ApnaResume - Complete Project Documentation

Table of Contents

Project Overview

ApnaResume is a professional AI-powered resume builder web application that allows users to create, edit, and share polished resumes effortlessly. The application leverages Google's Gemini AI to generate professional content suggestions for resume sections like summaries, experience descriptions, and education details.

Key Highlights

- **AI-Powered Content Generation**: Uses Google Gemini AI for intelligent content suggestions
- **Multiple Templates**: Offers Classic, Modern, and Minimal resume templates
- **Real-time Preview**: Live preview of resume changes
- **PDF Export**: Download resumes as PDF files
- **Shareable Links**: Generate and share resume URLs
- **User Authentication**: Secure authentication via Clerk
- **Drag-and-Drop Section Ordering**: Customize resume section order
- **Theme Customization**: Choose from 15 predefined theme colors

---

Technology Stack

Frontend

Technology	Version	Purpose
Next.js	14.2.35	React framework with App Router
React	18.x	UI library
TypeScript	5.x	Type-safe JavaScript
Tailwind CSS	3.4.1	Utility-first CSS framework
Radix UI	Various	Headless UI primitives
Lucide React	0.408.0	Icon library
React Hook Form	7.52.1	Form handling

Zod	3.23.8	Schema validation
React Quill	2.0.0	Rich text editor
Draft.js	0.11.7	Rich text framework
@hello-pangea/dnd	18.0.1	Drag and drop functionality

Backend & Database

Technology	Version	Purpose
MongoDB	6.8.0	NoSQL database (driver)
Mongoose	8.8.3	MongoDB ODM

Authentication

Technology	Version	Purpose
Clerk	5.7.5	User authentication & management
Clerk Themes	2.1.10	Clerk UI customization

AI Integration

Technology	Version	Purpose
@google/generative-ai	0.14.1	Google Gemini AI integration

PDF & Share

Technology	Version	Purpose
html2pdf.js	0.13.0	PDF generation
react-web-share	2.0.2	Web share API wrapper

Utilities

Technology	Version	Purpose
uuid	10.0.0	Unique ID generation
clsx	2.1.1	Conditional classnames
tailwind-merge	2.4.0	Tailwind class merging
tailwindcss-animate	1.0.7	Animation utilities
next-nprogress-bar	2.3.12	Page transition progress bar
class-variance-authority	0.7.0	Component variants

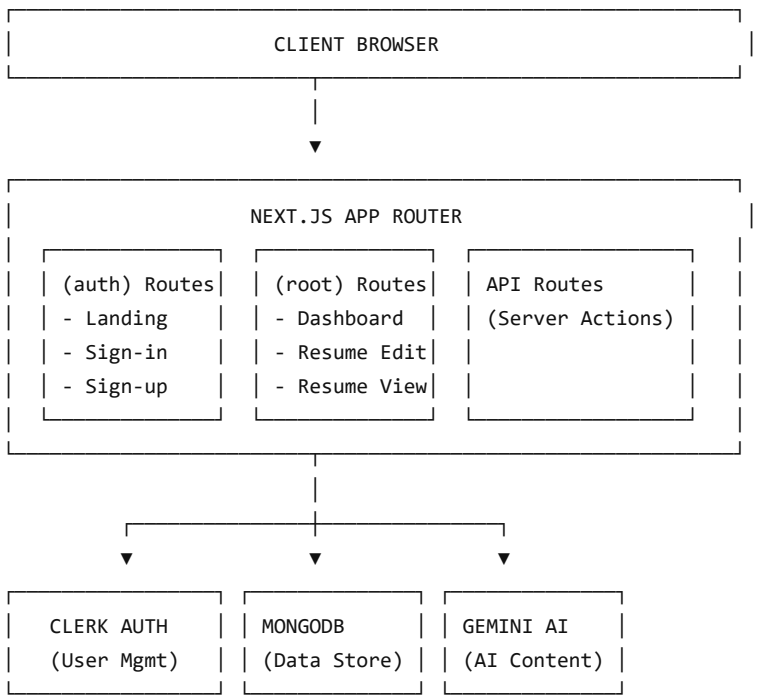
Development Tools

Technology	Version	Purpose
PostCSS	8.x	CSS processing

Docker	-	Containerization
Docker Compose	3.9	Multi-container orchestration

Project Architecture

The project follows Next.js 14 App Router architecture with a modular component structure:



