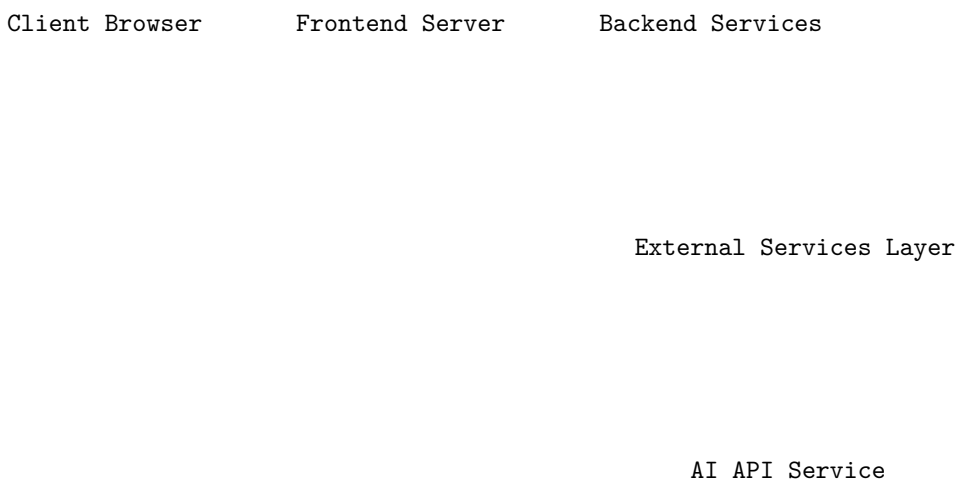
System Architecture Document

AI-Powered LaTeX CV Generator

1. Architecture Overview

1.1 System Description The AI-Powered LaTeX CV Generator is a web application that allows users to create professional CVs using LaTeX templates through a simple interface. The system leverages AI to convert user input into properly formatted LaTeX code, which is then compiled into downloadable PDF documents.

1.2 Architecture Diagram



1.3 Architecture Style The system follows a client-server architecture with a RESTful API approach. The frontend and backend are decoupled, allowing for independent development and scaling. The architecture incorporates elements of:

- **Microservices**: Separating CV generation, LaTeX processing, and PDF compilation
- **Event-driven**: For handling the asynchronous nature of AI processing and LaTeX compilation
- **Layered architecture**: Clear separation between presentation, business logic, and external services

2. Component Description

2.1 Frontend Component

- **Purpose**: Provide user interface for CV data input and PDF preview/download
- **Technologies**: React.js, Redux/Context API, Material-UI/Tailwind CSS
- **Responsibilities**:
  - Render UI components
  - Validate user input
  - Manage form state
  - Handle API communication
  - Preview generated PDFs
  - Manage download process

2.2 Backend Component

- **Purpose**: Process requests, coordinate with AI and LaTeX services
- **Technologies**: Node.js, Express
- **Responsibilities**:
  - Handle API requests
  - Validate incoming data
  - Coordinate with AI service
  - Process LaTeX generation
  - Manage PDF compilation
  - Error handling and reporting

2.3 AI Integration Component

- **Purpose**: Transform structured data into formatted LaTeX code
- **Technologies**: OpenAI API (primary), Claude/Mistral (fallback)
- **Responsibilities**:
  - Receive structured CV data
  - Generate appropriate prompts
  - Process AI responses
  - Handle rate limiting and quotas
  - Implement fallback mechanisms

2.4 LaTeX Processing Component

- **Purpose**: Compile LaTeX code into PDF documents
- **Technologies**: pdfLaTeX, Node-LaTeX/LaTeX.js
- **Responsibilities**:
  - Parse LaTeX code
  - Apply templates
  - Compile to PDF
  - Handle compilation errors
  - Optimize PDF output

3. Interface Specifications

3.1 User Interface

- **Web Interface**: Browser-based responsive design
- **Mobile Interface**: Touch-friendly UI optimized for smaller screens

3.2 API Interfaces

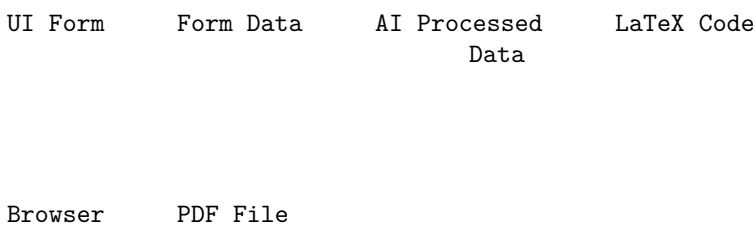
- **Frontend to Backend**: RESTful JSON API
- **Backend to AI Service**: HTTPS with API key authentication
- **Backend to LaTeX Service**: Internal service calls

