

CareerPilot System Architecture

Project: CareerPilot - AI-Powered Career Path Advisor

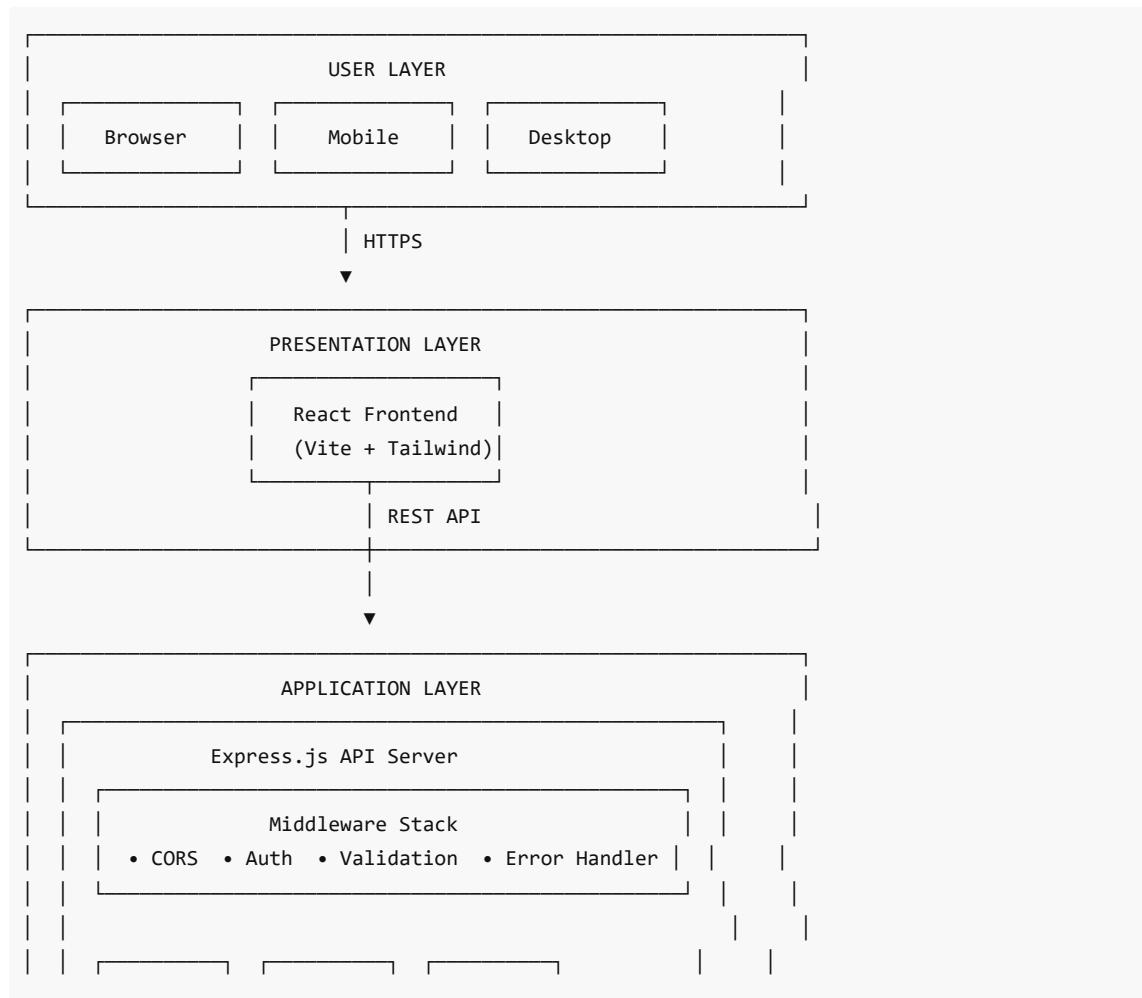
Architecture Style: Microservices-Ready Monolith