Application Flow

1. **Authentication Layer**: Clerk handles user authentication with protected routes
2. **Route Groups**: `(auth)` for public pages, `(root)` for authenticated pages
3. **Server Actions**: Backend logic using Next.js Server Actions
4. **Database**: MongoDB with Mongoose ODM
5. **AI Integration**: Google Gemini for content generation

Directory Structure

```
bumblebee/
├── app/                        # Next.js App Router
│   ├── (auth)/                 # Public authentication group
│   │   ├── page.tsx            # Landing page (Home)
│   │   ├── layout.tsx          # Auth layout
│   │   ├── sign-in/            # Clerk sign-in page
│   │   └── sign-up/            # Clerk sign-up page
│   ├── (root)/                 # Protected routes group
│   │   ├── dashboard/          # User dashboard
│   │   │   └── page.tsx
│   │   └── my-resume/          # Resume management
│   │       └── [id]/           # Dynamic resume routes
│   │           ├── edit/       # Resume editor
│   │           └── view/       # Resume viewer
```

```
│   ├── globals.css              # Global styles
│   └── layout.tsx               # Root layout
│
├── components/                  # React Components
│   ├── common/                  # Shared components
│   │   ├── AddResume.tsx        # New resume dialog
│   │   ├── PageWrapper.tsx      # Page wrapper
│   │   ├── ProgressBarProvider.tsx # Progress bar
│   │   ├── ResumeCard.tsx       # Resume card display
│   │   └── RichTextEditor.tsx   # Quill editor wrapper
│   ├── layout/                  # Layout components
│   │   ├── DashboardCards.tsx   # Dashboard grid
│   │   ├── Header.tsx           # Navigation header
│   │   ├── HeaderText.tsx       # Header text
│   │   ├── ResumeView.tsx       # Resume view page
│   │   ├── TemplatePreview.tsx  # Template preview (500 lines)
│   │   ├── TemplateSelector.tsx # Template selector
│   │   ├── ThemeColor.tsx       # Theme color picker
│   │   ├── TopBar.tsx           # Top navigation bar
│   │   └── my-resume/           # Resume-specific components
│   │       ├── ResumeEditForm.tsx     # Main edit form
│   │       ├── ResumeEditor.tsx       # Editor wrapper
│   │       ├── ResumePreview.tsx      # Preview component
│   │       ├── SectionOrderBoard.tsx  # Drag-drop ordering
│   │       ├── forms/                 # Input forms
│   │       │   ├── PersonalDetailsForm.tsx  (339 lines)
│   │       │   ├── SummaryForm.tsx
│   │       │   ├── ExperienceForm.tsx
│   │       │   ├── EducationForm.tsx
│   │       │   └── SkillsForm.tsx
│   │       ├── previews/              # Section previews
│   │       │   ├── PersonalDetailsPreview.tsx
│   │       │   ├── SummaryPreview.tsx
│   │       │   ├── ExperiencePreview.tsx
│   │       │   ├── EducationalPreview.tsx
│   │       │   └── SkillsPreview.tsx
│   │       └── templates/             # Resume templates
│   │           ├── ClassicTemplate.tsx  (246 lines)
│   │           ├── ModernTemplate.tsx   (294 lines)
│   │           ├── MinimalTemplate.tsx  (266 lines)
│   │           └── index.ts
│   └── ui/                      # Radix UI components
│       ├── alert-dialog.tsx
│       ├── button.tsx
│       ├── card.tsx
│       ├── dialog.tsx
│       ├── dropdown-menu.tsx
│       ├── form.tsx
│       ├── input.tsx
│       ├── label.tsx
│       ├── popover.tsx
│       ├── textarea.tsx
│       ├── toast.tsx
│       ├── toaster.tsx
│       └── use-toast.ts
│
```

```
├── lib/                          # Core utilities
│   ├── actions/                  # Server Actions
│   │   ├── resume.actions.ts     # Resume CRUD (307 lines)
│   │   ├── gemini.actions.ts     # AI integration (61 lines)
│   │   └── user.actions.ts       # User actions
│   ├── context/                  # React Context
│   │   └── FormProvider.tsx      # Form state provider
│   ├── models/                   # Mongoose Models
│   │   ├── resume.model.ts       # Resume schema
│   │   ├── experience.model.ts   # Experience schema
│   │   ├── education.model.ts    # Education schema
│   │   └── skill.model.ts        # Skill schema
│   ├── validations/              # Zod schemas
│   │   └── resume.ts             # Resume validation
│   ├── mongoose.ts               # DB connection
│   ├── templates.ts              # Template definitions
│   ├── utils.ts                  # Utility functions
│   ├── dummy.ts                  # Dummy data
│   └── dummyResume.ts            # Sample resume data
│
├── public/                       # Static assets
│   ├── icons/                    # App icons
│   └── img/                      # Images
│
├── types/                        # TypeScript types
│
├── middleware.ts                 # Auth middleware
├── tailwind.config.ts            # Tailwind configuration
├── next.config.mjs               # Next.js configuration
├── components.json               # Shadcn/ui config
├── docker-compose.yml            # Docker orchestration
├── Dockerfile                    # Docker build
├── package.json                  # Dependencies
└── tsconfig.json                 # TypeScript config
```

