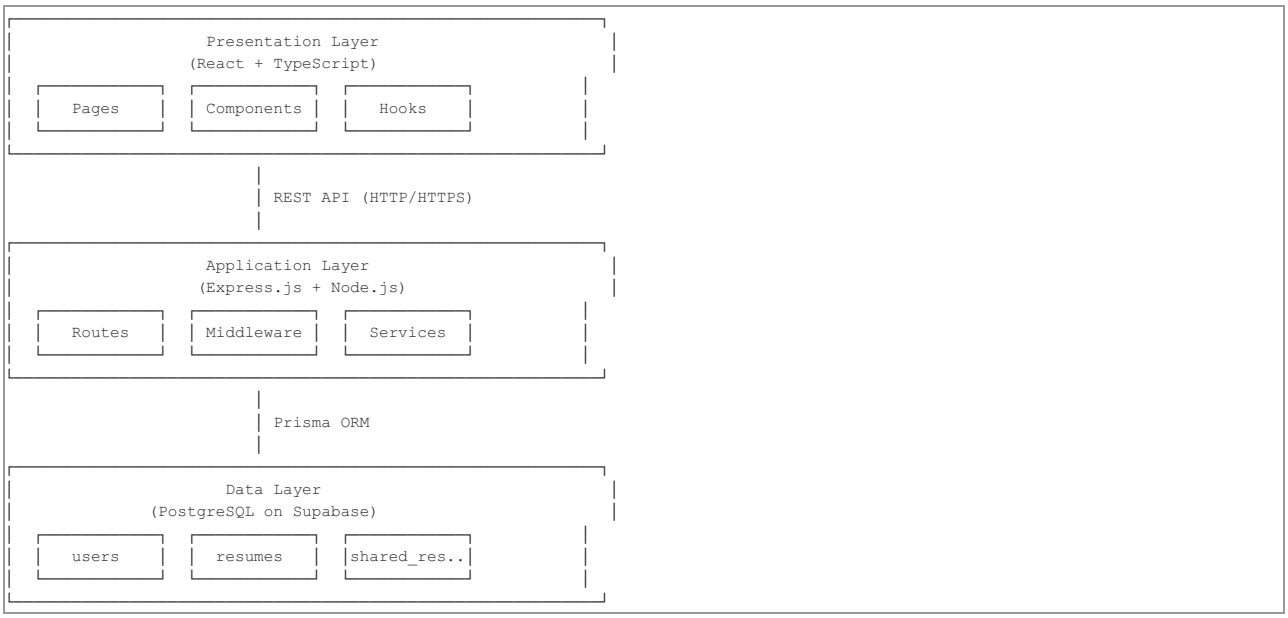
ImpactCV - Technical Documentation

Table of Contents

System Architecture

High-Level Overview

ImpactCV follows a modern three-tier architecture:



Architecture Principles

1. **Separation of Concerns**: Clear boundaries between layers
2. **Modularity**: Independent, reusable components
3. **Scalability**: Horizontal scaling capability
4. **Security**: Defense in depth approach
5. **Performance**: Optimized for speed and efficiency

Technology Stack

Frontend Stack

Core Technologies

- **React 18.3.1**: UI library with concurrent features
- **TypeScript 5.6.2**: Type-safe JavaScript
- **Vite 5.4.2**: Fast build tool and dev server

UI & Styling

- **Tailwind CSS 3.4.1**: Utility-first CSS framework
- **Radix UI**: Accessible component primitives
  - `@radix-ui/react-dialog`
  - `@radix-ui/react-label`
  - `@radix-ui/react-select`
  - `@radix-ui/react-slot`
- **Lucide React 0.344.0**: Icon library

- **class-variance-authority**: CSS variant management
- **clsx**: Conditional className utility
- **tailwind-merge**: Merge Tailwind classes

Routing & Navigation

- **React Router DOM 6.22.0**: Client-side routing

PDF & Export

- **jsPDF 2.5.1**: PDF generation
- **html2canvas 1.4.1**: HTML to canvas conversion

Notifications

- **Sonner 1.4.0**: Toast notifications

Backend Stack

Core Technologies

- **Node.js 18+**: JavaScript runtime
- **Express.js 4.18.2**: Web framework
- **JavaScript (ES Modules)**: Modern JS syntax

Database & ORM

- **Prisma 6.19.0**: Next-generation ORM
- **@prisma/client 6.19.0**: Prisma client
- **PostgreSQL 15+**: Relational database
- **Supabase**: Database hosting

