PROJECT REPORT

ON

SSL Automation Tool

StartMySafari Innovations Private Limited (smsipl)

Submitted by

Dipak Nivrutti Rathod

Seat No: MCD004036

UNDER THE GUIDANCE OF

Dr. Kavari Lad

In partial fulfilment for the award of the degree of

MCA (Master of Computer Application)

Dr. Babasaheb Ambedkar Marathwada University, Chhatrapati Sambhajinagar

Department of Management Science

2023-24

Dr. Babasaheb Ambedkar Marathwada University,
Chhatrapati Sambhajinagar
Department of Management Science

CERTIFICATE

This is to certify that **Dipak Nivrutti Rathod** (Seat No: **MCD004036**) has satisfactorily carried out the project work entitled **"SSL Automation Tool"** as per requirement of Department of Management Science, Dr. Babasaheb Ambedkar Marathwada University, Chhatrapati Sambhajinagar in the fulfillment of Master of Computer Application for the academic Year 2023-24.

Dr. Kavari Lad
(Project Guide)

Prof. Abhijeet Shelke
(Director)

Date: ______________

Place: Chhatrapati Sambhajinagar

External Examiner

Acknowledgements

I would like to express my sincere gratitude to all those who contributed to the successful completion of this SSL Automation Tool project.

First and foremost, I extend my heartfelt thanks to **Dr. Kavari Lad**, my project guide, for his invaluable guidance, continuous support, and expert advice throughout the development of this project. His insights into modern web technologies and security practices were instrumental in shaping this work.

I am deeply grateful to **StartMySafari Innovations Private Limited (smsipl)** for providing the opportunity to work on this practical and industry-relevant project. The real-world exposure and access to modern development tools and technologies significantly enhanced my learning experience.

Special thanks to **Prof. Abhijeet Shelke**, Director, for his administrative support and for creating an environment conducive to learning and innovation.

I would like to acknowledge the faculty members of the Department of Management Science at **Dr. Babasaheb Ambedkar Marathwada University, Chhatrapati Sambhajinagar**, for their academic support and for providing the foundational knowledge that enabled me to undertake this project.

My appreciation extends to the open-source community, particularly the developers of React, Next.js, FastAPI, and Let's Encrypt, whose technologies formed the backbone of this SSL automation solution.

I am thankful to my colleagues and peers who provided feedback and suggestions during the development and testing phases of the project.

Finally, I express my gratitude to my family and friends for their constant encouragement and support throughout my academic journey.

This project has been an enriching experience that has significantly enhanced my understanding of web development, SSL certificate management, and modern software engineering practices.

Dipak Nivrutti Rathod
June 2025

Abstract

This project presents the design, implementation, and evaluation of an SSL Automation Tool developed for StartMySafari Innovations Private Limited (smsipl). The system addresses critical challenges in SSL certificate management by automating the entire certificate lifecyclefrom generation and validation to deployment and renewal. The solution integrates with Let's Encrypt to provide free, automated SSL certificates using DNS-01 validation methods, particularly focusing on GoDaddy DNS provider integration.

The system architecture follows a modern microservices approach with a React-based frontend and FastAPI backend. The frontend delivers an intuitive user interface with real-time certificate status monitoring, while the backend handles complex certificate operations through Let's Encrypt's ACME protocol. Container-based deployment ensures scalability and consistent operation across environments.

Key innovations include automated DNS challenge validation, certificate bundling for various server configurations, and proactive expiration monitoring. Performance testing demonstrates significant time savings compared to manual certificate management processesreducing certificate generation time from approximately 30 minutes to under 2 minutes per domain.

The project successfully addresses the industry requirements for automated SSL management while providing a foundation for future enhancements. Implementation results show improved operational efficiency, reduced human error, and enhanced security compliance for web applications requiring HTTPS connections.

Keywords SSL Certificate Management, Let's Encrypt, DNS Validation, Web Security, Automation, FastAPI, React, Microservices Architecture

Contents

List of Figures

List of Tables

Abbreviations

SSL	**S**ecure **S**ockets **L**ayer
TLS	**T**ransport **L**ayer **S**ecurity
HTTPS	**H**yper**T**ext **T**ransfer **P**rotocol **S**ecure
CA	**C**ertificate **A**uthority
DNS	**D**omain **N**ame **S**ystem
API	**A**pplication **P**rogramming **I**nterface
CSR	**C**ertificate **S**igning **R**equest
PKI	**P**ublic **K**ey **I**nfrastructure
ACME	**A**utomated **C**ertificate **M**anagement **E**nvironment
JSON	**J**avaScript **O**bject **N**otation
REST	**RE**presentational **S**tate **T**ransfer
UI	**U**ser **I**nterface
UX	**U**ser **E**xperience
CI/CD	**C**ontinuous **I**ntegration/**C**ontinuous **D**eployment
SPA	**S**ingle **P**age **A**pplication
JWT	**J**SON **W**eb **T**oken
DRY	**D**on't **R**epeat **Y**ourself
SOLID	**S**ingle Responsibility, **O**pen-closed, **L**iskov Substitution, **I**nterface Segregation, **D**ependenc
MVC	**M**odel-**V**iew-**C**ontroller
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol

Symbols

$X.509$	Standard format for public key certificates
RSA	Rivest-Shamir-Adleman public-key cryptosystem
$ECDSA$	Elliptic Curve Digital Signature Algorithm
$SHA-256$	Secure Hash Algorithm (256-bit)
$AES-256$	Advanced Encryption Standard (256-bit)
$DNS-01$	DNS challenge type for domain validation
$HTTP-01$	HTTP challenge type for domain validation
$PKCS\#12$	Public Key Cryptography Standards format for storing certificates
PEM	Privacy Enhanced Mail format for storing certificates
DER	Distinguished Encoding Rules format for certificates
CN	Common Name in certificate subject field
SAN	Subject Alternative Name in certificate
$OCSP$	Online Certificate Status Protocol
CRL	Certificate Revocation List
CT	Certificate Transparency
TTL	Time To Live for DNS records
$FQDN$	Fully Qualified Domain Name
$CORS$	Cross-Origin Resource Sharing
$OAuth$	Open Authorization protocol
$RBAC$	Role-Based Access Control

Chapter 1

Introduction

1.1 Company Profile

StartMySafari Innovations Private Limited (smsipl) is a technology innovation company focused on developing modern web solutions and automation tools. The company specializes in creating scalable web applications, SSL certificate management systems, and enterprise-grade software solutions.

