id -> cardData)

\*/

function buildIndex(state) {

const byId = new Map();

state.cards.forEach(c => byId.set(c.id, c));

return byId;

}

5.3.4 pickParent(byId, line)

/\*\*

\* Определяет родительскую и дочернюю карточку в линии

\* Родитель - карточка с меньшим Y (выше на экране)

\* @param {Map} byId - Индекс карточек

\* @param {Object} line - Объект линии {startId, endId, startSide, endSide}

\* @returns {{parentId, childId, parentSide} | null}

\*/

function pickParent(byId, line) {

const s = byId.get(line.startId);

const e = byId.get(line.endId);

if (!s || !e) return null;

let parent, child, sideOnParent;

if ((s.y ?? 0) <= (e.y ?? 0)) {

parent = s; child = e; sideOnParent = line.startSide;

} else {

parent = e; child = s; sideOnParent = line.endSide;

}

return {

parentId: parent.id,

childId: child.id,

parentSide: normalizeSide(sideOnParent)

};

}

Алгоритм:

* Получить обе карточки линии
* Сравнить Y-координаты
* Карточка с меньшим Y = родитель
* Вернуть структуру с ID и стороной родителя

5.3.5 calcStagesAndCycles(totalBalls)

/\*\*

\* Рассчитывает количество циклов и текущий этап

\* Этапы: 6, 12, 18, 36 баллов (сумма = 72 на цикл)

\* @param {number} totalBalls - Общее количество баллов

\* @returns {{cycles, stage, toNext}}

\*/

function calcStagesAndCycles(totalBalls) {

const thresholds = [6, 12, 18, 36];

const cycles = Math.floor(totalBalls / 72);

let within = totalBalls % 72;

let stage = 0, toNext = 6;

let accumulated = 0;

for (let i = 0; i < thresholds.length; i++) {

if (within >= accumulated + thresholds[i]) {

accumulated += thresholds[i];

stage = i + 1;

} else {

toNext = accumulated + thresholds[i] - within;

break;

}

}

// Если прошли все этапы

if (stage === 4) {

toNext = 72 - within;

}

return { cycles, stage, toNext };

}

Логика:

* 1 цикл = 72 балла
* Этап 1: 0–5 → 6
* Этап 2: 6–17 → 12
* Этап 3: 18–35 → 18
* Этап 4: 36–71 → 36
* После 72 → новый цикл

5.4 Публичный метод Engine.recalc(state)

/\*\*

\* Основной метод пересчёта всей структуры

\* @param {Object} state - Сериализованное состояние {cards, lines}

\* @returns {{result, meta}}

\*/

Engine.recalc(state) {

// 1. ПОСТРОЕНИЕ ИНДЕКСОВ

const byId = buildIndex(state);

// 2. ОПРЕДЕЛЕНИЕ ИЕРАРХИИ

const parentOf = {}; // child -> {parentId, side}

const children = {}; // parent -> {left: Set, right: Set}

state.cards.forEach(c => {

children[c.id] = { left: new Set(), right: new Set() };

});

// Обработка линий

state.lines.forEach(line => {

const pr = pickParent(byId, line);

if (!pr) return;

const { parentId, childId, parentSide } = pr;

children[parentId][parentSide].add(childId);

// Выбор самого верхнего родителя при конфликте

const existed = parentOf[childId];

if (!existed) {

parentOf[childId] = { parentId, side: parentSide };

} else {

const oldP = byId.get(existed.parentId);

const newP = byId.get(parentId);

if (oldP && newP && (newP.y ?? 0) < (oldP.y ?? 0)) {

parentOf[childId] = { parentId, side: parentSide };

}

}

});

// 3. ПАРСИНГ PV

const pv = {};

const isFull = {};

state.cards.forEach(c => {

const { value, isFull: full } = parsePv330(c.bodyHTML);

pv[c.id] = value;

isFull[c.id] = full;

});

// 4. РАСЧЁТ БАЛЛОВ В НОГАХ

const balls = {};

state.cards.forEach(c => {

balls[c.id] = { L: 0, R: 0, total: 0 };

});

// Поднимаем 1 балл от каждого полного (330/330) вверх

state.cards.forEach(c => {

if (!isFull[c.id]) return;

let cur = c.id;

let guard = 0;

while (parentOf[cur] && guard++ < 5000) {

const { parentId, side } = parentOf[cur];

if (side === 'right') balls[parentId].R += 1;

else balls[parentId].L += 1;

balls[parentId].total += 1;

cur = parentId;

}

});

// 5. РАСЧЁТ ЭТАПОВ И ЦИКЛОВ

const result = {};

state.cards.forEach(c => {

const b = balls[c.id];

const { cycles, stage, toNext } = calcStagesAndCycles(b.total);

result[c.id] = {

L: b.L,

R: b.R,

total: b.total,

cycles,

stage,

toNext

};

});

return {

result,

meta: { pv, isFull, parentOf, children }

};

}

Возвращаемая структура:

{

result: {

"card\_id": {

L: number, // Баллы в левой ноге

R: number, // Баллы в правой ноге

total: number, // Общее количество баллов

cycles: number, // Количество завершённых циклов

stage: number, // Текущий этап (0-4)

toNext: number // Баллов до следующего этапа/цикла

}

},

meta: {

pv: { // PV каждой карточки

"card\_id": number

},

isFull: { // Флаг полноты (330/330)

"card\_id": boolean

},

parentOf: { // Родитель каждого ребёнка

"child\_id": {

parentId: string,

side: "left" | "right"

}

},

children: { // Дети каждого родителя

"parent\_id": {

left: Set<string>,

right: Set<string>

}

}

}

}

1. Модуль функционала карточек (card-features.js)

6.1 Назначение

Управление значками (badges) и контекстным меню для карточек.

6.2 Глобальная функция applyCardBadges(cardData)

/\*\*

\* Применяет значки к карточке на основе cardData.badges

\* @param {Object} cardData - Объект карточки с полем badges

\*/

function applyCardBadges(cardData) {

if (!cardData || !cardData.element) return;

const cardElement = cardData.element;

const badges = cardData.badges || {};

// Получение элементов значков

const fendouBadge = cardElement.querySelector('.fendou-badge');

const slfBadge = cardElement.querySelector('.slf-badge');

const rankBadge = cardElement.querySelector('.rank-badge');

// Применение видимости

if (fendouBadge) fendouBadge.classList.toggle('visible', !!badges.fendou);

if (slfBadge) slfBadge.classList.toggle('visible', !!badges.slf);

if (rankBadge) {

if (badges.rank) {

rankBadge.src = `rank-${badges.rank}.png`;

rankBadge.classList.add('visible');

} else {

rankBadge.classList.remove('visible');

}

}

}

6.3 Глобальная функция initializeCardFeatures(getCards, saveState)

/\*\*

\* Инициализирует контекстное меню для управления значками

\* @param {Function} getCards - Функция получения массива карточек

\* @param {Function} saveState - Функция сохранения состояния

\*/

function initializeCardFeatures(getCards, saveState) {

// 1. СОЗДАНИЕ КОНТЕКСТНОГО МЕНЮ

const contextMenu = document.createElement('div');

contextMenu.className = 'card-context-menu';

document.body.appendChild(contextMenu);

let currentCardData = null;

// 2. ОПРЕДЕЛЕНИЕ ДОСТУПНЫХ РАНГОВ

const ranks = {

'emerald': 'Изумруд',

'sapphire': 'Сапфир',

'diamond': 'Алмаз',

'brilliant3': 'Бриллиант 3',

'brilliant5': 'Бриллиант 5',

'brilliant7': 'Бриллиант 7',

'ambassador': 'Амбасадор'

};

// 3. ФУНКЦИЯ ОТОБРАЖЕНИЯ МЕНЮ

function showContextMenu(e, cardData) {

e.preventDefault();

if (!cardData) return;

currentCardData = cardData;

const badges = cardData.badges || {};

// Генерация HTML для рангов

let rankHtml = '';

for (const key in ranks) {

rankHtml += `<div class="context-menu-item ${badges.rank === key ? 'active' : ''}"

data-type="rank" data-value="${key}">${ranks[key]}</div>`;

}

// Полная структура меню

contextMenu.innerHTML = `

<div class="context-menu-section">

<div class="context-menu-item ${badges.fendou ? 'active' : ''}"

data-type="badge" data-value="fendou">FENDOU</div>

</div>

<div class="context-menu-section">

<div class="context-menu-item ${badges.slf ? 'active' : ''}"

data-type="badge" data-value="slf">SLF</div>

</div>

<div class="context-menu-section">

${rankHtml}

</div>

`;

// Позиционирование меню

contextMenu.style.display = 'block';

const menuWidth = contextMenu.offsetWidth;

const menuHeight = contextMenu.offsetHeight;

const winWidth = window.innerWidth;

const winHeight = window.innerHeight;

let x = e.clientX;

let y = e.clientY;

// Защита от выхода за пределы экрана

if (x + menuWidth > winWidth) {

x = winWidth - menuWidth - 5;

}

if (y + menuHeight > winHeight) {

y = winHeight - menuHeight - 5;

}

contextMenu.style.left = `${x}px`;

contextMenu.style.top = `${y}px`;

}

// 4. ФУНКЦИЯ СКРЫТИЯ МЕНЮ

function hideContextMenu() {

contextMenu.style.display = 'none';

currentCardData = null;

}

// 5. ОБРАБОТЧИК КОНТЕКСТНОГО МЕНЮ НА ЗАГОЛОВКЕ КАРТОЧКИ

document.addEventListener('contextmenu', (e) => {

const header = e.target.closest('.card-header');

if (!header) return;

const cardElement = header.closest('.card');

const cardData = getCards().find(c => c.id === cardElement.id);

if (cardData) {

showContextMenu(e, cardData);

}

});

// 6. ЗАКРЫТИЕ МЕНЮ ПРИ КЛИКЕ ВНЕ НЕГО

document.addEventListener('mousedown', (e) => {

if (!e.target.closest('.card-context-menu')) {

hideContextMenu();

}

});

// 7. ОБРАБОТКА ВЫБОРА ПУНКТА МЕНЮ

contextMenu.addEventListener('click', (e) => {

const item = e.target.closest('.context-menu-item');

if (!item || !currentCardData) return;

// Инициализация badges при необходимости

if (!currentCardData.badges) {

currentCardData.badges = {};

}

const { type, value } = item.dataset;

if (type === 'badge') {

// Переключение badge (toggle)

currentCardData.badges[value] = !currentCardData.badges[value];

} else if (type === 'rank') {

// Переключение ранга (toggle или установка)

currentCardData.badges.rank = currentCardData.badges.rank === value ? null : value;

}

// Применение изменений

applyCardBadges(currentCardData);

saveState();

hideContextMenu();

});

}

6.4 Структура данных badges

{

fendou: boolean, // Значок FENDOU

slf: boolean, // Значок SLF

rank: string | null // Ранг ('emerald', 'sapphire', и т.д.) или null

}

6.5 HTML-структура значков в карточке

<div class="card-header">

<div class="slf-badge">SLF</div>

<span class="card-title">...</span>

<div class="fendou-badge">FENDOU</div>

<img class="rank-badge" src="" alt="Rank">

</div>

6.6 CSS-классы для значков

.slf-badge, .fendou-badge, .rank-badge {

position: absolute;

display: none;

user-select: none;

pointer-events: none;

}

.slf-badge.visible,

.fendou-badge.visible,

.rank-badge.visible {

display: block;

}

.slf-badge {

top: 15px;

left: 15px;

color: #ffc700;

font-weight: 900;

font-size: 36px;

text-shadow: 1px 1px 2px rgba(0,0,0,0.5);

}

.fendou-badge {

top: -25px;

left: 50%;

transform: translateX(-50%);

color: red;

font-weight: 900;

font-size: 56px;

text-shadow: 2px 2px 4px rgba(0,0,0,0.3);

}

.rank-badge {

top: -15px;

right: 15px;

width: 80px;

height: auto;

transform: rotate(15deg);

}

1. Основной модуль приложения (script.js)

7.1 Структура модуля

document.addEventListener('DOMContentLoaded', () => {

// 1. Получение ссылок на DOM-элементы

// 2. Определение констант

// 3. Инициализация состояния

// 4. Настройка обработчиков событий

// 5. Вспомогательные функции

// 6. Основная логика

});

7.2 Константы приложения

const GRID\_SIZE = 70; // Размер сетки для привязки

const MARKER\_OFFSET = 12; // Отступ маркера на линии

const HISTORY\_LIMIT = 50; // Лимит истории Undo/Redo

const SNAP\_TOLERANCE = 5; // Допуск привязки (px)

const ACTIVE\_PV\_BASE = 330; // Базовое значение PV

const DEFAULT\_LINE\_COLOR = '#0f62fe'; // Цвет линии по умолчанию

const DEFAULT\_ANIMATION\_DURATION = 2000; // Длительность анимации (мс)

const MIN\_ANIMATION\_DURATION = 2000; // Минимальная длительность

const MAX\_ANIMATION\_DURATION = 999000; // Максимальная длительность

const MAX\_ANIMATION\_LOOP\_DURATION = 3000;// Лимит одного цикла анимации

const MAX\_CANVAS\_DIMENSION = 16384; // Макс. размер canvas

const MAX\_EXPORT\_DIMENSION = 16384; // Макс. размер экспорта

7.3 Глобальное состояние приложения

7.3.1 canvasState — состояние canvas

let canvasState = {

x: 0, // Смещение canvas по X

y: 0, // Смещение canvas по Y

scale: 1, // Масштаб canvas

isPanning: false, // Флаг панорамирования

lastPointerX: 0, // Последняя X-координата указателя

lastPointerY: 0, // Последняя Y-координата указателя

panPointerId: null, // ID указателя для пана

panCaptureTarget: null // Элемент захвата для пана

};

7.3.2 activeState — активное состояние UI

let activeState = {

currentThickness: 5, // Текущая толщина линии

currentColor: DEFAULT\_LINE\_COLOR, // Текущий цвет линии

selectedLine: null, // Выбранная линия

selectedCards: new Set(), // Набор выбранных карточек

lastFocusedCard: null, // Последняя сфокусированная карточка

isSelecting: false, // Флаг выделения рамкой

isSelectionMode: false, // Режим выделения

isHierarchicalDragMode: false, // Режим иерархического перемещения

isGlobalLineMode: false, // Режим глобального изменения линий

guidesEnabled: true, // Направляющие включены

lineStart: null, // Начало рисования линии

previewLine: null, // Превью линии (SVG элемент)

linePointerId: null, // ID указателя для рисования

lineCaptureTarget: null, // Элемент захвата для линии

selectionPointerId: null, // ID указателя для выделения

selectionCaptureTarget: null // Элемент захвата для выделения

};

