Based on my search, here are the files associated with each of the dashboard views and tabs you asked about:

Front Week View (Weekly Calendar)

Main Component: client/src/components/calendar/WeeklyCalendarGrid.tsx

Header: client/src/components/common/Header.tsx

Export Functions:

client/src/utils/currentWeeklyExport.ts

client/src/utils/perfectDashboardExport.ts

Styling: src/index.css (weekly calendar grid styles)

Daily View

Main Component: client/src/components/calendar/DailyView.tsx

Export Functions:

client/src/utils/browserReplicaPDF.ts (main daily export)

client/src/utils/completePDFExport.ts

client/src/components/PerfectDailyReplicaPDF.tsx

client/src/components/ApplicationReplicaPDF.tsx

Styling: src/index.css (daily view appointment layout, time display, etc.)

Appointment Tab

Main Components:

client/src/components/calendar/AppointmentStatusView.tsx (displays appointment list and stats)

client/src/components/calendar/AppointmentStats.tsx (within the same file - shows statistics)

Modal: AppointmentStatusModal component (in the same file)

Export Tab/Dropdown

Main Component: client/src/components/sidebar/ExportToPDF.tsx

Export Handler: In client/src/pages/planner.tsx (handleExportPDF function)

Various Export Utilities:

client/src/utils/bidirectionalLinkedPDFExport.ts

client/src/utils/unifiedBidirectionalExport.ts

client/src/utils/combineExactTemplates.ts

client/src/utils/htmlTemplateDailyPDF.ts

Location Marking, Event Notes, and Action Items

These features are integrated into the daily view and export components:

Display: In DailyView.tsx - renders event notes and action items in the appointment layout

Styling: In index.css - styles for .appointment-notes, .appointment-actions, etc.

Export: In browserReplicaPDF.ts - handles rendering of event notes and action items in PDF exports

Data Model: Event notes and action items are part of the CalendarEvent type (would be in shared/schema.ts)

The main entry point for all these views is client/src/pages/planner.tsx, which contains the tab navigation and coordinates between all these components.

import { useState } from 'react';

import { generateTimeSlots, getEventDurationInSlots, isEventInTimeSlot } from '../../utils/timeSlots';

import { formatDateShort } from '../../utils/dateUtils';

import { cleanEventTitle } from '../../utils/textCleaner';

import { wrapText } from '../../utils/textWrappers';

import { CalendarEvent, CalendarDay } from '../../types/calendar';

import { cn } from '@/lib/utils';

import { getLocationDisplay } from '../../utils/locationUtils';

interface WeeklyCalendarGridProps {

week: CalendarDay[];

events: CalendarEvent[];

onDayClick: (date: Date) => void;

onTimeSlotClick: (date: Date, time: string) => void;

onEventClick: (event: CalendarEvent) => void;

onEventMove?: (eventId: string, newStartTime: Date, newEndTime: Date) => void;

}

export const WeeklyCalendarGrid = ({

week,

events,

onDayClick,

onTimeSlotClick,

onEventClick,

onEventMove

}: WeeklyCalendarGridProps) => {

const timeSlots = generateTimeSlots();

const [draggedEventId, setDraggedEventId] = useState<string | null>(null);

const [dropZone, setDropZone] = useState<{date: Date, time: string} | null>(null);

const handleDragStart = (e: React.DragEvent, event: CalendarEvent) => {

// Ensure dates are properly parsed

const startTime = event.startTime instanceof Date ? event.startTime : new Date(event.startTime);

const endTime = event.endTime instanceof Date ? event.endTime : new Date(event.endTime);

// Skip invalid dates

if (isNaN(startTime.getTime()) || isNaN(endTime.getTime())) {

return;

}

// Set visual state

setDraggedEventId(event.id);

// Set drag data

e.dataTransfer.setData('text/plain', JSON.stringify({

eventId: event.id,

originalStartTime: startTime.toISOString(),

originalEndTime: endTime.toISOString(),

duration: endTime.getTime() - startTime.getTime()

}));

// Set drag effect

e.dataTransfer.effectAllowed = 'move';

// Add some visual feedback

e.dataTransfer.setDragImage(e.currentTarget as HTMLElement, 10, 10);

};

const handleDragOver = (e: React.DragEvent, date: Date, timeSlot: { hour: number; minute: number }) => {

e.preventDefault();

e.dataTransfer.dropEffect = 'move';

// Set drop zone for visual feedback

setDropZone({ date, time: `${timeSlot.hour.toString().padStart(2, '0')}:${timeSlot.minute.toString().padStart(2, '0')}` });

};

const handleDragLeave = (e: React.DragEvent) => {

// Clear drop zone when leaving

setDropZone(null);

};

const handleDragEnd = (e: React.DragEvent) => {

// Clear drag state

setDraggedEventId(null);

setDropZone(null);

};

const handleDrop = (e: React.DragEvent, date: Date, timeSlot: { hour: number; minute: number }) => {

e.preventDefault();

if (!onEventMove) return;

try {

const dragData = JSON.parse(e.dataTransfer.getData('text/plain'));

const newStartTime = new Date(date);

newStartTime.setHours(timeSlot.hour, timeSlot.minute, 0, 0);

const newEndTime = new Date(newStartTime.getTime() + dragData.duration);

onEventMove(dragData.eventId, newStartTime, newEndTime);

} catch (error) {

console.error('Error handling drop:', error);

} finally {

// Clear drag state

setDraggedEventId(null);

setDropZone(null);

}

};

const getAllDayEventsForDate = (date: Date) => {

return events.filter(event => {

// Ensure dates are properly parsed

const startTime = event.startTime instanceof Date ? event.startTime : new Date(event.startTime);

const endTime = event.endTime instanceof Date ? event.endTime : new Date(event.endTime);

// Skip invalid dates

if (isNaN(startTime.getTime()) || isNaN(endTime.getTime())) {

return false;

}

const eventDate = startTime;

// Check if backend marked it as all-day

const isMarkedAllDay = (event as any).isAllDay;

};

import { Badge } from '@/components/ui/badge';

import { Button } from '@/components/ui/button';

import { ChevronLeft, ChevronRight } from 'lucide-react';

interface HeaderProps {

weekRangeString: string;

isOnline: boolean;

isCurrentWeek: boolean;

onConnectGoogle: () => void;

onPreviousWeek: () => void;

onToday: () => void;

onNextWeek: () => void;

}

export const Header = ({

weekRangeString,

isOnline,

isCurrentWeek,

onConnectGoogle,

onPreviousWeek,

onToday,

onNextWeek

}: HeaderProps) => {

return (

<div className="space-y-6">

<div className="flex justify-between items-center">

<div>

<h1 className="text-xl font-bold text-gray-900">{weekRangeString}</h1>

<div className="flex items-center space-x-3 mt-1">

<Badge variant={isOnline ? "default" : "secondary"} className="bg-green-100 text-green-800">

<div className="w-1.5 h-1.5 bg-green-400 rounded-full mr-1"></div>

{isOnline ? "Online" : "Offline"}

</Badge>

<span className="text-sm text-gray-500">

Weekly Overview • Click any day to view details

</span>

</div>

</div>

<Button onClick={onConnectGoogle} className="bg-blue-600 hover:bg-blue-700">

Connect Google Calendar

</Button>

</div>

<div className="flex justify-center">

<div className="flex space-x-3">

<Button

variant="outline"

onClick={onPreviousWeek}

className="flex items-center px-4 py-2 bg-gray-50 border-gray-300 hover:bg-gray-100 hover:border-gray-400 transition-colors"

>

<ChevronLeft className="w-4 h-4 mr-2" />

Previous Week

</Button>

<Button

variant={isCurrentWeek ? "default" : "outline"}

onClick={onToday}

className={isCurrentWeek

? "bg-blue-600 hover:bg-blue-700 text-white px-6 py-2 font-medium"

: "px-6 py-2 bg-gray-50 border-gray-300 hover:bg-gray-100 hover:border-gray-400 transition-colors"

}

>

Today

</Button>

<Button

variant="outline"

onClick={onNextWeek}

className="flex items-center px-4 py-2 bg-gray-50 border-gray-300 hover:bg-gray-100 hover:border-gray-400 transition-colors"

>

Next Week

<ChevronRight className="w-4 h-4 ml-2" />

</Button>

</div>

</div>

</div>

);

};

import jsPDF from 'jspdf';

import { CalendarEvent } from '../types/calendar';

// Import comprehensive emoji cleaning function

import { cleanTitleForPDF } from './emojiCleaner';

// Clean event title utility function

function cleanEventTitle(title: string): string {

return cleanTitleForPDF(title);

}

export interface CurrentWeeklyExportConfig {

pageWidth: number;

pageHeight: number;

margins: number;

headerHeight: number;

timeColumnWidth: number;

dayColumnWidth: number;

timeSlotHeight: number;

fonts: {

title: number;

weekInfo: number;

dayHeader: number;

timeLabel: number;

eventTitle: number;

eventSource: number;

eventTime: number;

