🏗️ CareerPilot System Architecture

Project: CareerPilot - AI-Powered Career Path Advisor
Architecture Style: Microservices-Ready Monolith
Version: 1.0.0

📋 Table of Contents

🌐 High-Level Overview

```
┌─────────────────────────────────────────────────────────────────┐
│                          USER LAYER                             │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐          │
│  │   Browser    │  │   Mobile     │  │   Desktop    │          │
│  └──────────────┘  └──────────────┘  └──────────────┘          │
└──────────────────────────┬──────────────────────────────────────┘
                           │ HTTPS
                           ▼
┌─────────────────────────────────────────────────────────────────┐
│                     PRESENTATION LAYER                          │
│                  ┌─────────────────────┐                        │
│                  │   React Frontend    │                        │
│                  │  (Vite + Tailwind)│                        │
│                  └─────────┬───────────┘                        │
│                            │ REST API                            │
└────────────────────────────┼─────────────────────────────────────┘
                             │
                             ▼
┌─────────────────────────────────────────────────────────────────┐
│                      APPLICATION LAYER                          │
│  ┌───────────────────────────────────────────────────────┐     │
│  │              Express.js API Server                     │     │
│  │  ┌─────────────────────────────────────────────────┐  │     │
│  │  │             Middleware Stack                     │  │     │
│  │  │  • CORS  • Auth  • Validation  • Error Handler │  │     │
│  │  └─────────────────────────────────────────────────┘  │     │
│  │                                                          │     │
│  │  ┌──────────┐  ┌──────────┐  ┌──────────┐            │     │
│  │  │Controllers│  │ Services │  │    AI     │           │     │
│  │  └───────────┘  └──────────┘  └───────────┘          │     │
│  └───────────────────────────────────────────────────────┘     │
└───────────────────┬───────────────────┬────────────────────────┘
                    │                   │
                    │                   │
           ┌────────┼───────────────────┼──────────────┐
           │        │                   │              │
           ▼        ▼                   ▼              ▼
┌──────────┐  ┌──────────┐  ┌─────────────────┐  ┌──────────┐
│PostgreSQL│  │ OpenAI   │  │  File Storage   │  │  Redis   │
│ Database │  │   API    │  │    (Local/S3)   │  │  Cache   │
└──────────┘  └──────────┘  └─────────────────┘  └──────────┘
```

🧩 System Components

1. Frontend (React + Vite)

Purpose: User interface and client-side logic

Key Responsibilities:

- Render UI components
- Handle user interactions
- Manage client-side state
- Make API requests
- Display real-time feedback

Technology Stack:

```
React 18          → UI Library
Vite              → Build tool & dev server
React Router v7   → Client-side routing
TailwindCSS v4    → Styling framework
Recharts          → Data visualization
Axios             → HTTP client
React Icons       → Icon library
```

Folder Architecture:

```
src/
├── components/       # Reusable UI components
│   ├── common/       # Shared components (Button, Card, etc.)
│   ├── auth/         # Authentication components
│   ├── dashboard/    # Dashboard widgets
│   ├── career/       # Career-related components
│   └── chat/         # Chat interface
├── pages/            # Route-based page components
├── hooks/            # Custom React hooks
├── services/         # API communication layer
├── context/          # React Context providers
├── utils/            # Helper functions
└── assets/           # Static files
```

2. Backend (Node.js + Express)

Purpose: Business logic, API endpoints, data processing

Architecture Pattern: MVC + Service Layer

Layer Structure:

```
┌─────────────────────────────────────────────┐
│             Routes Layer                    │
│  (URL mapping, request routing)             │
└─────────────────────────────────────────────┘
               │
               ▼
┌─────────────────────────────────────────────┐
│          Controllers Layer                  │
│  (Request handling, validation, response)   │
└─────────────────────────────────────────────┘
               │
               ▼
┌─────────────────────────────────────────────┐
│           Services Layer                    │
│  (Business logic, data processing)          │
└─────────────────────────────────────────────┘
               │
               ▼
┌─────────────────────────────────────────────┐
│            Models Layer                     │
│  (Database schema, ORM models)              │
└─────────────────────────────────────────────┘
               │
               ▼
          PostgreSQL
```