7.3.3 Массивы данных

let cards = []; // Массив карточек

let lines = []; // Массив линий

let notesDropdownApi = null; // API выпадающего списка заметок

7.3.4 История и кеш

let undoStack = []; // Стек отмены

let redoStack = []; // Стек повтора

let clipboard = null; // Буфер обмена

let lastSavedSnapshot = null; // Последний сохранённый снимок

let lastEngineMeta = null; // Последние метаданные расчётов

const imageDataUriCache = new Map(); // Кеш Data URI изображений

const activePointers = new Map(); // Активные указатели (для мультитача)

let pinchState = null; // Состояние жеста pinch

7.3.5 Состояние анимации и подсветки

const lastCalculatedBalances = new Map(); // Последние рассчитанные балансы

let highlightedBalanceParts = new Map(); // Подсвеченные части баланса

const lastActivePvValues = new Map(); // Последние значения Актив-PV

const partHighlightTimers = new WeakMap(); // Таймеры подсветки частей

const lineHighlightTimers = new WeakMap(); // Таймеры подсветки линий

const animationSettings = {

durationMs: DEFAULT\_ANIMATION\_DURATION // Длительность анимации

};

7.4 Инициализация модулей

// Инициализация card-features.js

if (window.initializeCardFeatures) {

initializeCardFeatures(() => cards, saveState);

}

7.5 Основные функции по категориям

7.5.1 БЛОК: Управление Canvas

updateCanvasTransform()

/\*\*

\* Обновляет трансформацию canvas

\*/

function updateCanvasTransform() {

canvas.style.transform =

`translate(${canvasState.x}px, ${canvasState.y}px) scale(${canvasState.scale})`;

}

getCanvasCoordinates(clientX, clientY)

/\*\*

\* Преобразует экранные координаты в координаты canvas

\* @param {number} clientX - X в viewport

\* @param {number} clientY - Y в viewport

\* @returns {{x: number, y: number}} - Координаты в canvas

\*/

function getCanvasCoordinates(clientX, clientY) {

return {

x: (clientX - canvasState.x) / canvasState.scale,

y: (clientY - canvasState.y) / canvasState.scale

};

}

7.5.2 БЛОК: Управление карточками

createCard(opts = {})

/\*\*

\* Создаёт новую карточку

\* @param {Object} opts - Параметры карточки

\* @param {number} opts.x - X-координата

\* @param {number} opts.y - Y-координата

\* @param {string} opts.title - Заголовок

\* @param {string} opts.bodyHTML - HTML тела

\* @param {boolean} opts.isDarkMode - Тёмная тема

\* @param {boolean} opts.isLarge - Большая карточка (494px)

\* @param {string} opts.width - Ширина карточки

\* @param {string} opts.headerBg - Цвет заголовка

\* @param {string} opts.bodyClass - CSS класс тела

\* @param {number} opts.colorIndex - Индекс цвета

\* @param {Object} opts.note - Заметка

\* @param {Object} opts.badges - Значки

\* @param {boolean} opts.isTemplate - Флаг шаблона

\* @returns {Object} - Объект cardData

\*/

function createCard(opts = {}) {

// 1. СОЗДАНИЕ ID

const cardId = `card\_${Date.now()}\_${Math.floor(Math.random()\*1000)}`;

// 2. СОЗДАНИЕ DOM-ЭЛЕМЕНТА

const card = document.createElement('div');

card.className = 'card';

card.id = cardId;

if (opts.isDarkMode) card.classList.add('dark-mode');

// 3. УСТАНОВКА ШИРИНЫ

if (opts.isLarge) {

card.style.width = '494px';

} else if (opts.width) {

card.style.width = opts.width;

}

const CARD\_WIDTH = card.offsetWidth || (opts.isLarge ? 494 : 380);

// 4. РАСЧЁТ НАЧАЛЬНОЙ ПОЗИЦИИ

const CARD\_HEIGHT = 280, PADDING = 50;

let initialX, initialY;

if (opts.x != null) {

initialX = opts.x;

initialY = opts.y;

} else {

// Позиционирование в видимой области

const viewL = -canvasState.x / canvasState.scale;

const viewT = -canvasState.y / canvasState.scale;

const viewR = (window.innerWidth - canvasState.x) / canvasState.scale;

const viewB = (window.innerHeight - canvasState.y) / canvasState.scale;

initialX = Math.max(viewL + PADDING, viewR - CARD\_WIDTH - PADDING);

initialY = Math.max(viewT + PADDING, viewB - CARD\_HEIGHT - PADDING);

}

// 5. ПРИВЯЗКА К СЕТКЕ (если не шаблон)

if (opts.isTemplate) {

card.style.left = `${initialX}px`;

card.style.top = `${initialY}px`;

} else {

card.style.left = `${Math.round(initialX / GRID\_SIZE) \* GRID\_SIZE}px`;

card.style.top = `${Math.round(initialY / GRID\_SIZE) \* GRID\_SIZE}px`;

}

// 6. УСТАНОВКА СОДЕРЖИМОГО

const titleText = opts.title ?? 'RUY1234567890';

const bodyHTML = opts.bodyHTML ?? `

<div class="card-row">

<svg class="coin-icon" viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg">

<circle cx="50" cy="50" r="45" fill="#ffd700" stroke="#DAA520" stroke-width="5"/>

</svg>

<span class="value" contenteditable="true">30/330pv</span>

</div>

<div class="card-row"><span class="label">Баланс:</span><span class="value" contenteditable="true">0 / 0</span></div>

<div class="card-row"><span class="label">Актив-заказы PV:</span><span class="value" contenteditable="true">0 / 0</span></div>

<div class="card-row"><span class="label">Цикл:</span><span class="value" contenteditable="true">0</span></div>

`;

card.innerHTML = `

<div class="card-header" style="${opts.headerBg ? `background:${opts.headerBg}` : ''}">

<div class="slf-badge">SLF</div>

<span class="card-title" contenteditable="false">${titleText}</span>

<div class="fendou-badge">FENDOU</div>

<img class="rank-badge" src="" alt="Rank">

<button class="card-close-btn" type="button" title="Удалить карточку">×</button>

</div>

<div class="card-body ${opts.bodyClass || ''}">${bodyHTML}</div>

<div class="connection-point top" data-side="top"></div>

<div class="connection-point right" data-side="right"></div>

<div class="connection-point bottom" data-side="bottom"></div>

<div class="connection-point left" data-side="left"></div>

<div class="card-controls">

<button class="card-control-btn note-btn" title="Заметка">📝</button>

</div>

`;

// 7. ДОБАВЛЕНИЕ В DOM

canvas.appendChild(card);

ensureActiveControls(card);

// 8. СОЗДАНИЕ ОБЪЕКТА ДАННЫХ

const cardData = {

id: cardId,

element: card,

locked: false,

note: opts.note || null,

badges: opts.badges || { fendou: false, slf: false, rank: null }

};

cards.push(cardData);

// 9. ПРИМЕНЕНИЕ ЗНАЧКОВ

if (window.applyCardBadges) {

applyCardBadges(cardData);

}

// 10. УСТАНОВКА ОБРАБОТЧИКОВ

// 10.1. Выделение с Ctrl

card.addEventListener('pointerdown', (e) => {

if (e.ctrlKey) {

e.stopPropagation();

toggleCardSelection(cardData);

if (activeState.selectedCards.size > 0) {

let last = null;

activeState.selectedCards.forEach(cd => { last = cd; });

if (last) setLastFocusedCard(last);

} else {

setLastFocusedCard(cardData);

}

} else {

setLastFocusedCard(cardData);

}

});

// 10.2. Редактирование заголовка по двойному клику

const titleEl = card.querySelector('.card-title');

let isTitleEditing = false;

if (titleEl) {

titleEl.addEventListener('dblclick', (e) => {

e.stopPropagation();

if (isTitleEditing) return;

isTitleEditing = true;

titleEl.setAttribute('contenteditable', 'true');

titleEl.focus();

const selection = window.getSelection();

if (selection) {

const range = document.createRange();

range.selectNodeContents(titleEl);

selection.removeAllRanges();

selection.addRange(range);

}

});

const finishTitleEditing = () => {

if (!isTitleEditing) return;

isTitleEditing = false;

titleEl.setAttribute('contenteditable', 'false');

saveState();

};

titleEl.addEventListener('blur', finishTitleEditing);

titleEl.addEventListener('keydown', (e) => {

if (e.key === 'Enter' || e.key === 'Escape') {

e.preventDefault();

titleEl.blur();

}

});

}

// 10.3. Перемещение карточки

makeDraggable(card, cardData);

// 10.4. Установка цвета заголовка

const header = card.querySelector('.card-header');

const savedColorIndex = Number.parseInt(opts.colorIndex ?? '0', 10);

if (Number.isFinite(savedColorIndex) && savedColorIndex > -1) {

setCardHeaderColor(cardData, cardColors[savedColorIndex % cardColors.length], String(savedColorIndex));

} else if (opts.headerBg) {

setCardHeaderColor(cardData, opts.headerBg, '-1');

} else if (header) {

setCardHeaderColor(cardData, cardColors[0], '0');

}

setLastFocusedCard(cardData);

// 10.5. Переключение цвета монетки

const coin = card.querySelector('.coin-icon circle');

if (coin) {

coin.addEventListener('click', () => {

coin.setAttribute('fill', coin.getAttribute('fill') === '#ffd700' ? '#3d85c6' : '#ffd700');

saveState();

});

}

// 10.6. Кнопка закрытия

const closeBtn = card.querySelector('.card-close-btn');

if (closeBtn) {

closeBtn.addEventListener('click', (e) => {

e.stopPropagation();

deleteCard(cardData);

saveState();

});

}

// 10.7. Кнопка заметки

const noteBtn = card.querySelector('.note-btn');

noteBtn.addEventListener('click', (e) => {

e.stopPropagation();

toggleNote(cardData);

updateNotesButtonState();

});

if (cardData.note && cardData.note.visible) createNoteWindow(cardData);

updateNoteButtonAppearance(cardData);

// 10.8. Ручное редактирование баланса

setupBalanceManualEditing(card);

card.querySelectorAll('[contenteditable="true"]').forEach(el =>

el.addEventListener('blur', () => {

handleBalanceManualBlur(el);

saveState();

})

);

// 10.9. Точки соединения

card.querySelectorAll('.connection-point').forEach(point => {

point.addEventListener('pointerdown', (e) => {

e.stopPropagation();

if (pinchState && e.pointerType === 'touch') return;

if (!activeState.isDrawingLine) {

activeState.linePointerId = e.pointerId;

if (point.setPointerCapture) {

try {

point.setPointerCapture(e.pointerId);

activeState.lineCaptureTarget = point;

} catch (\_) {

activeState.lineCaptureTarget = null;

}

}

startDrawingLine(cardData, point.dataset.side);

} else if (!activeState.linePointerId || activeState.linePointerId === e.pointerId) {

endDrawingLine(cardData, point.dataset.side);

saveState();

if (activeState.lineCaptureTarget && activeState.lineCaptureTarget.releasePointerCapture) {

try { activeState.lineCaptureTarget.releasePointerCapture(e.pointerId); } catch (\_) {}

}

activeState.linePointerId = null;

activeState.lineCaptureTarget = null;

}

});

point.addEventListener('pointerup', (e) => {

if (activeState.linePointerId === e.pointerId && activeState.lineCaptureTarget === point) {

if (point.releasePointerCapture) {

try { point.releasePointerCapture(e.pointerId); } catch (\_) {}

}

activeState.lineCaptureTarget = null;

}

});

point.addEventListener('pointercancel', (e) => {

if (activeState.linePointerId === e.pointerId) {

if (activeState.lineCaptureTarget && activeState.lineCaptureTarget.releasePointerCapture) {

try { activeState.lineCaptureTarget.releasePointerCapture(e.pointerId); } catch (\_) {}

}

cancelDrawing();

activeState.linePointerId = null;

activeState.lineCaptureTarget = null;

}

});

});

updateNotesButtonState();

return cardData;

}

deleteCard(cardData)

/\*\*

\* Удаляет карточку и связанные линии

\* @param {Object} cardData - Объект карточки

\*/

function deleteCard(cardData) {

// 1. Удаление связанных линий

lines = lines.filter(line => {

if (line.startCard.id === cardData.id || line.endCard.id === cardData.id) {

clearLineHighlightState(line.element);

line.element.remove();

return false;

}

return true;

});

// 2. Удаление окна заметки

if (cardData.note && cardData.note.window) cardData.note.window.remove();

// 3. Удаление DOM-элемента

cardData.element.remove();

// 4. Удаление из массива cards

cards = cards.filter(c => c.id !== cardData.id);

// 5. Удаление из выделения

activeState.selectedCards.delete(cardData);

// 6. Обновление lastFocusedCard

if (activeState.lastFocusedCard && activeState.lastFocusedCard.id === cardData.id) {

setLastFocusedCard(null);

if (activeState.selectedCards.size > 0) {

const next = activeState.selectedCards.values().next().value;

if (next) setLastFocusedCard(next);

}

}

// 7. Обновление состояния кнопки заметок

updateNotesButtonState();

}

makeDraggable(element, cardData)

/\*\*

\* Делает карточку перемещаемой

\* @param {HTMLElement} element - DOM-элемент карточки

\* @param {Object} cardData - Объект данных карточки

\*/

function makeDraggable(element, cardData) {

const interactiveDragBlockSelector = '.card-control-btn, .note-btn, .active-btn, .card-title, [contenteditable="true"], button, input, textarea, select, label, a[href]';

element.addEventListener('pointerdown', (e) => {

// 1. ПРОВЕРКА УСЛОВИЙ

if (e.button !== 0 || e.ctrlKey || activeState.isSelectionMode) return;

if (pinchState && e.pointerType === 'touch') return;

if (e.target.closest(interactiveDragBlockSelector)) return;

let dragSet = new Set();