};

}

// Optimized configuration for complete time range display

export const CURRENT\_WEEKLY\_CONFIG: CurrentWeeklyExportConfig = {

pageWidth: 792, // 11" landscape

pageHeight: 612, // 8.5" landscape

margins: 16, // Perfect centering

headerHeight: 40,

timeColumnWidth: 60,

dayColumnWidth: 100, // Clean 100px for 7 days = 700px total

timeSlotHeight: 13, // Slightly reduced to fit all time slots

fonts: {

title: 16,

weekInfo: 12,

dayHeader: 9,

timeLabel: 7,

eventTitle: 5, // Small but readable

eventSource: 4, // Very small for source/location

eventTime: 4, // Very small for time

},

};

export const exportCurrentWeeklyView = (

events: CalendarEvent[],

weekStart: Date,

weekEnd: Date

): void => {

// Normalize week start to beginning of Monday

const normalizedWeekStart = new Date(weekStart);

normalizedWeekStart.setHours(0, 0, 0, 0);

// Ensure week end covers full Sunday

const normalizedWeekEnd = new Date(weekEnd);

normalizedWeekEnd.setHours(23, 59, 59, 999);

console.log(`📊 Exporting weekly view: ${normalizedWeekStart.toDateString()} to ${normalizedWeekEnd.toDateString()}`);

console.log(`📊 Total events available: ${events.length}`);

const pdf = new jsPDF({

orientation: 'landscape',

unit: 'pt',

format: [CURRENT\_WEEKLY\_CONFIG.pageWidth, CURRENT\_WEEKLY\_CONFIG.pageHeight]

});

pdf.setFont('helvetica');

// Draw header

drawCurrentWeeklyHeader(pdf, normalizedWeekStart, normalizedWeekEnd);

// Draw grid and events

drawCurrentWeeklyGrid(pdf, events, normalizedWeekStart);

// Debug Monday events before saving

const mondayEvents = events.filter(event => {

const eventDate = new Date(event.startTime);

return eventDate.getDay() === 1 && // Monday

eventDate >= normalizedWeekStart &&

eventDate <= normalizedWeekEnd;

});

console.log(`🔍 Monday events found: ${mondayEvents.length}`);

mondayEvents.forEach(event => {

const eventDate = new Date(event.startTime);

console.log(` - "${event.title}" at ${eventDate.toLocaleString()}`);

});

// Save the PDF with dynamic filename

const weekStartStr = normalizedWeekStart.toLocaleDateString('en-US', {

month: 'short',

day: 'numeric',

year: 'numeric'

});

const weekEndStr = normalizedWeekEnd.toLocaleDateString('en-US', {

import jsPDF from 'jspdf';

**Error! Filename not specified.**import { CalendarEvent } from '../types/calendar';

import { cleanEventTitle } from './titleCleaner';

/\*\*

\* Perfect Dashboard Export System - Exact Screenshot Replication

\*

\* This system creates pixel-perfect PDF exports that exactly match the dashboard screenshots.

\* Based on analyzing the PERFECT weekly and daily screenshots provided by the user.

\*/

// Configuration for PERFECT WEEKLY view (based on the perfect weekly screenshot)

const PERFECT\_WEEKLY\_CONFIG = {

pageWidth: 1190, // A3 landscape

pageHeight: 842, // A3 landscape

// Header configuration

headerHeight: 120,

titleFontSize: 18,

subtitleFontSize: 14,

// Statistics section

statsHeight: 50,

statsFontSize: 11,

statsValueFontSize: 16,

// Legend section

legendHeight: 30,

legendFontSize: 10,

// Grid configuration (matching the perfect screenshot exactly)

margin: 20,

timeColumnWidth: 80,

dayColumnWidth: 155, // (1190 - 40 - 80) / 7 = 152.8 ≈ 155

rowHeight: 18,

// Colors based on perfect screenshot analysis

colors: {

// Header colors

headerBg: '#ffffff',

headerText: '#000000',

// Grid colors

gridLine: '#000000',

gridBorder: '#000000',

timeColumnBg: '#ffffff',

dayHeaderBg: '#ffffff',

// Event colors (from perfect screenshot)

simplePractice: '#d4e3fc', // Light blue background

simplePracticeBorder: '#4285f4', // Blue border

google: '#ffffff', // White background with dashed green border

googleBorder: '#34a853', // Green border

holiday: '#fff3cd', // Light yellow background

holidayBorder: '#ffc107' // Yellow border

}

};

// Configuration for PERFECT DAILY view (based on the perfect daily screenshot)

const PERFECT\_DAILY\_CONFIG = {

pageWidth: 612, // 8.5 inches portrait

pageHeight: 792, // 11 inches portrait

// Header configuration

headerHeight: 100,

titleFontSize: 16,

subtitleFontSize: 12,

// Statistics section

statsHeight: 60,

statsFontSize: 10,

statsValueFontSize: 14,

// Legend section

legendHeight: 25,

legendFontSize: 9,

// Grid configuration (matching the perfect screenshot exactly)

margin: 15,

timeColumnWidth: 80,

appointmentColumnWidth: 500, // Remaining width for appointments

rowHeight: 24, // Larger rows for better readability

// Colors based on perfect screenshot analysis

colors: {

// Header colors

headerBg: '#ffffff',

headerText: '#000000',

// Grid colors

gridLine: '#000000',

gridBorder: '#000000',

timeColumnBg: '#ffffff',

dayHeaderBg: '#ffffff',

// Event colors (from perfect screenshot)

simplePractice: '#ffffff', // White background

simplePracticeBorder: '#4285f4', // Blue border

google: '#e3f2fd', // Light blue background

googleBorder: '#2196f3', // Blue border

holiday: '#fff3cd', // Light yellow background

holidayBorder: '#ffc107' // Yellow border

}

};

/\*\*

\* Generate complete time slots from 06:00 to 23:30

\*/

function generateTimeSlots() {

const slots = [];

for (let hour = 6; hour <= 23; hour++) {

slots.push({

hour,

minute: 0,

display: `${hour.toString().padStart(2, '0')}:00`,

}

@tailwind base;

@tailwind components;

@tailwind utilities;

/\* Accessibility styles \*/

.high-contrast {

--background: 255 255 255;

--foreground: 0 0 0;

--card: 255 255 255;

--card-foreground: 0 0 0;

--primary: 0 0 0;

--primary-foreground: 255 255 255;

--secondary: 245 245 245;

--secondary-foreground: 0 0 0;

--border: 0 0 0;

}

.reduced-motion,

.reduced-motion \*,

.reduced-motion \*:before,

.reduced-motion \*:after {

animation-duration: 0.01ms !important;

animation-iteration-count: 1 !important;

transition-duration: 0.01ms !important;

scroll-behavior: auto !important;

}

.large-text {

font-size: 1.125rem !important;

line-height: 1.75rem !important;

}

.large-text h1 { font-size: 2.5rem !important; }

.large-text h2 { font-size: 2rem !important; }

.large-text h3 { font-size: 1.75rem !important; }

.large-text button {

font-size: 1.125rem !important;

padding: 0.75rem 1.5rem !important;

}

.focus-indicators \*:focus {

outline: 3px solid **Error! Filename not specified.**#3b82f6 !important;

outline-offset: 2px !important;

}

.focus-indicators button:focus,

.focus-indicators input:focus,

.focus-indicators select:focus,

.focus-indicators textarea:focus {

box-shadow: 0 0 0 3px **Error! Filename not specified.**rgba(59, 130, 246, 0.3) !important;

}

/\* Skip link for keyboard navigation \*/

.skip-link {

position: absolute;

top: -40px;

left: 6px;

background: **Error! Filename not specified.**#000;

color: **Error! Filename not specified.**#fff;

padding: 8px;

text-decoration: none;

z-index: 1000;

border-radius: 4px;

}

.skip-link:focus {

top: 6px;

}

/\* Prevent viewport overflow \*/

html, body {

overflow-x: hidden;

height: auto;

min-height: 100vh;

}

#root {

height: auto;

min-height: 100vh;

}

/\* PROFESSIONAL PLANNER STYLING - EXACT MATCH TO HTML TEMPLATE \*/

.planner-container {

width: 100%;

max-width: 100vw;

margin: 0 auto;

background: **Error! Filename not specified.**white;

border: 2px solid **Error! Filename not specified.**black;

font-family: 'Times New Roman', serif;

overflow-x: auto;

height: fit-content;

display: flex;

flex-direction: column;

box-sizing: border-box;