Backend Modules:

```
src/
├── controllers/      # Request handlers
│   ├── authController.js
│   ├── userController.js
│   ├── cvController.js
│   ├── careerController.js
│   └── chatController.js
│
├── routes/           # API route definitions
│   ├── authRoutes.js
│   ├── userRoutes.js
│   └── ...
│
├── services/         # Business logic
│   ├── authService.js
│   ├── cvService.js
│   ├── careerService.js
│   └── progressService.js
│
├── models/           # Sequelize models
```

```
│   ├── User.js
│   ├── Profile.js
│   ├── CV.js
│   └── ...
│
├── ai/              # AI integration
│   ├── openaiClient.js
│   ├── careerAdvisor.js
│   ├── roadmapGenerator.js
│   └── chatbot.js
│
├── cv_parser/       # CV processing
│   ├── pdfParser.js
│   ├── skillExtractor.js
│   └── experienceParser.js
│
└── utils/           # Utilities
    ├── jwtHelper.js
    ├── validators.js
    └── errorHandler.js
```

---

3. Database Layer (PostgreSQL)

Purpose: Persistent data storage

Design Principles:

- Normalized schema (3NF)
- Foreign key constraints
- Indexed for performance
- JSONB for flexible data

Core Tables:

```
users → profiles → user_skills ← skills
  ↓                                  ↓
 cvs                            career_paths
  ↓                                  ↓
progress                          roadmaps
  ↓                                  ↓
chat_messages                   roadmap_tasks
```

See DATABASE.md for complete schema.

---

4. AI Integration Layer

Purpose: Intelligent career guidance using LLMs

Components:

OpenAI Client

```
// ai/openaiClient.js
- Initialize OpenAI API
- Manage API keys
- Handle rate limiting
- Error handling
```

Career Advisor

```
// ai/careerAdvisor.js
- Analyze user skills
- Match to career paths
- Generate recommendations
- Calculate match scores
```

Roadmap Generator

```
// ai/roadmapGenerator.js
- Create learning plans
- Structure by phases
- Generate weekly tasks
- Recommend resources
```

Chatbot

```
// ai/chatbot.js
- Contextual conversations
- Career Q&A
- Progress feedback
- Motivational support
```

AI Workflow:

```
User Input → Context Gathering → Prompt Engineering → OpenAI API
                                                          ↓
Response Processing ← JSON Parsing ← AI Response ←--------┘
         ↓
    Store in DB
```

📐 Architecture Patterns

1. **MVC Pattern** (Controllers-Services-Models)

```
Request → Route → Controller → Service → Model → Database
                     ↓            ↓
                  Validation   Business Logic
                     ↓
                  Response
```

2. Repository Pattern (Data Access)

```
// models/repositories/UserRepository.js
class UserRepository {
  async findByEmail(email) {
    return await User.findOne({ where: { email } });
  }

  async create(userData) {
    return await User.create(userData);
  }
}
```

3. Dependency Injection

```
// Inject services into controllers
class CareerController {
  constructor(careerService, aiService) {
    this.careerService = careerService;
    this.aiService = aiService;
  }
}
```

4. Middleware Pipeline

```
app.use(cors());
app.use(express.json());
app.use(authMiddleware);
app.use(validationMiddleware);
app.use(errorHandler);
```

