JobSync - Technology Stack Code Snippets

Project: JobSync - AI-Powered Job Application Management System
Client: Asuncion Municipal Hall, Davao del Norte
Purpose: Real code examples showing how each technology is implemented

What Are Code Snippets?

Code snippets are small examples of real code from the JobSync project that demonstrate how each technology is actually used. These are NOT tools - they are sample code taken directly from the codebase to show practical implementation.

Table of Contents

1. Next.js - Frontend Framework

Next.js is the foundation of JobSync, providing server-side rendering, routing, and API routes.

Example 1: App Router Page with Auto-Redirect

File: `src/app/(public)/page.tsx`

```tsx
'use client';
import { useEffect } from 'react';
import { useRouter } from 'next/navigation';
import { useAuth } from '@/contexts/AuthContext';

export default function Home() {
  const { isAuthenticated, role, isLoading } = useAuth();
  const router = useRouter();

  useEffect(() => {
    if (!isLoading && isAuthenticated && role) {
      const dashboardMap: Record<string, string> = {
        ADMIN: '/admin/dashboard',
        HR: '/hr/dashboard',
        PESO: '/peso/dashboard',
        APPLICANT: '/applicant/dashboard',
      };
      const redirectPath = dashboardMap[role] || '/applicant/dashboard';
      router.push(redirectPath);
    }
  }, [isAuthenticated, role, isLoading, router]);

  return (
    <main className="min-h-screen">
      {/* Landing page content */}
    </main>
  );
}
```

What it does: This Next.js page uses the `'use client'` directive to enable client-side features like `useEffect` and `useRouter`. It automatically redirects authenticated users to their role-specific dashboard (HR, PESO, ADMIN, or APPLICANT).

Example 2: Server-Side Metadata Export

File: `src/app/layout.tsx`

```
import type { Metadata } from "next";
import { Geist, Geist_Mono } from "next/font/google";

export const metadata: Metadata = {
  title: "JobSync - AI-Powered Job Matching System",
  description: "Gemini AI-powered job matching system for the Municipality of Asuncion, Davao de
  icons: {
    icon: [
      { url: '/favicon.ico', sizes: '48x48' },
      { url: '/icon-192.png', sizes: '192x192', type: 'image/png' },
    ],
  },
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <body className="antialiased">
        {children}
      </body>
    </html>
  );
}
```

What it does: Server Component that exports metadata for SEO optimization. This runs only on the server, improving performance by not sending unnecessary JavaScript to the client.

Example 3: API Route Handler

File: `src/app/api/applications/export/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import { createClient } from '@/lib/supabase/server';

export async function GET(request: NextRequest) {
  try {
    const supabase = await createClient();

    // Authenticate user
    const { data: { user }, error: authError } = await supabase.auth.getUser();
    if (authError || !user) {
      return NextResponse.json(
        { success: false, error: 'Unauthorized - Please login' },
        { status: 401 }
      );
    }

    // Parse query parameters
    const searchParams = request.nextUrl.searchParams;
    const jobId = searchParams.get('job_id');

    // Database query
    let query = supabase
      .from('applications')
      .select(`
        id, status, rank, match_score,
        jobs:job_id (title, location),
        applicant_profiles:applicant_profile_id (surname, first_name)
      `)
      .order('created_at', { ascending: false });

    if (jobId) {
      query = query.eq('job_id', jobId);
    }

    const { data: applications, error } = await query;

    // Generate Excel file and return
    return new NextResponse(excelBuffer, {
      headers: {
        'Content-Type': 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
        'Content-Disposition': 'attachment; filename="Applications.xlsx"',
      },
    });
  } catch (error: any) {
    return NextResponse.json({ error: error.message }, { status: 500 });
  }
}
```

What it does: Next.js API route that handles GET requests, authenticates users, queries the database with filters, and returns an Excel file for download. This is a serverless function that runs on-demand.

2. React - UI Library

React provides the component-based UI system with hooks for state management and side effects.

Example 1: Component with State and Effects

File: `src/components/charts/MonthlyApplicantsChart.tsx`