The company's expertise lies in:

- Modern web application development using React.js and Next.js
- Backend API development with Python and FastAPI
- SSL certificate automation and security solutions
- Container-based deployment and DevOps practices
- DNS provider integrations and automation

1.2 Existing System and Need for System

1.2.1 Current SSL Certificate Management Challenges

Traditional SSL certificate management involves several manual processes that are time-consuming and error-prone:

- **Manual Certificate Generation**: Administrators must manually generate CSRs (Certificate Signing Requests), validate domain ownership, and configure certificates on servers.
- **DNS Validation Complexity**: DNS-01 challenge validation requires manual creation and cleanup of TXT records, leading to potential configuration errors.
- **Certificate Renewal Management**: Tracking expiration dates and renewing certificates before they expire requires constant monitoring and manual intervention.
- **Multi-Domain Management**: Managing certificates for multiple domains across different DNS providers becomes increasingly complex and time-consuming.
- **Deployment Inconsistencies**: Manual certificate deployment often leads to configuration inconsistencies across different server environments.

1.2.2 Industry Requirements

Modern web applications require:

- Automated SSL certificate generation and renewal
- Support for wildcard certificates through DNS validation

- Integration with multiple DNS providers
- Centralized certificate management dashboard
- API-driven certificate operations for DevOps workflows

1.3 Scope of Work

The SSL Automation Tool project encompasses the development of a comprehensive SSL certificate management system with the following scope:

1.3.1 Frontend Application Scope

- React-based user interface with modern design principles
- Certificate generation forms with validation
- Real-time certificate status monitoring
- Certificate download and deployment assistance
- Comprehensive documentation system
- Responsive design for multiple device types

1.3.2 Backend API Scope

- FastAPI-based REST API for certificate operations
- Let's Encrypt integration using Certbot
- DNS provider automation (initially GoDaddy)
- Certificate file management and retrieval

- Error handling and logging systems
- Docker-based deployment configuration

1.3.3 Integration Scope

- ACME protocol compliance for Let's Encrypt communication
- DNS provider API integration for automated challenge response
- Certificate parsing and validation utilities
- Automated backup and recovery mechanisms

1.4 Operating Environment Hardware and Software

1.4.1 Development Environment

- **Operating System**: Ubuntu 20.04+ / Windows 10+ / macOS 12+
- **Development Tools**: Visual Studio Code, Git, Docker Desktop
- **Browser Support**: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+

1.4.2 Frontend Requirements

- **Runtime**: Node.js 18+ or Bun runtime
- **Package Manager**: pnpm, npm, or yarn
- **Build Tool**: Next.js 15 with Turbopack
- **Memory**: Minimum 4GB RAM for development

1.4.3 Backend Requirements

- **Runtime**: Python 3.13+
- **Container**: Docker 20.10+ and Docker Compose 2.0+
- **System Dependencies**: Certbot 4.1.1+, OpenSSL, curl, wget
- **Memory**: Minimum 2GB RAM for API operations
- **Storage**: Persistent volume for certificate storage

1.4.4 Production Environment

- **Server**: Linux-based server (Ubuntu 20.04+ recommended)
- **Container Orchestration**: Docker Swarm or Kubernetes (optional)
- **Load Balancer**: Nginx or HAProxy for production deployments
- **DNS Provider**: GoDaddy account with API credentials
- **Network**: Internet connectivity for Let's Encrypt ACME server and DNS APIs

1.5 Detail Description of Technology Used

1.5.1 Frontend Technologies

1.5.1.1 Next.js 15 Framework

Next.js provides the foundation for the frontend application with:

- App Router for modern routing architecture

- Server-side rendering capabilities
- Built-in optimization for performance
- TypeScript support for type safety

1.5.1.2 React 19

The latest version of React offers:

- Concurrent features for improved performance
- Enhanced hooks for state management
- Improved error boundaries and debugging
- Modern component patterns

1.5.1.3 ShadCN UI Component Library

A modern component library providing:

- Accessible UI components built on Radix UI
- Customizable design system with Tailwind CSS
- Dark mode support
- Responsive design components

1.5.1.4 TypeScript

Type safety is ensured through:

- Strict type checking configuration
- Interface definitions for all data models
- Compile-time error detection
- Enhanced IDE support and autocompletion

1.5.2 Backend Technologies

1.5.2.1 FastAPI Framework

FastAPI provides:

- High-performance async web framework
- Automatic API documentation generation
- Built-in data validation using Pydantic
- OpenAPI specification compliance

1.5.2.2 Python 3.13

Modern Python features utilized:

- Async/await for concurrent operations
- Type hints for better code documentation
- Enhanced error handling capabilities
- Improved performance optimizations

1.5.2.3 Certbot Integration

Let's Encrypt automation through:

- ACME protocol implementation
- DNS-01 challenge automation
- Certificate lifecycle management
- Plugin architecture for DNS providers

1.5.3 Supporting Technologies

1.5.3.1 Docker Containerization

Container-based deployment provides:

- Environment consistency across development and production
- Simplified dependency management
- Scalable deployment architecture
- Volume persistence for certificate storage

1.5.3.2 UV Package Manager

Modern Python dependency management:

- Fast dependency resolution
- Reproducible builds with lock files

- Cross-platform compatibility
- Improved security with dependency auditing

1.5.3.3 DNS Provider APIs

Currently supporting GoDaddy with:

- REST API integration for DNS record management
- Automated TXT record creation and cleanup
- Rate limiting and error handling
- Extensible architecture for additional providers

Chapter 2

Proposed System

2.1 Proposed System

The SSL Automation Tool is a comprehensive web-based solution designed to automate the entire lifecycle of SSL certificate management. The system consists of two main components:

- **Frontend Application**: A React-based web interface providing user-friendly certificate management
- **Backend API**: A FastAPI-based service handling SSL certificate operations and Let's Encrypt integration

2.1.1 System Architecture Overview

The proposed system follows a modern microservices architecture with clear separation of concerns (detailed component diagram is provided in Chapter 3):

2.1.1.1 Frontend Layer

- **User Interface**: Modern React-based SPA with responsive design
- **State Management**: Local component state with React hooks
- **API Communication**: Axios-based HTTP client with error handling
- **Authentication**: Token-based authentication (future enhancement)

2.1.1.2 Backend Layer

- **API Gateway**: FastAPI framework with automatic documentation
- **Business Logic**: Certificate generation and management services
- **External Integrations**: Let's Encrypt ACME and DNS provider APIs
- **File Management**: Certificate storage and retrieval system

2.1.1.3 Infrastructure Layer

- **Containerization**: Docker-based deployment with volume persistence
- **DNS Integration**: GoDaddy API for automated DNS challenge validation
- **Certificate Authority**: Let's Encrypt for free SSL certificates
- **Storage**: Persistent volumes for certificate and configuration files

2.1.1.4 Frontend Layer

- **User Interface**: Modern React-based SPA with responsive design
- **State Management**: Local component state with React hooks

- **API Communication**: Axios-based HTTP client with error handling
- **Authentication**: Token-based authentication (future enhancement)

2.1.1.5 Backend Layer

- **API Gateway**: FastAPI framework with automatic documentation
- **Business Logic**: Certificate generation and management services
- **External Integrations**: Let's Encrypt ACME and DNS provider APIs
- **File Management**: Certificate storage and retrieval system

2.1.1.6 Infrastructure Layer

- **Containerization**: Docker-based deployment with volume persistence
- **DNS Integration**: GoDaddy API for automated DNS challenge validation
- **Certificate Authority**: Let's Encrypt for free SSL certificates
- **Storage**: Persistent volumes for certificate and configuration files

2.2 Objectives of System

2.2.1 Primary Objectives

1. **Automate SSL Certificate Generation**
    - Eliminate manual CSR creation and domain validation processes
    - Provide one-click certificate generation for single and wildcard domains
    - Support DNS-01 challenge validation for enhanced security

2. **Simplify Certificate Management**
    - Centralized dashboard for all certificate operations
    - Real-time monitoring of certificate status and expiration dates
    - Easy certificate download and deployment assistance
3. **Enhance Security and Compliance**
    - Implement best practices for SSL certificate handling
    - Ensure ACME protocol compliance for Let's Encrypt integration
    - Provide secure credential management for DNS provider APIs
4. **Improve Operational Efficiency**
    - Reduce manual intervention in certificate lifecycle management
    - Provide comprehensive documentation for server configuration
    - Enable bulk certificate operations for enterprise use

2.2.2 Secondary Objectives

1. **User Experience Enhancement**
    - Intuitive web interface with modern design principles
    - Responsive design for multiple device types
    - Dark mode support for improved usability
2. **System Reliability**
    - Robust error handling and recovery mechanisms
    - Comprehensive logging for troubleshooting
    - Scalable architecture for growing certificate needs

3. **Integration Capabilities**

   - RESTful API design for third-party integrations
   - Support for multiple DNS providers (extensible architecture)
   - CI/CD pipeline compatibility for DevOps workflows

2.3 User Requirements

2.3.1 Functional Requirements

2.3.1.1 Certificate Generation Requirements

1. The system shall generate SSL certificates using Let's Encrypt ACME protocol
2. The system shall support both single domain and wildcard certificate generation
3. The system shall validate domain ownership using DNS-01 challenge method
4. The system shall automatically handle DNS TXT record creation and cleanup
5. The system shall provide real-time feedback during certificate generation process

2.3.1.2 Certificate Management Requirements

1. The system shall display all certificates with their status and expiration dates
2. The system shall provide certificate download capabilities in multiple formats
3. The system shall show certificate validity and remaining days until expiration
4. The system shall support certificate bundle downloads (ZIP format)
5. The system shall provide certificate information parsing and display

2.3.1.3 DNS Provider Integration Requirements

1. The system shall integrate with GoDaddy DNS API for automated DNS operations
2. The system shall support configurable TTL values for DNS records
3. The system shall handle DNS propagation delays automatically
4. The system shall provide extensible architecture for additional DNS providers
5. The system shall validate DNS provider credentials before certificate generation

2.3.1.4 User Interface Requirements

1. The system shall provide a responsive web interface accessible on multiple devices
2. The system shall support dark and light theme modes
3. The system shall display comprehensive documentation for server configuration
4. The system shall provide form validation with user-friendly error messages
5. The system shall show loading states and progress indicators for long operations

2.3.2 Non-Functional Requirements

2.3.2.1 Performance Requirements

1. Certificate generation process shall complete within 5 minutes under normal conditions
2. The web interface shall load within 3 seconds on standard broadband connections
3. The system shall support concurrent certificate generation requests
4. API responses shall have a maximum timeout of 60 seconds for certificate operations

2.3.2.2 Security Requirements

1. DNS provider credentials shall not be stored persistently in the system
2. All API communications shall use HTTPS in production environments
3. Certificate files shall be stored with appropriate file permissions
4. The system shall validate all input parameters to prevent injection attacks

2.3.2.3 Reliability Requirements

1. The system shall have 99% uptime availability during business hours
2. Certificate files shall be persistently stored with backup capabilities
3. The system shall gracefully handle DNS provider API rate limits
4. Error recovery mechanisms shall be implemented for failed certificate generation

2.3.2.4 Usability Requirements

1. The system shall be operable by users with basic web application knowledge
2. Documentation shall be comprehensive and easily accessible within the application
3. Error messages shall be clear and provide actionable guidance
4. The interface shall follow modern web design principles and accessibility standards

2.3.2.5 Scalability Requirements

1. The system shall support management of up to 100 certificates per installation
2. The architecture shall be designed for horizontal scaling through containerization

3. The system shall support multiple DNS provider integrations simultaneously
4. Database integration shall be considered for future certificate metadata management

Chapter 3

Analysis and Design

3.1 System Analysis

The SSL Automation Tool system analysis involves understanding the interaction between various components and the data flow within the system. This chapter presents the comprehensive analysis and design of the system through various UML diagrams and architectural specifications.