Database Models

Resume Model

File: `lib/models/resume.model.ts`

```
{
  resumeId: String,        // Unique UUID identifier
  userId: String,          // Clerk user ID (owner)
  title: String,           // Resume title
  updatedAt: Date,         // Last modified timestamp
  firstName: String,       // User's first name
  lastName: String,        // User's last name
  jobTitle: String,        // Target job title
  address: String,         // Location/address
  phone: String,           // Phone number
  email: String,           // Email address
  summary: String,         // Professional summary
  socialLinks: {           // Social media links
    linkedin: String,
```

```
    twitter: String,
    portfolio: String,
    leetcode: String,
    codeforces: String,
    github: String,
    instagram: String
  },
  experience: [ObjectId], // References to Experience documents
  education: [ObjectId],  // References to Education documents
  skills: [ObjectId],     // References to Skill documents
  themeColor: String,     // Hex color code (default: "#90A4AE")
  template: String,       // Template type (default: "classic")
  sectionOrder: [String]  // Order of sections (default: ["summary", "experience", "education",
"skills"])
}
```

Experience Model

File: `lib/models/experience.model.ts`

```
{
  title: String,          // Job title
  companyName: String,    // Company name
  city: String,           // City
  state: String,          // State/Province
  startDate: String,      // Start date
  endDate: String,        // End date (empty = current)
  workSummary: String     // Work description (HTML supported)
}
```

Education Model

File: `lib/models/education.model.ts`

```
{
  universityName: String, // Institution name
  degree: String,         // Degree type
  major: String,          // Field of study
  startDate: String,      // Start date
  endDate: String,        // End date
  description: String     // Additional details
}
```

Skill Model

File: `lib/models/skill.model.ts`

```
{
  category: String,       // Skill category (required)
  skills: [String]        // Array of skills in this category
}
```

API & Server Actions

Resume Actions (`lib/actions/resume.actions.ts`)

Function	Parameters	Description
`createResume`	`resumeId, userId, title`	Creates a new resume document
`fetchResume`	`resumeId`	Fetches single resume with populated relations
`fetchUserResumes`	`userId`	Fetches all resumes for a user
`checkResumeOwnership`	`userId, resumeId`	Verifies if user owns the resume
`updateResume`	`resumeId, updates`	Updates resume fields
`addExperienceToResume`	`resumeId, experienceDataArray`	Adds/updates experiences
`addEducationToResume`	`resumeId, educationDataArray`	Adds/updates education
`addSkillToResume`	`resumeId, skillDataArray`	Adds/updates skills
`deleteResume`	`resumeId, path`	Deletes a resume

Gemini AI Actions (`lib/actions/gemini.actions.ts`)

Function	Parameters	Description
`generateSummary`	`jobTitle`	Generates 3 summary options for different experience levels
`generateEducationDescription`	`educationInfo`	Generates education descriptions at 3 activity levels
`generateExperienceDescription`	`experienceInfo`	Generates work experience descriptions at 3 activity levels

AI Configuration:

- Model: `gemini-2.5-flash`
- Temperature: 1
- TopP: 0.95
- TopK: 64
- Max Output Tokens: 8192
- Response Format: JSON

Authentication

Clerk Configuration

Middleware: `middleware.ts`

Protected Routes:

- `/dashboard` - User resume dashboard
- `/my-resume/:resumeId/edit` - Resume editing page

Public Routes:

- `/` - Landing page
- `/sign-in` - Sign in page
- `/sign-up` - Sign up page
- `/my-resume/:resumeId/view` - Resume viewing (public sharing)

Features:

- Social login buttons (bottom placement)
- Custom logo via `logoImageUrl`
- Post-logout redirect to `/`

UI Components

Radix UI Components (Shadcn/ui)

Located in `components/ui/` :