// 2. ОПРЕДЕЛЕНИЕ РЕЖИМА ПЕРЕМЕЩЕНИЯ

if (activeState.isHierarchicalDragMode) {

let dragMode = null;

const target = e.target;

if (target.closest('.card-header')) {

dragMode = 'all';

} else if (target.closest('.card-body')) {

const bodyRect = target.closest('.card-body').getBoundingClientRect();

dragMode = (e.clientX - bodyRect.left < bodyRect.width / 2) ? 'left' : 'right';

}

if (!dragMode) return;

e.stopPropagation();

dragSet = getBranchDescendants(cardData, dragMode);

dragSet.add(cardData);

} else {

e.stopPropagation();

if (activeState.selectedCards.has(cardData)) {

dragSet = new Set(activeState.selectedCards);

} else {

clearSelection();

dragSet.add(cardData);

}

}

setSelectionSet(dragSet);

// 3. ПОДГОТОВКА МАССИВА ПЕРЕМЕЩАЕМЫХ КАРТОЧЕК

const draggedCards = [];

activeState.selectedCards.forEach(selectedCard => {

if (selectedCard.locked) return;

draggedCards.push({

card: selectedCard,

element: selectedCard.element,

startX: parseFloat(selectedCard.element.style.left),

startY: parseFloat(selectedCard.element.style.top),

noteStartX: (selectedCard.note && selectedCard.note.window) ? selectedCard.note.window.offsetLeft : 0,

noteStartY: (selectedCard.note && selectedCard.note.window) ? selectedCard.note.window.offsetTop : 0,

});

});

if (draggedCards.length === 0) {

clearSelection();

return;

}

// 4. ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕЩЕНИЯ

const pointerId = e.pointerId;

const startPointerX = e.clientX;

const startPointerY = e.clientY;

const staticCards = cards.filter(c => !activeState.selectedCards.has(c));

if (element.setPointerCapture) {

try { element.setPointerCapture(pointerId); } catch (\_) {}

}

// 5. ОБРАБОТЧИК ПЕРЕМЕЩЕНИЯ

const onPointerMove = (e2) => {

if (e2.pointerId !== pointerId) return;

let dx\_canvas = (e2.clientX - startPointerX) / canvasState.scale;

let dy\_canvas = (e2.clientY - startPointerY) / canvasState.scale;

const dx\_viewport = e2.clientX - startPointerX;

const dy\_viewport = e2.clientY - startPointerY;

// 6. ПРИВЯЗКА К НАПРАВЛЯЮЩИМ

if (activeState.guidesEnabled) {

let snapX = null, snapY = null;

// Расчёт границ перемещаемой группы

const draggedBounds = {

left: Math.min(...draggedCards.map(d => d.startX + dx\_canvas)),

top: Math.min(...draggedCards.map(d => d.startY + dy\_canvas)),

right: Math.max(...draggedCards.map(d => d.startX + d.element.offsetWidth + dx\_canvas)),

bottom: Math.max(...draggedCards.map(d => d.startY + d.element.offsetHeight + dy\_canvas))

};

draggedBounds.centerX = draggedBounds.left + (draggedBounds.right - draggedBounds.left) / 2;

draggedBounds.centerY = draggedBounds.top + (draggedBounds.bottom - draggedBounds.top) / 2;

// Проверка привязки к статическим карточкам

for (const staticCard of staticCards) {

const sElem = staticCard.element;

const s = {

left: parseFloat(sElem.style.left),

top: parseFloat(sElem.style.top),

width: sElem.offsetWidth,

height: sElem.offsetHeight

};

s.right = s.left + s.width;

s.bottom = s.top + s.height;

s.centerX = s.left + s.width / 2;

s.centerY = s.top + s.height / 2;

// Привязка по Y

if (Math.abs(draggedBounds.top - s.top) < SNAP\_TOLERANCE) {

snapY = s.top;

dy\_canvas = s.top - Math.min(...draggedCards.map(d => d.startY));

} else if (Math.abs(draggedBounds.bottom - s.bottom) < SNAP\_TOLERANCE) {

snapY = s.bottom;

dy\_canvas = s.bottom - Math.max(...draggedCards.map(d => d.startY + d.element.offsetHeight));

} else if (Math.abs(draggedBounds.centerY - s.centerY) < SNAP\_TOLERANCE) {

snapY = s.centerY;

dy\_canvas = s.centerY - (Math.min(...draggedCards.map(d => d.startY)) + (draggedBounds.bottom - draggedBounds.top) / 2);

}

// Привязка по X

if (Math.abs(draggedBounds.left - s.left) < SNAP\_TOLERANCE) {

snapX = s.left;

dx\_canvas = s.left - Math.min(...draggedCards.map(d => d.startX));

} else if (Math.abs(draggedBounds.right - s.right) < SNAP\_TOLERANCE) {

snapX = s.right;

dx\_canvas = s.right - Math.max(...draggedCards.map(d => d.startX + d.element.offsetWidth));

} else if (Math.abs(draggedBounds.centerX - s.centerX) < SNAP\_TOLERANCE) {

snapX = s.centerX;

dx\_canvas = s.centerX - (Math.min(...draggedCards.map(d => d.startX)) + (draggedBounds.right - draggedBounds.left) / 2);

}

}

// Отображение направляющих

if (snapX !== null) {

vGuide.style.left = `${(snapX \* canvasState.scale) + canvasState.x}px`;

vGuide.style.display = 'block';

} else {

vGuide.style.display = 'none';

}

if (snapY !== null) {

hGuide.style.top = `${(snapY \* canvasState.scale) + canvasState.y}px`;

hGuide.style.display = 'block';

} else {

hGuide.style.display = 'none';

}

}

// 7. ПРИМЕНЕНИЕ ПЕРЕМЕЩЕНИЯ

draggedCards.forEach(dragged => {

let newX = dragged.startX + dx\_canvas;

let newY = dragged.startY + dy\_canvas;

// Привязка к сетке (если не используются направляющие)

const snappedX = (activeState.guidesEnabled && vGuide.style.display === 'block')

? newX

: Math.round(newX / GRID\_SIZE) \* GRID\_SIZE;

const snappedY = (activeState.guidesEnabled && hGuide.style.display === 'block')

? newY

: Math.round(newY / GRID\_SIZE) \* GRID\_SIZE;

dragged.element.style.left = `${snappedX}px`;

dragged.element.style.top = `${snappedY}px`;

// Обновление линий

updateLinesForCard(dragged.element.id);

// Перемещение окна заметки

if (dragged.card.note && dragged.card.note.window) {

dragged.card.note.window.style.left = `${dragged.noteStartX + dx\_viewport}px`;

dragged.card.note.window.style.top = `${dragged.noteStartY + dy\_viewport}px`;

}

});

};

});

}

// 8. ЗАВЕРШЕНИЕ ПЕРЕМЕЩЕНИЯ

const finishDrag = (e2) => {

if (e2.pointerId !== pointerId) return;

if (element.releasePointerCapture) {

try { element.releasePointerCapture(pointerId); } catch (\_) {}

}

element.removeEventListener('pointermove', onPointerMove);

element.removeEventListener('pointerup', finishDrag);

element.removeEventListener('pointercancel', cancelDrag);

vGuide.style.display = 'none';

hGuide.style.display = 'none';

// Сохранение новых позиций заметок

draggedCards.forEach(dragged => {

if (dragged.card.note && dragged.card.note.window) {

dragged.card.note.x = parseFloat(dragged.card.note.window.style.left);

dragged.card.note.y = parseFloat(dragged.card.note.window.style.top);

}

});

saveState();

};

// 9. ОТМЕНА ПЕРЕМЕЩЕНИЯ

const cancelDrag = (e2) => {

if (e2.pointerId !== pointerId) return;

if (element.releasePointerCapture) {

try { element.releasePointerCapture(pointerId); } catch (\_) {}

}

element.removeEventListener('pointermove', onPointerMove);

element.removeEventListener('pointerup', finishDrag);

element.removeEventListener('pointercancel', cancelDrag);

vGuide.style.display = 'none';

hGuide.style.display = 'none';

};

// 10. УСТАНОВКА СЛУШАТЕЛЕЙ

element.addEventListener('pointermove', onPointerMove);

element.addEventListener('pointerup', finishDrag);

element.addEventListener('pointercancel', cancelDrag);

});

}

getBranchDescendants(startCard, branchFilter)

/\*\*

\* Получает всех потомков в указанной ветке

\* @param {Object} startCard - Начальная карточка

\* @param {string} branchFilter - Фильтр ветки ('left', 'right', 'all')

\* @returns {Set} - Набор карточек-потомков

\*/

function getBranchDescendants(startCard, branchFilter) {

const descendants = new Set();

const queue = [startCard];

const visited = new Set([startCard.id]);

let head = 0;

let isInitialCard = true;

while(head < queue.length) {

const currentCard = queue[head++];

const childLines = lines.filter(line => line.startCard.id === currentCard.id);

for (const line of childLines) {

if (isInitialCard) {

const isBranchMatch =

(branchFilter === 'all' && ['left', 'right', 'bottom'].includes(line.startSide)) ||

(line.startSide === branchFilter);

if (!isBranchMatch) continue;

}

const childCard = line.endCard;

if (!visited.has(childCard.id)) {

visited.add(childCard.id);

descendants.add(childCard);

queue.push(childCard);

}

}

isInitialCard = false;

}

return descendants;

}

7.5.3 БЛОК: Управление линиями

startDrawingLine(card, side)

/\*\*

\* Начинает рисование линии

\* @param {Object} card - Карточка начала

\* @param {string} side - Сторона ('top', 'right', 'bottom', 'left')

\*/

function startDrawingLine(card, side) {

activeState.isDrawingLine = true;

activeState.lineStart = { card, side };

// Создание превью линии

activeState.previewLine = document.createElementNS('http://www.w3.org/2000/svg', 'path');

activeState.previewLine.setAttribute('class', 'line');

const lineColor = activeState.currentColor || DEFAULT\_LINE\_COLOR;

activeState.previewLine.setAttribute('stroke', lineColor);

activeState.previewLine.setAttribute('stroke-dasharray', '5,5');

activeState.previewLine.setAttribute('stroke-width', activeState.currentThickness);

activeState.previewLine.style.setProperty('--line-color', lineColor);

activeState.previewLine.setAttribute('marker-start', 'url(#marker-dot)');

activeState.previewLine.setAttribute('marker-end', 'url(#marker-dot)');

svgLayer.appendChild(activeState.previewLine);

}

endDrawingLine(card, side)

/\*\*

\* Завершает рисование линии

\* @param {Object} card - Карточка окончания

\* @param {string} side - Сторона

\*/

function endDrawingLine(card, side) {

// Защита от соединения с самим собой

if (!activeState.lineStart || activeState.lineStart.card.id === card.id) {

cancelDrawing();

return;

}

const lineElement = activeState.previewLine;

lineElement.removeAttribute('stroke-dasharray');

const lineColor = activeState.currentColor || DEFAULT\_LINE\_COLOR;

// Создание объекта линии

const lineData = {

id: `line\_${Date.now()}\_${Math.floor(Math.random()\*1000)}`,

startCard: activeState.lineStart.card,

startSide: activeState.lineStart.side,

endCard: card,

endSide: side,

color: lineColor,

thickness: activeState.currentThickness,

element: lineElement

};

lines.push(lineData);

// Обработчик клика для выделения

lineElement.addEventListener('click', (e) => {

e.stopPropagation();

selectLine(lineData);

});

updateAllLines();

// Сброс состояния рисования

activeState.isDrawingLine = false;

activeState.lineStart = null;

activeState.previewLine = null;

activeState.linePointerId = null;

activeState.lineCaptureTarget = null;

}

cancelDrawing()

/\*\*

\* Отменяет рисование линии

\*/

function cancelDrawing() {

if (activeState.previewLine) activeState.previewLine.remove();

activeState.isDrawingLine = false;

activeState.lineStart = null;

activeState.previewLine = null;

if (activeState.lineCaptureTarget && activeState.linePointerId != null &&

activeState.lineCaptureTarget.releasePointerCapture) {

try {

activeState.lineCaptureTarget.releasePointerCapture(activeState.linePointerId);

} catch (\_) {}

}

activeState.linePointerId = null;

activeState.lineCaptureTarget = null;

}

updateLinePath(pathElement, p1, p2, side1, side2)

/\*\*

\* Обновляет путь SVG-линии

\* @param {SVGPathElement} pathElement - SVG path элемент

\* @param {{x, y}} p1 - Начальная точка

\* @param {{x, y}} p2 - Конечная точка

\* @param {string} side1 - Сторона начала

\* @param {string} side2 - Сторона конца (может быть null)

\*/

function updateLinePath(pathElement, p1, p2, side1, side2) {

let finalP2 = { ...p2 };

let midP1 = { ...p1 };

// Определение промежуточной точки для Г-образной линии

if (side1 === 'left' || side1 === 'right') {

midP1 = { x: p2.x, y: p1.y };

if (side2) finalP2.y = p2.y + (p2.y > p1.y ? -MARKER\_OFFSET : MARKER\_OFFSET);

} else {

midP1 = { x: p1.x, y: p2.y };

if (side2) finalP2.x = p2.x + (p2.x > p1.x ? -MARKER\_OFFSET : MARKER\_OFFSET);

}

// Установка пути

pathElement.setAttribute('d', `M ${p1.x} ${p1.y} L ${midP1.x} ${midP1.y} L ${finalP2.x} ${finalP2.y}`);

}

getPointCoords(cardData, side)

/\*\*

\* Получает координаты точки соединения

\* @param {Object} cardData - Данные карточки

\* @param {string} side - Сторона

\* @returns {{x: number, y: number}} - Координаты точки

\*/

function getPointCoords(cardData, side) {

const card = cardData.element;

const x = parseFloat(card.style.left);

const y = parseFloat(card.style.top);

const width = card.offsetWidth;

const height = card.offsetHeight;

switch (side) {

case 'top': return { x: x + width / 2, y: y };

case 'bottom': return { x: x + width / 2, y: y + height };

case 'left': return { x: x, y: y + height / 2 };

case 'right': return { x: x + width, y: y + height / 2 };

}

}

updateLinesForCard(cardId)

/\*\*

\* Обновляет все линии, связанные с карточкой

\* @param {string} cardId - ID карточки

\*/

function updateLinesForCard(cardId) {

lines.forEach(line => {

if (line.startCard.id === cardId || line.endCard.id === cardId) {

const startPoint = getPointCoords(line.startCard, line.startSide);

const endPoint = getPointCoords(line.endCard, line.endSide);

updateLinePath(line.element, startPoint, endPoint, line.startSide, line.endSide);