3.1.1 System Context

The system operates within the following context:

- Integration with Let's Encrypt Certificate Authority for SSL certificate issuance
- Integration with DNS provider APIs (GoDaddy) for domain validation
- User interaction through modern web browser interfaces
- Containerized deployment on server infrastructure

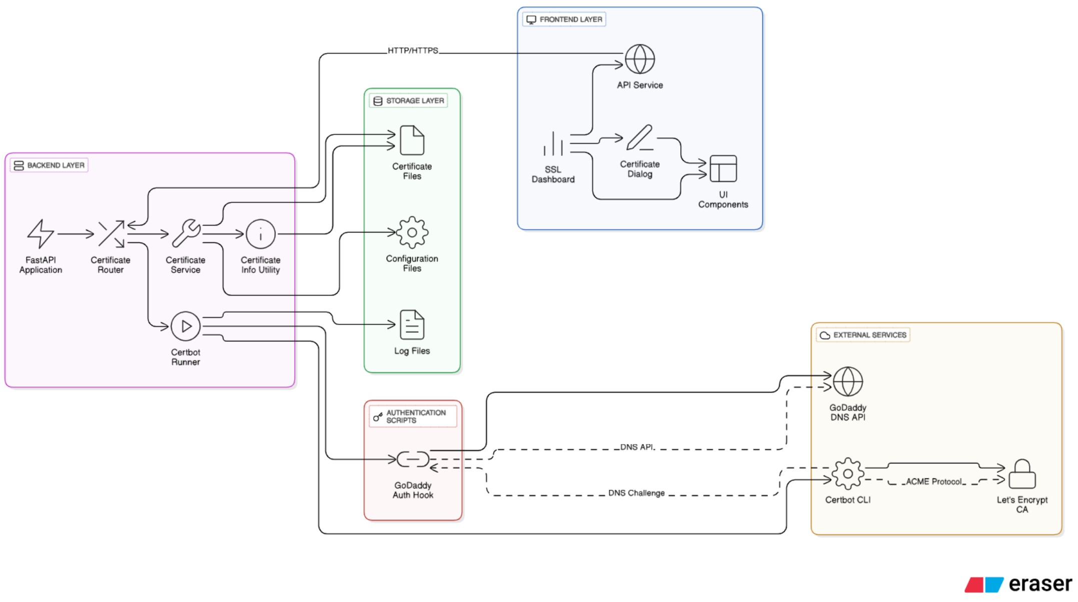

FIGURE 3.1: System Component Architecture

3.2 Activity Diagram

The activity diagram illustrates the workflow of SSL certificate generation process, showing the sequence of activities from user initiation to certificate delivery.

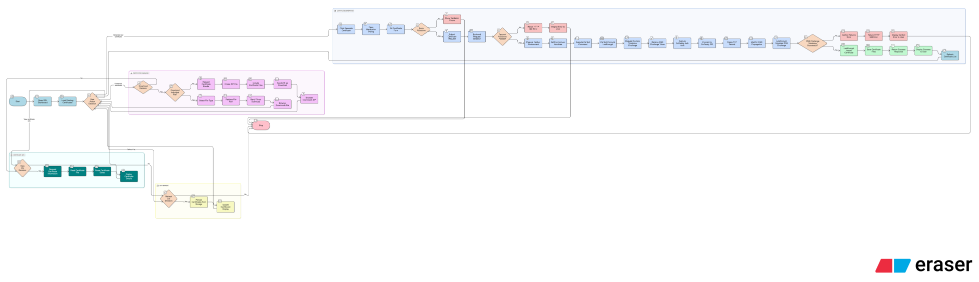

FIGURE 3.2: SSL Certificate Generation Activity Diagram

The certificate generation process involves the following key activities:

1. User provides domain and DNS provider credentials
2. System validates input parameters and credentials
3. Certbot initiates ACME challenge with Let's Encrypt

4. DNS hook script creates TXT record via provider API
5. Let's Encrypt validates domain ownership
6. Certificate files are generated and stored
7. User receives confirmation and download links

3.3 Use Case Diagrams

The use case diagram shows the functional requirements of the system from the user's perspective, identifying the main actors and their interactions with the system.

3.3.1 Primary Use Cases

1. **Generate SSL Certificate**: User initiates certificate generation for a domain
2. **View Certificate Status**: User monitors certificate validity and expiration
3. **Download Certificate**: User downloads certificate files for deployment
4. **Manage DNS Settings**: User configures DNS provider credentials
5. **View Documentation**: User accesses server configuration guides

3.3.2 Secondary Use Cases

1. **List All Certificates**: User views comprehensive certificate inventory
2. **Download Certificate Bundle**: User downloads ZIP archive of certificate files
3. **Check Certificate Info**: User views detailed certificate metadata
4. **Configure Advanced Settings**: User adjusts TTL and propagation settings