Component	File	Description
AlertDialog	`alert-dialog.tsx`	Confirmation dialogs
Button	`button.tsx`	Styled buttons with variants
Card	`card.tsx`	Card containers
Dialog	`dialog.tsx`	Modal dialogs
DropdownMenu	`dropdown-menu.tsx`	Dropdown menus
Form	`form.tsx`	Form components with React Hook Form
Input	`input.tsx`	Text inputs
Label	`label.tsx`	Form labels
Popover	`popover.tsx`	Popover containers
Textarea	`textarea.tsx`	Text areas
Toast	`toast.tsx`	Toast notifications
Toaster	`toaster.tsx`	Toast container

Custom Components

AddResume (`components/common/AddResume.tsx`)

- Dialog for creating new resumes
- Form validation with Zod
- UUID generation for resume IDs
- Redirects to edit page on creation

ResumeCard (`components/common/ResumeCard.tsx`)

- Displays resume card with gradient background
- Theme color indicator
- Dropdown menu (Edit, View, Delete)
- Delete confirmation dialog
- Loading skeleton state

Header (`components/layout/Header.tsx`)

- Sticky navigation header
- Clerk UserButton integration
- Conditional login/dashboard buttons
- Responsive design

ResumeEditForm (`components/layout/my-resume/ResumeEditForm.tsx`)

- 6-step wizard form:

1. Personal Details
2. Summary
3. Experience
4. Education
5. Skills
6. Section Order

- Navigation (Prev/Next/Finish)
- Theme and template selectors
- Batch save on finish

SectionOrderBoard (`components/layout/my-resume/SectionOrderBoard.tsx`)

- Drag-and-drop section reordering
- Uses @hello-pangea/dnd
- Default order: Summary, Experience, Education, Skills

Resume Templates

Template Types

1. **Classic** - Traditional professional layout with top border accent
2. **Modern** - Two-column layout with sidebar for skills and contact
3. **Minimal** - Clean and simple design with elegant typography

Template Structure

Each template component (`ClassicTemplate.tsx` , `ModernTemplate.tsx` , `MinimalTemplate.tsx`):

```
// Key functions
getSocialIcon(platform: string)    // Returns icon for social platform
getPlatformLabel(platform: string) // Returns label for social platform
renderSection(key: string)         // Renders individual section by key
```

Supported Social Platforms

- LinkedIn
- Twitter
- Portfolio (Website)
- LeetCode
- Codeforces
- GitHub
- Instagram

Section Order

Templates respect `sectionOrder` array for dynamic section positioning.

Styling System

Tailwind Configuration (`tailwind.config.ts`)

Color Palette (Primary):

```
50:  #eff6ff  |  500: #3b82f6
100: #dbeafe  |  600: #2563eb
200: #bfdbfe  |  700: #1d4ed8
300: #93c5fd  |  800: #1e40af
```

```
400: #60a5fa  |  900: #1e3a8a
               950: #172554
```

Theme Colors for Resumes (15 options):

```
#90A4AE | #E57373 | #64B5F6 | #81C784 | #FFF176
#FFB74D | #BA68C8 | #4DB6AC | #AED581 | #FF8A65
#A1887F | #F06292 | #7986CB | #FFD54F | #FFEB3B
```

Typography:

- Font Family: Inter (primary), Nunito (secondary)
- CSS Variables: `--font-inter`, `--font-nunito`

Features:

- Dark mode support (class-based)
- Custom animations (accordion, shimmer)
- Container max-width: 1400px

Global CSS (`app/globals.css`)

Custom Utilities:

- `.no-focus` - Removes focus ring
- `.btn-ghost` - Ghost button style
- `.no-scrollbar` - Hide scrollbar
- `.skeleton` - Loading skeleton with shimmer
- `.btn-gradient` - Animated gradient button

Rich Text Editor Styles:

- Quill toolbar styling
- Form preview HTML rendering
- List and blockquote styling

Print Styles:

- `#no-print` - Hidden on print
- `#print-area` - Full width on print

Environment Configuration

Required Environment Variables

```
# MongoDB Connection
NEXT_MONGODB_URL=mongodb://...

# Clerk Authentication
NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=pk_...
CLERK_SECRET_KEY=sk_...
NEXT_PUBLIC_CLERK_SIGN_IN_URL=/sign-in
NEXT_PUBLIC_CLERK_SIGN_UP_URL=/sign-up

# Google Gemini AI
GEMINI_API_KEY=...
```

```
# Application URL
NEXT_PUBLIC_API_URL=http://localhost:3000
```

Docker Configuration

Dockerfile (Multi-stage Build)