}

});

}

updateAllLines()

/\*\*

\* Обновляет все линии на canvas

\*/

function updateAllLines() {

lines.forEach(line => updateLinesForCard(line.startCard.id));

}

selectLine(lineData)

/\*\*

\* Выделяет линию и обновляет элементы управления

\* @param {Object} lineData - Данные линии

\*/

function selectLine(lineData) {

// Снятие предыдущего выделения

if (activeState.selectedLine) {

activeState.selectedLine.element.classList.remove('selected');

}

clearSelection();

activeState.selectedLine = lineData;

lineData.element.classList.add('selected');

// Обновление элементов управления

thicknessSlider.value = lineData.thickness;

thicknessValue.textContent = lineData.thickness;

const color = lineData.color || DEFAULT\_LINE\_COLOR;

hiddenLineColorPicker.value = color;

lineColorTrigger.style.backgroundColor = color;

activeState.currentThickness = lineData.thickness;

activeState.currentColor = color;

setupLineControls();

}

deleteLine(lineData)

/\*\*

\* Удаляет линию

\* @param {Object} lineData - Данные линии

\*/

function deleteLine(lineData) {

clearLineHighlightState(lineData.element);

lineData.element.remove();

lines = lines.filter(l => l.id !== lineData.id);

if (activeState.selectedLine && activeState.selectedLine.id === lineData.id) {

activeState.selectedLine = null;

}

}

toggleCardSelection(cardData)

/\*\*

\* Переключает выделение карточки

\* @param {Object} cardData - Данные карточки

\*/

function toggleCardSelection(cardData) {

if (activeState.selectedCards.has(cardData)) {

activeState.selectedCards.delete(cardData);

cardData.element.classList.remove('selected');

} else {

if (activeState.selectedLine) {

activeState.selectedLine.element.classList.remove('selected');

activeState.selectedLine = null;

}

activeState.selectedCards.add(cardData);

cardData.element.classList.add('selected');

}

}

setSelectionSet(newSet)

/\*\*

\* Устанавливает новый набор выделенных карточек

\* @param {Set} newSet - Новый набор карточек

\*/

function setSelectionSet(newSet) {

// Снятие старого выделения

activeState.selectedCards.forEach(card => card.element.classList.remove('selected'));

activeState.selectedCards.clear();

let lastCard = null;

newSet.forEach(cd => {

activeState.selectedCards.add(cd);

cd.element.classList.add('selected');

lastCard = cd;

});

if (lastCard) {

setLastFocusedCard(lastCard);

}

}

clearSelection()

/\*\*

\* Снимает выделение со всех карточек

\*/

function clearSelection() {

activeState.selectedCards.forEach(card => card.element.classList.remove('selected'));

activeState.selectedCards.clear();

}

startMarqueeSelection(e)

/\*\*

\* Начинает выделение рамкой

\* @param {PointerEvent} e - Событие указателя

\*/

function startMarqueeSelection(e) {

if (!e.ctrlKey) clearSelection();

activeState.isSelecting = true;

activeState.selectionPointerId = e.pointerId;

marqueeStart.x = e.clientX;

marqueeStart.y = e.clientY;

baseSelection = e.ctrlKey ? new Set(activeState.selectedCards) : new Set();

// Создание элемента рамки

if (!selectionBox) {

selectionBox = document.createElement('div');

selectionBox.className = 'selection-box';

document.body.appendChild(selectionBox);

}

selectionBox.style.left = `${marqueeStart.x}px`;

selectionBox.style.top = `${marqueeStart.y}px`;

selectionBox.style.width = '0px';

selectionBox.style.height = '0px';

selectionBox.style.display = 'block';

}

updateMarqueeSelection(e)

/\*\*

\* Обновляет выделение рамкой

\* @param {PointerEvent} e - Событие указателя

\*/

function updateMarqueeSelection(e) {

if (!activeState.isSelecting) return;

// Расчёт размеров рамки

const x = Math.min(e.clientX, marqueeStart.x);

const y = Math.min(e.clientY, marqueeStart.y);

const width = Math.abs(e.clientX - marqueeStart.x);

const height = Math.abs(e.clientY - marqueeStart.y);

selectionBox.style.left = `${x}px`;

selectionBox.style.top = `${y}px`;

selectionBox.style.width = `${width}px`;

selectionBox.style.height = `${height}px`;

// Определение пересечений

const selectionRect = selectionBox.getBoundingClientRect();

const newSet = new Set(baseSelection);

cards.forEach(cardData => {

const rect = cardData.element.getBoundingClientRect();

const intersect = rect.left < selectionRect.right &&

rect.right > selectionRect.left &&

rect.top < selectionRect.bottom &&

rect.bottom > selectionRect.top;

if (intersect) newSet.add(cardData);

});

setSelectionSet(newSet);

}

endMarqueeSelection()

/\*\*

\* Завершает выделение рамкой

\*/

function endMarqueeSelection() {

activeState.isSelecting = false;

if (activeState.selectionCaptureTarget && activeState.selectionPointerId != null &&

activeState.selectionCaptureTarget.releasePointerCapture) {

try {

activeState.selectionCaptureTarget.releasePointerCapture(activeState.selectionPointerId);

} catch (\_) {}

}

activeState.selectionPointerId = null;

activeState.selectionCaptureTarget = null;

if (selectionBox) {

selectionBox.style.display = 'none';

selectionBox.style.width = '0px';

selectionBox.style.height = '0px';

}

// Автоматический выход из режима выделения

if (activeState.selectedCards.size > 0) {

activeState.isSelectionMode = false;

updateDragModeButtons();

}

}

7.5.5 БЛОК: История операций (Undo/Redo)

serializeState()

/\*\*

\* Сериализует текущее состояние приложения

\* @returns {Object} - Сериализованное состояние

\*/

function serializeState() {

return {

cards: cards.map(c => ({

id: c.id,

x: parseFloat(c.element.style.left),

y: parseFloat(c.element.style.top),

width: c.element.style.width || null,

title: c.element.querySelector('.card-title')?.innerText ?? '',

bodyHTML: c.element.querySelector('.card-body')?.innerHTML ?? '',

isDarkMode: c.element.classList.contains('dark-mode'),

bodyClass: c.element.querySelector('.card-body')?.className.replace('card-body', '').trim() ?? '',

headerBg: c.element.querySelector('.card-header')?.style.background ?? '',

colorIndex: parseInt(c.element.querySelector('.color-changer')?.dataset.colorIndex || '0', 10),

note: c.note ? { ...c.note, window: null } : null,

badges: c.badges,

})),

lines: lines.map(l => ({

startId: l.startCard.id,

startSide: l.startSide,

endId: l.endCard.id,

endSide: l.endSide,

color: l.color,

thickness: l.thickness

}))

};

}

loadState(state, pushHistory = false)

/\*\*

\* Загружает состояние приложения

\* @param {Object} state - Сериализованное состояние

\* @param {boolean} pushHistory - Добавлять ли в историю

\*/

function loadState(state, pushHistory = false) {

// 1. Очистка текущего состояния

lines.forEach(l => {

clearLineHighlightState(l.element);

l.element.remove();

});

lines = [];

cards.forEach(c => {

if (c.note && c.note.window) c.note.window.remove();

c.element.remove();

});

cards = [];

activeState.selectedCards.clear();

activeState.selectedLine = null;

// 2. Создание карточек

const idMap = new Map();

state.cards.forEach(cd => {

const cardData = createCard({

...cd,

isLarge: (parseInt(cd.width, 10) === 494),

isTemplate: true

});

idMap.set(cd.id, cardData);

});

// 3. Создание линий

state.lines.forEach(ld => {

const startCard = idMap.get(ld.startId);

const endCard = idMap.get(ld.endId);

if (!startCard || !endCard) return;

const path = document.createElementNS('http://www.w3.org/2000/svg', 'path');

path.setAttribute('class', 'line');

const color = ld.color || DEFAULT\_LINE\_COLOR;

path.setAttribute('stroke', color);

path.setAttribute('stroke-width', ld.thickness);

path.style.setProperty('--line-color', color);

path.setAttribute('marker-start', 'url(#marker-dot)');

path.setAttribute('marker-end', 'url(#marker-dot)');

svgLayer.appendChild(path);

const lineData = {

id: `line\_${Date.now()}\_${Math.floor(Math.random()\*1000)}`,

startCard,

startSide: ld.startSide,

endCard,

endSide: ld.endSide,

color,

thickness: ld.thickness,

element: path

};

lines.push(lineData);

path.addEventListener('click', (e) => {

e.stopPropagation();

selectLine(lineData);

});

const p1 = getPointCoords(startCard, ld.startSide);

const p2 = getPointCoords(endCard, ld.endSide);

updateLinePath(path, p1, p2, ld.startSide, ld.endSide);

});

updateNotesButtonState();

// 4. Обработка истории

const currentState = serializeState();

if (pushHistory) {

saveState(currentState);

} else {

recalculateAndRender(currentState);

lastSavedSnapshot = JSON.stringify(currentState);

}

}

saveState(precomputedState = null)

/\*\*

\* Сохраняет текущее состояние в историю

\* @param {Object} precomputedState - Предварительно рассчитанное состояние

\*/

function saveState(precomputedState = null) {

const snapshot = precomputedState ?? serializeState();

// Защита от сохранения пустого начального состояния

if (undoStack.length === 0 && snapshot.cards.length === 0 && snapshot.lines.length === 0) {

recalculateAndRender(snapshot);

return;

}

const serialized = JSON.stringify(snapshot);

// Проверка на изменения

if (serialized === lastSavedSnapshot) {

recalculateAndRender(snapshot);

return;

}

// Добавление в стек

undoStack.push(serialized);

if (undoStack.length > HISTORY\_LIMIT) undoStack.shift();

// Очистка Redo при новом действии

redoStack = [];

lastSavedSnapshot = serialized;

recalculateAndRender(snapshot);

}

undo()

/\*\*

\* Отменяет последнее действие

\*/

function undo() {

if (undoStack.length < 2) return;

const current = undoStack.pop();

redoStack.push(current);

const prev = JSON.parse(undoStack[undoStack.length - 1]);

loadState(prev, false);

}

redo()

/\*\*

\* Повторяет отменённое действие

\*/

function redo() {

if (redoStack.length === 0) return;

const snapshot = redoStack.pop();

undoStack.push(snapshot);

loadState(JSON.parse(snapshot), false);

}

7.5.6 БЛОК: Буфер обмена

copySelection()

/\*\*

\* Копирует выделенные карточки в буфер

\*/

function copySelection() {

if (activeState.selectedCards.size === 0) return;

const selectedIds = new Set([...activeState.selectedCards].map(c => c.id));

// Копирование карточек

const copiedCards = [];

activeState.selectedCards.forEach(cd => {

const state = serializeState().cards.find(c => c.id === cd.id);

if (state) copiedCards.push(state);

});

// Копирование линий между выделенными карточками

const copiedLines = [];

lines.forEach(l => {

if (selectedIds.has(l.startCard.id) && selectedIds.has(l.endCard.id)) {

copiedLines.push({

startId: l.startCard.id,

startSide: l.startSide,

endId: l.endCard.id,

endSide: l.endSide,

color: l.color || DEFAULT\_LINE\_COLOR,

thickness: l.thickness

});

}

});

clipboard = { cards: copiedCards, lines: copiedLines };

}

pasteSelection()

/\*\*

\* Вставляет содержимое буфера обмена

\*/

function pasteSelection() {

if (!clipboard || !clipboard.cards || clipboard.cards.length === 0) return;

const OFFSET = GRID\_SIZE;

const idMap = new Map();

const newSelection = new Set();

// Создание карточек со смещением

clipboard.cards.forEach(cd => {

const newCard = createCard({

...cd,

x: cd.x + OFFSET,

y: cd.y + OFFSET,

isLarge: (parseInt(cd.width, 10) === 494),

note: cd.note ? { ...cd.note, x: cd.note.x + OFFSET, y: cd.note.y + OFFSET, visible: false } : null,

});

idMap.set(cd.id, newCard);

newSelection.add(newCard);

});

setSelectionSet(newSelection);

// Создание линий с задержкой

setTimeout(() => {

clipboard.lines.forEach(ld => {

const startCard = idMap.get(ld.startId);

const endCard = idMap.get(ld.endId);

if (!startCard || !endCard) return;

const path = document.createElementNS('http://www.w3.org/2000/svg', 'path');

path.setAttribute('class', 'line');

const color = ld.color || DEFAULT\_LINE\_COLOR;

path.setAttribute('stroke', color);

path.setAttribute('stroke-width', ld.thickness);

path.style.setProperty('--line-color', color);

path.setAttribute('marker-start', 'url(#marker-dot)');

path.setAttribute('marker-end', 'url(#marker-dot)');

svgLayer.appendChild(path);

const lineData = {

id: `line\_${Date.now()}\_${Math.floor(Math.random()\*1000)}`,

startCard,

startSide: ld.startSide,

endCard,

endSide: ld.endSide,

color,

thickness: ld.thickness,

element: path

};

lines.push(lineData);

path.addEventListener('click', (e) => {

e.stopPropagation();

selectLine(lineData);

});

const p1 = getPointCoords(startCard, ld.startSide);

const p2 = getPointCoords(endCard, ld.endSide);

updateLinePath(path, p1, p2, ld.startSide, ld.endSide);

});

saveState();

}, 0);

}

7.5.7 БЛОК: Система заметок

Структура данных заметки:

/\*\*

\* {

\* text: string, // Устаревшее поле (для совместимости)

\* entries: { // Записи по датам

\* "2025-01-15": {

\* text: string,

\* updatedAt: string // ISO timestamp

\* }

\* },

\* colors: { // Цвета по датам

\* "2025-01-15": "#f44336"

\* },

\* selectedDate: string, // Текущая выбранная дата (YYYY-MM-DD)

\* highlightColor: string, // Цвет индикатора по умолчанию

\* width: number, // Ширина окна

\* height: number, // Высота окна

\* visible: boolean, // Видимость окна

\* window: HTMLElement|null, // DOM-элемент окна

\* x: number, // X-координата окна

\* y: number // Y-координата окна

\* }

\*/

toggleNote(cardData)