}

/\* NAVIGATION BUTTON STYLING - COMPREHENSIVE IMPLEMENTATION \*/

}

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

**Error! Filename not specified.**import { ChevronLeft, ChevronRight, ArrowLeft } from 'lucide-react';

import { Button } from '@/components/ui/button';

import { Textarea } from '@/components/ui/textarea';

import { formatDate } from '../../utils/dateUtils';

import { generateTimeSlots } from '../../utils/timeSlots';

import { CalendarEvent } from '../../types/calendar';

import { getLocationDisplay } from '../../utils/locationUtils';

interface DailyViewProps {

selectedDate: Date;

events: CalendarEvent[];

dailyNotes: string;

onPreviousDay: () => void;

onNextDay: () => void;

onBackToWeek: () => void;

onEventClick: (event: CalendarEvent) => void;

onUpdateEvent: (eventId: string, updates: Partial<CalendarEvent>) => void;

onUpdateDailyNotes: (notes: string) => void;

onEventMove?: (eventId: string, newStartTime: Date, newEndTime: Date) => void;

onCreateEvent?: (startTime: Date, endTime: Date) => void;

onDeleteEvent?: (eventId: string) => void;

}

export const DailyView = ({

selectedDate,

events,

dailyNotes,

onPreviousDay,

onNextDay,

onBackToWeek,

onEventClick,

onUpdateEvent,

onUpdateDailyNotes,

onEventMove,

onCreateEvent,

onDeleteEvent

}: DailyViewProps) => {

const [currentNotes, setCurrentNotes] = useState(dailyNotes);

const [expandedEventId, setExpandedEventId] = useState<string | null>(null);

const [noteTimers, setNoteTimers] = useState<{[key: string]: NodeJS.Timeout}>({});

const [draggedEventId, setDraggedEventId] = useState<string | null>(null);

const [dragOverSlot, setDragOverSlot] = useState<number | null>(null);

// Get events for the selected date with null checks and proper date conversion

const dayEvents = events.filter(event => {

if (!event || !event.startTime || !event.endTime || !selectedDate) return false;

try {

// Convert startTime and endTime to Date objects if they aren't already

const startTime = event.startTime instanceof Date ? event.startTime : new Date(event.startTime);

const endTime = event.endTime instanceof Date ? event.endTime : new Date(event.endTime);

// Validate that dates are valid

if (!startTime || !endTime || isNaN(startTime.getTime()) || isNaN(endTime.getTime()) || isNaN(selectedDate.getTime())) {

// Invalid date detected

return false;

}

const selectedDateString = selectedDate.toDateString();

const eventDateString = startTime.toDateString();

const matches = eventDateString === selectedDateString;

// Debug specific events

if (event.title.toLowerCase().includes('calvin') || event.title.toLowerCase().includes('hill')) {

console.log(`🎯 Calvin Hill Event Debug:`, {

title: event.title,

selectedDate: selectedDateString,

eventDate: eventDateString,

matches: matches,

startTime: startTime.toISOString(),

endTime: endTime.toISOString()

});

}

return matches;

} catch (error) {

// Invalid date in event

return false;

}

});

// Debug logging for event filtering

console.log(`📊 Daily View Debug:`, {

selectedDate: selectedDate?.toDateString(),

totalEvents: events.length,

filteredEvents: dayEvents.length,

eventTitles: dayEvents.map(e => e.title)

});

// Null check for selectedDate before using

if (!selectedDate) {

// DailyView: selectedDate is undefined

return (

<div className="planner-container daily-planner">

<div className="flex items-center justify-center h-64">

<p>Loading daily view...</p>

</div>

</div>

);

}

// Daily view event filtering completed

// Calculate daily statistics with error handling

const totalEvents = dayEvents.length;

};

import html2canvas from 'html2canvas';

**Error! Filename not specified.**import jsPDF from 'jspdf';

import { CalendarEvent } from '../types/calendar';

export async function exportBrowserReplicaPDF(events: CalendarEvent[], selectedDate: Date): Promise<void> {

try {

console.log('🚀 Starting TRULY Fixed Calendar Export');

console.log('📅 Selected date:', selectedDate.toDateString());

console.log('📊 Total events:', events.length);

const dayName = selectedDate.toLocaleDateString('en-US', { weekday: 'long' });

const dateString = selectedDate.toLocaleDateString('en-US', {

year: 'numeric',

month: 'long',

day: 'numeric'

});

// Filter events for the selected date

const filteredEvents = events.filter(event => {

if (!event?.startTime || !selectedDate) return false;

try {

const eventStart = event.startTime instanceof Date ? event.startTime : new Date(event.startTime);

if (isNaN(eventStart.getTime())) return false;

return eventStart.toDateString() === selectedDate.toDateString();

} catch (error) {

console.warn('Date parsing error for event:', event.title, error);

return false;

}

});

console.log(`✅ Filtered events: ${filteredEvents.length} found`);

// Calculate real statistics

const totalAppointments = filteredEvents.length;

const scheduledHours = filteredEvents.reduce((sum, event) => {

const start = new Date(event.startTime);

const end = new Date(event.endTime);

const duration = (end.getTime() - start.getTime()) / (1000 \* 60 \* 60);

return sum + duration;

}, 0);

const workdayHours = 17.5; // 6 AM to 11:30 PM

const availableHours = Math.max(0, workdayHours - scheduledHours);

const freeTimePercentage = Math.round((availableHours / workdayHours) \* 100);

// Create container with EXACT browser structure

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

container.style.position = 'absolute';

container.style.left = '-9999px';

container.style.top = '0';

container.style.width = '1200px';

container.style.backgroundColor = '#ffffff';

container.style.fontFamily = 'Inter, -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif';

container.innerHTML = `

<style>

.planner-container {

width: 1200px;

margin: 0 auto;

background: white;

padding: 20px;

font-family: Inter, -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;

}

.nav-header {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 20px;

padding: 16px 24px;

background: white;

border: 3px solid #3b82f6;

border-radius: 8px;

}

.weekly-overview-btn {

padding: 8px 16px;

border: 1px solid #d1d5db;

border-radius: 6px;

background: #f8fafc;

font-size: 14px;

font-weight: 500;

color: #374151;

display: flex;

align-items: center;

gap: 6px;

min-width: 140px;

justify-content: center;

}

.page-title {

text-align: center;

flex: 1;

}

// Add this to your utils folder as completePDFExport.ts

import { CalendarEvent } from '../types/calendar';

interface ExportData {

date: string;

appointments: {

time: string;

title: string;

source: string;

duration: string;

notes?: string;

actionItems?: string;

}[];

totalAppointments: number;

dailyNotes?: string;

}

export const generateCompleteExportData = (

selectedDate: Date,

events: CalendarEvent[],

dailyNotes: string = ''

): ExportData => {

console.log('Generating complete export data...');

console.log('Selected date:', selectedDate);

console.log('Total events passed:', events.length);

// Filter events for the selected day

const dayEvents = events.filter(event => {

const eventDate = new Date(event.startTime);

const selectedDateStr = selectedDate.toDateString();

const eventDateStr = eventDate.toDateString();

const matches = eventDateStr === selectedDateStr;

if (matches) {

console.log(`Event matches date: ${event.title} at ${event.startTime}`);

}

return matches;

});

console.log(`Found ${dayEvents.length} events for ${selectedDate.toDateString()}`);

// Sort events by start time

dayEvents.sort((a, b) => new Date(a.startTime).getTime() - new Date(b.startTime).getTime());

// Format appointments

const appointments = dayEvents.map(event => {

const startTime = new Date(event.startTime);

const endTime = new Date(event.endTime);

const timeString = `${formatTime(startTime)}-${formatTime(endTime)}`;

const duration = calculateDuration(startTime, endTime);

return {

time: timeString,

title: event.title,

source: getSourceName(event.source),

duration: duration,

notes: event.notes || '',

actionItems: event.actionItems || ''

};

});

return {

date: selectedDate.toLocaleDateString('en-US', {

weekday: 'long',

year: 'numeric',

month: 'long',

day: 'numeric'

}),

appointments,

totalAppointments: appointments.length,

dailyNotes

};

};

const formatTime = (date: Date): string => {

return date.toLocaleTimeString('en-US', {

hour: '2-digit',

minute: '2-digit',

hour12: false

});

};

const calculateDuration = (start: Date, end: Date): string => {

const diffMs = end.getTime() - start.getTime();

const diffMins = Math.round(diffMs / (1000 \* 60));

const hours = Math.floor(diffMins / 60);

const minutes = diffMins % 60;

if (hours === 0) {

return `${minutes}min`;

} else if (minutes === 0) {

return `${hours}hr`;

} else {

return `${hours}hr ${minutes}min`;

}

};

const getSourceName = (source: string): string => {

};

import React from 'react';

import html2canvas from 'html2canvas';

import jsPDF from 'jspdf';

import { CalendarEvent } from '../types/calendar';

import { generateTimeSlots } from '../utils/timeSlots';

import { getLocationDisplay } from '../utils/locationUtils';

interface PerfectDailyReplicaProps {

selectedDate: Date;

events: CalendarEvent[];