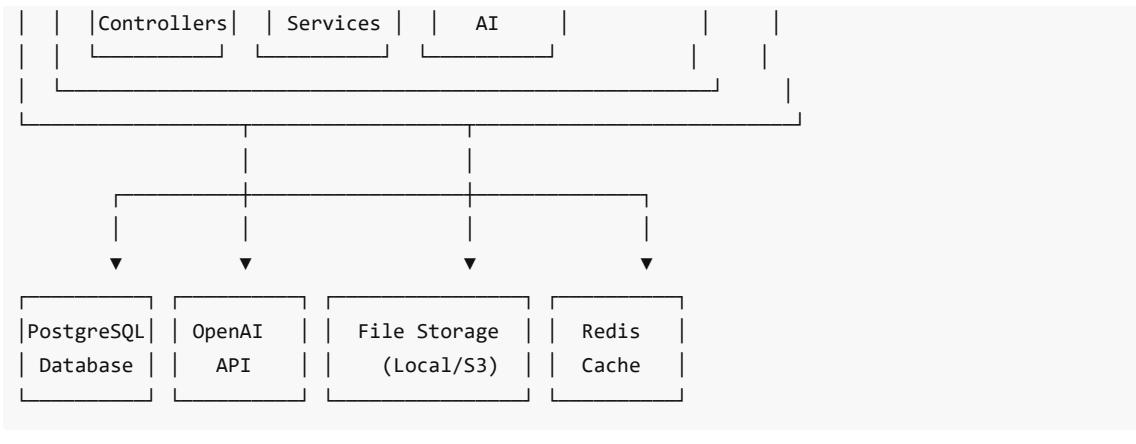
Version: 1.0.0

Table of Contents

- [High-Level Overview](#)
 - [System Components](#)
 - [Architecture Patterns](#)
 - [Data Flow](#)
 - [Authentication Flow](#)
 - [AI Integration Architecture](#)
 - [File Upload Flow](#)
 - [Deployment Architecture](#)
 - [Security Architecture](#)
 - [Performance & Scaling](#)
-

High-Level Overview





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:

```

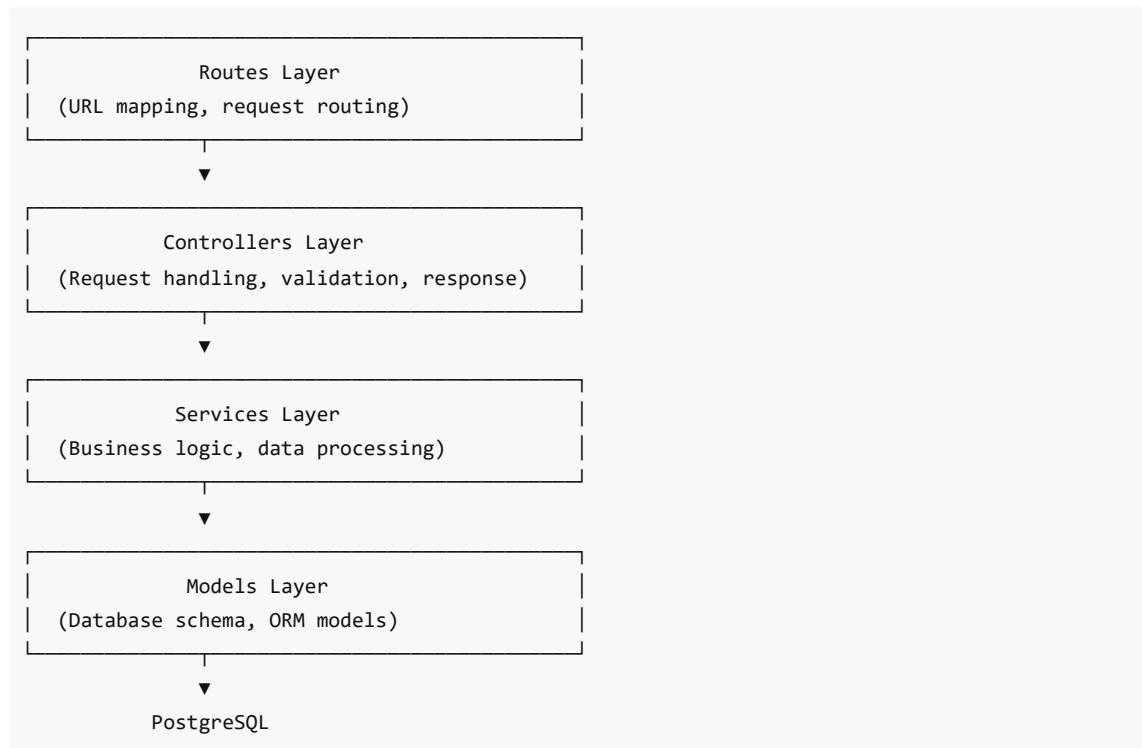
src/
├── components/      # Reusable UI components
│   ├── common/      # Shared components (Button, Card, etc.)
│   ├── auth/        # Authentication components
│   ├── dashboard/   # Dashboard widgets
│   ├── career/      # Career-related components
│   └── chat/        # Chat interface
├── pages/           # 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:



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
```