```tsx
'use client';
import React, { useState, useEffect } from 'react';
import { LineChart, Line, XAxis, YAxis, Tooltip, ResponsiveContainer } from 'recharts';
import { supabase } from '@/lib/supabase/auth';
import { Loader2 } from 'lucide-react';

interface MonthlyData {
  month: string;
  applications: number;
}

export const MonthlyApplicantsChart: React.FC = () => {
  const [data, setData] = useState<MonthlyData[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchMonthlyData();
  }, []);

  const fetchMonthlyData = async () => {
    try {
      const { data: applications, error } = await supabase
        .from('applications')
        .select('created_at');

      if (error) throw error;

      // Group by month
      const monthCounts: Record<string, number> = {};
      applications?.forEach((app) => {
        const date = new Date(app.created_at);
        const monthKey = `${date.getFullYear()}-${String(date.getMonth() + 1).padStart(2, '0')}`
        monthCounts[monthKey] = (monthCounts[monthKey] || 0) + 1;
      });

      setData(Object.entries(monthCounts).map(([month, count]) => ({
        month,
        applications: count,
      })));
    } catch (error) {
      console.error('Error fetching monthly data:', error);
    } finally {
      setLoading(false);
      }
    };

    if (loading) {
      return (
        <div className="h-64 flex items-center justify-center">
          <Loader2 className="animate-spin" />
        </div>
      );
    }

    return (
      <ResponsiveContainer width="100%" height="100%">
        <LineChart data={data}>
          <XAxis dataKey="month" />
          <YAxis />
          <Tooltip />
          <Line type="monotone" dataKey="applications" stroke="#22A555" />
        </LineChart>
      </ResponsiveContainer>
    );
  };
```

What it does: React component demonstrating:

- `useState` for managing component state (data and loading)
- `useEffect` for fetching data when component mounts
- Conditional rendering (loading spinner vs chart)
- TypeScript interfaces for type safety

Example 2: Context API for Global State

File: `src/contexts/AuthContext.tsx`

```
'use client';
import React, { createContext, useContext, useState, useEffect } from 'react';

interface User {
  id: string;
  email: string;
  fullName: string;
}

interface AuthContextType {
  user: User | null;
  role: 'ADMIN' | 'HR' | 'PESO' | 'APPLICANT' | null;
  isAuthenticated: boolean;
  login: (email: string, password: string) => Promise<void>;
  logout: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export function AuthProvider({ children }: { children: React.ReactNode }) {
  const [user, setUser] = useState<User | null>(null);
  const [role, setRole] = useState<'ADMIN' | 'HR' | 'PESO' | 'APPLICANT' | null>(null);

  const login = async (email: string, password: string) => {
    const result = await authLoginUser({ email, password });
    setUser({
      id: result.data.profile.id,
      email,
      fullName: result.data.profile.fullName
    });
    setRole(result.data.profile.role);
  };

  const logout = async () => {
    setUser(null);
    setRole(null);
  };

  return (
    <AuthContext.Provider value={{ user, role, isAuthenticated: !!user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
}
```

What it does: Implements React Context API for global authentication state management. The custom `useAuth` hook allows any component to access authentication state and functions without prop drilling.

3. TypeScript - Programming Language

TypeScript adds static typing to JavaScript, catching errors during development.

Example 1: Interface Definitions

File: `src/lib/gemini/scoringAlgorithms.ts`

```
export interface JobRequirements {
  title?: string;
  description?: string;
  degreeRequirement: string;
  eligibilities: string[];
  skills: string[];
  yearsOfExperience: number;
}

export interface ApplicantData {
  highestEducationalAttainment: string;
  eligibilities: Array<{ eligibilityTitle: string }>;
  skills: string[];
  totalYearsExperience: number;
  workExperienceTitles?: string[];
}

export interface ScoreBreakdown {
  educationScore: number;
  experienceScore: number;
  skillsScore: number;
  eligibilityScore: number;
  totalScore: number;
  algorithmUsed: string;
  reasoning: string;
  matchedSkillsCount: number;
  matchedEligibilitiesCount: number;
}
```

What it does: Defines TypeScript interfaces that specify the exact shape of data structures. This ensures type safety across the entire ranking system.

Example 2: Type-Safe Functions

File: `src/lib/gemini/scoringAlgorithms.ts`