🔄 Data Flow

Example: Get Career Recommendations

```
┌─────────┐
│ User    │ Click "Get Recommendations"
└────┬────┘
     │
     ▼
┌─────────────────┐
│ React Component │ onClick → careerService.getRecommendations()
└────┬────────────┘
     │ HTTP POST /api/career/recommend
     ▼
┌──────────────────┐
│ Express Router   │ Route to careerController.recommend()
```

```
└────┬────────────┘
     │
     ▼
┌─────────────────┐
│ Auth Middleware │ Verify JWT token → Extract user ID
└────┬────────────┘
     │
     ▼
┌─────────────────────┐
│ Career Controller   │ Validate request → Call service
└────┬────────────────┘
     │
     ▼
┌─────────────────────┐
│ Career Service      │ 1. Fetch user profile & skills
└────┬────────────────┘    2. Get career paths from DB
     │                     3. Calculate matches
     ▼
┌─────────────────────┐
│ AI Service          │ Send to OpenAI for analysis
└────┬────────────────┘
     │
     ▼
┌─────────────────────┐
│ OpenAI API          │ Generate recommendations
└────┬────────────────┘
     │
     ▼
┌─────────────────────┐
│ Career Service      │ Process AI response
└────┬────────────────┘    Format data
     │                     Store in cache
     ▼
┌─────────────────────┐
│ Career Controller   │ Return JSON response
└────┬────────────────┘
     │ HTTP 200 + JSON
     ▼
┌─────────────────┐
│ React Component │ Update state → Render results
└─────────────────┘
```

🔐 Authentication Flow

```
┌─────────┐
│  User   │ Enter email + password
└────┬────┘
     │
     ▼
┌─────────────────┐
```

```
│ Login Component │ POST /api/auth/login
└────┬────────────┘
     │
     ▼
┌─────────────────┐
│ Auth Controller │ Validate credentials
└────┬────────────┘
     │
     ▼
┌─────────────────┐
│  Auth Service   │ 1. Find user by email
└────┬────────────┘    2. Compare password hash
     │                 3. Generate JWT token
     ▼
┌─────────────────┐
│  User Model     │ Query database
└────┬────────────┘
     │
     ▼
┌─────────────────┐
│  JWT Helper     │ Sign token with secret
└────┬────────────┘    Set expiration (7 days)
     │
     ▼
┌─────────────────┐
│ Auth Controller │ Return { user, token }
└────┬────────────┘
     │ HTTP 200 + JWT
     ▼
┌─────────────────┐
│ Login Component │ 1. Store token in localStorage
└────┬────────────┘    2. Set axios default header
     │                 3. Redirect to dashboard
     ▼
┌─────────────────┐
│   Dashboard     │ All future requests include:
└─────────────────┘    Authorization: Bearer <token>
```

Protected Route Middleware:

```
// middleware/authMiddleware.js
async function authenticateToken(req, res, next) {
  const token = req.headers.authorization?.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'No token provided' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = await User.findByPk(decoded.id);
    next();
  } catch (error) {
    return res.status(401).json({ error: 'Invalid token' });
  }
}
```

🤖 AI Integration Architecture

Prompt Engineering Strategy

```
// ai/careerAdvisor.js
function buildRecommendationPrompt(userProfile, userSkills, careerPaths) {
  return `
You are an expert career advisor. Analyze this profile:

USER PROFILE:
- Skills: ${userSkills.join(', ')}
- Experience: ${userProfile.experience}
- Education: ${userProfile.education}
- Goals: ${userProfile.careerGoals}

AVAILABLE CAREER PATHS:
${careerPaths.map(c => `- ${c.title}: ${c.description}`).join('\n')}

TASK:
Recommend the top 3 most suitable career paths.
For each, provide:
1. Match score (0-100)
2. Reasoning
3. Missing skills
4. Recommended certifications

Return as JSON.
`;
}
```