Authentication & Security

- **jsonwebtoken 9.0.2**: JWT implementation
- **bcrypt 5.1.1**: Password hashing
- **cors 2.8.5**: Cross-origin resource sharing

Utilities

- **dotenv 16.4.5**: Environment variables
- **multer 1.4.5**: File upload handling
- **pg 8.11.3**: PostgreSQL client

Development Tools

- **ESLint**: Code linting
- **TypeScript**: Type checking
- **Vite**: Development server
- **Git**: Version control

Project Structure

Directory Structure

```
ImpactCV-master/
├── src/                          # Frontend source code
│   ├── components/               # React components
│   │   ├── ui/                   # Reusable UI components
│   │   │   ├── button.tsx
│   │   │   ├── card.tsx
│   │   │   ├── input.tsx
│   │   │   └── ...
│   │   ├── ProtectedRoute.tsx    # Auth guard component
│   │   ├── ShareOptions.tsx      # Resume sharing component
│   │   └── ...
│   ├── pages/                    # Page components
│   │   ├── HomePage.tsx          # Landing page
│   │   ├── LoginPage.tsx         # Login page
│   │   ├── SignupPage.tsx        # Registration page
│   │   ├── DashboardPage.tsx     # User dashboard
│   │   ├── EditorPage.tsx        # Resume editor
│   │   └── SharedResumePage.tsx  # Public resume view
│   ├── utils/                    # Utility functions
│   │   ├── auth.ts               # Auth helpers
│   │   └── ...
│   ├── lib/                      # Library configurations
│   │   └── utils.ts              # Shared utilities
│   ├── App.tsx                   # Root component
│   ├── main.tsx                  # Entry point
│   └── index.css                 # Global styles
│
├── server/                       # Backend source code
│   ├── routes/                   # API routes
│   │   └── auth.js               # Authentication routes
│   ├── middleware/               # Express middleware
│   │   └── auth.js               # JWT verification
│   ├── utils/                    # Utility functions
│   │   ├── jwt.js                # JWT helpers
│   │   └── password.js           # Password hashing
│   ├── prisma.js                 # Prisma client instance
│   ├── server.js                 # Express server setup
│   └── uploads/                  # File upload directory
│
├── prisma/                       # Prisma ORM
│   └── schema.prisma             # Database schema
│
├── public/                       # Static assets
│   └── _redirects                # Netlify redirects
│
├── .env                          # Environment variables (gitignored)
├── .env.example                  # Environment template
├── .gitignore                    # Git ignore rules
├── package.json                  # Dependencies
├── vite.config.ts                # Vite configuration
├── tsconfig.json                 # TypeScript configuration
├── tailwind.config.js            # Tailwind configuration
├── postcss.config.js             # PostCSS configuration
├── README.md                     # Project documentation
├── SETUP.md                      # Setup instructions
├── DATABASE_SCHEMA.md            # Database documentation
└── TECHNICAL.md                  # This file
```

Key Files Explained

Frontend Configuration

vite.config.ts

```typescript
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import path from 'path'

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
    },
  },
  server: {
    port: 3000,
    proxy: {
      '/api': {
        target: 'http://localhost:3001',
        changeOrigin: true,
      },
    },
  },
})
```

tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2020",
    "useDefineForClassFields": true,
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "skipLibCheck": true,
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx",
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true,
    "baseUrl": ".",
    "paths": {
      "@/*": ["./src/*"]
    }
  },
  "include": ["src"],
  "references": [{ "path": "./tsconfig.node.json" }]
}
```

Backend Configuration

server/server.js (Main server file)

- Express app initialization
- Middleware setup (CORS, JSON parsing)
- Route mounting
- Error handling
- Server startup

server/prisma.js (Database client)

```
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient({
  log: ['query', 'error', 'warn'],
});

export default prisma;
```

Frontend Architecture

Component Hierarchy

```
App
├── Router
│   ├── HomePage
│   ├── LoginPage
│   ├── SignupPage
│   ├── ProtectedRoute
│   │   ├── DashboardPage
│   │   │   └── ResumeCard (multiple)
│   │   └── EditorPage
│   │       ├── EditorSidebar
│   │       ├── ResumePreview
│   │       └── ShareOptions
│   └── SharedResumePage
│       └── ResumePreview
```

State Management

Local State (useState)

Used for component-specific state:

- Form inputs
- UI toggles
- Loading states

Context API (useContext)

Used for global state:

- Authentication state
- User information
- Theme preferences

Local Storage

Used for persistence:

- JWT token
- User preferences
- Draft resume data

Routing Strategy

Protected Routes