```typescript
export function algorithm1_WeightedSum(
  job: JobRequirements,
  applicant: ApplicantData
): ScoreBreakdown {
  const jobDegree = job.degreeRequirement.toLowerCase().trim();
  const applicantDegree = applicant.highestEducationalAttainment.toLowerCase().trim();

  let educationScore = matchDegreeRequirement(jobDegree, applicantDegree);

  const weights = {
    education: 0.30,
    experience: 0.25,
    skills: 0.25,
    eligibility: 0.20
  };

  const totalScore =
    weights.education * educationScore +
    weights.experience * experienceScore +
    weights.skills * skillsScore +
    weights.eligibility * eligibilityScore;

  return {
    educationScore,
    experienceScore,
    skillsScore,
    eligibilityScore,
    totalScore: Math.round(totalScore * 100) / 100,
    algorithmUsed: 'Multi-Factor Assessment',
    reasoning: `Education (30%): ${educationScore.toFixed(1)}%...`,
    matchedSkillsCount: 0,
    matchedEligibilitiesCount: 0
  };
}
```

What it does: Function with typed parameters and return type. TypeScript ensures you can only pass `JobRequirements` and `ApplicantData` objects, and the return value will always be a complete `ScoreBreakdown`.

Example 3: Type Inference with Zod

File: `src/lib/validation/profileSchema.ts`

```
import { z } from 'zod';

export const updateProfileSchema = z.object({
  full_name: z.string().min(2).max(100),
  email: z.string().email(),
  phone: z.string().regex(/^\+?[0-9\s\-\(\)]+$/).optional(),
});

// TypeScript type automatically inferred from Zod schema
export type UpdateProfileData = z.infer<typeof updateProfileSchema>;
```

What it does: Uses Zod's `z.infer<>` to automatically generate TypeScript types from validation schemas, ensuring your validation and types stay in sync.

4. Tailwind CSS - Styling Framework

Tailwind provides utility-first CSS classes for rapid UI development.

Example 1: Responsive Layout with Utility Classes

File: `src/app/(public)/page.tsx`

```
<section className="relative bg-gradient-to-br from-[#22A555] via-[#1A7F3E] to-[#22A555] text-wl
  <div className="container mx-auto px-6 py-24 lg:py-32 relative z-10">
    <div className="max-w-4xl mx-auto text-center">
      <div className="inline-block px-4 py-2 bg-white/20 backdrop-blur-sm rounded-full text-sm
        Municipality of Asuncion - Davao del Norte
      </div>

      <h1 className="text-5xl lg:text-6xl font-extrabold mb-6 leading-tight">
        Find Your Perfect Career Match with <span className="text-[#D4F4DD]">JobSync</span>
      </h1>

      <p className="text-xl lg:text-2xl mb-8 text-white/90 leading-relaxed">
        AI-powered job matching system connecting talent with opportunity
      </p>

      <div className="flex flex-col sm:flex-row gap-4 justify-center">
        <button className="px-8 py-4 bg-white text-[#22A555] rounded-lg font-bold hover:bg-gray
          Get Started
        </button>
      </div>
    </div>
  </div>
</section>
```

What it does: Demonstrates Tailwind utility classes for:

- Gradients (`bg-gradient-to-br`)
- Custom colors (`from-[#22A555]`)
- Responsive text sizes (`text-5xl lg:text-6xl`)
- Flexbox layout (`flex flex-col sm:flex-row`)
- Spacing (`px-6 py-24 lg:py-32`)
- Effects (`backdrop-blur-sm` , `hover:bg-gray-100`)

Example 2: Grid Layout with Breakpoints

File: `src/components/dashboard/StatsGrid.tsx`