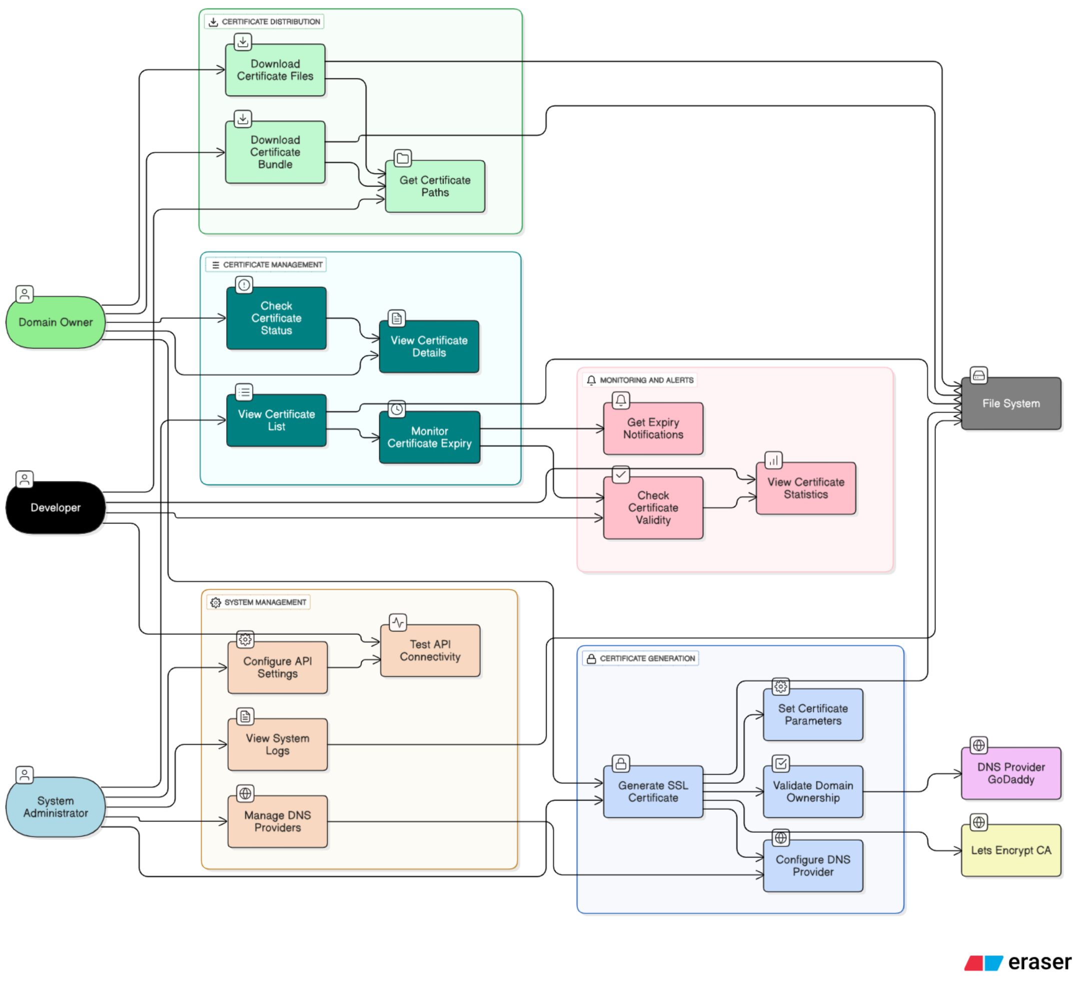

FIGURE 3.3: SSL Automation Tool Use Case Diagram

3.4 Component Diagram

The component diagram shows the high-level software components and their relationships within the SSL Automation Tool architecture.

3.4.1 Frontend Components

- **React Application**: Main user interface component
- **SSL Dashboard**: Certificate management interface
- **Certificate Forms**: Data input and validation components

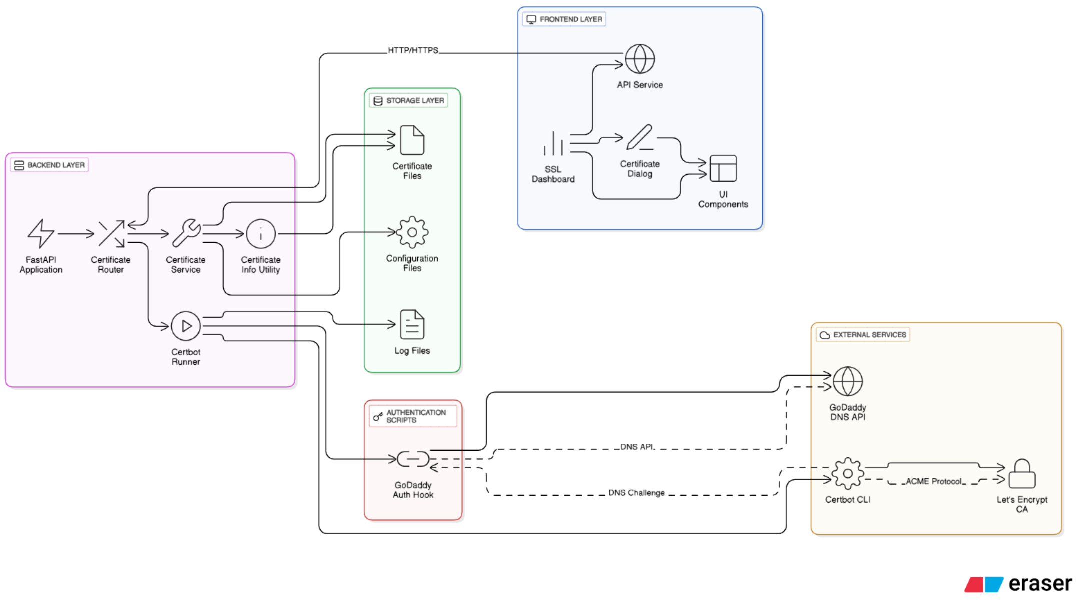

FIGURE 3.4: System Component Architecture

- **Documentation System**: Server configuration guides
- **API Service Layer**: HTTP client for backend communication

3.4.2 Backend Components

- **FastAPI Application**: REST API server
- **Certificate Service**: Business logic for certificate operations
- **Certbot Runner**: Let's Encrypt integration service
- **DNS Hook Scripts**: DNS provider automation components
- **File Management**: Certificate storage and retrieval

3.5 Deployment Diagram

The deployment diagram illustrates the physical architecture and runtime environment of the SSL Automation Tool system.