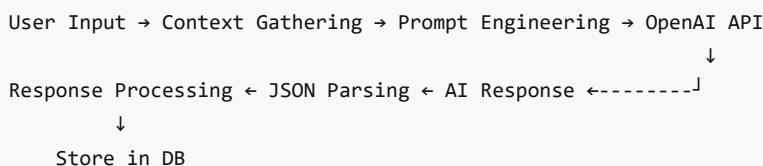
Roadmap Generator

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

Chatbot

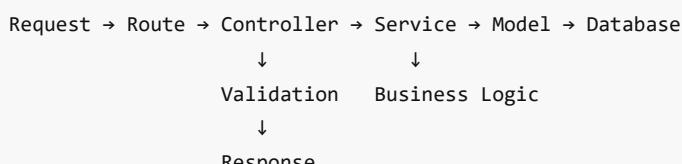
```
// ai/chatbot.js
- Contextual conversations
- Career Q&A
- Progress feedback
- Motivational support
```

AI Workflow:



Architecture Patterns

1. MVC Pattern (Controllers-Services-Models)



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

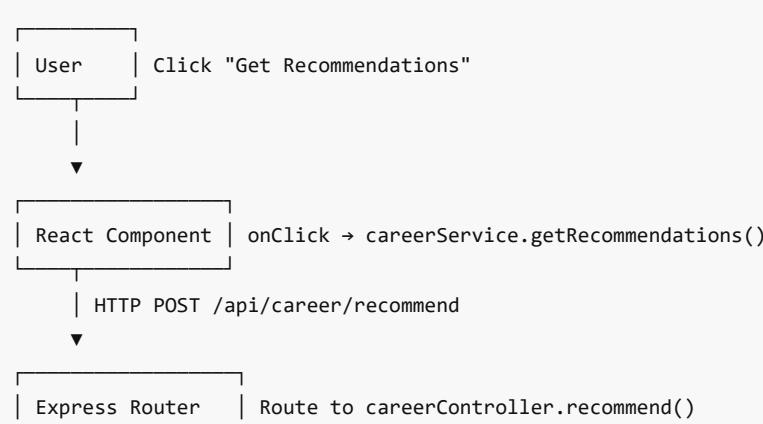
```
// Inject services into controllers
class CareerController {
  constructor(careerService, aiService) {
    this.careerService = careerService;
    this.aiService = aiService;
  }
}
```