```
<Route element={<ProtectedRoute />}>
  <Route path="/dashboard" element={<DashboardPage />} />
  <Route path="/editor/:id" element={<EditorPage />} />
</Route>
```

Public Routes

```
<Route path="/" element={<HomePage />} />
<Route path="/login" element={<LoginPage />} />
<Route path="/signup" element={<SignupPage />} />
<Route path="/shared/:shareId" element={<SharedResumePage />} />
```

Component Patterns

1. Presentational Components

Pure UI components without business logic:

```
interface ButtonProps {
  children: React.ReactNode;
  onClick?: () => void;
  variant?: 'primary' | 'secondary';
}

export const Button: React.FC<ButtonProps> = ({
  children,
  onClick,
  variant = 'primary'
}) => {
  return (
    <button
      onClick={onClick}
      className={cn(buttonVariants({ variant }))}
    >
      {children}
    </button>
  );
};
```

2. Container Components

Components with business logic:

```
export const DashboardPage = () => {
  const [resumes, setResumes] = useState([]);
  const [loading, setLoading] = useState(true);

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

  const fetchResumes = async () => {
    // API call logic
  };

  return <DashboardView resumes={resumes} loading={loading} />;
};
```

3. Higher-Order Components

Components that wrap other components:

```
export const ProtectedRoute = () => {
  const token = localStorage.getItem('token');

  if (!token) {
    return <Navigate to="/login" />;
  }

  return <Outlet />;
};
```

Backend Architecture

Express Server Structure

Middleware Stack

```
app.use(cors());                    // Enable CORS
app.use(express.json());            // Parse JSON bodies
app.use('/uploads', express.static(...)); // Serve static files
app.use('/api/users', authRoutes);  // Mount auth routes
app.use(authMiddleware);            // Protect routes below
```

Route Organization

Authentication Routes (`/api/users`)

- POST `/register` - User registration
- POST `/login` - User login

Resume Routes (`/api`)

- GET `/resumes` - Get all user resumes
- GET `/resumes/:id` - Get specific resume
- POST `/resumes` - Create new resume
- PUT `/resumes/:id` - Update resume
- DELETE `/resume/:id` - Delete resume

Sharing Routes (`/api`)

- POST `/share-resume` - Create share link
- GET `/share-resume/:shareId` - Get shared resume

Utility Routes

- GET `/health` - Health check

Middleware Implementation

Authentication Middleware

```
import jwt from 'jsonwebtoken';

export default function authMiddleware(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 = decoded;
    next();
  } catch (error) {
    return res.status(401).json({ error: 'Invalid token' });
  }
}
```

Error Handling Middleware

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    error: 'Internal server error',
    message: process.env.NODE_ENV === 'development' ? err.message : undefined
  });
});
```

Service Layer Pattern

User Service

```
class UserService {
  async createUser(userData) {
    const hashedPassword = await hashPassword(userData.password);
    return prisma.user.create({
      data: {
        name: userData.name,
        email: userData.email,
        passwordHash: hashedPassword
      }
    });
  }

  async findByEmail(email) {
    return prisma.user.findUnique({ where: { email } });
  }
}
```

Resume Service

```
class ResumeService {
  async getUserResumes(userId) {
    return prisma.resume.findMany({
      where: { userId },
      orderBy: { updatedAt: 'desc' }
    });
  }

  async createResume(userId, resumeData) {
    return prisma.resume.create({
      data: {
        userId,
        title: resumeData.title,
        theme: resumeData.theme,
        data: resumeData.content
      }
    });
  }
}
```