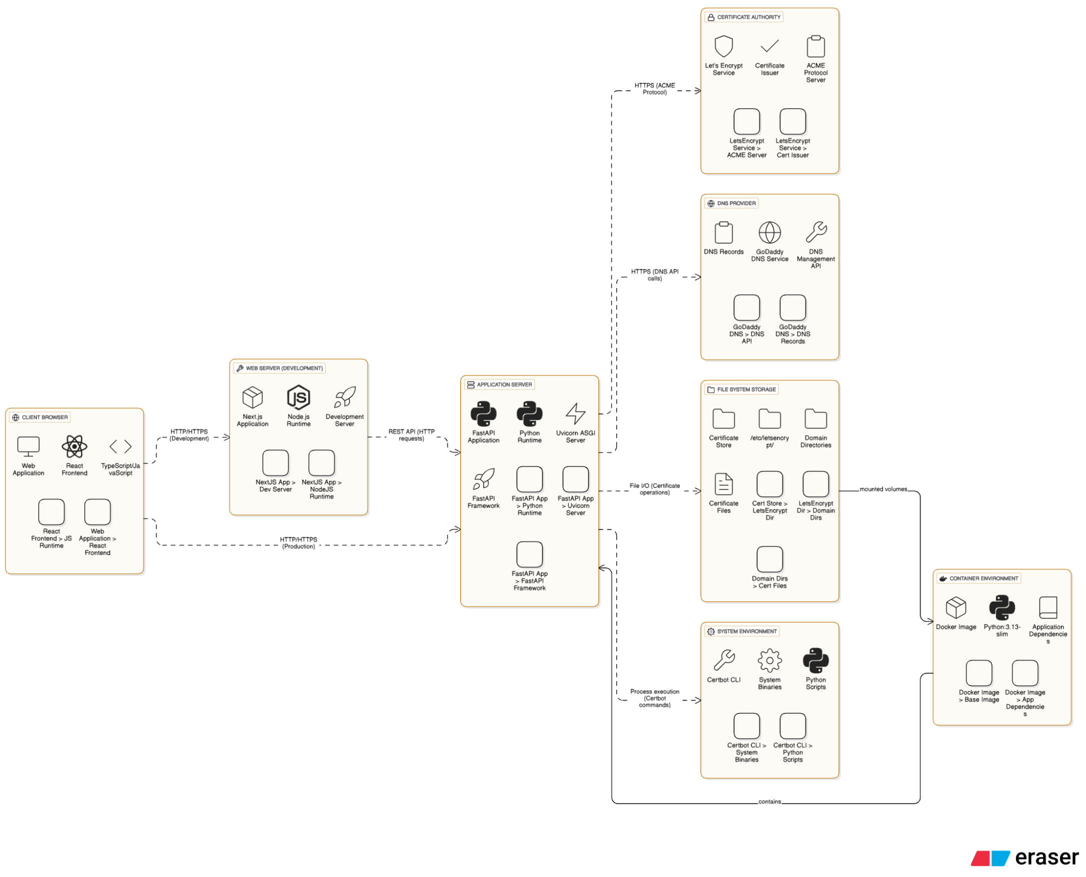

FIGURE 3.5: System Deployment Architecture

3.5.1 Deployment Architecture

- **Client Browser**: User interface execution environment
- **Docker Container**: Application runtime environment
- **Host Server**: Physical or virtual server infrastructure
- **Volume Storage**: Persistent certificate file storage

- **External APIs**: Let's Encrypt and DNS provider services

3.6 Data Flow Diagram

The data flow diagram shows how data moves through the system during certificate generation and management operations.

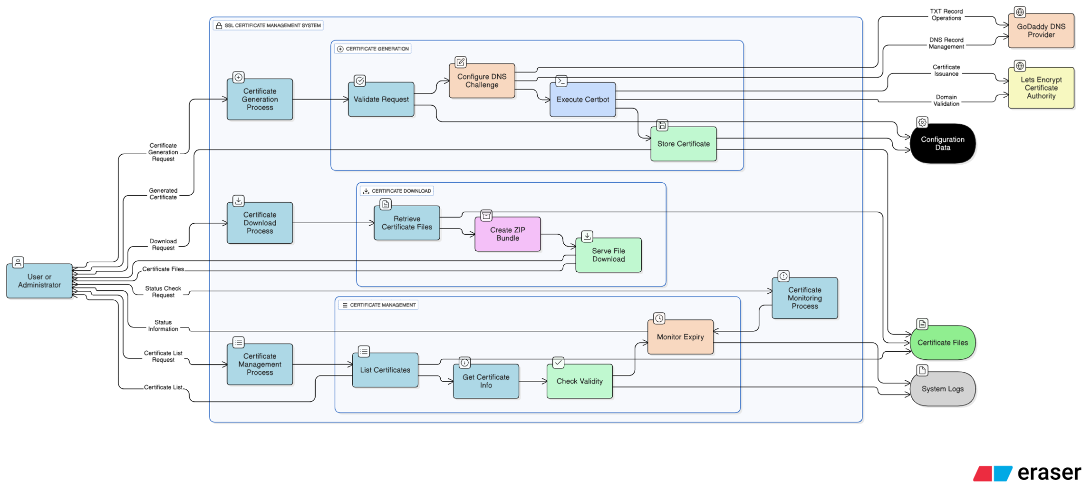

FIGURE 3.6: SSL Certificate Data Flow Diagram

3.6.1 Data Flow Process

1. User submits certificate request with domain and credentials
2. Frontend validates and sends request to backend API
3. Backend initiates Certbot process with challenge data
4. DNS hook script creates TXT record via provider API
5. Let's Encrypt validates domain and issues certificate
6. Certificate files are stored in persistent volume
7. Certificate metadata is returned to user interface

3.7 Functional Decomposition

The functional decomposition diagram breaks down the system functionality into hierarchical components showing the relationship between different system functions.