}

// Perfect Daily Replica Component for PDF Export

const PerfectDailyReplica: React.FC<PerfectDailyReplicaProps> = ({ selectedDate, events }) => {

// Filter events for the selected date

const dayEvents = events.filter(event => {

if (!event || !event.startTime || !event.endTime || !selectedDate) return false;

try {

const startTime = event.startTime instanceof Date ? event.startTime : new Date(event.startTime);

const endTime = event.endTime instanceof Date ? event.endTime : new Date(event.endTime);

if (!startTime || !endTime || isNaN(startTime.getTime()) || isNaN(endTime.getTime()) || isNaN(selectedDate.getTime())) {

return false;

}

const selectedDateString = selectedDate.toDateString();

const eventDateString = startTime.toDateString();

return eventDateString === selectedDateString;

} catch (error) {

return false;

}

});

// Calculate daily statistics

const totalEvents = dayEvents.length;

const totalHours = dayEvents.reduce((sum, event) => {

try {

const startTime = event.startTime instanceof Date ? event.startTime : new Date(event.startTime);

const endTime = event.endTime instanceof Date ? event.endTime : new Date(event.endTime);

if (!startTime || !endTime || isNaN(startTime.getTime()) || isNaN(endTime.getTime())) return sum;

return sum + (endTime.getTime() - startTime.getTime()) / (1000 \* 60 \* 60);

} catch (error) {

return sum;

}

}, 0);

const availableHours = 24 - totalHours;

const freeTimePercentage = Math.round((availableHours / 24) \* 100);

// Generate time slots (6:00 to 23:30)

const timeSlots = generateTimeSlots().map(slot => ({

...slot,

isHour: slot.minute === 0

}));

// Get event positioning and styling

const getEventStyle = (event: CalendarEvent) => {

const startTime = event.startTime instanceof Date ? event.startTime : new Date(event.startTime);

const endTime = event.endTime instanceof Date ? event.endTime : new Date(event.endTime);

if (!startTime || !endTime || isNaN(startTime.getTime()) || isNaN(endTime.getTime())) {

return { className: 'appointment', style: { gridRowStart: 1, gridRowEnd: 2, zIndex: 10 } };

}

const startHour = startTime.getHours();

const startMinute = startTime.getMinutes();

const endHour = endTime.getHours();

const endMinute = endTime.getMinutes();

// Calculate grid positions (each 30-minute slot = 1 grid row)

const startSlot = Math.max(0, (startHour - 6) \* 2 + Math.floor(startMinute / 30));

const endSlot = Math.max(startSlot + 1, (endHour - 6) \* 2 + Math.ceil(endMinute / 30));

const gridRowStart = startSlot + 1;

const gridRowEnd = endSlot + 1;

// Source-specific styling

let className = 'appointment ';

const isSimplePractice = event.source === 'simplepractice' ||

};

// EXACT REPLICA of your application interface - analyzing your screenshots carefully

**Error! Filename not specified.**import React from 'react';

import html2canvas from 'html2canvas';

import jsPDF from 'jspdf';

interface AppointmentData {

id: string;

startTime: string;

endTime: string;

clientName: string;

status: 'confirmed' | 'clinician\_canceled' | 'client\_canceled';

title?: string;

}

const TrueApplicationReplica: React.FC<{ appointments: AppointmentData[] }> = ({ appointments }) => {

// Generate time slots matching your app exactly

const generateTimeSlots = () => {

const slots = [];

for (let hour = 6; hour <= 19; hour++) {

// Top of hour (shaded)

slots.push({

time: `${hour.toString().padStart(2, '0')}:00`,

isTopOfHour: true

});

// Half hour (not shaded)

if (hour < 19) {

slots.push({

time: `${hour.toString().padStart(2, '0')}:30`,

isTopOfHour: false

});

}

}

return slots;

};

const timeSlots = generateTimeSlots();

// Map appointments to time slots

const appointmentsByTime = appointments.reduce((acc, apt) => {

acc[apt.startTime] = apt;

return acc;

}, {} as Record<string, AppointmentData>);

return (

<div style={{

fontFamily: '-apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif',

backgroundColor: '#ffffff',

width: '100%',

maxWidth: '1200px',

margin: '0 auto',

padding: '16px'

}}>

{/\* EXACT Header Layout from your app \*/}

<div style={{

display: 'flex',

justifyContent: 'space-between',

alignItems: 'flex-start',

marginBottom: '16px',

padding: '0 8px'

}}>

{/\* Left side - Date and appointment count \*/}

<div>

<h1 style={{

fontSize: '28px',

fontWeight: '600',

margin: '0',

color: '#1f2937',

lineHeight: '1.2'

}}>

Friday, July 18, 2025

</h1>

<p style={{

fontSize: '14px',

color: '#6b7280',

margin: '4px 0 0 0'

}}>

8 appointments

</p>

</div>

{/\* Right side - Metrics HORIZONTALLY arranged like your app \*/}

<div style={{

display: 'flex',

alignItems: 'center',

gap: '40px'

}}>

<div style={{ textAlign: 'center' }}>

<div style={{

fontSize: '24px',

fontWeight: '600',

color: '#1f2937',

lineHeight: '1'

}}>8</div>

<div style={{

fontSize: '11px',

color: '#6b7280',

textAlign: 'center',

marginTop: '2px'

}}>Appointments</div>

];

import React, { useState } from 'react';

**Error! Filename not specified.**import { CalendarEvent } from '@/types/calendar';

import { AppointmentStatusModal } from './AppointmentStatusModal';

import {

getAppointmentStatusStyles,

getStatusBadgeInfo,

shouldShowStrikethrough,

isAppointmentEvent,

getAppointmentStatusLabel

} from '@/utils/appointmentStatusUtils';

import { Badge } from '@/components/ui/badge';

import { Card, CardContent } from '@/components/ui/card';

interface AppointmentStatusViewProps {

events: CalendarEvent[];

selectedDate: Date;

onEventClick?: (event: CalendarEvent) => void;

}

export function AppointmentStatusView({ events, selectedDate, onEventClick }: AppointmentStatusViewProps) {

const [selectedEvent, setSelectedEvent] = useState<CalendarEvent | null>(null);

const [modalOpen, setModalOpen] = useState(false);

// Filter events for the selected date and appointments only

const appointmentEvents = events.filter(event => {

const eventDate = new Date(event.startTime);

const selectedDateOnly = new Date(selectedDate.getFullYear(), selectedDate.getMonth(), selectedDate.getDate());

const eventDateOnly = new Date(eventDate.getFullYear(), eventDate.getMonth(), eventDate.getDate());

return eventDateOnly.getTime() === selectedDateOnly.getTime() && isAppointmentEvent(event);

}).sort((a, b) => new Date(a.startTime).getTime() - new Date(b.startTime).getTime());

const handleEventClick = (event: CalendarEvent) => {

setSelectedEvent(event);

setModalOpen(true);

onEventClick?.(event);

};

const formatTime = (date: Date) => {

return date.toLocaleTimeString('en-US', {

hour: '2-digit',

minute: '2-digit',

hour12: false

});

};

const formatTimeRange = (startTime: Date, endTime: Date) => {

return `${formatTime(startTime)}-${formatTime(endTime)}`;

};

if (appointmentEvents.length === 0) {

return (

<Card className="w-full">

<CardContent className="p-6">

<div className="text-center text-muted-foreground">

No appointments scheduled for {selectedDate.toLocaleDateString()}

</div>

</CardContent>

</Card>

);

}

return (

<div className="space-y-4 h-full">

<div className="text-lg font-semibold">

{selectedDate.toLocaleDateString('en-US', {

weekday: 'long',

year: 'numeric',

month: 'long',

day: 'numeric'

})}

</div>

<div className="text-sm text-muted-foreground">

{appointmentEvents.length} appointment{appointmentEvents.length !== 1 ? 's' : ''} scheduled

</div>

<div className="space-y-3 max-h-[600px] overflow-y-auto">

{appointmentEvents.map((event) => {

const badgeInfo = getStatusBadgeInfo(event.status || 'scheduled');

const isStrikethrough = shouldShowStrikethrough(event.status || 'scheduled');

const getBackgroundColor = (status: string) => {

switch(status) {

case 'cancelled': return '#fff3cd';

case 'no\_show': return '#f8d7da';

case 'clinician\_canceled': return '#f8f9fa';

default: return '#ffffff';

}

};

const getBorderColor = (status: string) => {

switch(status) {

case 'cancelled': return '#ffc107';

case 'no\_show': return '#dc3545';

case 'clinician\_canceled': return '#6c757d';

}

import React, { useState } from 'react';