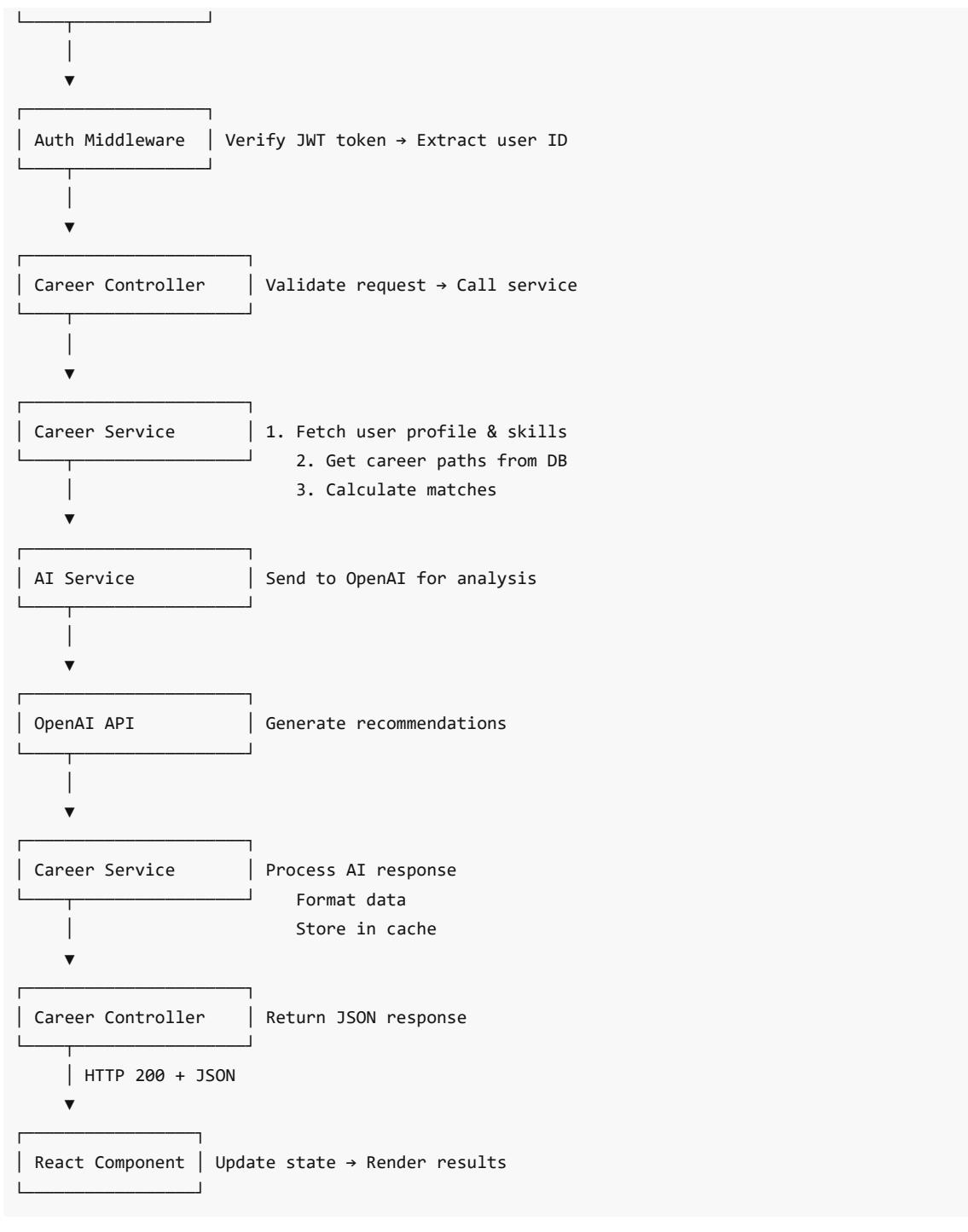
4. Middleware Pipeline

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

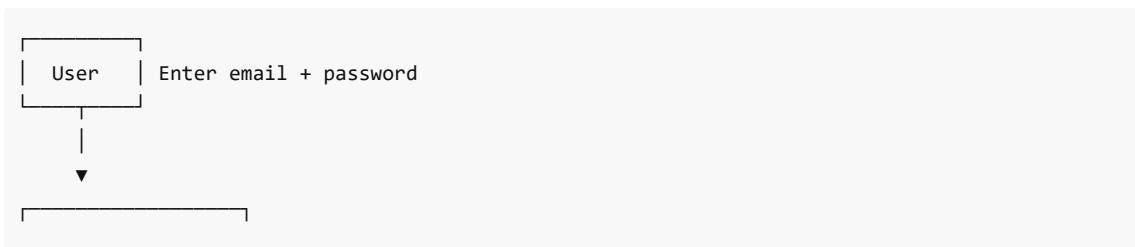
Data Flow

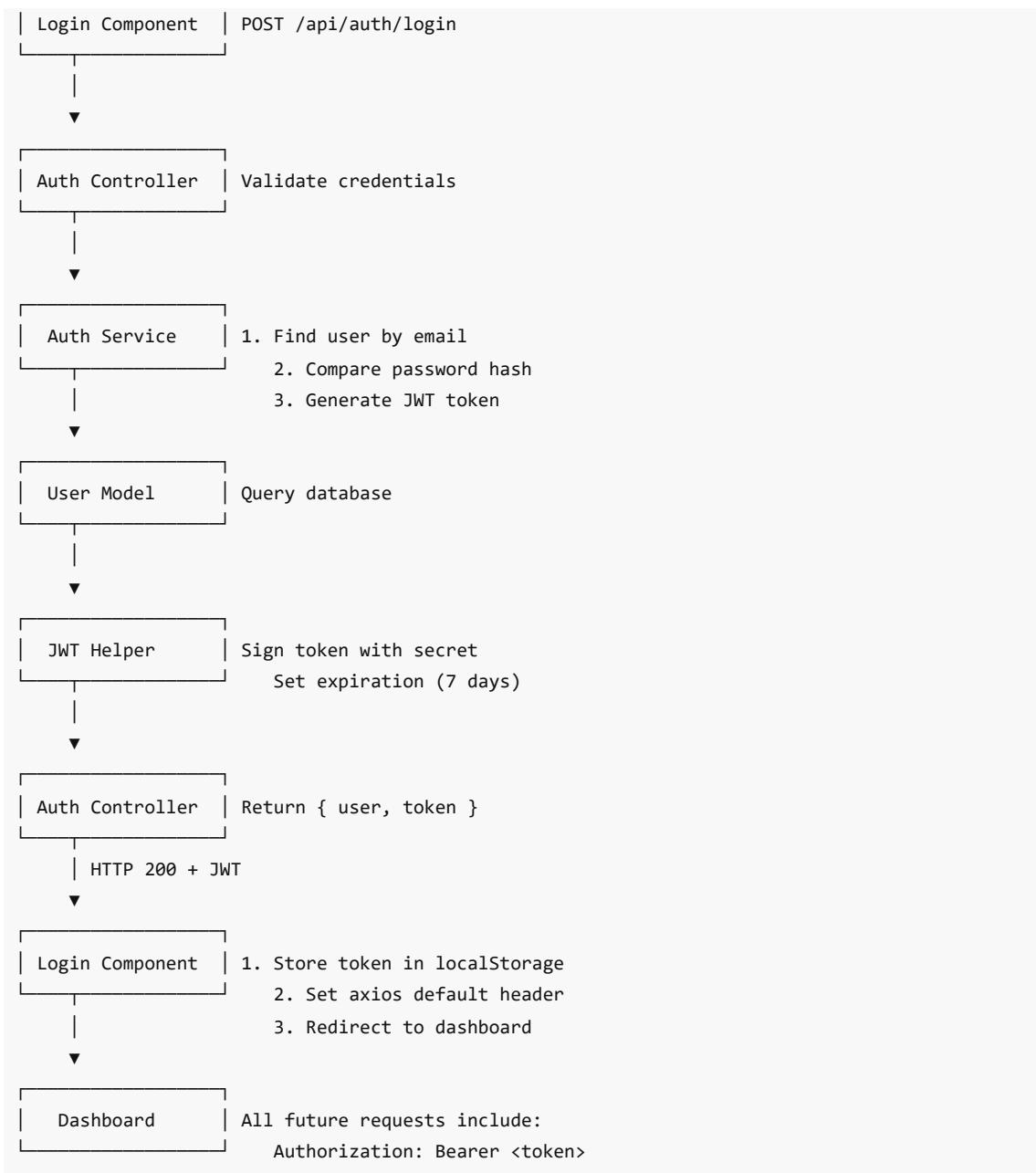
Example: Get Career Recommendations





🔒 Authentication Flow





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.findById(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.findById(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' }
  });