```
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
  <div className="bg-white rounded-lg shadow-md p-6 border-l-4 border-[#22A555]">
    <div className="flex items-center justify-between">
      <div>
        <p className="text-sm font-medium text-gray-600">Total Applications</p>
        <p className="text-3xl font-bold text-gray-900 mt-2">156</p>
      </div>
      <div className="p-3 bg-[#22A555]/10 rounded-full">
        <FileText className="w-8 h-8 text-[#22A555]" />
      </div>
    </div>
  </div>
  {/* More stat cards */}
</div>
```

What it does: Responsive grid that shows:

- 1 column on mobile (`grid-cols-1`)
- 2 columns on tablets (`md:grid-cols-2`)
- 4 columns on desktops (`lg:grid-cols-4`)
- Custom green accent colors (`border-[#22A555]` , `bg-[#22A555]/10`)

5. Supabase - Database and Authentication

Supabase provides PostgreSQL database, authentication, and real-time subscriptions.

Example 1: Client Initialization

File: `src/lib/supabase/client.ts`

```
import { createBrowserClient } from '@supabase/ssr';

export function createClient() {
  return createBrowserClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
  );
}
```

What it does: Creates a Supabase client for browser-side operations with cookie-based session management.

Example 2: Database Query with Joins

File: `src/app/api/applications/export/route.ts`

```
const supabase = await createClient();

// Complex query with nested joins
const { data: applications, error } = await supabase
  .from('applications')
  .select(`
    id,
    job_id,
    status,
    rank,
    match_score,
    created_at,
    jobs:job_id (
      title,
      location,
      employment_type
    ),
    applicant_profiles:applicant_profile_id (
      surname,
      first_name,
      middle_name,
      phone_number,
      email,
      highest_educational_attainment
    )
  `)
  .eq('status', 'pending')
  .order('created_at', { ascending: false });

if (error) {
  console.error('Database query error:', error);
  return NextResponse.json({ error: error.message }, { status: 500 });
}
```

What it does: Demonstrates Supabase query builder with:

- Table selection (`.from('applications')`)
- Nested joins (foreign key relationships)
- Filtering (`.eq('status', 'pending')`)
- Sorting (`.order('created_at')`)
- Error handling

Example 3: Authentication

File: `src/lib/supabase/auth.ts`

```
import { createClient } from '@/lib/supabase/client';

export async function loginUser(email: string, password: string) {
  const supabase = createClient();

  const { data, error } = await supabase.auth.signInWithPassword({
    email,
    password,
  });

  if (error) {
    throw new Error(error.message);
  }

  // Get user profile with role
  const { data: profile } = await supabase
    .from('profiles')
    .select('id, role, full_name, email')
    .eq('id', data.user.id)
    .single();

  return {
    user: data.user,
    profile,
  };
}

export async function logoutUser() {
  const supabase = createClient();
  const { error } = await supabase.auth.signOut();

  if (error) {
    throw new Error(error.message);
  }
}
```

What it does: Handles user authentication with Supabase Auth, including login with email/password and logout. Retrieves user profile data after authentication.

6. Google Gemini AI - AI Integration

Gemini AI powers the intelligent applicant ranking system with three mathematical algorithms.

Example 1: Gemini Model Initialization

File: `src/lib/gemini/rankApplicants.ts`

```
import { GoogleGenerativeAI } from '@google/generative-ai';

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY!);

async function addGeminiInsights(
  job: any,
  topCandidates: RankedApplicant[]
): Promise<void> {
  const model = genAI.getGenerativeModel({ model: 'gemini-2.0-flash-exp' });

  const prompt = `You are an HR expert analyzing job applicants.