**Error! Filename not specified.**import { CalendarEvent } from '@/types/calendar';

import { AppointmentStatusModal } from './AppointmentStatusModal';

import {

getAppointmentStatusStyles,

getStatusBadgeInfo,

shouldShowStrikethrough,

isAppointmentEvent,

getAppointmentStatusLabel

} from '@/utils/appointmentStatusUtils';

import { Badge } from '@/components/ui/badge';

import { Card, CardContent } from '@/components/ui/card';

interface AppointmentStatusViewProps {

events: CalendarEvent[];

selectedDate: Date;

onEventClick?: (event: CalendarEvent) => void;

}

export function AppointmentStatusView({ events, selectedDate, onEventClick }: AppointmentStatusViewProps) {

const [selectedEvent, setSelectedEvent] = useState<CalendarEvent | null>(null);

const [modalOpen, setModalOpen] = useState(false);

// Filter events for the selected date and appointments only

const appointmentEvents = events.filter(event => {

const eventDate = new Date(event.startTime);

const selectedDateOnly = new Date(selectedDate.getFullYear(), selectedDate.getMonth(), selectedDate.getDate());

const eventDateOnly = new Date(eventDate.getFullYear(), eventDate.getMonth(), eventDate.getDate());

return eventDateOnly.getTime() === selectedDateOnly.getTime() && isAppointmentEvent(event);

}).sort((a, b) => new Date(a.startTime).getTime() - new Date(b.startTime).getTime());

const handleEventClick = (event: CalendarEvent) => {

setSelectedEvent(event);

setModalOpen(true);

onEventClick?.(event);

};

const formatTime = (date: Date) => {

return date.toLocaleTimeString('en-US', {

hour: '2-digit',

minute: '2-digit',

hour12: false

});

};

const formatTimeRange = (startTime: Date, endTime: Date) => {

return `${formatTime(startTime)}-${formatTime(endTime)}`;

};

if (appointmentEvents.length === 0) {

return (

<Card className="w-full">

<CardContent className="p-6">

<div className="text-center text-muted-foreground">

No appointments scheduled for {selectedDate.toLocaleDateString()}

</div>

</CardContent>

</Card>

);

}

return (

<div className="space-y-4 h-full">

<div className="text-lg font-semibold">

{selectedDate.toLocaleDateString('en-US', {

weekday: 'long',

year: 'numeric',

month: 'long',

day: 'numeric'

})}

</div>

<div className="text-sm text-muted-foreground">

{appointmentEvents.length} appointment{appointmentEvents.length !== 1 ? 's' : ''} scheduled

</div>

<div className="space-y-3 max-h-[600px] overflow-y-auto">

{appointmentEvents.map((event) => {

const badgeInfo = getStatusBadgeInfo(event.status || 'scheduled');

const isStrikethrough = shouldShowStrikethrough(event.status || 'scheduled');

const getBackgroundColor = (status: string) => {

switch(status) {

case 'cancelled': return '#fff3cd';

case 'no\_show': return '#f8d7da';

case 'clinician\_canceled': return '#f8f9fa';

default: return '#ffffff';

}

};

const getBorderColor = (status: string) => {

switch(status) {

case 'cancelled': return '#ffc107';

case 'no\_show': return '#dc3545';

case 'clinician\_canceled': return '#6c757d';

}

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

**Error! Filename not specified.**import { useMutation, useQueryClient } from '@tanstack/react-query';

import { apiRequest } from '@/lib/queryClient';

import {

Dialog,

DialogContent,

DialogHeader,

DialogTitle,

DialogDescription,

DialogFooter

} from '@/components/ui/dialog';

import { Button } from '@/components/ui/button';

import {

Select,

SelectContent,

SelectItem,

SelectTrigger,

SelectValue

} from '@/components/ui/select';

import { Textarea } from '@/components/ui/textarea';

import { Label } from '@/components/ui/label';

import { useToast } from '@/hooks/use-toast';

import { CalendarEvent } from '@/types/calendar';

import { AppointmentStatus } from '../../../../shared/schema';

import { getLocationDisplay } from '@/utils/locationUtils';

interface AppointmentStatusModalProps {

event: CalendarEvent | null;

isOpen: boolean;

onClose: () => void;

}

const statusOptions = [

{ value: AppointmentStatus.SCHEDULED, label: 'Scheduled', color: '#4285f4' },

{ value: AppointmentStatus.CONFIRMED, label: 'Confirmed', color: '#34a853' },

{ value: AppointmentStatus.CANCELLED, label: 'Cancelled', color: '#ffc107' },

{ value: AppointmentStatus.NO\_SHOW, label: 'No Show', color: '#dc3545' },

{ value: AppointmentStatus.CLINICIAN\_CANCELED, label: 'Clinician Canceled', color: '#6c757d' },

{ value: AppointmentStatus.COMPLETED, label: 'Completed', color: '#28a745' }

];

const locationOptions = [

{ value: 'none', label: 'No Location' },

{ value: 'woodbury', label: '🏢 Woodbury' },

{ value: 'rvc', label: '🏢 RVC' },

{ value: 'telehealth', label: '💻 Telehealth' }

];

export function AppointmentStatusModal({ event, isOpen, onClose }: AppointmentStatusModalProps) {

const [selectedStatus, setSelectedStatus] = useState<string>('');

const [selectedLocation, setSelectedLocation] = useState<string>('');

const [reason, setReason] = useState('');

const { toast } = useToast();

const queryClient = useQueryClient();

// Initialize location state when event changes

useEffect(() => {

if (event) {

setSelectedLocation(event.location || 'none');

}

}, [event]);

const updateStatusMutation = useMutation({

mutationFn: async ({ eventId, status, reason, location }: { eventId: string; status: string; reason: string; location?: string }) => {

console.log('Updating status and location for event:', { eventId, status, reason, location });

// Update the event with both status and location

const response = await apiRequest('PUT', `/api/events/${eventId}`, {

status,

cancellationReason: reason.trim() || null,

location: location === 'none' ? null : location

});

return response.json();

},

onSuccess: (data) => {

toast({

title: "Event Updated",

description: `Event successfully updated`,

variant: "default"

});

// Invalidate and refetch calendar events

queryClient.invalidateQueries({ queryKey: ['/api/events'] });

queryClient.invalidateQueries({ queryKey: ['/api/calendar/events'] });

}

import { Button } from '@/components/ui/button';

**Error! Filename not specified.**import { runPixelPerfectAudit } from '@/utils/pixelPerfectAudit';

import { pdfPerfectionTester } from '@/utils/pdfPerfectionTest';

import { pixelPerfectReviewer } from '@/utils/pixelPerfectReview';

import { comprehensivePixelAnalyzer } from '@/utils/comprehensivePixelAnalysis';

import { exportPixelPerfectPDF } from '@/utils/pixelPerfectPDFExportFixed';

import { exportExactGridPDF } from '../../utils/exactGridPDFExport';

import { exportDailyToPDF } from '../../utils/dailyPDFExport';

interface ExportToPDFProps {

isGoogleConnected: boolean;

onExportCurrentView: (type?: string) => void;

onExportWeeklyPackage: () => void;

onExportDailyView: () => void;

onExportFullMonth: () => void;

onExportToGoogleDrive: (type: string) => void;

}

export const ExportToPDF = ({

isGoogleConnected,

onExportCurrentView,

onExportWeeklyPackage,

onExportDailyView,

onExportFullMonth,

onExportToGoogleDrive

}: ExportToPDFProps) => {

// ExportToPDF component rendered

// Make the export function globally available for testing

(window as any).testDailyExport = () => {

// Global test export called

onExportCurrentView('Daily View');

};

// Also make it available with a simple name

(window as any).export = () => {

// Simple export called

onExportCurrentView('Daily View');

};

// Add simple PDF export test

(window as any).testSimplePDF = async () => {

try {

// Testing Simple PDF Export

// Import the simple PDF export function

const { exportSimplePDF } = await import('../../utils/simplePDFExport');

// Use current date and get events for today

const testDate = new Date();

const events = (window as any).currentEvents || [];

const todayEvents = events.filter(event => {

const eventDate = new Date(event.startTime);

return eventDate.toDateString() === testDate.toDateString();

});

// Exporting Simple PDF for date

// Events for this date

await exportSimplePDF(testDate, todayEvents);

// Simple PDF test completed successfully

} catch (error) {

// Simple PDF test failed

}

};

// Add pixel-perfect review function

(window as any).runPixelPerfectReview = async () => {

try {

console.log('🔍 Running Pixel-Perfect Review');

// Use current date and get events

const testDate = new Date();

const events = (window as any).currentEvents || [];

console.log('Reviewing for date:', testDate.toDateString());

console.log('Events for analysis:', events.length);

const results = await pixelPerfectReviewer.runPixelPerfectReview(testDate, events);

console.log('\n🎯 PIXEL-PERFECT REVIEW RESULTS:');

console.log('='.repeat(80));

console.log(`📊 Overall Score: ${results.overallScore}/${results.maxScore} (${results.percentage}%)`);

console.log(`🔧 Issues Found: ${results.issues.length}`);

console.log(`💡 Recommendations: ${results.recommendations.length}`);

if (results.issues.length > 0) {

console.log('\n❌ ISSUES FOUND:');

results.issues.forEach((issue, index) => {

console.log(`${index + 1}. [${issue.severity.toUpperCase()}] ${issue.description}`);

console.log(` Expected: ${issue.expected}`);

console.log(` Actual: ${issue.actual}`);

console.log(` Fix: ${issue.fixRecommendation}\n`);