Database Design

Prisma Schema

```
datasource db {
  provider  = "postgresql"
  url       = env("DATABASE_URL")
  directUrl = env("DIRECT_URL")
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id           Int       @id @default(autoincrement())
  name         String    @db.VarChar(255)
  email        String    @unique @db.VarChar(255)
  passwordHash String    @map("password_hash") @db.VarChar(255)
  createdAt    DateTime? @default(now()) @map("created_at")
  updatedAt    DateTime? @default(now()) @map("updated_at")

  resumes      Resume[]

  @@index([email])
  @@map("users")
}

model Resume {
  id        Int       @id @default(autoincrement())
  userId    Int?      @map("user_id")
  title     String    @db.VarChar(255)
  theme     String    @db.VarChar(100)
  data      Json
  createdAt DateTime? @default(now()) @map("created_at")
  updatedAt DateTime? @default(now()) @map("updated_at")

  user      User?     @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId])
  @@map("resumes")
}

model SharedResume {
  id        Int       @id @default(autoincrement())
  shareId   String    @unique @map("share_id") @db.VarChar(255)
  data      Json
  createdAt DateTime? @default(now()) @map("created_at")

  @@index([shareId])
  @@map("shared_resumes")
}
```

Database Operations

Using Prisma Client

Create

```
const user = await prisma.user.create({
  data: {
    name: 'John Doe',
    email: 'john@example.com',
    passwordHash: hashedPassword
  }
});
```

Read

```
const resumes = await prisma.resume.findMany({
  where: { userId: 1 },
  include: { user: true },
  orderBy: { updatedAt: 'desc' }
});
```

Update

```
const resume = await prisma.resume.update({
  where: { id: 1 },
  data: {
    title: 'Updated Title',
    data: updatedContent
  }
});
```

Delete

```
await prisma.resume.delete({
  where: { id: 1 }
});
```

Connection Pooling

Configuration

- Pooled connection (port 6543) for application queries
- Direct connection (port 5432) for migrations
- PgBouncer in transaction mode
- Connection limit: Managed by Supabase

Authentication & Security

JWT Authentication Flow

```
1. User Registration/Login
   ↓
2. Server validates credentials
   ↓
3. Server generates JWT token
   ↓
4. Client stores token (localStorage)
   ↓
5. Client includes token in requests
   ↓
6. Server verifies token
   ↓
7. Server processes request
```

JWT Implementation

Token Generation

```
import jwt from 'jsonwebtoken';

export const generateToken = (payload) => {
  return jwt.sign(
    payload,
    process.env.JWT_SECRET,
    { expiresIn: process.env.JWT_EXPIRES_IN || '24h' }
  );
};
```

Token Verification

```
export const verifyToken = (token) => {
  try {
    return jwt.verify(token, process.env.JWT_SECRET);
  } catch (error) {
    throw new Error('Invalid token');
  }
};
```

Password Security

Hashing with Bcrypt

```
import bcrypt from 'bcrypt';

const SALT_ROUNDS = 10;

export const hashPassword = async (password) => {
  return await bcrypt.hash(password, SALT_ROUNDS);
};

export const comparePassword = async (password, hash) => {
  return await bcrypt.compare(password, hash);
};
```

Security Best Practices

1. **Password Requirements**
   - Minimum 8 characters
   - Validated on both client and server
2. **Token Storage**
   - Stored in localStorage (consider httpOnly cookies for production)
   - Cleared on logout
   - Expires after 24 hours
3. **API Security**
   - CORS enabled with specific origins
   - Rate limiting (recommended for production)
   - Input validation and sanitization
4. **Database Security**
   - Parameterized queries via Prisma
   - No raw SQL with user input
   - Connection string in environment variables
5. **Environment Variables**
   - Never commit `.env` to version control
   - Use different secrets for dev/prod
   - Rotate JWT secrets regularly

API Design

RESTful Principles

- **Resource-based URLs**: `/api/resumes`, `/api/users`
- **HTTP Methods**: GET, POST, PUT, DELETE
- **Status Codes**: 200, 201, 400, 401, 404, 500
- **JSON Responses**: Consistent format

Request/Response Format

Success Response

```
{
  "success": true,
  "data": { /* resource data */ },
  "message": "Operation successful"
}
```

Error Response

```
{
  "error": "Error type",
  "message": "Human-readable error message",
  "details": "Additional error information"
}
```

API Versioning

Currently using implicit v1. Future versions:

- `/api/v2/resumes`
- Maintain backward compatibility
- Deprecation notices

Rate Limiting (Recommended)

```
import rateLimit from 'express-rate-limit';

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
});

app.use('/api/', limiter);
```

State Management

Frontend State Architecture

1. Local Component State

```
const [formData, setFormData] = useState({
  name: '',
  email: '',
  password: ''
});
```

2. Authentication State

```
// utils/auth.ts
export const isAuthenticated = (): boolean => {
  const token = localStorage.getItem('token');
  if (!token) return false;

  try {
    const decoded = jwt.decode(token);
    return decoded.exp * 1000 > Date.now();
  } catch {
    return false;
  }
};

export const getUser = () => {
  const token = localStorage.getItem('token');
  if (!token) return null;

  try {
    return jwt.decode(token);
  } catch {
    return null;
  }
};
```