/\*\*

\* Переключает видимость заметки

\* @param {Object} cardData - Данные карточки

\*/

function toggleNote(cardData) {

if (cardData.note && cardData.note.window) {

// Закрытие существующей заметки

cardData.note.window.remove();

cardData.note.window = null;

cardData.note.visible = false;

setCardNoteHighlight(cardData, false);

} else {

// Создание новой заметки

if (!cardData.note) {

const cardRect = cardData.element.getBoundingClientRect();

cardData.note = {

text: '',

entries: {},

colors: {},

selectedDate: formatLocalYMD(new Date()),

highlightColor: '#f44336',

width: 260,

height: 380,

visible: false,

window: null,

x: cardRect.right + 15,

y: cardRect.top

};

}

cardData.note.visible = true;

createNoteWindow(cardData);

setCardNoteHighlight(cardData, true);

}

updateNoteButtonAppearance(cardData);

saveState();

updateNotesButtonState();

}

createNoteWindow(cardData)

/\*\*

\* Создаёт окно заметки

\* @param {Object} cardData - Данные карточки

\*/

function createNoteWindow(cardData) {

const note = cardData.note;

ensureNoteStructure(note);

// 1. СОЗДАНИЕ ЭЛЕМЕНТА ОКНА

const noteWindow = document.createElement('div');

noteWindow.className = 'note-window';

noteWindow.style.left = `${note.x}px`;

noteWindow.style.top = `${note.y}px`;

if (Number.isFinite(note.width) && note.width >= 200) {

noteWindow.style.width = `${note.width}px`;

}

if (Number.isFinite(note.height) && note.height >= 200) {

noteWindow.style.height = `${note.height}px`;

}

// 2. HTML-СТРУКТУРА

noteWindow.innerHTML = `

<style>

/\* Встроенные стили календаря \*/

.note-header .note-close-btn { font-size: 20px; cursor: pointer; padding: 0 8px; border: none; background: transparent; }

.note-header{display:flex;align-items:center;gap:8px;justify-content:space-between}

.note-cal-wrap{padding:6px 8px 0 8px}

.cal-head{display:flex;align-items:center;justify-content:space-between;margin-bottom:4px;font-weight:700}

.cal-month{font-size:12px}

.cal-nav{display:flex;gap:6px}

.cal-btn{border:none;border-radius:6px;padding:2px 6px;cursor:pointer;box-shadow:0 1px 3px rgba(0,0,0,.2);background:#fff}

.cal-grid{display:grid;grid-template-columns:repeat(7,1fr);gap:4px;font-size:11px}

.cal-dow{opacity:.7;text-align:center}

.cal-cell{height:24px;display:flex;align-items:center;justify-content:center;border-radius:6px;cursor:pointer;background:#fff}

.cal-cell.out{opacity:.35}

.cal-cell.selected{outline:2px solid #4caf50}

.cal-cell.has-entry{box-shadow: inset 0 0 0 2px rgba(0,0,0,.08)}

.note-tools{display:flex;gap:6px;align-items:center;margin-left:auto;margin-right:6px}

.clr-dot{width:18px;height:18px;border-radius:50%;border:2px solid #333;cursor:pointer;box-shadow:0 1px 3px rgba(0,0,0,.2)}

.clr-dot.active{box-shadow:0 0 0 2px rgba(0,0,0,.25), inset 0 0 0 2px #fff}

</style>

<div class="note-header">

<button class="note-close-btn" type="button" title="Закрыть">×</button>

<div class="note-tools">

<div class="clr-dot" data-color="#f44336" title="Красный" style="background:#f44336"></div>

<div class="clr-dot" data-color="#4caf50" title="Зелёный" style="background:#4caf50"></div>

<div class="clr-dot" data-color="#42a5f5" title="Синий" style="background:#42a5f5"></div>

</div>

</div>

<div class="note-content-scroller">

<div class="note-cal-wrap">

<div class="cal-head">

<button class="cal-btn prev">‹</button>

<div class="cal-month"></div>

<button class="cal-btn next">›</button>

</div>

<div class="cal-grid">

<div class="cal-dow">Пн</div><div class="cal-dow">Вт</div><div class="cal-dow">Ср</div>

<div class="cal-dow">Чт</div><div class="cal-dow">Пт</div><div class="cal-dow">Сб</div><div class="cal-dow">Вс</div>

</div>

</div>

<textarea class="note-textarea" placeholder="Введите текст заметки..."></textarea>

</div>

<div class="note-resize-handle"></div>

`;

document.body.appendChild(noteWindow);

note.window = noteWindow;

setCardNoteHighlight(cardData, true);

// 3. ПОЛУЧЕНИЕ ССЫЛОК НА ЭЛЕМЕНТЫ

const header = noteWindow.querySelector('.note-header');

const textarea = noteWindow.querySelector('.note-textarea');

const colorDots = noteWindow.querySelectorAll('.clr-dot');

const calMonthEl = noteWindow.querySelector('.cal-month');

const calGrid = noteWindow.querySelector('.cal-grid');

const prevBtn = noteWindow.querySelector('.prev');

const nextBtn = noteWindow.querySelector('.next');

let viewDate = parseYMDToDate(note.selectedDate) || new Date();

// 4. ФУНКЦИИ КАЛЕНДАРЯ

const updateTextareaValue = () => {

const info = getNoteEntryInfo(note.entries[note.selectedDate]);

textarea.value = info.text;

};

const applyAccent = () => {

const accent = note.colors[note.selectedDate] || note.highlightColor;

if (accent) {

note.highlightColor = accent;

noteWindow.style.setProperty('--note-accent', accent);

} else {

noteWindow.style.removeProperty('--note-accent');

}

};

function updateColorDotsActive() {

const currentColor = note.colors[note.selectedDate] || note.highlightColor;

colorDots.forEach((d) =>

d.classList.toggle('active', d.getAttribute('data-color') === currentColor)

);

applyAccent();

}

function renderCalendar() {

calGrid.innerHTML = `

<div class="cal-dow">Пн</div><div class="cal-dow">Вт</div><div class="cal-dow">Ср</div>

<div class="cal-dow">Чт</div><div class="cal-dow">Пт</div><div class="cal-dow">Сб</div><div class="cal-dow">Вс</div>

`;

const first = new Date(viewDate.getFullYear(), viewDate.getMonth(), 1);

const last = new Date(viewDate.getFullYear(), viewDate.getMonth()+1, 0);

const startIndex = (first.getDay() + 6) % 7;

calMonthEl.textContent = viewDate.toLocaleDateString('ru-RU',{month:'long', year:'numeric'});

// Дни предыдущего месяца

const daysInPrev = new Date(viewDate.getFullYear(), viewDate.getMonth(), 0).getDate();

for (let i=0; i<startIndex; i++){

const d = new Date(viewDate.getFullYear(), viewDate.getMonth()-1, daysInPrev - startIndex + 1 + i);

calGrid.appendChild(makeCell(d, true));

}

// Дни текущего месяца

for (let day=1; day<=last.getDate(); day++){

const d = new Date(viewDate.getFullYear(), viewDate.getMonth(), day);

calGrid.appendChild(makeCell(d, false));

}

// Дни следующего месяца (до 42 ячеек)

const totalCells = calGrid.querySelectorAll('.cal-cell').length;

if (totalCells < 42) {

for (let i=1; i<=42 - totalCells; i++){

const d = new Date(viewDate.getFullYear(), viewDate.getMonth()+1, i);

calGrid.appendChild(makeCell(d, true));

}

}

}

function makeCell(dateObj, outMonth) {

const cell = document.createElement('div');

cell.className = 'cal-cell' + (outMonth ? ' out' : '');

const dateStr = formatLocalYMD(dateObj);

cell.textContent = String(dateObj.getDate());

// Выделение выбранной даты

if (dateStr === note.selectedDate) cell.classList.add('selected');

// Отображение записи

const entryInfo = getNoteEntryInfo(note.entries[dateStr]);

const hasEntry = entryInfo.text.trim().length > 0;

if (hasEntry) {

cell.classList.add('has-entry');

const dayColor = note.colors[dateStr] || note.highlightColor;

cell.style.background = dayColor;

cell.style.color = '#fff';

}

// Клик по дню

cell.addEventListener('click', () => {

note.selectedDate = normalizeYMD(dateStr) || dateStr;

renderCalendar();

updateTextareaValue();

updateColorDotsActive();

});

return cell;

}

// 5. ОБРАБОТЧИКИ СОБЫТИЙ

// 5.1. Навигация по месяцам

prevBtn.addEventListener('click', () => {

viewDate.setMonth(viewDate.getMonth() - 1);

renderCalendar();

updateColorDotsActive();

updateTextareaValue();

});

nextBtn.addEventListener('click', () => {

viewDate.setMonth(viewDate.getMonth() + 1);

renderCalendar();

updateColorDotsActive();

updateTextareaValue();

});

// 5.2. Выбор цвета

colorDots.forEach(dot => {

dot.addEventListener('click', () => {

const c = dot.getAttribute('data-color');

if (!c) return;

note.colors[note.selectedDate] = c;

note.highlightColor = c;

updateColorDotsActive();

renderCalendar();

saveState();

if (notesDropdownApi?.isOpen()) notesDropdownApi.refresh?.();

updateNoteButtonAppearance(cardData);

});

});

// 5.3. Редактирование текста

textarea.addEventListener('input', () => {

const val = textarea.value;

setNoteEntryValue(note, note.selectedDate, val);

updateNoteButtonAppearance(cardData);

renderCalendar();

updateNotesButtonState();

if (notesDropdownApi?.isOpen()) notesDropdownApi.refresh?.();

});

textarea.addEventListener('blur', () => {

saveState();

if (notesDropdownApi?.isOpen()) notesDropdownApi.refresh?.();

});

// 5.4. Кнопка закрытия

const closeBtn = noteWindow.querySelector('.note-close-btn');

if (closeBtn) {

closeBtn.addEventListener('click', (e) => {

e.stopPropagation();

note.visible = false;

noteWindow.remove();

note.window = null;

setCardNoteHighlight(cardData, false);

saveState();

updateNotesButtonState();

updateNoteButtonAppearance(cardData);

});

}

// 5.5. Перемещение окна

if (header) {

header.addEventListener('pointerdown', (e) => {

if (e.target.closest('.note-close-btn')) return;

if (e.target.closest('.note-tools')) return;

e.preventDefault();

if (pinchState && e.pointerType === 'touch') return;

const pointerId = e.pointerId;

const startX = e.clientX, startY = e.clientY;

const startNoteX = note.x, startNoteY = note.y;

if (header.setPointerCapture) {

try { header.setPointerCapture(pointerId); } catch (\_) {}

}

const onMove = (e2) => {

if (e2.pointerId !== pointerId) return;

const dx = e2.clientX - startX;

const dy = e2.clientY - startY;

note.x = startNoteX + dx;

note.y = startNoteY + dy;

noteWindow.style.left = `${note.x}px`;

noteWindow.style.top = `${note.y}px`;

};

const finish = (e2) => {

if (e2.pointerId !== pointerId) return;

if (header.releasePointerCapture) {

try { header.releasePointerCapture(pointerId); } catch (\_) {}

}

header.removeEventListener('pointermove', onMove);

header.removeEventListener('pointerup', finish);

header.removeEventListener('pointercancel', cancel);

saveState();

};

const cancel = (e2) => {

if (e2.pointerId !== pointerId) return;

if (header.releasePointerCapture) {

try { header.releasePointerCapture(pointerId); } catch (\_) {}

}

header.removeEventListener('pointermove', onMove);

header.removeEventListener('pointerup', finish);

header.removeEventListener('pointercancel', cancel);

};

header.addEventListener('pointermove', onMove);

header.addEventListener('pointerup', finish);

header.addEventListener('pointercancel', cancel);

});

}

// 5.6. Отслеживание изменения размеров

new ResizeObserver(() => {

const w = noteWindow.offsetWidth;

const h = noteWindow.offsetHeight;

if (w >= 200) note.width = w;

if (h >= 200) note.height = h;

}).observe(noteWindow);

// 6. ИНИЦИАЛИЗАЦИЯ

renderCalendar();

updateColorDotsActive();

updateTextareaValue();

}

Вспомогательные функции заметок:

/\*\*

\* Нормализует дату к формату YYYY-MM-DD

\*/

function normalizeYMD(value) {

if (typeof value === 'string' && /^\d{4}-\d{2}-\d{2}$/.test(value)) return value;

const parsed = new Date(value);

if (Number.isNaN(parsed.getTime())) return '';

return formatLocalYMD(parsed);

}

/\*\*

\* Форматирует дату в YYYY-MM-DD

\*/

function formatLocalYMD(date) {

if (!(date instanceof Date) || Number.isNaN(date.getTime())) return '';

const y = date.getFullYear();

const m = String(date.getMonth() + 1).padStart(2, '0');

const d = String(date.getDate()).padStart(2, '0');

return `${y}-${m}-${d}`;

}

/\*\*

\* Парсит YYYY-MM-DD в Date

\*/

function parseYMDToDate(value) {

const normalized = normalizeYMD(value);

if (!normalized) return null;

const parts = normalized.split('-').map(Number);

if (parts.length !== 3) return null;

const [year, month, day] = parts;

return new Date(year, month - 1, day);

}

/\*\*

\* Получает информацию о записи

\*/

function getNoteEntryInfo(entry) {

if (!entry) return { text: '', updatedAt: null };

if (typeof entry === 'string') return { text: entry, updatedAt: null };

if (typeof entry === 'object') {

const text = typeof entry.text === 'string' ? entry.text : '';

const updatedAt = typeof entry.updatedAt === 'string' ? entry.updatedAt : null;

return { text, updatedAt };

}

return { text: '', updatedAt: null };

}

/\*\*

\* Устанавливает значение записи

\*/

function setNoteEntryValue(note, date, value) {

if (!note.entries) note.entries = {};

const key = normalizeYMD(date);

if (!key) return;

const raw = value ?? '';

const trimmed = raw.trim();

if (trimmed) {

note.entries[key] = { text: raw, updatedAt: new Date().toISOString() };

} else {

delete note.entries[key];

}

}

/\*\*

\* Проверяет наличие записей в заметке

\*/

function hasAnyEntry(note) {

if (!note) return false;

if (note.entries && typeof note.entries === 'object') {

return Object.values(note.entries).some((entry) => {

const info = getNoteEntryInfo(entry);

return info.text.trim().length > 0;