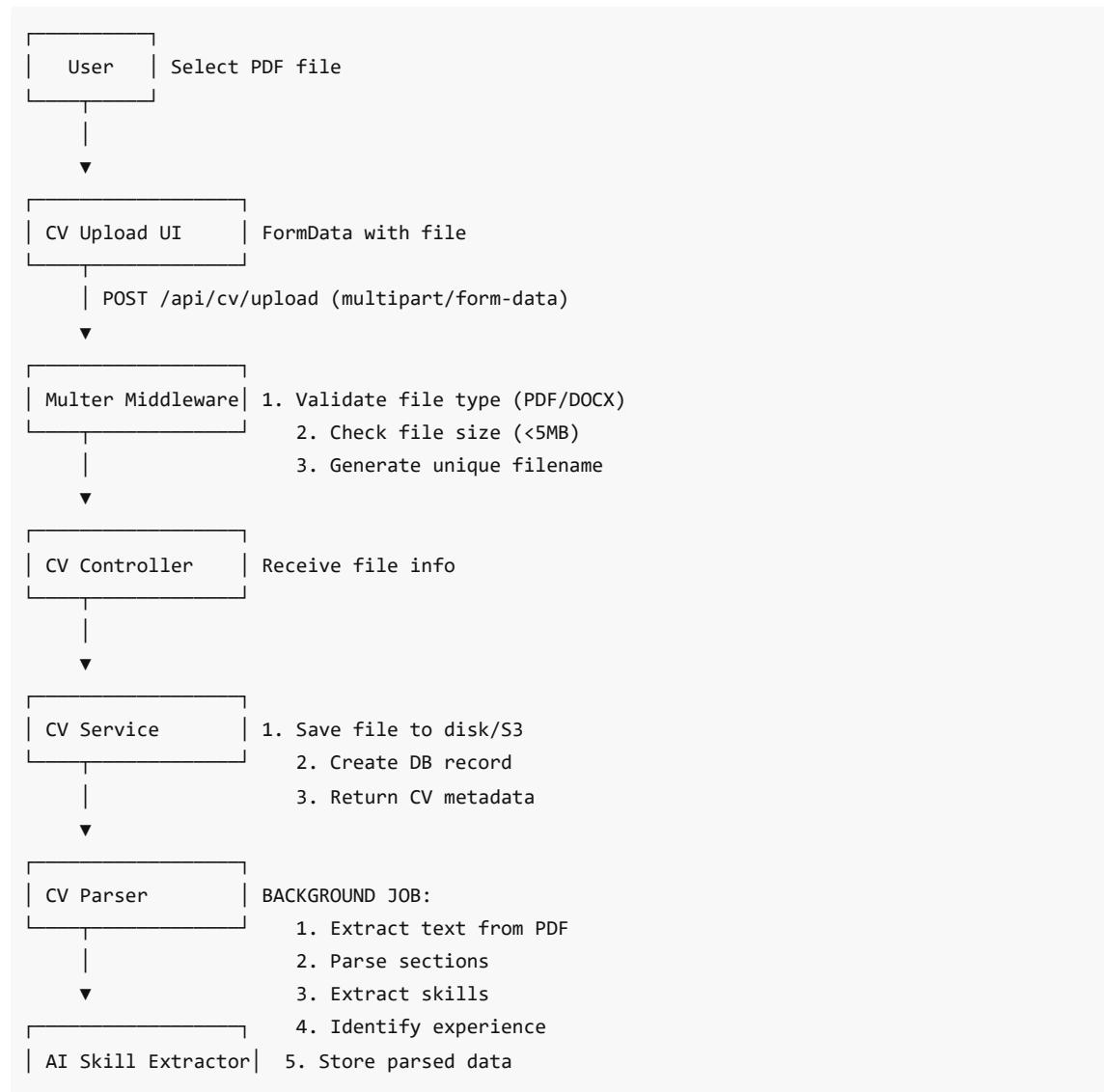
// 4. Parse and validate
const recommendations = JSON.parse(response.choices[0].message.content);

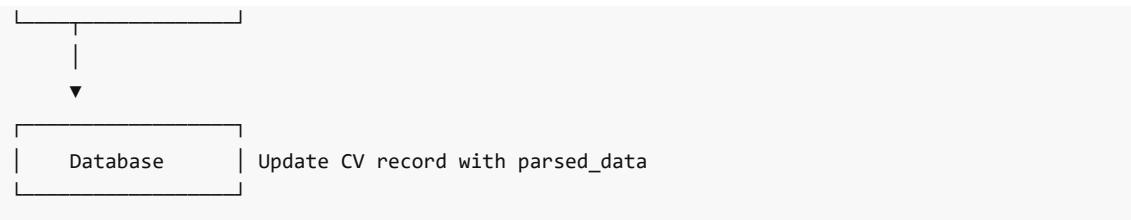
// 5. Enrich with database data
return enrichRecommendations(recommendations, careerPaths);
}

```

File Upload Flow

CV Upload Process





Multer Configuration:

```

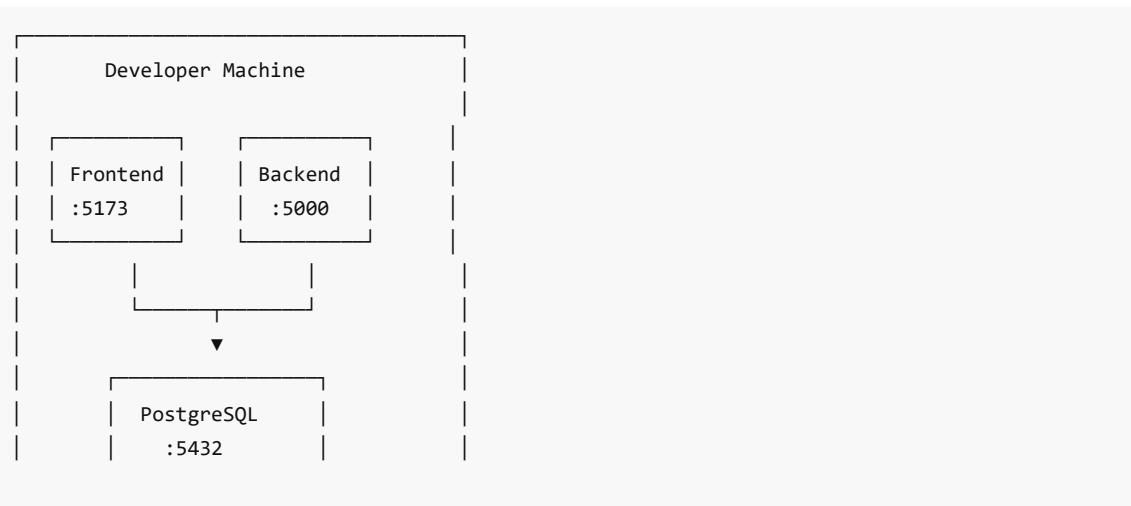
// middleware/upload.js
const storage = multer.diskStorage({
  destination: './uploads/cvs/',
  filename: (req, file, cb) => {
    const uniqueName = `${req.user.id}_${Date.now()}_${file.originalname}`;
    cb(null, uniqueName);
  }
});