Job Position: ${job.title}
Job Description: ${job.description}

Requirements:
- Education: ${job.degree_requirement}
- Experience: ${job.years_of_experience} years
- Skills: ${job.skills.join(', ')}
- Eligibilities: ${job.eligibilities.join(', ')}

Top ${topCandidates.length} Candidates:
${topCandidates.map((c, i) => `
${i + 1}. ${c.applicantName}
   - Match Score: ${c.matchScore.toFixed(1)}%
   - Education: ${c.educationScore.toFixed(1)}%
   - Experience: ${c.experienceScore.toFixed(1)}%
   - Skills: ${c.skillsScore.toFixed(1)}%
   - Eligibility: ${c.eligibilityScore.toFixed(1)}%
`).join('\n')}

For each candidate, provide brief (2-3 sentences) professional insight about their fit for this

  const result = await model.generateContent(prompt);
  const response = result.response;
  const text = response.text();

  // Parse and assign insights
  const insights = text.split(/Candidate \d+:/g).slice(1);
  topCandidates.forEach((candidate, index) => {
    if (insights[index]) {
      candidate.geminiInsights = insights[index].trim();
    }
    });
  }
```

What it does: Initializes Gemini AI model and generates qualitative insights for top candidates using natural language processing.

Example 2: AI-Powered Scoring Algorithm

File: `src/lib/gemini/scoringAlgorithms.ts`

```
export function algorithm1_WeightedSum(
  job: JobRequirements,
  applicant: ApplicantData
): ScoreBreakdown {
  // Education Score using Levenshtein distance
  const jobDegree = job.degreeRequirement.toLowerCase().trim();
  const applicantDegree = applicant.highestEducationalAttainment.toLowerCase().trim();
  let educationScore = matchDegreeRequirement(jobDegree, applicantDegree);

  // Apply degree level bonuses/penalties
  const degreeLevels = ['elementary', 'secondary', 'vocational', 'bachelor', 'master', 'doctora
  const jobLevel = degreeLevels.findIndex(level => jobDegree.includes(level));
  const applicantLevel = degreeLevels.findIndex(level => applicantDegree.includes(level));

  if (jobLevel === applicantLevel) {
    // Same level: tiered floor based on similarity
    if (educationScore >= 40) {
      educationScore = Math.max(educationScore, 60);
    } else {
      educationScore = Math.max(educationScore, 40);
    }
  }

  // Experience Score (70% years + 30% relevance)
  const requiredYears = job.yearsOfExperience;
  const applicantYears = applicant.totalYearsExperience;
  let yearsScore = Math.min((applicantYears / requiredYears) * 100, 100);

  const experienceScore = (yearsScore * 0.7) + (relevanceScore * 0.3);

  // Skills Score: Token-based fuzzy matching
  const skillsMatch = calculateSkillMatch(job.skills, applicant.skills);
  const skillsScore = skillsMatch.score;

  // Weighted sum with normalized weights
  const weights = { education: 0.30, experience: 0.25, skills: 0.25, eligibility: 0.20 };
  const totalScore =
    weights.education * educationScore +
    weights.experience * experienceScore +
    weights.skills * skillsScore +
    weights.eligibility * eligibilityScore;

  return {
    educationScore,
    experienceScore,
    skillsScore,
    eligibilityScore,
    totalScore: Math.round(totalScore * 100) / 100,
    algorithmUsed: 'Multi-Factor Assessment',
    reasoning: `Weighted scoring: Education (30%), Experience (25%), Skills (25%), Eligibility (
    matchedSkillsCount: skillsMatch.matchedCount,
    matchedEligibilitiesCount: 0
  };
}
```

What it does: Implements AI-powered mathematical algorithm that scores applicants using:

- Levenshtein distance for fuzzy string matching
- Tiered scoring floors for education levels
- Weighted sum model (30% education, 25% experience, 25% skills, 20% eligibility)
- Token-based skill matching

7. Recharts, jsPDF, xlsx - Visualization and Reporting

These libraries handle data visualization and document generation.

Example 1: Recharts Line Chart

File: `src/components/charts/MonthlyApplicantsChart.tsx`