});

}

return false;

}

/\*\*

\* Обеспечивает правильную структуру заметки

\*/

function ensureNoteStructure(note) {

if (!note.entries || typeof note.entries !== 'object') note.entries = {};

if (!note.colors) note.colors = {};

// Нормализация дат в colors

const normalizedColors = {};

Object.entries(note.colors).forEach(([date, color]) => {

const normalizedDate = normalizeYMD(date);

if (normalizedDate) normalizedColors[normalizedDate] = color;

});

note.colors = normalizedColors;

// Установка selectedDate

if (!note.selectedDate) {

note.selectedDate = formatLocalYMD(new Date());

} else {

note.selectedDate = normalizeYMD(note.selectedDate) || formatLocalYMD(new Date());

}

if (!note.highlightColor) note.highlightColor = '#f44336';

// Нормализация и очистка entries

Object.entries({ ...note.entries }).forEach(([date, entry]) => {

const info = getNoteEntryInfo(entry);

if (!info.text.trim()) {

delete note.entries[date];

} else {

const normalizedDate = normalizeYMD(date);

if (!normalizedDate) {

delete note.entries[date];

return;

}

if (normalizedDate !== date) {

delete note.entries[date];

note.entries[normalizedDate] = { text: info.text, updatedAt: info.updatedAt || null };

} else {

note.entries[date] = { text: info.text, updatedAt: info.updatedAt || null };

}

}

});

// Миграция старого формата

if (note.text && !note.entries[note.selectedDate]) {

note.entries[note.selectedDate] = { text: note.text, updatedAt: null };

note.text = '';

}

}

7.5.8 БЛОК: Расчёты и отображение

recalculateAndRender(stateOverride = null)

/\*\*

\* Пересчитывает баллы и обновляет отображение

\* @param {Object} stateOverride - Предварительное состояние

\*/

function recalculateAndRender(stateOverride = null) {

try {

if (!window.Engine) {

lastEngineMeta = null;

return;

}

const state = stateOverride ?? serializeState();

const { result, meta } = Engine.recalc(state);

lastEngineMeta = meta;

const id2el = new Map(cards.map(c => [c.id, c.element]));

const highlightCandidates = new Map();

// Обновление каждой карточки

cards.forEach(cd => {

const el = cd.element;

const cardResult = result[cd.id];

ensureActiveControls(el);

// Обновление цвета монетки

const circle = el.querySelector('.coin-icon circle');

if (circle) {

const full = meta.isFull[cd.id];

circle.setAttribute('fill', full ? '#ffd700' : '#3d85c6');

}

// Получение бонусов

const hidden = el.querySelector('.active-pv-hidden');

const aBonusL = hidden ? parseInt(hidden.dataset.abonusl || '0', 10) : 0;

const aBonusR = hidden ? parseInt(hidden.dataset.abonusr || '0', 10) : 0;

// Обновление строк данных

const rows = el.querySelectorAll('.card-row');

rows.forEach(row => {

const label = row.querySelector('.label');

const value = row.querySelector('.value');

if (!label || !value) return;

const name = (label.textContent || '').trim().toLowerCase();

if (name.startsWith('баланс')) {

// Обработка ручного ввода баланса

const manual = value.dataset.manualBalance ? readStoredManualBalance(el) : null;

let manualLeft = 0;

let manualRight = 0;

let hasManualNumbers = false;

if (manual) {

if (Number.isFinite(manual.left) && Number.isFinite(manual.right)) {

manualLeft = manual.left;

manualRight = manual.right;

hasManualNumbers = true;

} else if (manual.raw) {

value.textContent = manual.raw;

return;

} else {

delete value.dataset.manualBalance;

}

}

// Расчёт баланса

const r = cardResult || { L: 0, R: 0, total: 0 };

let localL = hidden ? parseInt(hidden.dataset.locall || '0', 10) : 0;

let localR = hidden ? parseInt(hidden.dataset.localr || '0', 10) : 0;

if (!Number.isFinite(localL)) localL = 0;

if (!Number.isFinite(localR)) localR = 0;

if (localL < 0) localL = 0;

if (localR < 0) localR = 0;

const leftBalance = Math.max(0, (r.L || 0) + aBonusL + localL);

const rightBalance = Math.max(0, (r.R || 0) + aBonusR + localR);

// Определение необходимости подсветки

if (!hasManualNumbers && cardResult) {

const prev = lastCalculatedBalances.get(cd.id);

const prevL = prev && Number.isFinite(prev.L) ? prev.L : null;

const prevR = prev && Number.isFinite(prev.R) ? prev.R : null;

const nextL = Number.isFinite(cardResult.L) ? cardResult.L : 0;

const nextR = Number.isFinite(cardResult.R) ? cardResult.R : 0;

if (prevL !== null && nextL > prevL) {

const entry = highlightCandidates.get(cd.id) || { left: false, right: false };

entry.left = true;

highlightCandidates.set(cd.id, entry);

}

if (prevR !== null && nextR > prevR) {

const entry = highlightCandidates.get(cd.id) || { left: false, right: false };

entry.right = true;

highlightCandidates.set(cd.id, entry);

}

}

// Отображение

if (hasManualNumbers) {

value.textContent = formatManualBalance({ left: manualLeft, right: manualRight });

} else {

renderSplitValue(value, leftBalance, rightBalance);

}

}

});

});

// Применение подсветки

applyBalanceHighlights(highlightCandidates, id2el);

// Обновление кеша балансов

Object.entries(result || {}).forEach(([cardId, data]) => {

const nextL = data && Number.isFinite(data.L) ? data.L : 0;

const nextR = data && Number.isFinite(data.R) ? data.R : 0;

lastCalculatedBalances.set(cardId, { L: nextL, R: nextR });

});

// Очистка удалённых карточек из кеша

Array.from(lastCalculatedBalances.keys()).forEach((cardId) => {

if (!result[cardId]) {

lastCalculatedBalances.delete(cardId);

}

});

} catch (e) {

console.warn('Recalc/render error:', e);

}

}

Функции подсветки:

/\*\*

\* Применяет подсветку изменённых балансов

\*/

function applyBalanceHighlights(highlightCandidates, idToElementMap) {

const previous = new Map();

highlightedBalanceParts.forEach((sides, id) => {

if (idToElementMap.has(id)) {

previous.set(id, { left: !!sides.left, right: !!sides.right });

}

});

const timing = getHighlightTimingConfig();

const highlightOptions = {

autoRemoveDuration: timing.autoRemoveDuration,

animationDuration: timing.cssDuration

};

// Снятие старой подсветки

if (!highlightCandidates || highlightCandidates.size === 0) {

previous.forEach((\_, id) => {

const cardEl = idToElementMap.get(id);

const valueEl = cardEl ? getBalanceValueElement(cardEl) : null;

if (!valueEl) return;

setPartHighlight(valueEl, 'L', false);

setPartHighlight(valueEl, 'R', false);

});

highlightedBalanceParts = new Map();

return;

}

// Снятие подсветки с неактивных карточек

previous.forEach((\_, id) => {

if (highlightCandidates.has(id)) return;

const cardEl = idToElementMap.get(id);

const valueEl = cardEl ? getBalanceValueElement(cardEl) : null;

if (!valueEl) return;

setPartHighlight(valueEl, 'L', false);

setPartHighlight(valueEl, 'R', false);

});

// Применение новой подсветки

const applied = new Map();

highlightCandidates.forEach((sides, id) => {

const cardEl = idToElementMap.get(id);

const valueEl = cardEl ? getBalanceValueElement(cardEl) : null;

if (!valueEl) return;

// Пропуск ручного баланса

if (valueEl.dataset.manualBalance === 'true') {

setPartHighlight(valueEl, 'L', false);

setPartHighlight(valueEl, 'R', false);

return;

}

const appliedSides = { left: false, right: false };

if (sides.left) {

if (setPartHighlight(valueEl, 'L', true, highlightOptions)) appliedSides.left = true;

} else {

setPartHighlight(valueEl, 'L', false);

}

if (sides.right) {

if (setPartHighlight(valueEl, 'R', true, highlightOptions)) appliedSides.right = true;

} else {

setPartHighlight(valueEl, 'R', false);

}

if (appliedSides.left || appliedSides.right) {

applied.set(id, appliedSides);

}

});

highlightedBalanceParts = applied;

// Подсветка линий

applied.forEach((sides, id) => {

if (sides.left) highlightLinesForParentSide(id, 'L', 'balance');

if (sides.right) highlightLinesForParentSide(id, 'R', 'balance');

});

}

/\*\*

* Устанавливает подсветку части значения \*/

function setPartHighlight(valueEl, side, shouldHighlight, options = {}) {

let part = null;

const selector = side === 'L' ? '.value-left' : '.value-right';

if (valueEl.dataset.manualBalance === 'true') {

part = valueEl.querySelector(selector);

} else {

part = getValuePart(valueEl, side);

}

if (!part) return false;

const { autoRemoveDuration, animationDuration } = options;

if (!shouldHighlight) {

part.classList.remove('balance-highlight');

const timer = partHighlightTimers.get(part);

if (timer) {

clearTimeout(timer);

partHighlightTimers.delete(part);

}

return false;

}

// Установка длительности анимации

if (Number.isFinite(animationDuration)) {

const sanitizedDuration = Math.min(MAX\_ANIMATION\_DURATION, Math.max(MIN\_ANIMATION\_DURATION, animationDuration));

part.style.setProperty('--balance-animation-duration', `${sanitizedDuration}ms`);

}

// Перезапуск анимации

part.classList.remove('balance-highlight');

void part.offsetWidth;

part.classList.add('balance-highlight');

// Установка таймера автоудаления

if (autoRemoveDuration == null || autoRemoveDuration <= 0) {

const prevTimer = partHighlightTimers.get(part);

if (prevTimer) {

clearTimeout(prevTimer);

partHighlightTimers.delete(part);

}

return true;

}

const prevTimer = partHighlightTimers.get(part);

if (prevTimer) clearTimeout(prevTimer);

const newTimer = setTimeout(() => {

part.classList.remove('balance-highlight');

partHighlightTimers.delete(part);

}, autoRemoveDuration);

partHighlightTimers.set(part, newTimer);

return true;

}

/\*\*

* Подсвечивает линии для стороны родителя \*/

function highlightLinesForParentSide(parentId, side, type) {

if (!parentId) return;

const normalizedSide = normalizeBranchSide(side);

lines.forEach((line) => {

const relation = getLineRelation(line);

if (!relation) return;

if (relation.parent.id !== parentId) return;

if (relation.side !== normalizedSide) return;

applyLineHighlight(line.element, type);

});

}

/\*\*

* Подсвечивает линию между родителем и ребёнком \*/

function highlightLineBetween(parentId, childId, side, type) {

if (!parentId || !childId) return;

const normalizedSide = normalizeBranchSide(side);

for (const line of lines) {

const relation = getLineRelation(line);

if (!relation) continue;

if (relation.parent.id !== parentId) continue;

if (relation.child.id !== childId) continue;

if (relation.side !== normalizedSide) continue;

applyLineHighlight(line.element, type);

break;

}

}

/\*\*

* Применяет подсветку к линии \*/

function applyLineHighlight(lineEl, type) {

if (!lineEl) return;

clearLineHighlightState(lineEl);

const { cssDuration, autoRemoveDuration } = getHighlightTimingConfig();

const className = type === 'pv' ? 'line--pv-highlight' : 'line--balance-highlight';

lineEl.classList.remove(className);

try {

void lineEl.getBBox();

} catch (\_) {

try {

lineEl.getBoundingClientRect();

} catch (\_\_) {}

}

lineEl.style.setProperty('--line-animation-duration', `${cssDuration}ms`);

lineEl.classList.add(className);

let perLine = lineHighlightTimers.get(lineEl);

if (perLine && perLine.has(className)) {

clearTimeout(perLine.get(className));

perLine.delete(className);

if (perLine.size === 0) {

lineHighlightTimers.delete(lineEl);

perLine = null;

}

}

if (autoRemoveDuration == null) return;

if (!perLine) {

perLine = new Map();

lineHighlightTimers.set(lineEl, perLine);

}

const timer = setTimeout(() => {

lineEl.classList.remove(className);

const stored = lineHighlightTimers.get(lineEl);

if (stored) {

stored.delete(className);

if (stored.size === 0) lineHighlightTimers.delete(lineEl);

}

}, autoRemoveDuration);

perLine.set(className, timer);

}

/\*\*

* Очищает состояние подсветки линии \*/

function clearLineHighlightState(lineEl) {

const timers = lineHighlightTimers.get(lineEl);

if (timers) {

timers.forEach((timer) => clearTimeout(timer));

lineHighlightTimers.delete(lineEl);

}

if (lineEl) {

lineEl.classList.remove('line--balance-highlight', 'line--pv-highlight');

}

}

/\*\*

* Получает конфигурацию времени подсветки \*/

function getHighlightTimingConfig() {

const base = Number.isFinite(animationSettings.durationMs) ? animationSettings.durationMs : DEFAULT\_ANIMATION\_DURATION;

const sanitizedDuration = Math.min(MAX\_ANIMATION\_DURATION, Math.max(MIN\_ANIMATION\_DURATION, base));

const cssDuration = Math.min(MAX\_ANIMATION\_LOOP\_DURATION, sanitizedDuration);

return { cssDuration, autoRemoveDuration: sanitizedDuration };

}

7.5.9 БЛОК: Активные PV (Актив-заказы)

ensureActiveControls(cardEl)

/\*\*

\* Обеспечивает наличие контролов для Актив-PV

\* @param {HTMLElement} cardEl - Элемент карточки

\*/

function ensureActiveControls(cardEl) {

const rows = cardEl.querySelectorAll('.card-row');

let activeRow = null;

rows.forEach(r => {

const lab = r.querySelector('.label');

if (lab && (lab.textContent || '').trim().toLowerCase().startsWith('актив-заказы')) {

activeRow = r;

}

});

if (!activeRow) return;

activeRow.classList.add('active-pv-row');