});

}

if (results.recommendations.length > 0) {

};

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

import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';

import { apiRequest } from '@/lib/queryClient';

import { CalendarEvent, CalendarDay, ViewMode, CalendarState } from '@/types/calendar';

import { WeeklyCalendarGrid } from '@/components/calendar/WeeklyCalendarGrid';

import { DailyView } from '@/components/calendar/DailyView';

import { MonthlyView } from '@/components/calendar/MonthlyView';

import { YearlyView } from '@/components/calendar/YearlyView';

import { CalendarLegend } from '@/components/calendar/CalendarLegend';

import { Button } from '@/components/ui/button';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';

import { Badge } from '@/components/ui/badge';

import { Loader2, Calendar, FileText, Download, Upload, Eye, Settings, AlertCircle } from 'lucide-react';

import { useToast } from '@/hooks/use-toast';

import { useAuthenticatedUser } from '@/hooks/useAuthenticatedUser';

import { LoadingState } from '@/components/common/LoadingState';

import { generateWeekDays } from '@/utils/dateUtils';

import { pixelPerfectAuditSystem } from '@/utils/pixelPerfectAuditSystem';

import { auditSystem, AuditResults } from '@/utils/comprehensiveAuditSystem';

import { exportExactGridPDF } from '@/utils/exactGridPDFExport';

import { exportDailyToPDF } from '@/utils/dailyPDFExport';

import { exportWeeklyPackage } from '@/utils/weeklyPackageExport';

import { exportBidirectionalWeeklyPackage } from '@/utils/bidirectionalWeeklyPackage';

import { exportDynamicDailyPlannerPDF } from '@/utils/dynamicDailyPlannerPDF';

import { exportTrulyPixelPerfectWeeklyPDF } from '@/utils/trulyPixelPerfectExport';

import { exportExactWeeklySpec } from '@/utils/exactWeeklySpecExport';

import { exportExactWeeklyPackage } from '@/utils/exactWeeklyPackageExport';

import { exportLinkedWeeklyPackage } from '@/utils/bidirectionalWeeklyPackageLinked';

import { export100PercentPixelPerfectPDF } from '@/utils/pixelPerfectPDFExport';

import { exportEnhancedWeeklyPDF } from '@/utils/enhancedWeeklyPDFExport';

import { exportEnhancedDailyPDF } from '@/utils/enhancedDailyPDFExport';

import { exportEnhancedWeeklyPackage } from '@/utils/enhancedWeeklyPackageExport';

import { exportHtmlTemplateDailyPDF } from '@/utils/htmlTemplateDailyExport';

import { exportHTMLTemplatePerfect } from '@/utils/htmlTemplatePerfectExport';

import { exportPerfectDailyCalendarPDF } from '@/utils/perfectDailyCalendarPDF';

import { exportIsolatedCalendarPDF } from '@/utils/isolatedCalendarPDF';

import { runIsolatedCalendarAudit } from '@/utils/isolatedCalendarAudit';

import { exportBrowserReplicaPDF } from '@/utils/browserReplicaPDF';

import { SimpleOAuth } from '@/components/auth/SimpleOAuth';

import { AuthMonitor } from '@/components/auth/AuthMonitor';

import { dailyPDFAudit } from '@/utils/dailyPDFAudit';

import { NewOAuthTest } from '@/components/auth/NewOAuthTest';

import { DateRangeInfo } from '@/components/DateRangeInfo';

import { runAuthenticationFix } from '@/utils/authenticationFix';

import { runSimpleAuthFix, testAuthenticationStatus, forceCalendarSync } from '@/utils/simpleAuthFix';

import { AppointmentStatusView, AppointmentStats } from '@/components/calendar/AppointmentStatusView';

import { AppointmentStatusModal } from '@/components/calendar/AppointmentStatusModal';

export default function Planner() {

const { user, isLoading: userLoading, refetch: refetchAuth } = useAuthenticatedUser();

const { toast } = useToast();

const queryClient = useQueryClient();

// Force refresh authentication after OAuth callback

useEffect(() => {

const urlParams = new URLSearchParams(window.location.search);

// Check for successful authentication

if (urlParams.get('auth') === 'success') {

setTimeout(() => {

refetchAuth();

// Clear the URL parameters

window.history.replaceState({}, document.title, window.location.pathname);

}, 1000);

}

// Check for failed authentication

if (urlParams.get('error') === 'oauth\_failed') {

toast({

title: "Authentication Failed",

description: "Google OAuth authentication failed. Please try again.",

variant: "destructive"

});

// Clear the URL parameters

window.history.replaceState({}, document.title, window.location.pathname);

}

// Also check for Google OAuth callback parameters

if (urlParams.has('code') && urlParams.has('scope')) {

setTimeout(() => {

refetchAuth();

}, 1000);

}

}, [refetchAuth, toast]);

// Add authentication refresh function for debugging

const refreshAuth = async () => {

try {

await refetchAuth();

await queryClient.invalidateQueries({ queryKey: ['/api/events'] });

await queryClient.invalidateQueries({ queryKey: ['/api/simplepractice/events'] });

await queryClient.invalidateQueries({ queryKey: ['/api/calendar/events'] });

} catch (error) {

// Auth refresh failed

}

}

import jsPDF from 'jspdf';

import { CalendarEvent } from '../types/calendar';

import { exportCurrentWeeklyView } from './currentWeeklyExport';

import { exportBrowserReplicaPDF } from './browserReplicaPDF';

/\*\*

\* Enhanced Bidirectional PDF Export System

\* Creates a single linked PDF with navigation between weekly and daily views

\*/

interface PDFPage {

type: 'weekly' | 'daily';

title: string;

date?: Date;

pageNumber: number;

content: any;

}

export class BidirectionalPDFManager {

private pages: PDFPage[] = [];

private pdf: jsPDF;

private linkColor = [0, 0, 255]; // Blue for links

private navigationHeight = 30;

constructor() {

this.pdf = new jsPDF({

orientation: 'landscape',

unit: 'pt',

format: 'a4'

});

}

/\*\*

\* Add navigation links to each page

\*/

private addNavigationLinks(pageNumber: number, pageType: 'weekly' | 'daily', currentDate?: Date) {

const pageWidth = this.pdf.internal.pageSize.getWidth();

const pageHeight = this.pdf.internal.pageSize.getHeight();

// Navigation bar background

this.pdf.setFillColor(240, 240, 240);

this.pdf.rect(0, pageHeight - this.navigationHeight, pageWidth, this.navigationHeight, 'F');

// Navigation text

this.pdf.setFontSize(8);

this.pdf.setTextColor(60, 60, 60);

let navText = '';

const links: Array<{text: string, x: number, y: number, page: number}> = [];

if (pageType === 'weekly') {

navText = 'Weekly Overview - Navigate to: ';

let x = 50;

// Add links to each daily page

for (let i = 0; i < 7; i++) {

const dayName = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'][i];

const targetPage = i + 2; // Daily pages start at page 2

links.push({

text: dayName,

x: x,

y: pageHeight - 15,

page: targetPage

});

x += 40;

}

} else if (pageType === 'daily' && currentDate) {

const dayName = currentDate.toLocaleDateString('en-US', { weekday: 'long' });

const dateStr = currentDate.toLocaleDateString('en-US', {

month: 'short',

day: 'numeric'

});

navText = `${dayName} ${dateStr} - `;

// Link back to weekly overview

links.push({

text: 'Weekly Overview',

x: 50,

y: pageHeight - 15,

page: 1

});

// Links to other daily pages

let x = 150;

for (let i = 0; i < 7; i++) {

const dayName = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'][i];

const targetPage = i + 2;

if (targetPage !== pageNumber) {

links.push({

text: dayName,

x: x,

y: pageHeight - 15,

};

import jsPDF from 'jspdf';

**Error! Filename not specified.**import { CalendarEvent } from '../types/calendar';

import { exportCurrentWeeklyView } from './currentWeeklyExport';

import { exportBrowserReplicaPDF } from './browserReplicaPDF';

import { exportHTMLTemplatePDF } from './htmlTemplatePDF';

/\*\*

\* TRUE UNIFIED BIDIRECTIONAL WEEKLY PACKAGE EXPORT

\*

\* Creates a single PDF with bidirectional navigation by:

\* 1. Executing existing template functions to generate their content

\* 2. Intercepting the PDF output and adding it to our unified document

\* 3. Adding clickable navigation links between pages

\* 4. Using the ACTUAL existing templates without modification

\*/

class UnifiedBidirectionalExporter {

private events: CalendarEvent[];

private weekStart: Date;

private weekEnd: Date;

private pdf: jsPDF;

private linkColor = [0, 0, 255]; // Blue for links

constructor(events: CalendarEvent[], weekStart: Date) {

this.events = events;

this.weekStart = new Date(weekStart);

this.weekStart.setHours(0, 0, 0, 0);

this.weekEnd = new Date(weekStart);

this.weekEnd.setDate(weekStart.getDate() + 6);

this.weekEnd.setHours(23, 59, 59, 999);

// Initialize PDF in landscape for weekly view (matching Current Weekly Export)

this.pdf = new jsPDF({

orientation: 'landscape',

unit: 'pt',

format: [792, 612] // Exact dimensions from Current Weekly Export

});

}

/\*\*

\* Main export function - Creates single bidirectional PDF using ACTUAL existing templates

\*/

async export(): Promise<string> {

try {

console.log('🔗 TRUE UNIFIED BIDIRECTIONAL EXPORT STARTING...');

console.log('📊 Using ACTUAL existing templates: Current Weekly Export + Browser Replica PDF');

// Step 1: Use ACTUAL Current Weekly Export template logic for Page 1

console.log('📄 Page 1: Calling ACTUAL Current Weekly Export template...');

// Import and replicate the EXACT logic from currentWeeklyExport.ts

await this.createActualWeeklyPage();

// Step 2: Use ACTUAL Browser Replica PDF template logic for Pages 2-8

console.log('📄 Pages 2-8: Calling ACTUAL Browser Replica PDF templates...');

const days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'];

for (let dayIndex = 0; dayIndex < 7; dayIndex++) {

const currentDate = new Date(this.weekStart);

currentDate.setDate(this.weekStart.getDate() + dayIndex);

console.log(`📄 Page ${dayIndex + 2}: Creating ${days[dayIndex]} using ACTUAL Browser Replica template...`);

// Add new page in portrait orientation

this.pdf.addPage([612, 792], 'portrait'); // US Letter portrait

// Call ACTUAL Browser Replica PDF template logic

await this.createActualDailyPage(currentDate, dayIndex + 2);

}

console.log('🎯 UNIFIED BIDIRECTIONAL EXPORT COMPLETE');

// Save the file

const filename = `unified-bidirectional-weekly-package-${this.weekStart.toISOString().split('T')[0]}.pdf`;

this.pdf.save(filename);

return filename;

} catch (error) {

console.error('❌ Unified Bidirectional Export failed:', error);

throw error;

}

}

/\*\*

\* Create page 1 using EXACT exportCurrentWeeklyView function

\* Instead of extracting logic, use the actual existing export function

\*/

private async createActualWeeklyPage(): Promise<void> {

console.log('📄 Page 1: Using EXACT exportCurrentWeeklyView function...');

// CRITICAL: The original functions create their own PDF and save it

// We need to call them but somehow capture their output for our unified PDF

};

import { CalendarEvent } from '../types/calendar';

import { exportCurrentWeeklyView } from './currentWeeklyExport';

import { exportBrowserReplicaPDF } from './browserReplicaPDF';

/\*\*

\* COMBINE EXACT TEMPLATES INTO SINGLE PDF

\* This function calls the EXACT existing template functions as-is

\* NOTE: The original functions save PDFs directly to the user's downloads

\*/

export const combineExactTemplatesIntoBidirectionalPDF = async (

events: CalendarEvent[],

weekStart: Date

): Promise<string> => {

console.log('🎯 CALLING EXACT TEMPLATES - GENERATING 8 SEPARATE PDFs...');

console.log('⚠️ NOTE: The EXACT templates save individual PDFs directly');

console.log('📄 You will get 8 separate PDF files that can be manually combined');

try {

// Setup week dates

const normalizedWeekStart = new Date(weekStart);

normalizedWeekStart.setHours(0, 0, 0, 0);

const normalizedWeekEnd = new Date(weekStart);

normalizedWeekEnd.setDate(weekStart.getDate() + 6);

normalizedWeekEnd.setHours(23, 59, 59, 999);

console.log('📄 Step 1: Calling EXACT exportCurrentWeeklyView function...');

await exportCurrentWeeklyView(normalizedWeekStart, normalizedWeekEnd, events);

console.log('✅ Weekly PDF saved to downloads');

console.log('📄 Step 2: Calling EXACT exportBrowserReplicaPDF for each day...');

const currentDate = new Date(normalizedWeekStart);

for (let dayIndex = 0; dayIndex < 7; dayIndex++) {

const dayName = currentDate.toLocaleDateString('en-US', { weekday: 'long' });

console.log(`📄 Calling EXACT exportBrowserReplicaPDF for ${dayName}...`);

await exportBrowserReplicaPDF(currentDate, events);

console.log(`✅ ${dayName} PDF saved to downloads`);

currentDate.setDate(currentDate.getDate() + 1);

}

console.log('✅ EXACT TEMPLATES CALLED - 8 PDFs generated');

console.log('📄 Check your downloads folder for:');

console.log(' - 1 weekly overview PDF (landscape)');

console.log(' - 7 daily planner PDFs (portrait)');

console.log('📝 These can be manually combined using a PDF editor');

return `8 PDFs generated using EXACT templates`;

} catch (error) {

console.error('❌ Error calling EXACT templates:', error);

throw error;

}

};

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

import { ChevronLeft, ChevronRight, ArrowLeft } from 'lucide-react';

import { Button } from '@/components/ui/button';

import { Textarea } from '@/components/ui/textarea';

import { formatDate } from '../../utils/dateUtils';

import { generateTimeSlots } from '../../utils/timeSlots';

import { CalendarEvent } from '../../types/calendar';

import { getLocationDisplay } from '../../utils/locationUtils';

interface DailyViewProps {

selectedDate: Date;

events: CalendarEvent[];

dailyNotes: string;

onPreviousDay: () => void;

onNextDay: () => void;

onBackToWeek: () => void;

onEventClick: (event: CalendarEvent) => void;

onUpdateEvent: (eventId: string, updates: Partial<CalendarEvent>) => void;

onUpdateDailyNotes: (notes: string) => void;

onEventMove?: (eventId: string, newStartTime: Date, newEndTime: Date) => void;

onCreateEvent?: (startTime: Date, endTime: Date) => void;

onDeleteEvent?: (eventId: string) => void;

}

export const DailyView = ({

selectedDate,

events,

dailyNotes,

onPreviousDay,

onNextDay,

onBackToWeek,

onEventClick,

onUpdateEvent,

onUpdateDailyNotes,

onEventMove,

onCreateEvent,

onDeleteEvent

}: DailyViewProps) => {

const [currentNotes, setCurrentNotes] = useState(dailyNotes);

const [expandedEventId, setExpandedEventId] = useState<string | null>(null);

const [noteTimers, setNoteTimers] = useState<{[key: string]: NodeJS.Timeout}>({});

const [draggedEventId, setDraggedEventId] = useState<string | null>(null);

const [dragOverSlot, setDragOverSlot] = useState<number | null>(null);

// Get events for the selected date with null checks and proper date conversion

const dayEvents = events.filter(event => {

if (!event || !event.startTime || !event.endTime || !selectedDate) return false;

try {

// Convert startTime and endTime to Date objects if they aren't already

const startTime = event.startTime instanceof Date ? event.startTime : new Date(event.startTime);

const endTime = event.endTime instanceof Date ? event.endTime : new Date(event.endTime);

// Validate that dates are valid

if (!startTime || !endTime || isNaN(startTime.getTime()) || isNaN(endTime.getTime()) || isNaN(selectedDate.getTime())) {

// Invalid date detected

return false;

}

const selectedDateString = selectedDate.toDateString();

const eventDateString = startTime.toDateString();

const matches = eventDateString === selectedDateString;

// Debug specific events

if (event.title.toLowerCase().includes('calvin') || event.title.toLowerCase().includes('hill')) {

console.log(`🎯 Calvin Hill Event Debug:`, {

title: event.title,

selectedDate: selectedDateString,

eventDate: eventDateString,

matches: matches,

startTime: startTime.toISOString(),

endTime: endTime.toISOString()

});

}

return matches;

} catch (error) {

// Invalid date in event

return false;

}

});

// Debug logging for event filtering

console.log(`📊 Daily View Debug:`, {

selectedDate: selectedDate?.toDateString(),

totalEvents: events.length,

filteredEvents: dayEvents.length,

eventTitles: dayEvents.map(e => e.title)

});

// Null check for selectedDate before using

if (!selectedDate) {

// DailyView: selectedDate is undefined

return (

<div className="planner-container daily-planner">

<div className="flex items-center justify-center h-64">

<p>Loading daily view...</p>

</div>

</div>

);

}

// Daily view event filtering completed

// Calculate daily statistics with error handling

const totalEvents = dayEvents.length;