```
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer } from 'rec

export const MonthlyApplicantsChart: React.FC = () => {
  return (
    <div className="h-64">
      <ResponsiveContainer width="100%" height="100%">
        <LineChart
          data={monthlyData}
          margin={{ top: 10, right: 30, left: 0, bottom: 0 }}
        >
          <defs>
            <linearGradient id="colorApplications" x1="0" y1="0" x2="0" y2="1">
              <stop offset="5%" stopColor="#22A555" stopOpacity={0.8} />
              <stop offset="95%" stopColor="#22A555" stopOpacity={0.1} />
            </linearGradient>
          </defs>
          <CartesianGrid strokeDasharray="3 3" stroke="#e5e7eb" />
          <XAxis dataKey="month" tick={{ fontSize: 12, fill: '#6b7280' }} />
          <YAxis tick={{ fontSize: 12, fill: '#6b7280' }} allowDecimals={false} />
          <Tooltip
            contentStyle={{
              backgroundColor: 'white',
              border: '1px solid #e5e7eb',
              borderRadius: '8px',
            }}
          />
          <Line
            type="monotone"
            dataKey="applications"
            stroke="#22A555"
            strokeWidth={3}
            fill="url(#colorApplications)"
            dot={{ fill: '#22A555', r: 4 }}
          />
        </LineChart>
      </ResponsiveContainer>
    </div>
  );
};
```

What it does: Creates responsive line chart with gradient fills, custom styling, tooltips, and real-time data updates.

Example 2: jsPDF PDF Generation

File: `src/lib/pds/pdfGenerator.ts`

```
import jsPDF from 'jspdf';
import autoTable from 'jspdf-autotable';

export async function generatePDSPDF(pdsData: PDSData): Promise<void> {
  const doc = new jsPDF('p', 'mm', 'a4');
  const pageWidth = doc.internal.pageSize.getWidth();
  let yPosition = 15;

  // Header
  doc.setFontSize(16);
  doc.setFont('helvetica', 'bold');
  doc.text('PERSONAL DATA SHEET', pageWidth / 2, yPosition, { align: 'center' });
  yPosition += 10;

  // Personal Information Table
  const personalInfoData = [
    ['Full Name:', `${pdsData.surname}, ${pdsData.firstName} ${pdsData.middleName || ''}`],
    ['Date of Birth:', formatDateOnly(pdsData.dateOfBirth)],
    ['Place of Birth:', pdsData.placeOfBirth || 'N/A'],
    ['Sex:', pdsData.sex || 'N/A'],
    ['Civil Status:', pdsData.civilStatus || 'N/A'],
  ];

  autoTable(doc, {
    startY: yPosition,
    body: personalInfoData,
    theme: 'plain',
    styles: { fontSize: 9, cellPadding: 1.5 },
    columnStyles: {
      0: { fontStyle: 'bold', cellWidth: 45 },
      1: { cellWidth: 135 },
    },
  });

  yPosition = (doc as any).lastAutoTable.finalY + 8;

  // Educational Background Table
  if (pdsData.educationalBackground && pdsData.educationalBackground.length > 0) {
    doc.setFontSize(12);
    doc.setFont('helvetica', 'bold');
    doc.text('II. EDUCATIONAL BACKGROUND', 14, yPosition);
    yPosition += 6;
```

```
      const eduData = pdsData.educationalBackground.map((edu) => [
        edu.level,
        edu.schoolName,
        edu.course || 'N/A',
        `${edu.periodFrom || ''} - ${edu.periodTo || ''}`,
        edu.yearGraduated || 'N/A',
      ]);

      autoTable(doc, {
        startY: yPosition,
        head: [['Level', 'School Name', 'Course', 'Period', 'Year Graduated']],
        body: eduData,
        theme: 'grid',
        headStyles: { fillColor: [34, 165, 85], textColor: 255, fontSize: 9 },
        styles: { fontSize: 8, cellPadding: 2 },
      });

      yPosition = (doc as any).lastAutoTable.finalY + 8;
    }

    // Save PDF
    doc.save(`PDS_${pdsData.surname}_${pdsData.firstName}.pdf`);
  }
```

What it does: Generates formal PDF documents from PDS data with tables, formatted text, headers, and automatic page breaks.

Example 3: xlsx Excel Export

File: `src/app/api/applications/export/route.ts`