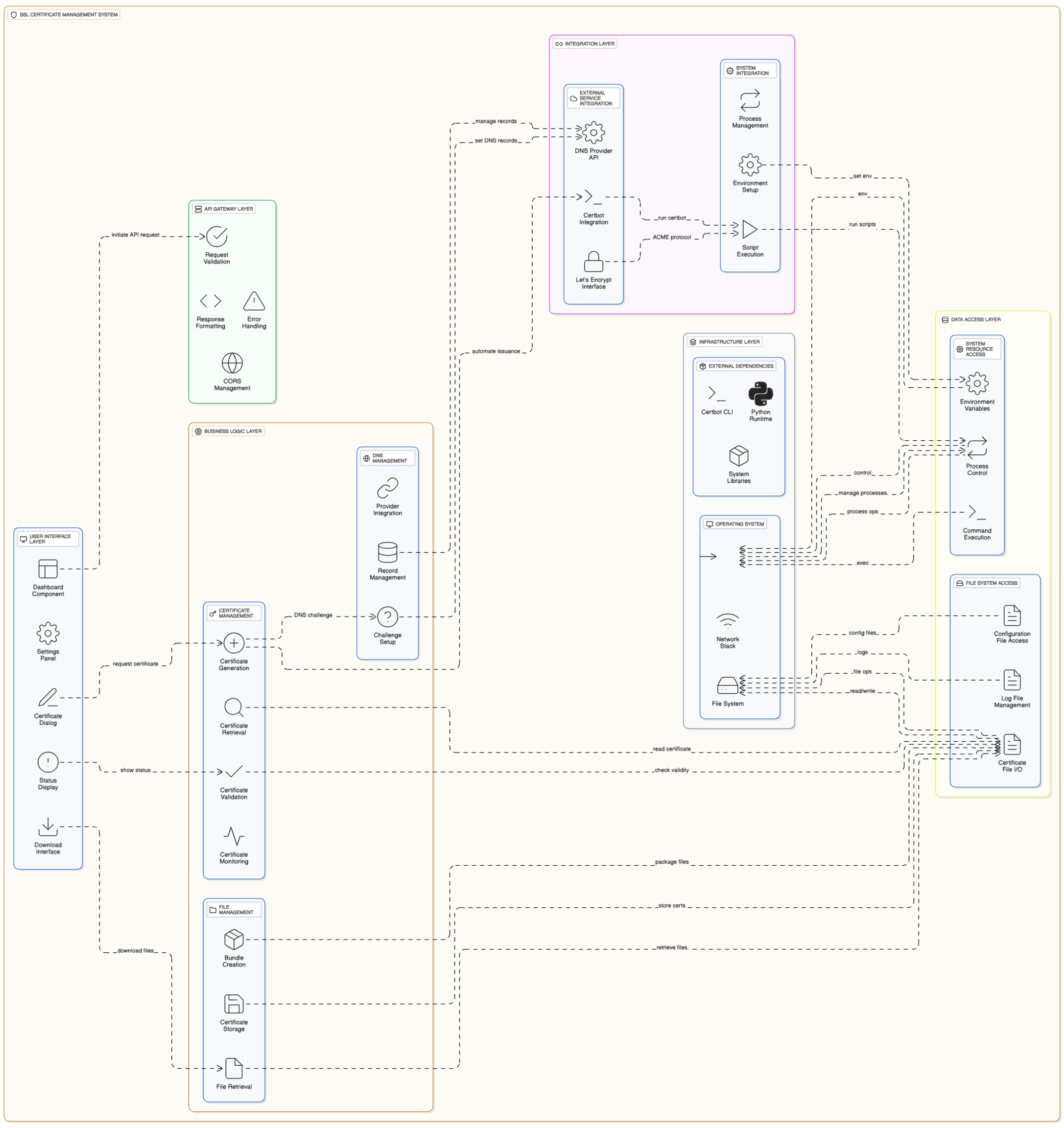

FIGURE 3.7: Functional Decomposition Diagram

3.7.1 Function Hierarchy

- **Certificate Management**
  - Certificate Generation
  - Certificate Validation
  - Certificate Storage
  - Certificate Retrieval
- **DNS Operations**
  - Provider Authentication
  - TXT Record Management
  - Domain Validation
  - Propagation Handling
- **User Interface**
  - Form Management
  - Status Monitoring
  - File Downloads
  - Documentation Display

3.8 Test Procedures and Implementation

The testing strategy ensures system reliability and functionality through comprehensive test coverage.

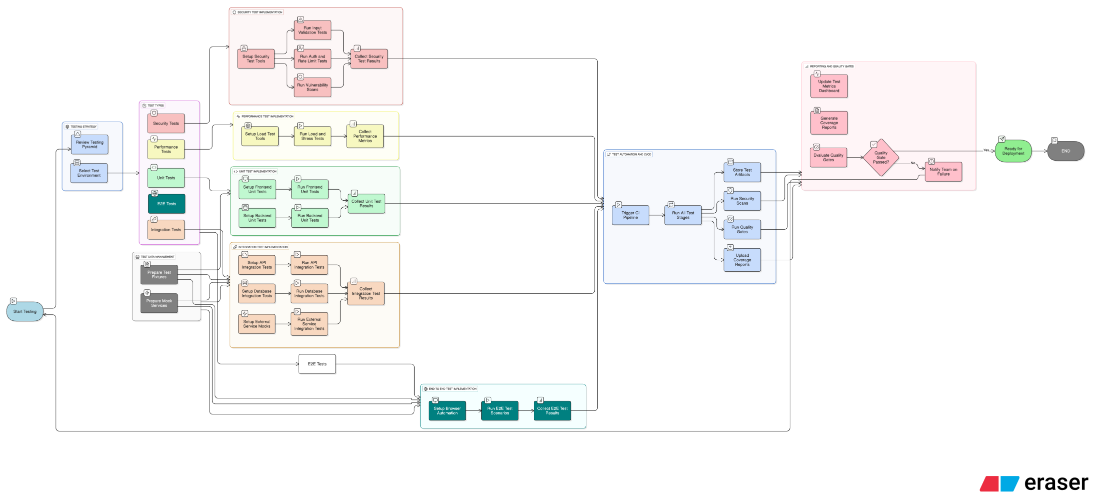
Start Testing
TESTING STRATEGY
Review Testing Pyramid
Select Test Environment
TEST TYPES
Security Tests
Performance Tests
Unit Tests
E2E Tests
Integration Tests
TEST DATA MANAGEMENT
Prepare Test Fixtures
Prepare Mock Services
SECURITY TEST IMPLEMENTATION
Setup Security Test Tools
Run Input Validation Tests
Run Auth and Rate Limit Tests
Run Vulnerability Scans
Collect Security Test Results
PERFORMANCE TEST IMPLEMENTATION
Setup Load Test Tools
Run Load and Stress Tests
Collect Performance Metrics
UNIT TEST IMPLEMENTATION
Setup Frontend Unit Tests
Run Frontend Unit Tests
Setup Backend Unit Tests
Run Backend Unit Tests
Collect Unit Test Results
INTEGRATION TEST IMPLEMENTATION
Setup API Integration Tests
Run API Integration Tests
Setup Database Integration Tests
Run Database Integration Tests
Setup External Service Mocks
Run External Service Integration Tests
Collect Integration Test Results
E2E Tests
END TO END TEST IMPLEMENTATION
Setup Browser Automation
Run E2E Test Scenarios
Collect E2E Test Results
TEST AUTOMATION AND CI/CD
Trigger CI Pipeline
Run All Test Stages
Store Test Artifacts
Run Security Scans
Run Quality Gates
Upload Coverage Reports
REPORTING AND QUALITY GATES
Update Test Metrics Dashboard
Generate Coverage Reports
Evaluate Quality Gates
Quality Gate Passed?
Yes
No
Notify Team on Failure
Ready for Deployment
END
eraser