// Создание контролов, если отсутствуют

if (!cardEl.querySelector('.active-pv-controls')) {

const controls = document.createElement('div');

controls.className = 'active-pv-controls';

controls.innerHTML = `

<div class="left-controls">

<button class="active-btn" data-dir="L" data-step="1">+1</button>

<button class="active-btn" data-dir="L" data-step="10">+10</button>

<button class="active-btn" data-dir="L" data-step="-10">-10</button>

<button class="active-btn" data-dir="L" data-step="-1">-1</button>

</div>

<div class="mid-controls">

<button class="active-btn active-clear">Очистить</button>

</div>

<div class="right-controls">

<button class="active-btn" data-dir="R" data-step="-1">-1</button>

<button class="active-btn" data-dir="R" data-step="-10">-10</button>

<button class="active-btn" data-dir="R" data-step="10">+10</button>

<button class="active-btn" data-dir="R" data-step="1">+1</button>

</div>`;

activeRow.insertAdjacentElement('afterend', controls);

}

// Создание скрытого элемента для хранения данных

let hidden = cardEl.querySelector('.active-pv-hidden');

if (!hidden) {

hidden = document.createElement('span');

hidden.className = 'active-pv-hidden';

hidden.style.display = 'none';

hidden.dataset.btnL = '0';

hidden.dataset.btnR = '0';

hidden.dataset.abonusl = '0';

hidden.dataset.abonusr = '0';

hidden.dataset.locall = '0';

hidden.dataset.localr = '0';

activeRow.insertAdjacentElement('afterend', hidden);

} else {

// Инициализация отсутствующих полей

hidden.dataset.btnL = hidden.dataset.btnL || '0';

hidden.dataset.btnR = hidden.dataset.btnR || '0';

hidden.dataset.abonusl = hidden.dataset.abonusl || '0';

hidden.dataset.abonusr = hidden.dataset.abonusr || '0';

hidden.dataset.locall = hidden.dataset.locall || '0';

hidden.dataset.localr = hidden.dataset.localr || '0';

}

hidden.dataset.manualBalanceLeft = hidden.dataset.manualBalanceLeft || '';

hidden.dataset.manualBalanceRight = hidden.dataset.manualBalanceRight || '';

hidden.dataset.manualBalanceRaw = hidden.dataset.manualBalanceRaw || '';

// Блокировка редактирования поля Актив-PV

const valEl = activeRow.querySelector('.value');

if (valEl) {

valEl.setAttribute('contenteditable', 'false');

['beforeinput', 'input', 'keydown', 'paste'].forEach(ev =>

valEl.addEventListener(ev, (e) => {

e.stopPropagation();

e.preventDefault();

}, { capture: true })

);

}

}

parseActivePV(cardEl)

/\*\*

\* Парсит значения Актив-PV из карточки

\* @param {HTMLElement} cardEl - Элемент карточки

\* @returns {{L, R, row, valEl}}

\*/

function parseActivePV(cardEl) {

const row = Array.from(cardEl.querySelectorAll('.card-row')).find(r => {

const lab = r.querySelector('.label');

return lab && (lab.textContent || '').trim().toLowerCase().startsWith('актив-заказы');

});

if (!row) return { L: 0, R: 0, row: null, valEl: null };

const valEl = row.querySelector('.value');

const txt = (valEl?.textContent || '').trim();

const m = /^(\d+)\s\*\/\s\*(\d+)/.exec(txt);

const L = m ? parseInt(m[1], 10) : 0;

const R = m ? parseInt(m[2], 10) : 0;

return { L, R, row, valEl };

}

setActivePV(cardEl, L, R)

/\*\*

\* Устанавливает значения Актив-PV

\* @param {HTMLElement} cardEl - Элемент карточки

\* @param {number} L - Левое значение

\* @param {number} R - Правое значение

\*/

function setActivePV(cardEl, L, R) {

const { valEl } = parseActivePV(cardEl);

if (!valEl) return;

renderSplitValue(valEl, L, R);

const cardData = findCardByElement(cardEl);

const cardId = cardData?.id;

if (!cardId) return;

const prev = lastActivePvValues.get(cardId);

const highlightLeft = prev ? L > prev.L : L > 0;

const highlightRight = prev ? R > prev.R : R > 0;

const timing = getHighlightTimingConfig();

const highlightOptions = {

autoRemoveDuration: timing.autoRemoveDuration,

animationDuration: timing.cssDuration

};

if (highlightLeft) {

setPartHighlight(valEl, 'L', true, highlightOptions);

} else {

setPartHighlight(valEl, 'L', false);

}

if (highlightRight) {

setPartHighlight(valEl, 'R', true, highlightOptions);

} else {

setPartHighlight(valEl, 'R', false);

}

lastActivePvValues.set(cardId, { L, R });

}

propagateActivePvUp(cardEl, side, amount)

/\*\*

\* Распространяет Актив-PV вверх по иерархии

\* @param {HTMLElement} cardEl - Элемент карточки

\* @param {string} side - Сторона ('L' или 'R')

\* @param {number} amount - Количество для добавления

\*/

function propagateActivePvUp(cardEl, side, amount) {

if (!amount) return;

const BASE = ACTIVE\_PV\_BASE;

let curEl = cardEl;

let curSide = side;

let carry = amount;

while (curEl) {

ensureActiveControls(curEl);

const apv = parseActivePV(curEl);

let L = apv.L, R = apv.R;

const prev = (curSide === 'L') ? L : R;

const curCard = findCardByElement(curEl);

const parentInfo = curCard ? getParentInfo(curCard.id) : null;

const hidden = curEl.querySelector('.active-pv-hidden');

const localKey = curSide === 'L' ? 'locall' : 'localr';

let storedUnits = hidden ? parseInt(hidden.dataset[localKey] || '0', 10) : 0;

if (!Number.isFinite(storedUnits)) storedUnits = 0;

if (storedUnits < 0) storedUnits = 0;

const beforeRem = prev;

const beforeUnits = storedUnits;

const beforeTotal = beforeRem + beforeUnits \* BASE;

let delta = carry;

if (delta < 0) {

const minDelta = -beforeRem;

if (delta < minDelta) delta = minDelta;

}

let remTotal = beforeRem + delta;

if (remTotal < 0) remTotal = 0;

let newUnits = beforeUnits;

if (remTotal >= BASE) {

const extraUnits = Math.floor(remTotal / BASE);

newUnits += extraUnits;

remTotal -= extraUnits \* BASE;

}

const afterTotal = remTotal + newUnits \* BASE;

const applied = afterTotal - beforeTotal;

if (curSide === 'L') L = remTotal;

else R = remTotal;

setActivePV(curEl, L, R);

if (hidden) {

hidden.dataset[localKey] = String(newUnits);

}

// Подсветка баланса при добавлении единиц

if (newUnits > beforeUnits) {

const balanceValueEl = getBalanceValueElement(curEl);

if (balanceValueEl) {

const timing = getHighlightTimingConfig();

setPartHighlight(balanceValueEl, curSide === 'L' ? 'L' : 'R', true, {

autoRemoveDuration: timing.autoRemoveDuration,

animationDuration: timing.cssDuration

});

}

}

// Подсветка линии

if (applied > 0 && parentInfo && curCard?.id) {

highlightLineBetween(parentInfo.parentId, curCard.id, parentInfo.side, 'pv');

}

if (!parentInfo || applied === 0) break;

const parentEl = findCardElementById(parentInfo.parentId);

if (!parentEl) break;

curEl = parentEl;

curSide = parentInfo.side;

carry = applied;

}

}

Обработчик кнопок Актив-PV:

canvas.addEventListener('click', (e) => {

const btn = e.target.closest('.active-btn');

if (!btn) return;

const cardEl = btn.closest('.card');

if (!cardEl) return;

ensureActiveControls(cardEl);

// Кнопка очистки

if (btn.classList.contains('active-clear')) {

setActivePV(cardEl, 0, 0);

const hidden = cardEl.querySelector('.active-pv-hidden');

if (hidden) {

hidden.dataset.btnL = '0';

hidden.dataset.btnR = '0';

hidden.dataset.locall = '0';

hidden.dataset.localr = '0';

}

saveState();

return;

}

const dir = btn.dataset.dir;

const rawStep = parseInt(btn.dataset.step, 10);

if (!dir || Number.isNaN(rawStep)) return;

const apv = parseActivePV(cardEl);

const hidden = cardEl.querySelector('.active-pv-hidden');

let step = rawStep;

// Ограничение отрицательного шага

if (step < 0) {

const current = dir === 'L' ? apv.L : apv.R;

const totalAvailable = Math.max(0, current);

if (-step > totalAvailable) step = -totalAvailable;

}

if (step === 0) return;

// Сохранение нажатий кнопок

if (hidden) {

if (dir === 'L') {

hidden.dataset.btnL = String((parseInt(hidden.dataset.btnL || '0', 10) + step));

} else {

hidden.dataset.btnR = String((parseInt(hidden.dataset.btnR || '0', 10) + step));

}

}

propagateActivePvUp(cardEl, dir, step);

saveState();

});

7.5.10 БЛОК: Экспорт

Экспорт в JSON:

if (saveProjectBtn) {

saveProjectBtn.addEventListener('click', () => {

const data = serializeState();

const blob = new Blob([JSON.stringify(data, null, 2)], { type: 'application/json' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = `project-${Date.now()}.json`;

a.click();

URL.revokeObjectURL(url);

});

}

Загрузка из JSON:

if (loadProjectBtn && loadProjectInput) {

loadProjectBtn.addEventListener('click', () => loadProjectInput.click());

loadProjectInput.addEventListener('change', async (e) => {

const file = e.target.files && e.target.files[0];

if (!file) return;

try {

const text = await file.text();

// Проверка на HTML-файл

const isHtml = /^\s\*<!doctype html|<html[\s>]/i.test(text);

if (isHtml) throw new Error('html-file');

const state = JSON.parse(text);

// Проверка структуры

const ok = state && typeof state === 'object' &&

Array.isArray(state.cards) && Array.isArray(state.lines);

if (!ok) throw new Error('bad-structure');

loadState(state, true);

} catch (err) {

console.error('Не удалось загрузить проект:', err);

if (String(err.message) === 'html-file') {

alert('Вы выбрали HTML-файл из «Экспорт HTML». Для продолжения выберите JSON, сохранённый кнопкой «💾 Сохранить проект».');

} else if (String(err.message) === 'bad-structure') {

alert('Файл прочитан, но структура не похожа на проект (нет полей cards/lines). Проверьте, что это JSON из «💾 Сохранить проект».');

} else {

alert('Файл повреждён или имеет неверный формат JSON.');

}

} finally {

loadProjectInput.value = '';

}

});

}

Экспорт в HTML: (упрощённая версия)

if (exportHtmlBtn) {

exportHtmlBtn.addEventListener('click', async () => {

// 1. Клонирование canvas

const canvasClone = canvas.cloneNode(true);

// 2. Удаление интерактивных элементов

const selectorsToRemove = [

'.note-resize-handle',

'.note-close-btn',

'.card-controls',

'.close-btn',

'.connection-point',

'.active-pv-controls'

];

canvasClone.querySelectorAll(selectorsToRemove.join(', ')).forEach(el => el.remove());

// 3. Отключение редактирования

canvasClone.querySelectorAll('[contenteditable]').forEach(el => {

el.setAttribute('contenteditable', 'false');

el.style.pointerEvents = 'none';

});

// 4. Встраивание изображений в Data URI

await inlineImages(canvasClone);

// 5. Генерация HTML

const htmlContent = `<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Просмотр Схемы</title>

<style>${cssText}</style>

</head>

<body style="background:${bodyStyle.background};">

${canvasClone.outerHTML}

${viewOnlyScript}

</body>

</html>`;

// 6. Скачивание

const blob = new Blob([htmlContent], { type: 'text/html' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = `scheme-${Date.now()}.html`;

a.click();

URL.revokeObjectURL(url);

});

}

Экспорт в SVG: (детальная версия в полном коде)

Экспорт в PDF/PNG через prepareForPrint(): (см. раздел 7.5.11)

7.5.11 БЛОК: Печать и экспорт в PDF/PNG

Этот блок реализует модальное окно с настройками печати и экспорт в различные форматы.

Константы печати:

const PAPER\_SIZES = {

a4: { width: 210, height: 297 },

a3: { width: 297, height: 420 },

a2: { width: 420, height: 594 },

a1: { width: 594, height: 841 },

};

const ORIGINAL\_FORMAT\_KEY = 'original';

const DEFAULT\_DPI = 96;

const MAX\_CANVAS\_DIMENSION = 16384;

const MAX\_EXPORT\_DIMENSION = 16384;

const PNG\_PPM\_FACTOR = 39.37007874015748;

Функции модуля:

* prepareForPrint() — главная функция экспорта
* createPrintModal() — создание модального окна
* canvasToPngWithDpi() — встраивание DPI в PNG
* injectDpiIntoPngBytes() — модификация PNG на уровне байтов

Полный код экспорта см. в script.js строки 2830–3400.