```
import * as XLSX from 'xlsx';

// Format application data for Excel
const excelData = applications.map((app: any) => {
  const profile = app.applicant_profiles;
  const fullName = `${profile?.first_name || ''} ${profile?.surname || ''}`.trim();

  return {
    'Application ID': app.id,
    'Applicant Name': fullName || 'Unknown',
    'Email': profile?.email || 'N/A',
    'Phone': profile?.phone_number || 'N/A',
    'Applied Position': app.jobs?.title || 'Unknown',
    'Location': app.jobs?.location || 'N/A',
    'Status': app.status,
    'Rank': app.rank || 'N/A',
    'Match Score': app.match_score ? `${app.match_score}%` : 'N/A',
    'Education Score': app.education_score ? `${app.education_score}%` : 'N/A',
    'Experience Score': app.experience_score ? `${app.experience_score}%` : 'N/A',
    'Skills Score': app.skills_score ? `${app.skills_score}%` : 'N/A',
    'Education': profile?.highest_educational_attainment || 'N/A',
    'Applied Date': new Date(app.created_at).toLocaleDateString('en-US'),
  };
});

// Create Excel workbook
const worksheet = XLSX.utils.json_to_sheet(excelData);
const workbook = XLSX.utils.book_new();
XLSX.utils.book_append_sheet(workbook, worksheet, 'Applications');

// Set column widths
const columnWidths = [
  { wch: 30 }, // Application ID
  { wch: 25 }, // Applicant Name
  { wch: 30 }, // Email
  { wch: 15 }, // Phone
  { wch: 25 }, // Applied Position
  { wch: 20 }, // Location
  { wch: 12 }, // Status
  { wch: 8 },  // Rank
  { wch: 12 }, // Match Score
];
worksheet['!cols'] = columnWidths;
```

```
// Generate Excel file
const excelBuffer = XLSX.write(workbook, { type: 'buffer', bookType: 'xlsx' });

// Return as download
const timestamp = new Date().toISOString().split('T')[0];
const filename = `JobSync_Applications_${timestamp}.xlsx`;

return new NextResponse(excelBuffer, {
  status: 200,
  headers: {
    'Content-Type': 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
    'Content-Disposition': `attachment; filename="${filename}"`,
  },
});
```

What it does: Exports application data to Excel format with:

- Formatted column headers
- Custom column widths for readability
- Timestamped filename
- HTTP download response

8. Zod & React Hook Form - Validation and Form Handling

Zod provides schema validation, React Hook Form manages complex form state.

Example 1: Zod Schema Definition

File: `src/lib/validation/profileSchema.ts`

```
import { z } from 'zod';

export const updateProfileSchema = z.object({
  full_name: z
    .string()
    .min(2, 'Name must be at least 2 characters')
    .max(100, 'Name must be less than 100 characters')
    .regex(/^[a-zA-Z\s\-\.\']+$/, 'Name can only contain letters, spaces, hyphens, dots, and apo

  email: z
    .string()
    .email('Invalid email address')
    .max(255, 'Email must be less than 255 characters'),

  phone: z
    .string()
    .regex(/^\+?[0-9\s\-\(\)]+$/, 'Invalid phone number format')
    .min(7, 'Phone number must be at least 7 characters')
    .max(20, 'Phone number must be less than 20 characters')
    .nullable()
    .optional(),

  address: z
    .string()
    .max(500, 'Address must be less than 500 characters')
    .optional(),
});

export const changePasswordSchema = z
  .object({
    currentPassword: z.string().min(1, 'Current password is required'),
    newPassword: z
      .string()
      .min(8, 'Password must be at least 8 characters')
      .regex(/[A-Z]/, 'Password must contain at least one uppercase letter')
      .regex(/[0-9]/, 'Password must contain at least one number'),
    confirmPassword: z.string(),
  })
  .refine((data) => data.newPassword === data.confirmPassword, {
    message: "Passwords don't match",
    path: ['confirmPassword'],
  });
```