3. Resume Editor State

```
interface ResumeData {
  basicInfo: BasicInfo;
  summary: string;
  experience: Experience[];
  education: Education[];
  skills: Skill[];
  projects: Project[];
  certifications: Certification[];
}

const [resumeData, setResumeData] = useState<ResumeData>(initialData);
```

State Persistence

LocalStorage Strategy

```
// Save to localStorage
const saveToLocalStorage = (key: string, data: any) => {
  localStorage.setItem(key, JSON.stringify(data));
};

// Load from localStorage
const loadFromLocalStorage = (key: string) => {
  const data = localStorage.getItem(key);
  return data ? JSON.parse(data) : null;
};

// Auto-save implementation
useEffect(() => {
  const timer = setTimeout(() => {
    saveToLocalStorage('resume-draft', resumeData);
  }, 1000);

  return () => clearTimeout(timer);
}, [resumeData]);
```

Performance Optimization

Frontend Optimizations

1. Code Splitting

```
// Lazy load pages
const EditorPage = lazy(() => import('./pages/EditorPage'));
const DashboardPage = lazy(() => import('./pages/DashboardPage'));

// Use Suspense
<Suspense fallback={<LoadingSpinner />}>
  <EditorPage />
</Suspense>
```

2. Memoization

```
// Memoize expensive calculations
const processedData = useMemo(() => {
  return expensiveOperation(data);
}, [data]);

// Memoize callbacks
const handleSave = useCallback(() => {
  saveResume(resumeData);
}, [resumeData]);
```

3. Debouncing

```
// Debounce auto-save
const debouncedSave = useMemo(
  () => debounce((data) => {
    saveResume(data);
  }, 1000),
  []
);

useEffect(() => {
  debouncedSave(resumeData);
}, [resumeData]);
```

4. Image Optimization

- Use WebP format
- Lazy load images
- Implement responsive images

Backend Optimizations

1. Database Query Optimization

```
// Use select to limit fields
const resumes = await prisma.resume.findMany({
  where: { userId },
  select: {
    id: true,
    title: true,
    theme: true,
    updatedAt: true
    // Don't fetch large 'data' field for list view
  }
});

// Use indexes for frequent queries
@@index([userId])
@@index([email])
```

2. Connection Pooling

- PgBouncer for connection management
- Reuse database connections
- Configure pool size based on load

3. Caching Strategy

```
// In-memory cache for frequently accessed data
const cache = new Map();

const getCachedData = async (key, fetchFn) => {
  if (cache.has(key)) {
    return cache.get(key);
  }

  const data = await fetchFn();
  cache.set(key, data);
  return data;
};
```

4. Response Compression

```
import compression from 'compression';
app.use(compression());
```

Build Optimizations

Vite Configuration

```
export default defineConfig({
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          'react-vendor': ['react', 'react-dom', 'react-router-dom'],
          'ui-vendor': ['@radix-ui/react-dialog', '@radix-ui/react-select'],
        }
      }
    },
    chunkSizeWarningLimit: 1000
  }
});
```

Testing Strategy

Unit Testing

Frontend Tests (Vitest + React Testing Library)

```
import { render, screen } from '@testing-library/react';
import { Button } from './Button';

describe('Button', () => {
  it('renders with correct text', () => {
    render(<Button>Click me</Button>);
    expect(screen.getByText('Click me')).toBeInTheDocument();
  });

  it('calls onClick when clicked', () => {
    const handleClick = vi.fn();
    render(<Button onClick={handleClick}>Click</Button>);
    screen.getByText('Click').click();
    expect(handleClick).toHaveBeenCalledOnce();
  });
});
```

Backend Tests (Jest)

```
import { hashPassword, comparePassword } from './password';

describe('Password Utils', () => {
  it('hashes password correctly', async () => {
    const password = 'test123';
    const hash = await hashPassword(password);
    expect(hash).not.toBe(password);
    expect(hash).toMatch(/^\$2b\$/);
  });

  it('compares passwords correctly', async () => {
    const password = 'test123';
    const hash = await hashPassword(password);
    const isMatch = await comparePassword(password, hash);
    expect(isMatch).toBe(true);
  });
});
```