3.3 Data Flow Interfaces

1. User inputs CV data through web interface
2. Frontend validates and sends data to backend
3. Backend formats data for AI service
4. AI service generates LaTeX code
5. Backend processes LaTeX code with template
6. LaTeX compiler generates PDF
7. Backend serves PDF to frontend
8. User downloads PDF from frontend

4. Data Architecture

4.1 Data Flow Diagram



4.2 Data Storage

- **Session Storage**: Temporary storage of user data during CV creation
- **Template Storage**: Static storage of LaTeX templates
- **No Persistent Database**: By design for MVP to minimize data concerns

4.3 Data Models

- CV Data Model (see Technical Specifications)

- Template Model (see Technical Specifications)

5. Technology Stack

5.1 Frontend Stack

- **Framework**: React.js
- **State Management**: Redux or Context API
- **UI Components**: Material-UI or Tailwind CSS
- **Form Handling**: Formik with Yup
- **HTTP Client**: Axios
- **PDF Preview**: React-PDF

5.2 Backend Stack

- **Runtime**: Node.js
- **Framework**: Express
- **Validation**: Joi or express-validator
- **LaTeX Processing**: Node-LaTeX or LaTeX.js
- **Error Handling**: Winston logger

5.3 Infrastructure Stack

- **Frontend Hosting**: Vercel or Netlify (static hosting)
- **Backend Hosting**: Render, Railway, or Fly.io
- **CI/CD**: GitHub Actions
- **Monitoring**: Sentry

6. Security Architecture

6.1 Authentication & Authorization

- No user authentication for MVP
- API endpoints secured with rate limiting
- Environment-based API key management

6.2 Data Protection

- HTTPS encryption for all communications
- No persistent storage of user data
- Input sanitization and validation
- LaTeX command injection prevention

6.3 External Service Security

- AI API keys stored as environment variables
- Request scoping to minimum necessary permissions
- Error handling to prevent information leakage

7. Performance Architecture

7.1 Scalability Considerations

- Stateless backend design for horizontal scaling
- Independent microservices for AI and LaTeX processing
- Caching layer for template assets

7.2 Optimizations

- Frontend code splitting and lazy loading
- Backend response compression
- PDF generation optimization
- Request batching for AI service

7.3 Performance Targets

- Page load time: < 3 seconds
- Total CV generation time: < 15 seconds
- Concurrent user support: 100+

8. Monitoring & Operations

8.1 Logging

- Request/response logging
- Error logging
- AI service interaction logging
- LaTeX compilation logging

8.2 Monitoring

- Application performance monitoring
- API usage tracking
- Error tracking
- User session analytics

8.3 Alerting

- Critical error notifications
- API quota warnings
- Performance degradation alerts

9. Deployment Architecture

9.1 Development Environment

- Local development setup with mocked AI services
- Docker containers for LaTeX environment

9.2 Staging Environment

- Deployed to same infrastructure as production
- Connected to test instances of external services
- Automated integration testing

9.3 Production Environment

- Static frontend on CDN
- Backend services on managed platform
- Automated deployment pipeline

10. Resilience & Fault Tolerance

10.1 Error Handling

- Graceful degradation of features
- User-friendly error messaging
- Automatic retry for transient failures

10.2 Fallback Mechanisms

- Alternative AI service providers
- Local LaTeX generation fallback
- Circuit breaker pattern for external services

10.3 Recovery Procedures

- Session recovery from browser storage
- Form data autosave
- Process resumption capabilities

11. Future Architecture Considerations

11.1 Scalability Enhancements

- Containerization with Kubernetes
- Message queue for asynchronous processing
- Redis caching layer

11.2 Feature Extensions

- User authentication and profiles
- Template marketplace
- Advanced customization options
- Multi-language support

11.3 Integration Possibilities

- Job board API integrations
- LinkedIn profile import
- ATS optimization services

This architecture document provides a comprehensive overview of the AI-Powered LaTeX CV Generator system design. It should guide implementation decisions while allowing for flexibility as requirements evolve.