};

import { pgTable, text, serial, integer, boolean, timestamp } from "drizzle-orm/pg-core";

import { createInsertSchema } from "drizzle-zod";

import { z } from "zod";

// Appointment Status enum

export const AppointmentStatus = {

SCHEDULED: 'scheduled',

CONFIRMED: 'confirmed',

CANCELLED: 'cancelled',

NO\_SHOW: 'no\_show',

CLINICIAN\_CANCELED: 'clinician\_canceled',

COMPLETED: 'completed'

} as const;

export type AppointmentStatusType = typeof AppointmentStatus[keyof typeof AppointmentStatus];

export const users = pgTable("users", {

id: serial("id").primaryKey(),

username: text("username").notNull().unique(),

password: text("password"),

googleId: text("google\_id").unique(),

email: text("email"),

name: text("name"),

});

export const events = pgTable("events", {

id: serial("id").primaryKey(),

userId: integer("user\_id").references(() => users.id),

title: text("title").notNull(),

description: text("description"),

startTime: timestamp("start\_time").notNull(),

endTime: timestamp("end\_time").notNull(),

source: text("source").default("manual"), // 'manual', 'google', 'simplepractice'

sourceId: text("source\_id"),

calendarId: text("calendar\_id"), // For Google Calendar events

color: text("color").default("#6495ED"),

notes: text("notes"),

actionItems: text("action\_items"),

// Location field for appointment location

location: text("location"), // 'woodbury', 'rvc', 'telehealth', or NULL

// Appointment status fields

status: text("status").default("scheduled"), // 'scheduled', 'confirmed', 'cancelled', 'no\_show', 'clinician\_canceled', 'completed'

statusChangedBy: integer("status\_changed\_by").references(() => users.id),

statusChangedAt: timestamp("status\_changed\_at"),

cancellationReason: text("cancellation\_reason"),

createdAt: timestamp("created\_at").defaultNow(),

updatedAt: timestamp("updated\_at").defaultNow(),

});

export const dailyNotes = pgTable("daily\_notes", {

id: serial("id").primaryKey(),

userId: integer("user\_id").references(() => users.id),

date: text("date").notNull(), // YYYY-MM-DD format

content: text("content"),

createdAt: timestamp("created\_at").defaultNow(),

updatedAt: timestamp("updated\_at").defaultNow(),

});

export const statusChangeLogs = pgTable("status\_change\_logs", {

id: serial("id").primaryKey(),

eventId: integer("event\_id").references(() => events.id),

oldStatus: text("old\_status").notNull(),

newStatus: text("new\_status").notNull(),

changedBy: integer("changed\_by").references(() => users.id),

reason: text("reason"),

changedAt: timestamp("changed\_at").defaultNow(),

});

export const insertUserSchema = createInsertSchema(users).pick({

username: true,

password: true,

email: true,

name: true,

});

export const insertEventSchema = createInsertSchema(events).omit({

id: true,

createdAt: true,

updatedAt: true,

});

export const insertDailyNotesSchema = createInsertSchema(dailyNotes).omit({

id: true,

createdAt: true,

updatedAt: true,

});

export const insertStatusChangeLogSchema = createInsertSchema(statusChangeLogs).omit({

id: true,

changedAt: true,

});

export type InsertUser = z.infer<typeof insertUserSchema>;

export type User = typeof users.$inferSelect;

export type Event = typeof events.$inferSelect;

export type InsertEvent = z.infer<typeof insertEventSchema>;

export type DailyNote = typeof dailyNotes.$inferSelect;

export type InsertDailyNote = z.infer<typeof insertDailyNotesSchema>;

export type StatusChangeLog = typeof statusChangeLogs.$inferSelect;

export type InsertStatusChangeLog = z.infer<typeof insertStatusChangeLogSchema>;

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

import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';

import { apiRequest } from '@/lib/queryClient';

import { CalendarEvent, CalendarDay, ViewMode, CalendarState } from '@/types/calendar';

import { WeeklyCalendarGrid } from '@/components/calendar/WeeklyCalendarGrid';

import { DailyView } from '@/components/calendar/DailyView';

import { MonthlyView } from '@/components/calendar/MonthlyView';

import { YearlyView } from '@/components/calendar/YearlyView';

import { CalendarLegend } from '@/components/calendar/CalendarLegend';

import { Button } from '@/components/ui/button';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';

import { Badge } from '@/components/ui/badge';

import { Loader2, Calendar, FileText, Download, Upload, Eye, Settings, AlertCircle } from 'lucide-react';

import { useToast } from '@/hooks/use-toast';

import { useAuthenticatedUser } from '@/hooks/useAuthenticatedUser';

import { LoadingState } from '@/components/common/LoadingState';

import { generateWeekDays } from '@/utils/dateUtils';

import { pixelPerfectAuditSystem } from '@/utils/pixelPerfectAuditSystem';

import { auditSystem, AuditResults } from '@/utils/comprehensiveAuditSystem';

import { exportExactGridPDF } from '@/utils/exactGridPDFExport';

import { exportDailyToPDF } from '@/utils/dailyPDFExport';

import { exportWeeklyPackage } from '@/utils/weeklyPackageExport';

import { exportBidirectionalWeeklyPackage } from '@/utils/bidirectionalWeeklyPackage';

import { exportDynamicDailyPlannerPDF } from '@/utils/dynamicDailyPlannerPDF';

import { exportTrulyPixelPerfectWeeklyPDF } from '@/utils/trulyPixelPerfectExport';

import { exportExactWeeklySpec } from '@/utils/exactWeeklySpecExport';

import { exportExactWeeklyPackage } from '@/utils/exactWeeklyPackageExport';

import { exportLinkedWeeklyPackage } from '@/utils/bidirectionalWeeklyPackageLinked';

import { export100PercentPixelPerfectPDF } from '@/utils/pixelPerfectPDFExport';

import { exportEnhancedWeeklyPDF } from '@/utils/enhancedWeeklyPDFExport';

import { exportEnhancedDailyPDF } from '@/utils/enhancedDailyPDFExport';

import { exportEnhancedWeeklyPackage } from '@/utils/enhancedWeeklyPackageExport';

import { exportHtmlTemplateDailyPDF } from '@/utils/htmlTemplateDailyExport';

import { exportHTMLTemplatePerfect } from '@/utils/htmlTemplatePerfectExport';

import { exportPerfectDailyCalendarPDF } from '@/utils/perfectDailyCalendarPDF';

import { exportIsolatedCalendarPDF } from '@/utils/isolatedCalendarPDF';

import { runIsolatedCalendarAudit } from '@/utils/isolatedCalendarAudit';

import { exportBrowserReplicaPDF } from '@/utils/browserReplicaPDF';

import { SimpleOAuth } from '@/components/auth/SimpleOAuth';

import { AuthMonitor } from '@/components/auth/AuthMonitor';

import { dailyPDFAudit } from '@/utils/dailyPDFAudit';

import { NewOAuthTest } from '@/components/auth/NewOAuthTest';

import { DateRangeInfo } from '@/components/DateRangeInfo';

import { runAuthenticationFix } from '@/utils/authenticationFix';

import { runSimpleAuthFix, testAuthenticationStatus, forceCalendarSync } from '@/utils/simpleAuthFix';

import { AppointmentStatusView, AppointmentStats } from '@/components/calendar/AppointmentStatusView';

import { AppointmentStatusModal } from '@/components/calendar/AppointmentStatusModal';

export default function Planner() {

const { user, isLoading: userLoading, refetch: refetchAuth } = useAuthenticatedUser();

const { toast } = useToast();

const queryClient = useQueryClient();

// Force refresh authentication after OAuth callback

useEffect(() => {

const urlParams = new URLSearchParams(window.location.search);

// Check for successful authentication

if (urlParams.get('auth') === 'success') {

setTimeout(() => {

refetchAuth();

// Clear the URL parameters

window.history.replaceState({}, document.title, window.location.pathname);

}, 1000);

}

// Check for failed authentication

if (urlParams.get('error') === 'oauth\_failed') {

toast({

title: "Authentication Failed",

description: "Google OAuth authentication failed. Please try again.",

variant: "destructive"

});

// Clear the URL parameters

window.history.replaceState({}, document.title, window.location.pathname);

}

// Also check for Google OAuth callback parameters

if (urlParams.has('code') && urlParams.has('scope')) {

setTimeout(() => {

refetchAuth();

}, 1000);

}

}, [refetchAuth, toast]);

// Add authentication refresh function for debugging

const refreshAuth = async () => {

try {

await refetchAuth();

await queryClient.invalidateQueries({ queryKey: ['/api/events'] });

await queryClient.invalidateQueries({ queryKey: ['/api/simplepractice/events'] });

await queryClient.invalidateQueries({ queryKey: ['/api/calendar/events'] });

} catch (error) {

// Auth refresh failed

}

}