Integration Testing

API Tests

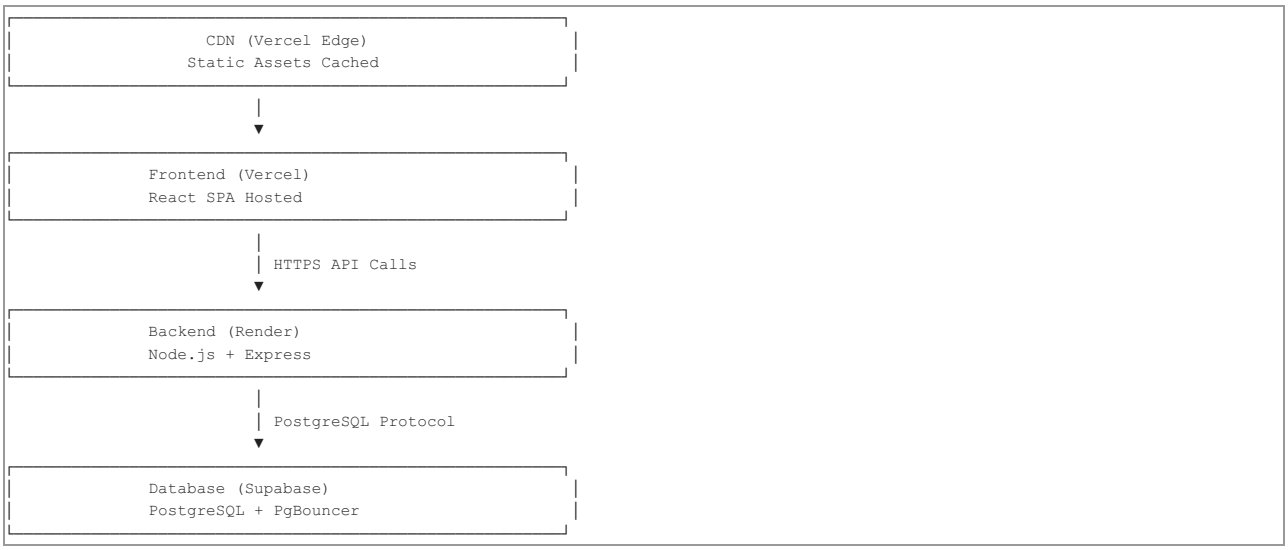
```
import request from 'supertest';
import app from './server';

describe('Auth API', () => {
  it('registers new user', async () => {
    const response = await request(app)
      .post('/api/users/register')
      .send({
        name: 'Test User',
        email: 'test@example.com',
        password: 'password123'
      });

    expect(response.status).toBe(201);
    expect(response.body).toHaveProperty('userId');
  });
});
```

E2E Testing (Playwright)

```
import { test, expect } from '@playwright/test';

test('user can create resume', async ({ page }) => {
  // Login
  await page.goto('/login');
  await page.fill('[name="email"]', 'test@example.com');
  await page.fill('[name="password"]', 'password123');
  await page.click('button[type="submit"]');

  // Navigate to editor
  await page.click('text=Create New Resume');

  // Fill resume data
  await page.fill('[name="name"]', 'John Doe');
  await page.fill('[name="email"]', 'john@example.com');

  // Save
  await page.click('text=Save');

  // Verify
  await expect(page.locator('text=Resume saved')).toBeVisible();
});
```

Deployment Architecture

Production Environment



Deployment Steps

Frontend (Vercel)

1. **Connect Repository**

   ```
   vercel link
   ```

2. **Configure Build**

   - Build Command: `npm run build`
   - Output Directory: `dist`
   - Install Command: `npm install`

3. **Set Environment Variables**

   ```
   VITE_API_URL=https://your-api.onrender.com
   ```

4. **Deploy**

   ```
   vercel --prod
   ```

Backend (Render)

1. **Create Web Service**

   - Environment: Node
   - Build Command: `cd server && npm install && npx prisma generate`
   - Start Command: `cd server && node server.js`

2. **Set Environment Variables**

```
DATABASE_URL=postgresql://...
DIRECT_URL=postgresql://...
JWT_SECRET=production-secret
PORT=3001
NODE_ENV=production
```