1. Система стилей (style.css)

8.1 Структура стилей

1. CSS-переменные (:root)
2. Базовые стили (html, body)
3. UI-панели
4. Кнопки и элементы управления
5. Canvas и SVG
6. Карточки
7. Линии и анимации
8. Заметки
9. Модальные окна
10. Направляющие
11. Печать

8.2 CSS-переменные

:root {

--card-width: 380px;

--brand: #0f62fe;

--ink: #111827;

--muted: #6b7280;

--panel: #ffffff;

--surface: #ffffff;

--bg: #f5f7fb;

--radius: 14px;

--shadow: 0 8px 20px rgba(0,0,0,.12);

}

8.3 Анимации

Подсветка баланса:

@keyframes balancePulse {

0% { transform: scale(1.8); }

50% { transform: scale(2.1); }

100% { transform: scale(1.8); }

}

.value-part.balance-highlight {

color: #d93025;

font-weight: 900;

animation: balancePulse var(--balance-animation-duration, 1.6s) ease-in-out infinite alternate;

transform-origin: center;

text-shadow: 0 4px 12px rgba(217,48,37,.35);

}

Подсветка линий:

.line--balance-highlight {

--line-highlight-color: rgba(217,48,37,.55);

stroke-dasharray: 16;

animation-name: lineBalanceFlow;

animation-duration: var(--line-animation-duration, 1.6s);

animation-iteration-count: infinite;

animation-timing-function: ease-in-out;

filter: drop-shadow(0 0 10px var(--line-highlight-color));

}

@keyframes lineBalanceFlow {

0% {

stroke-dashoffset: -24;

stroke: var(--line-color, currentColor);

}

50% {

stroke: #d93025;

}

100% {

stroke-dashoffset: 24;

stroke: var(--line-color, currentColor);

}

}

9. HTML-структура (index.html)

9.1 Основные разделы

<!DOCTYPE html>

<html lang="ru">

<head>

<!-- Метаданные -->

<!-- Подключение скриптов и стилей -->

</head>

<body>

<!-- Левая панель управления -->

<div class="ui-panel-left">...</div>

<!-- Правая панель настроек -->

<div class="ui-panel-right">...</div>

<!-- Рабочая область (canvas) -->

<div id="canvas">

<svg id="svg-layer">...</svg>

</div>

<!-- Модальное окно аутентификации -->

<div id="auth-modal">...</div>

<!-- Внешние библиотеки -->

<script src="html2canvas.min.js"></script>

<script src="jspdf.umd.min.js"></script>

<!-- Модули приложения -->

<script src="auth.js"></script>

<script src="engine.js"></script>

<script src="card-features.js"></script>

<script src="script.js"></script>

</body>

</html>

1. Форматы данных

10.1 Формат сохранения проекта (JSON)

{

"cards": [

{

"id": "card\_1234567890\_123",

"x": 140,

"y": 70,

"width": "380px" | null,

"title": "RUY1234567890",

"bodyHTML": "<div class=\"card-row\">...</div>",

"isDarkMode": false,

"bodyClass": "",

"headerBg": "rgb(93, 139, 244)",

"colorIndex": 0,

"note": {

"text": "",

"entries": {

"2025-01-15": {

"text": "Заметка за 15 января",

"updatedAt": "2025-01-15T10:30:00.000Z"

}

},

"colors": {

"2025-01-15": "#f44336"

},

"selectedDate": "2025-01-15",

"highlightColor": "#f44336",

"width": 260,

"height": 380,

"visible": false,

"window": null,

"x": 600,

"y": 100

},

"badges": {

"fendou": true,

"slf": false,

"rank": "emerald"

}

}

],

"lines": [

{

"startId": "card\_1234567890\_123",

"startSide": "left",

"endId": "card\_9876543210\_456",

"endSide": "top",

"color": "#0f62fe",

"thickness": 5

}

]

}

10.2 Структура состояния canvas

canvasState = {

x: 0, // Смещение по X (px)

y: 0, // Смещение по Y (px)

scale: 1, // Масштаб (0.1 - 3.0)

isPanning: false, // Флаг панорамирования

lastPointerX: 0, // Последняя X-координата указателя

lastPointerY: 0, // Последняя Y-координата указателя

panPointerId: null,

panCaptureTarget: null

}

10.3 Структура объекта карточки (в памяти)

cardData = {

id: string, // Уникальный ID

element: HTMLElement, // DOM-элемент

locked: boolean, // Блокировка перемещения

note: { // Заметка (или null)

text: string,

entries: Object,

colors: Object,

selectedDate: string,

highlightColor: string,

width: number,

height: number,

visible: boolean,

window: HTMLElement | null,

x: number,

y: number

},

badges: { // Значки

fendou: boolean,

slf: boolean,

rank: string | null

}

}

10.4 Структура объекта линии

lineData = {

id: string, // Уникальный ID

startCard: Object, // Ссылка на cardData начала

startSide: string, // 'top' | 'right' | 'bottom' | 'left'

endCard: Object, // Ссылка на cardData конца

endSide: string, // 'top' | 'right' | 'bottom' | 'left'

color: string, // HEX-цвет

thickness: number, // Толщина (1-20)

element: SVGPathElement // SVG path элемент

}

1. Алгоритмы и бизнес-логика

11.1 Алгоритм расчёта баллов

Входные данные:

* Карточки с координатами (x, y)
* Линии между карточками с указанием сторон
* PV каждой карточки (парсится из bodyHTML)

Шаги:

1. Построение индекса карточек по ID
2. Определение иерархии:
   * Родитель = карточка с меньшим Y
   * Сторона родителя определяет ногу (left/right)
3. Парсинг PV из HTML (формат X/330pv)
4. Определение "полных" карточек (≥330 PV)
5. Подъём 1 балла от каждой полной карточки вверх по иерархии
6. Расчёт этапов и циклов для каждой карточки

Формулы:

Цикл = Math.floor(totalBalls / 72)

Остаток = totalBalls % 72

Этапы:

0–5 → Этап 0 (до 6)

6–17 → Этап 1 (до 12)

18–35 → Этап 2 (до 18)

36–71 → Этап 3 (до 36)

72+ → Новый цикл

11.2 Алгоритм Актив-PV

Механизм:

1. Пользователь нажимает кнопки ±1, ±10 на карточке
2. Значение добавляется к текущему Актив-PV карточки
3. При достижении 330 PV:
   * Создаётся 1 «единица» (unit)
   * Остаток сбрасывается
   * Единица добавляется к балансу текущей карточки
4. Единица распространяется вверх к родителю
5. Процесс повторяется до корня дерева

Структура данных:

hidden.dataset = {

btnL: "15", // Нажатия кнопок слева

btnR: "20", // Нажатия кнопок справа

abonusl: "0", // Бонусы слева (устаревшее)

abonusr: "0", // Бонусы справа (устаревшее)

locall: "2", // Единицы слева (330 PV = 1 единица)

localr: "3" // Единицы справа

}

11.3 Алгоритм привязки к сетке

// Привязка к сетке 70px

const GRID\_SIZE = 70;

const snappedX = Math.round(x / GRID\_SIZE) \* GRID\_SIZE;

const snappedY = Math.round(y / GRID\_SIZE) \* GRID\_SIZE;

// Привязка к направляющим (при перемещении)

const SNAP\_TOLERANCE = 5; // px

if (Math.abs(draggedBounds.left - staticCard.left) < SNAP\_TOLERANCE) {

snapX = staticCard.left;

dx\_canvas = staticCard.left - draggedBounds.left;

}

11.4 Алгоритм иерархического перемещения

Режимы перемещения:

1. Обычный — перемещается только выделенная карточка
2. Выделение — перемещаются все выделённые карточки
3. Иерархический — перемещается ветка иерархии

Определение ветки:

// Клик по заголовку → всё дерево

if (target.closest('.card-header')) {

dragMode = 'all';

}

// Клик по левой половине body → левая ветка

else if (target.closest('.card-body')) {

const bodyRect = target.closest('.card-body').getBoundingClientRect();

dragMode = (e.clientX - bodyRect.left < bodyRect.width / 2) ? 'left' : 'right';

}

Алгоритм обхода:

// BFS (Breadth-First Search)

function getBranchDescendants(startCard, branchFilter) {

const descendants = new Set();

const queue = [startCard];

const visited = new Set([startCard.id]);

let head = 0;

let isInitialCard = true;

while(head < queue.length) {

const currentCard = queue[head++];

const childLines = lines.filter(line => line.startCard.id === currentCard.id);

for (const line of childLines) {

// Для стартовой карточки применяем фильтр

if (isInitialCard) {

const isBranchMatch =

(branchFilter === 'all' && ['left', 'right', 'bottom'].includes(line.startSide)) ||

(line.startSide === branchFilter);

if (!isBranchMatch) continue;

}

const childCard = line.endCard;

if (!visited.has(childCard.id)) {

visited.add(childCard.id);

descendants.add(childCard);

queue.push(childCard);

}

}

isInitialCard = false;

}

return descendants;

}

11.5 Алгоритм мультитач (Pinch-to-Zoom)

// Отслеживание активных указателей

const activePointers = new Map();

// Определение жеста pinch

function tryStartPinch() {

const touches = Array.from(activePointers.entries())

.filter(([, info]) => info.type === 'touch');

if (touches.length < 2) return;

const [first, second] = touches;

const distance = Math.hypot(

second[1].x - first[1].x,

second[1].y - first[1].y

);

pinchState = {

id1: first[0],

id2: second[0],

initialDistance: distance,

initialScale: canvasState.scale,

prevMidX: (first[1].x + second[1].x) / 2,

prevMidY: (first[1].y + second[1].y) / 2

};

}

// Обработка движения

function handlePinchMove() {

const first = activePointers.get(pinchState.id1);

const second = activePointers.get(pinchState.id2);

// Панорамирование по средней точке

const midX = (first.x + second.x) / 2;

const midY = (first.y + second.y) / 2;

canvasState.x += midX - pinchState.prevMidX;

canvasState.y += midY - pinchState.prevMidY;

// Масштабирование по расстоянию

const distance = Math.hypot(second.x - first.x, second.y - first.y);

const newScale = Math.max(0.1, Math.min(3,

pinchState.initialScale \* (distance / pinchState.initialDistance)

));

const ratio = newScale / canvasState.scale;

canvasState.x = midX - (midX - canvasState.x) \* ratio;

canvasState.y = midY - (midY - canvasState.y) \* ratio;

canvasState.scale = newScale;

pinchState.prevMidX = midX;

pinchState.prevMidY = midY;

updateCanvasTransform();

}

12. Взаимодействие компонентов

12.1 Диаграмма потока данных

Действие пользователя

↓

Обработчик события (script.js)

↓

Изменение состояния (cards/lines)

↓

saveState()

↓

┌─────────────────┬─────────────────┐

│ serializeState │ recalculateAndRender

│ │ ↓

│ │ Engine.recalc

│ │ ↓

│ │ Обновление DOM

└─────────────────┴─────────────────┘

↓

undoStack.push(snapshot)

12.2 Цикл обновления

// 1. Пользователь создаёт карточку

createCard()

→ cards.push(cardData)

→ saveState()

// 2. Сохранение состояния

saveState()

→ serializeState()

→ JSON.stringify()

→ undoStack.push()

→ recalculateAndRender()

// 3. Пересчёт

recalculateAndRender()

→ Engine.recalc(state)

→ Обновление значений в DOM

→ Применение подсветки

12.3 Взаимодействие модулей

auth.js

↓ (блокирует доступ до аутентификации)

index.html

↓ (загружает скрипты)

engine.js → глобальный объект Engine

↓

card-features.js → глобальные функции

↓

script.js

├─ использует Engine.recalc()

├─ использует initializeCardFeatures()

└─ управляет всем состоянием

12.4 События и их обработчики

Глобальные события (window):

* pointerdown → начало взаимодействия (пан, выделение, рисование)
* pointermove → перемещение (пан, drag, рисование линии)
* pointerup/pointercancel → завершение действия
* wheel → масштабирование
* keydown → горячие клавиши (Esc, Delete, Ctrl+Z, Ctrl+C/V)

События карточек:

* pointerdown на .card → начало перемещения или выделение с Ctrl
* dblclick на .card-title → редактирование заголовка
* click на .coin-icon → переключение цвета монетки
* click на .card-close-btn → удаление карточки
* click на .note-btn → открытие/закрытие заметки
* pointerdown на .connection-point → начало/завершение рисования линии
* contextmenu на .card-header → открытие меню значков

События линий:

* click на .line → выделение линии

События заметок:

* click на .clr-dot → выбор цвета
* click на .cal-cell → выбор даты
* input на .note-textarea → редактирование текста
* pointerdown на .note-header → перемещение окна

13. Рекомендации по портированию

13.1 Ключевые моменты для разработчика

1. Архитектура MVC:

\* Model: cards[], lines[], state

\* View: DOM-элементы, SVG

\* Controller: обработчики событий в script.js

2. Разделение ответственности:

\* engine.js — чистая математика (без DOM)

\* auth.js — независимый модуль безопасности

\* card-features.js — дополнительный функционал

\* script.js — основная логика и склейка

3. Состояние приложения:

\* Централизованное хранение в переменных модуля

\* История через Undo/Redo стеки

\* Сериализация/десериализация через JSON

4. Паттерны:

\* Observer (через recalculateAndRender)

\* Command (через saveState)

\* Factory (через createCard)

\* Strategy (режимы перемещения)

13.2 Приоритеты при реализации

**Этап 1: Базовый функционал**

1. Создание/удаление карточек
2. Перемещение карточек
3. Создание/удаление линий
4. Базовый расчёт баллов (engine.js)
5. Сохранение/загрузка JSON

**Этап 2: Расширенный функционал** 6. История (Undo/Redo) 7. Выделение и копирование 8. Панорамирование и масштабирование 9. Привязка к сетке и направляющие 10. Экспорт в HTML/SVG

**Этап 3: Продвинутые функции** 11. Система заметок с календарём 12. Значки и контекстное меню 13. Иерархическое перемещение 14. Актив-PV с распространением 15. Экспорт в PDF/PNG с настройками

**Этап 4: Полировка** 16. Мультитач и жесты 17. Анимации и подсветка 18. Аутентификация 19. Адаптивность и производительность 20. Тестирование и отладка

13.3 Критические алгоритмы для тестирования

1. Engine.recalc() — проверить на различных конфигурациях деревьев
2. getBranchDescendants() — проверить на циклических ссылках
3. propagateActivePvUp() — проверить переполнение и распространение
4. pinch-to-zoom — проверить на различных устройствах
5. saveState/loadState — проверить целостность данных

13.4 Потенциальные проблемы

1. **Производительность**: При большом количестве карточек (>100) может потребоваться виртуализация
2. **Память**: История ограничена 50 шагами, но большие проекты могут занимать много RAM
3. **Браузерная совместимость**: Pointer Events могут работать по-разному в разных браузерах
4. **Canvas limits**: MAX\_CANVAS\_DIMENSION = 16384 — лимит браузера
5. **PNG DPI**: Встраивание DPI работает через бинарные операции с PNG
6. Заключение

Данная документация описывает полную структуру проекта «Интерактивная доска» для визуализации иерархии лицензий.

Ключевые характеристики проекта:

* Язык: JavaScript (ES6+), HTML5, CSS3
* Архитектура: Модульная (4 JS-файла)
* Библиотеки: html2canvas, jsPDF
* Размер кодовой базы: ~4000 строк JavaScript
* Форматы экспорта: JSON, HTML, SVG, PNG, PDF
* Поддержка устройств: Desktop + Touch (планшеты, телефоны)

Для успешного портирования необходимо:

1. Сохранить модульную структуру
2. Реализовать Engine.recalc() идентично
3. Обеспечить работу Pointer Events или их аналога
4. Реализовать историю операций
5. Поддержать форматы экспорта

Разработчик должен иметь опыт работы с:

* Canvas/SVG рендерингом
* Событийной моделью (drag-and-drop, multi-touch)
* Алгоритмами обхода графов (BFS)
* Сериализацией данных
* Генерацией PDF/изображений