AI Response Processing

```
// ai/careerAdvisor.js
async function getRecommendations(userId) {
  // 1. Gather context
  const user = await User.findByPk(userId, { include: [Profile, Skills] });
  const careerPaths = await CareerPath.findAll();

  // 2. Build prompt
  const prompt = buildRecommendationPrompt(user.profile, user.skills, careerPaths);

  // 3. Call OpenAI
  const response = await openai.chat.completions.create({
    model: 'gpt-4o-mini',
    messages: [
      { role: 'system', content: 'You are a career advisor.' },
      { role: 'user', content: prompt }
    ],
    response_format: { type: 'json_object' }
  });

  // 4. Parse and validate
  const recommendations = JSON.parse(response.choices[0].message.content);

  // 5. Enrich with database data
  return enrichRecommendations(recommendations, careerPaths);
}
```

📤 File Upload Flow

CV Upload Process

```
┌──────────┐
│   User   │ Select PDF file
└────┬─────┘
     │
     ▼
┌─────────────────┐
│ CV Upload UI    │ FormData with file
└────┬────────────┘
     │ POST /api/cv/upload (multipart/form-data)
     ▼
┌─────────────────┐
│ Multer Middleware│ 1. Validate file type (PDF/DOCX)
└────┬────────────┘    2. Check file size (<5MB)
     │                 3. Generate unique filename
     ▼
┌─────────────────┐
│ CV Controller   │ Receive file info
└────┬────────────┘
     │
     ▼
┌─────────────────┐
│ CV Service      │ 1. Save file to disk/S3
└────┬────────────┘    2. Create DB record
     │                 3. Return CV metadata
     ▼
┌─────────────────┐
│ CV Parser       │ BACKGROUND JOB:
└────┬────────────┘    1. Extract text from PDF
     │                 2. Parse sections
     ▼                 3. Extract skills
┌─────────────────┐    4. Identify experience
│ AI Skill Extractor│  5. Store parsed data
```

```
└──────────────────┘
     │
     │
     ▼
┌──────────────────┐
│    Database      │ Update CV record with parsed_data
└──────────────────┘
```

Multer Configuration:

```
// middleware/upload.js
const storage = multer.diskStorage({
  destination: './uploads/cvs/',
  filename: (req, file, cb) => {
    const uniqueName = `${req.user.id}_${Date.now()}_${file.originalname}`;
    cb(null, uniqueName);
  }
});

const upload = multer({
  storage,
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB
  fileFilter: (req, file, cb) => {
    if (file.mimetype === 'application/pdf' ||
        file.mimetype === 'application/vnd.openxmlformats-
officedocument.wordprocessingml.document') {
      cb(null, true);
    } else {
      cb(new Error('Only PDF and DOCX files allowed'));
    }
  }
});
```

🚀 Deployment Architecture

Development Environment

```
┌─────────────────────────────────────┐
│       Developer Machine             │
│                                     │
│  ┌──────────┐    ┌──────────┐      │
│  │ Frontend │    │ Backend  │      │
│  │ :5173    │    │  :5000   │      │
│  └──────────┘    └──────────┘      │
│         │              │            │
│         └──────────────┘            │
│                │                    │
│                ▼                    │
│        ┌─────────────────┐          │
│        │   PostgreSQL    │          │
│        │     :5432       │          │
│       └──────────────────┘                │
└───────────────────────────────────────┘
```

Production Environment (Cloud)

```
                     Internet
                        │
                        ▼
              ┌──────────────────┐
              │   CDN/Cloudflare│
              │   (Static Assets)│
              └────────┬─────────┘
                       │
         ┌─────────────┼─────────────┐
         │             │             │
         ▼             ▼             ▼
┌──────────────┐ ┌───────────┐ ┌───────────┐
│   Vercel     │ │  Render   │ │    S3     │
│  (Frontend)  │ │ (Backend)│ │  (Files)  │
└──────────────┘ └─────┬─────┘ └───────────┘
                       │
         ┌─────────────┼─────────────┐
         │             │             │
         ▼             ▼             ▼
┌──────────┐   ┌──────────┐   ┌──────────┐
│PostgreSQL│   │  Redis   │   │ OpenAI   │
│  (Neon)  │   │ (Upstash)│   │  API     │
└──────────┘   └──────────┘   └──────────┘
```