3. **Auto-Deploy**
    - Connects to GitHub
    - Deploys on push to main branch

CI/CD Pipeline

GitHub Actions Example

```yaml
name: Deploy

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '18'
      - run: npm install
      - run: npm run build
      - run: npm test
      - uses: amondnet/vercel-action@v20
        with:
          vercel-token: ${{ secrets.VERCEL_TOKEN }}
          vercel-org-id: ${{ secrets.ORG_ID }}
          vercel-project-id: ${{ secrets.PROJECT_ID }}
```

Development Workflow

Git Workflow

Branch Strategy

```
main (production)
  ├── develop (staging)
  │   ├── feature/user-auth
  │   ├── feature/resume-editor
  │   └── bugfix/login-issue
```

Commit Convention

```
feat: Add resume sharing feature
fix: Resolve login redirect issue
docs: Update API documentation
style: Format code with prettier
refactor: Simplify auth middleware
test: Add unit tests for password utils
chore: Update dependencies
```

Development Commands

```bash
# Install dependencies
npm install

# Start development servers
npm run dev              # Frontend (port 3000)
cd server && node server.js  # Backend (port 3001)

# Build for production
npm run build

# Run tests
npm test

# Lint code
npm run lint

# Format code
npm run format

# Database commands
npx prisma generate      # Generate Prisma client
npx prisma studio        # Open database GUI
npx prisma migrate dev   # Create migration
npx prisma db push       # Push schema changes
```

Code Review Checklist

- [ ] Code follows project style guide
- [ ] All tests pass
- [ ] No console.log statements
- [ ] Error handling implemented
- [ ] Comments added for complex logic
- [ ] No sensitive data in code
- [ ] Performance considered
- [ ] Security best practices followed

Troubleshooting Guide

Common Issues

1. Database Connection Failed

Symptoms: "Can't reach database server"

Solutions:

- Check DATABASE_URL in `.env`
- Verify Supabase project is active
- Ensure password has no special characters that need escaping
- Test connection: `node test-supabase-connection.js`

2. Prisma Client Not Generated

Symptoms: "Cannot find module '@prisma/client'"

Solutions:

```
npx prisma generate
```

3. JWT Token Invalid

Symptoms: 401 Unauthorized errors

Solutions:

- Check JWT_SECRET matches between token generation and verification
- Verify token hasn't expired
- Clear localStorage and login again

4. CORS Errors

Symptoms: "Access-Control-Allow-Origin" errors

Solutions:

- Add frontend URL to CORS whitelist in server
- Check API URL in frontend matches backend URL

5. Port Already in Use

Symptoms: "EADDRINUSE: address already in use"

Solutions:

```
# Windows
netstat -ano | findstr :3001
taskkill /PID <PID> /F

# Linux/Mac
lsof -ti:3001 | xargs kill -9
```

Debug Mode

Enable Detailed Logging

```
// server/prisma.js
const prisma = new PrismaClient({
  log: ['query', 'info', 'warn', 'error'],
});

// server/server.js
app.use((req, res, next) => {
  console.log(`${req.method} ${req.path}`, req.body);
  next();
});
```

Performance Profiling

Frontend

```
// React DevTools Profiler
import { Profiler } from 'react';

<Profiler id="Editor" onRender={onRenderCallback}>
  <EditorPage />
</Profiler>
```

Backend

```
// Request timing middleware
app.use((req, res, next) => {
  const start = Date.now();
  res.on('finish', () => {
    const duration = Date.now() - start;
    console.log(`${req.method} ${req.path} - ${duration}ms`);
  });
  next();
});
```

Appendix

Useful Resources

- **React Documentation**: https://react.dev
- **TypeScript Handbook**: https://www.typescriptlang.org/docs
- **Prisma Docs**: https://www.prisma.io/docs
- **Supabase Docs**: https://supabase.com/docs
- **Express.js Guide**: https://expressjs.com/en/guide
- **Tailwind CSS**: https://tailwindcss.com/docs

Environment Variables Reference

```
# Database
DATABASE_URL="postgresql://..."
DIRECT_URL="postgresql://..."

# Server
PORT=3001
NODE_ENV=development|production

# Authentication
JWT_SECRET="your-secret-key"
JWT_EXPIRES_IN="24h"

# Frontend (Vite)
VITE_API_URL="http://localhost:3001"
```

Package Scripts

```
{
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "preview": "vite preview",
    "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warnings 0"
  }
}
```

Document Version: 1.0
Last Updated: January 2024
Maintained By: ImpactCV Development Team