```
// Type inference from schemas
export type UpdateProfileData = z.infer<typeof updateProfileSchema>;
export type ChangePasswordData = z.infer<typeof changePasswordSchema>;
```

What it does: Defines validation schemas with:

- Custom error messages
- Regex patterns for format validation
- Cross-field validation (password confirmation)
- Automatic TypeScript type generation

Example 2: React Hook Form Integration

File: `src/components/PDS/sections/PersonalInformationForm.tsx`

```
'use client';
import { useForm, Controller } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { PersonalInformation } from '@/types/pds.types';
import { personalInformationSchema } from '@/lib/pds/validation';

export const PersonalInformationForm: React.FC<Props> = ({ data, onChange }) => {
  const {
    control,
    watch,
    setValue,
    formState: { errors },
  } = useForm<PersonalInformation>({
    resolver: zodResolver(personalInformationSchema),
    defaultValues: data || {
      citizenship: 'Filipino',
      civilStatus: 'Single',
      residentialAddress: { barangay: '', cityMunicipality: '' },
    },
    mode: 'onBlur', // Validate on blur
  });

  const watchedData = watch();
  const watchCivilStatus = watch('civilStatus');

  // Auto-save when form changes
  useEffect(() => {
    const currentData = JSON.stringify(watchedData);
    if (currentData !== previousDataRef.current) {
      previousDataRef.current = currentData;
      onChange(watchedData);
    }
  }, [watchedData, onChange]);

  return (
    <div className="space-y-8">
      <h3 className="text-xl font-semibold">Personal Information</h3>

      {/* Name Fields */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
        <Controller
          name="surname"
          control={control}
          render={({ field }) => (
            <div>
              <label className="block text-sm font-medium mb-1">
                Surname <span className="text-red-500">*</span>
              </label>
              <input
                type="text"
                className="w-full px-4 py-2 border rounded-lg focus:ring-2 focus:ring-[#22A555]'
                value={field.value || ''}
                onChange={field.onChange}
                onBlur={field.onBlur}
              />
              {errors.surname && (
                <p className="text-red-500 text-sm mt-1">{errors.surname.message}</p>
              )}
            </div>
          )}
        />

        <Controller
          name="firstName"
          control={control}
          render={({ field }) => (
            <div>
              <label className="block text-sm font-medium mb-1">
                First Name <span className="text-red-500">*</span>
              </label>
              <input
                type="text"
                className="w-full px-4 py-2 border rounded-lg focus:ring-2 focus:ring-[#22A555]'
                {...field}
                value={field.value || ''}
              />
              {errors.firstName && (
                <p className="text-red-500 text-sm mt-1">{errors.firstName.message}</p>
              )}
            </div>
          )}
        />
      </div>

      {/* Conditional Fields */}
      {watchCivilStatus === 'Others' && (
        <Controller
          name="civilStatusOthers"
          control={control}
          render={({ field }) => (
            <input
              type="text"
              placeholder="Please specify"
              className="w-full px-4 py-2 border rounded-lg"
              {...field}
              value={field.value || ''}
            />
          )}
        />
      )}
    </div>
  );
};
```

What it does: Integrates React Hook Form with Zod validation:

- `useForm` with `zodResolver` for schema validation
- `Controller` for controlled inputs
- `watch` for reactive form values
- Real-time validation on blur
- Conditional field rendering based on watched values
- Auto-save to parent component
- Error message display

Summary

These code snippets demonstrate how JobSync uses modern web technologies to create a production-ready AI-powered recruitment system:

- **Next.js** provides server-side rendering and API routes
- **React** manages interactive UI with hooks and context
- **TypeScript** ensures type safety across the codebase
- **Tailwind CSS** delivers responsive, consistent styling
- **Supabase** handles authentication and database operations
- **Gemini AI** powers intelligent applicant ranking

- **Recharts, jsPDF, xlsx** generate visualizations and reports
- **Zod & React Hook Form** provide robust form validation

All examples are taken directly from the JobSync codebase and show real implementation patterns used in production.