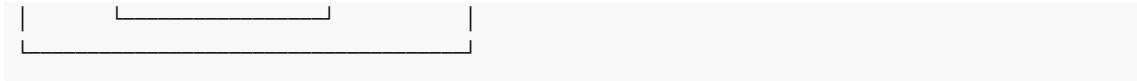
const upload = multer({
  storage,
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB
  fileFilter: (req, file, cb) => {
    if (file.mimetype === 'application/pdf' ||
        file.mimetype === 'application/vnd.openxmlformats-
officedocument.wordprocessingml.document') {
      cb(null, true);
    } else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

```

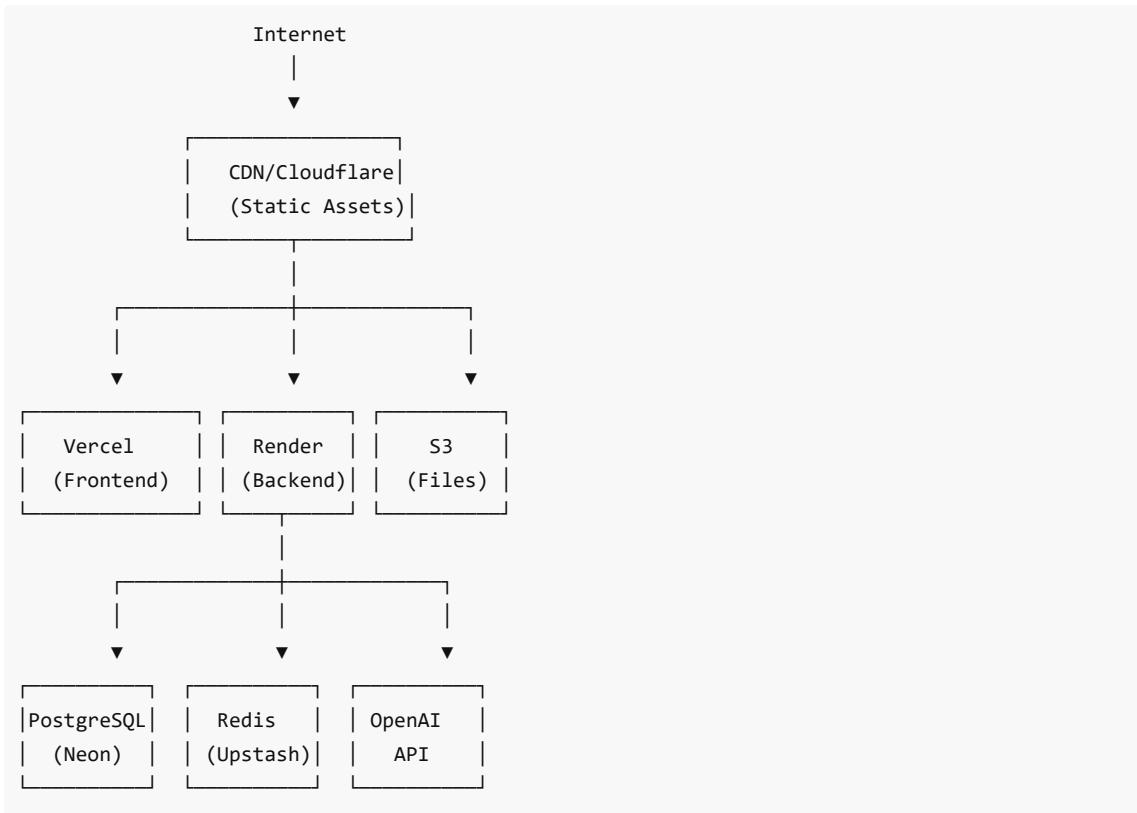
🚀 Deployment Architecture

Development Environment





Production Environment (Cloud)

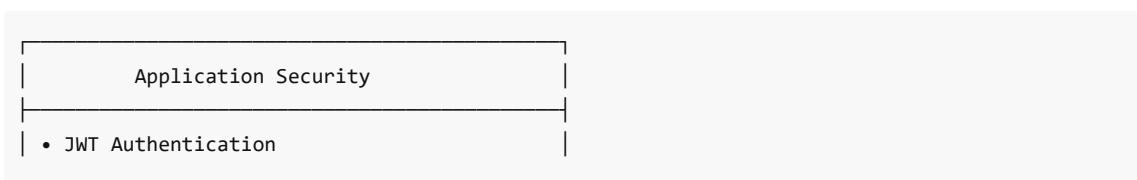


Hosting Options:

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



- 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

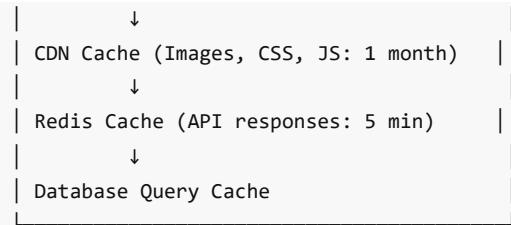
```
// Security middleware stack
app.use(helmet()); // Security headers
app.use(cors({
  origin: process.env.FRONTEND_URL,
  credentials: true
}));
app.use(rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
}));
app.use(express.json({ limit: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

Performance & Scaling

Caching Strategy

Caching Layers

- Browser Cache (Static assets: 1 year)



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

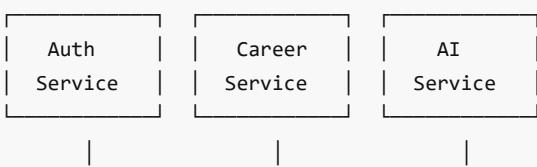
- Response compression (gzip)
- Request payload limits
- Async/await for concurrent operations
- Background jobs for heavy tasks (CV parsing)
- API response caching

🔮 Future Architecture Enhancements

Microservices Migration

Current: Monolith

Future:





Event-Driven Architecture



Last Updated: November 18, 2025