Hosting Options:

Component	Service	Alternative
Frontend	Vercel	Netlify, Cloudflare Pages
Backend	Render	Railway, Fly.io
Database	Neon	Supabase, Railway
File Storage	AWS S3	Cloudflare R2, DigitalOcean Spaces
Cache	Upstash Redis	Redis Cloud

🔒 Security Architecture

Security Layers

```
┌─────────────────────────────────────────────┐
│         Application Security                │
├─────────────────────────────────────────────┤
│ • JWT Authentication                        │
│ • Password Hashing (bcrypt)                  │
│ • Input Validation & Sanitization            │
│ • SQL Injection Prevention (ORM)             │
│ • XSS Protection (React escaping)            │
│ • CSRF Protection (SameSite cookies)         │
│ • Rate Limiting                              │
│ • CORS Configuration                         │
└──────────────────────────────────────────────┘

┌──────────────────────────────────────────────┐
│         Transport Security                   │
├──────────────────────────────────────────────┤
│ • HTTPS/TLS 1.3                              │
│ • Secure Headers (Helmet.js)                 │
│ • Content Security Policy                    │
└──────────────────────────────────────────────┘

┌──────────────────────────────────────────────┐
│         Data Security                        │
├──────────────────────────────────────────────┤
│ • Encrypted Database Connections             │
│ • Sensitive Data Encryption at Rest          │
│ • Secure File Storage                        │
│ • Environment Variable Management            │
└──────────────────────────────────────────────┘
```

Security Implementation

```javascript
// Security middleware stack
app.use(helmet()); // Security headers
app.use(cors({
  origin: process.env.FRONTEND_URL,
  credentials: true
}));
app.use(rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
}));
app.use(express.json({ limit: '10mb' }));
app.use(sanitize()); // Input sanitization
```

📊 Performance & Scaling

Caching Strategy

```
┌──────────────────────────────────────────┐
│          Caching Layers                  │
├──────────────────────────────────────────┤
│ Browser Cache (Static assets: 1 year)   │
│         ↓                                │
│ CDN Cache (Images, CSS, JS: 1 month)    │
│         ↓                                │
│ Redis Cache (API responses: 5 min)      │
│         ↓                                │
│ Database Query Cache                     │
└──────────────────────────────────────────┘
```

Redis Cache Implementation:

```
// services/cacheService.js
async function getCachedRecommendations(userId) {
  const cacheKey = `recommendations:${userId}`;
  const cached = await redis.get(cacheKey);

  if (cached) {
    return JSON.parse(cached);
  }

  const recommendations = await generateRecommendations(userId);
  await redis.setex(cacheKey, 300, JSON.stringify(recommendations)); // 5 min
  return recommendations;
}
```

Database Optimization

- Connection pooling (max 20 connections)
- Query optimization with indexes
- Prepared statements (SQL injection prevention)
- Pagination for large datasets
- Lazy loading relationships

API Performance

- Response compression (gzip)
- Request payload limits
- Async/await for concurrent operations
- Background jobs for heavy tasks (CV parsing)
- API response caching

🔮 Future Architecture Enhancements

Microservices Migration

```
Current: Monolith
Future:
┌────────────┐  ┌────────────┐  ┌────────────┐
│   Auth     │  │   Career   │  │     AI     │
│  Service   │  │  Service   │  │  Service   │
└────────────┘  └────────────┘  └────────────┘
      │                │                │
```

```
        └───────────────┼───────────────┘
                        │
                  ┌──────────┐
                  │ API      │
                  │ Gateway  │
                  └──────────┘
```

Event-Driven Architecture

```
User Action → Event Bus (RabbitMQ/Kafka)
                   ↓
         ┌───────────┼───────────┐
         ▼           ▼           ▼
   CV Parser   AI Service   Email Service
```

Last Updated: November 18, 2025