Stage 1: Builder

```
FROM node:20-alpine AS builder
- Install dependencies
- Copy source
- Build Next.js (standalone output)
```

Stage 2: Production Runner

```
FROM node:20-alpine AS runner
- Non-root user (appuser)
- Copy standalone build
- Copy static files
- Expose port 3000
```

docker-compose.yml

Development Service (`app-dev`):

- Port: 3000
- Hot reload with volume mounts
- Uses `.env.local`
- Command: `npm install && npm run dev`

Production Service (`app`):

- Port: 3001
- Uses built image
- NODE_ENV=production

Features

Core Features

1. **Resume Creation**
    - Quick resume creation with title
    - UUID-based unique identifiers
    - User-specific resume storage
2. **Resume Editing**
    - 6-step wizard interface
    - Live preview
    - Auto-save per section
    - Batch save on finish
3. **AI Content Generation**
    - Summary generation (3 experience levels)

- Education description generation (3 activity levels)
- Experience description generation (3 activity levels)

4. **Template Customization**
   - 3 professional templates
   - 15 theme color options
   - Drag-and-drop section ordering
5. **Export & Sharing**
   - PDF download (html2pdf.js)
   - Web share URL
   - Public view links

UI/UX Features

1. **Responsive Design**
   - Mobile-first approach
   - Adaptive grid layouts
   - Responsive navigation
2. **Loading States**
   - Skeleton loading cards
   - Button loading spinners
   - Progress bar for navigation
3. **Notifications**
   - Toast notifications
   - Success/error feedback
   - Form validation messages
4. **Accessibility**
   - Focus management
   - ARIA attributes (via Radix)
   - Semantic HTML

Data Flow

Resume Creation Flow

```
User clicks "+" → AddResume Dialog → Enter Title →
    ↓
createResume() Server Action → MongoDB Insert →
    ↓
Redirect to /my-resume/[id]/edit
```

Resume Edit Flow

```
Load /my-resume/[id]/edit → FormProvider fetches resume →
    ↓
Display PersonalDetailsForm → User edits →
    ↓
handleInputChange() updates context → Live preview updates →
    ↓
```

```
Click "Next" → Move to next form step →
    ↓
Click "Finish" → Batch save all sections via server actions
```

AI Generation Flow

```
User enters job title/info → Click AI generate button →
    ↓
gemini.actions.ts sends prompt → Gemini API responds →
    ↓
Parse JSON response → Display 3 options →
    ↓
User selects option → Update form field
```

PDF Export Flow

```
Click "Download PDF" → html2pdf.js processes #print-area →
    ↓
Generate canvas at 2x scale → Convert to PDF →
    ↓
Trigger download (A4 portrait)
```

Development Guidelines

File Naming Conventions

- Components: PascalCase (e.g., `ResumeCard.tsx`)
- Server actions: camelCase with `.actions.ts` suffix
- Models: camelCase with `.model.ts` suffix
- Utilities: camelCase (e.g., `utils.ts`)

Component Structure

```tsx
"use client"; // For client components

import { ... } from "...";

interface Props { ... }

const Component = ({ prop }: Props) => {
  // Hooks
  // State
  // Effects
  // Handlers
  // Render
};

export default Component;
```

Server Action Pattern

```tsx
"use server";
```

```
import { connectToDB } from "../mongoose";
import Model from "../models/model";

export async function actionName(params) {
  try {
    await connectToDB();
    // Database operations
    return { success: true, data: JSON.stringify(result) };
  } catch (error) {
    return { success: false, error: error.message };
  }
}
```

Form Validation Pattern

```
import { z } from "zod";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";

const schema = z.object({ ... });

const form = useForm({
  resolver: zodResolver(schema),
  defaultValues: { ... }
});
```

Scripts

```
{
  "dev": "next dev",       // Development server
  "build": "next build",   // Production build
  "start": "next start",   // Production server
  "lint": "next lint"      // ESLint
}
```

Deployment Options

1. **Vercel** (Recommended for Next.js)
   - Connect GitHub repository
   - Auto-deploy on push
2. **Docker**
   - Build: `docker-compose build app`
   - Run: `docker-compose up app`
   - Dev: `docker-compose up app-dev`
3. **Self-hosted**
   - `npm run build`
   - `npm run start`

Version Information

- **Project Name**: ApnaResume
- **Version**: 0.1.0
- **License**: MIT
- **Node.js**: 20.x (Alpine)
- **Next.js**: 14.2.35

This documentation provides a comprehensive overview of the ApnaResume project. For specific implementation details, refer to the source code files referenced throughout this document.