FIGURE 3.8: Test Procedures and Implementation

3.8.1 Testing Levels

1. **Unit Testing**

   - Individual component functionality testing
   - API endpoint validation
   - DNS integration testing
   - Certificate parsing validation

2. **Integration Testing**

   - Frontend-backend communication testing
   - Let's Encrypt ACME protocol integration
   - DNS provider API integration
   - End-to-end certificate generation workflow

3. **System Testing**

   - Complete system functionality validation
   - Performance testing under load

- Security testing for credential handling
- Browser compatibility testing

4. **User Acceptance Testing**
   - User interface usability testing
   - Documentation completeness validation
   - Real-world scenario testing
   - Error handling verification

3.8.2 Test Implementation Strategy

- **Automated Testing**: Unit tests for utility functions and API endpoints
- **Manual Testing**: User interface and integration scenarios
- **Staging Environment**: Let's Encrypt staging server for safe testing
- **Continuous Integration**: Automated test execution on code changes

Chapter 4

User Manual

4.1 User Manual

This chapter provides comprehensive guidance for using the SSL Automation Tool, including installation procedures, operation instructions, and troubleshooting guidelines.

4.1.1 System Requirements

Before using the SSL Automation Tool, ensure your system meets the following requirements:

4.1.1.1 For End Users

- Modern web browser (Chrome 90+, Firefox 88+, Safari 14+, Edge 90+)
- Internet connection for accessing Let's Encrypt and DNS provider APIs
- GoDaddy account with API credentials (for DNS validation)
- Domain administrative access for certificate deployment

4.1.1.2 For System Administrators

- Docker 20.10+ and Docker Compose 2.0+
- Linux-based server (Ubuntu 20.04+ recommended)
- Minimum 2GB RAM and 10GB storage space
- Network access to external APIs (Let's Encrypt, DNS providers)

4.1.2 Installation Guide

4.1.2.1 Quick Start with Docker

1. Clone the project repository:

```
git clone <repository-url>
cd ssl-automation-tool
```

2. Start the services using Docker Compose:

```
docker compose up -d
```

3. Access the application at `http://localhost:3000`

4.1.2.2 Development Setup

For developers who want to run the system in development mode:

1. **Backend Setup**:

```
cd ssl-automation-backend
pip install uv
uv sync
uv run uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
```

2. **Frontend Setup**:

```
cd ssl-automation-frontend
pnpm install
pnpm dev
```

4.2 Operations Manual / Menu Explanation

4.2.1 Main Dashboard

Upon accessing the application, users are presented with the SSL Certificate Dashboard containing the following sections:

4.2.1.1 Statistics Cards

- **Total Certificates**: Shows the total number of managed certificates
- **Valid Certificates**: Displays count of currently valid certificates
- **Expiring Soon**: Alerts for certificates expiring within 30 days
- **Invalid Certificates**: Count of expired or invalid certificates

4.2.1.2 Certificate List

The certificate list displays all managed certificates with the following information:

- **Domain**: The domain name for which the certificate was issued
- **Status**: Visual indicator showing certificate validity status
- **Issued Date**: When the certificate was generated
- **Expires On**: Certificate expiration date
- **Actions**: Download options for certificate files

4.2.1.3 Action Buttons

- **Generate Certificate**: Opens the certificate generation dialog
- **Refresh List**: Updates the certificate list with latest information
- **Download Individual Files**: Downloads specific certificate components
- **Download ZIP Bundle**: Downloads complete certificate package

4.2.2 Certificate Generation Process

4.2.2.1 Step 1: Access Generation Dialog

Click the "Generate Certificate" button to open the certificate generation form.

4.2.2.2 Step 2: Enter Domain Information

- **Domain**: Enter the target domain (e.g., example.com or *.example.com for wildcard)

- **Email**: Provide email address for Let’s Encrypt registration

4.2.2.3 Step 3: Configure DNS Provider

- **DNS Provider**: Select "GoDaddy" from the dropdown
- **API Key**: Enter your GoDaddy API key
- **API Secret**: Enter your GoDaddy API secret
- **Base Domain**: Specify the base domain for DNS operations

4.2.2.4 Step 4: Advanced Settings (Optional)

- **TTL**: DNS record time-to-live (default: 600 seconds)
- **Propagation Delay**: Wait time for DNS propagation (default: 60 seconds)

4.2.2.5 Step 5: Generate Certificate

Click "Generate Certificate" to start the process. Monitor the progress indicators for:

- Request validation
- DNS challenge setup
- Domain verification
- Certificate generation
- File storage completion

4.2.3 Certificate Download and Deployment

4.2.3.1 Individual File Downloads

For each certificate, users can download:

- **fullchain.pem**: Complete certificate chain (certificate + intermediate CA)
- **privkey.pem**: Private key file (keep secure)

4.2.3.2 ZIP Bundle Download

Download a complete ZIP archive containing both certificate files for convenient deployment.

4.2.3.3 Server Configuration

Access the Documentation section for detailed server configuration guides:

- Nginx SSL configuration
- Apache SSL setup
- Docker deployment strategies
- Troubleshooting common issues

4.3 Program Specifications / Flow Charts

4.3.1 Certificate Generation Workflow

The following flowchart illustrates the complete certificate generation process:

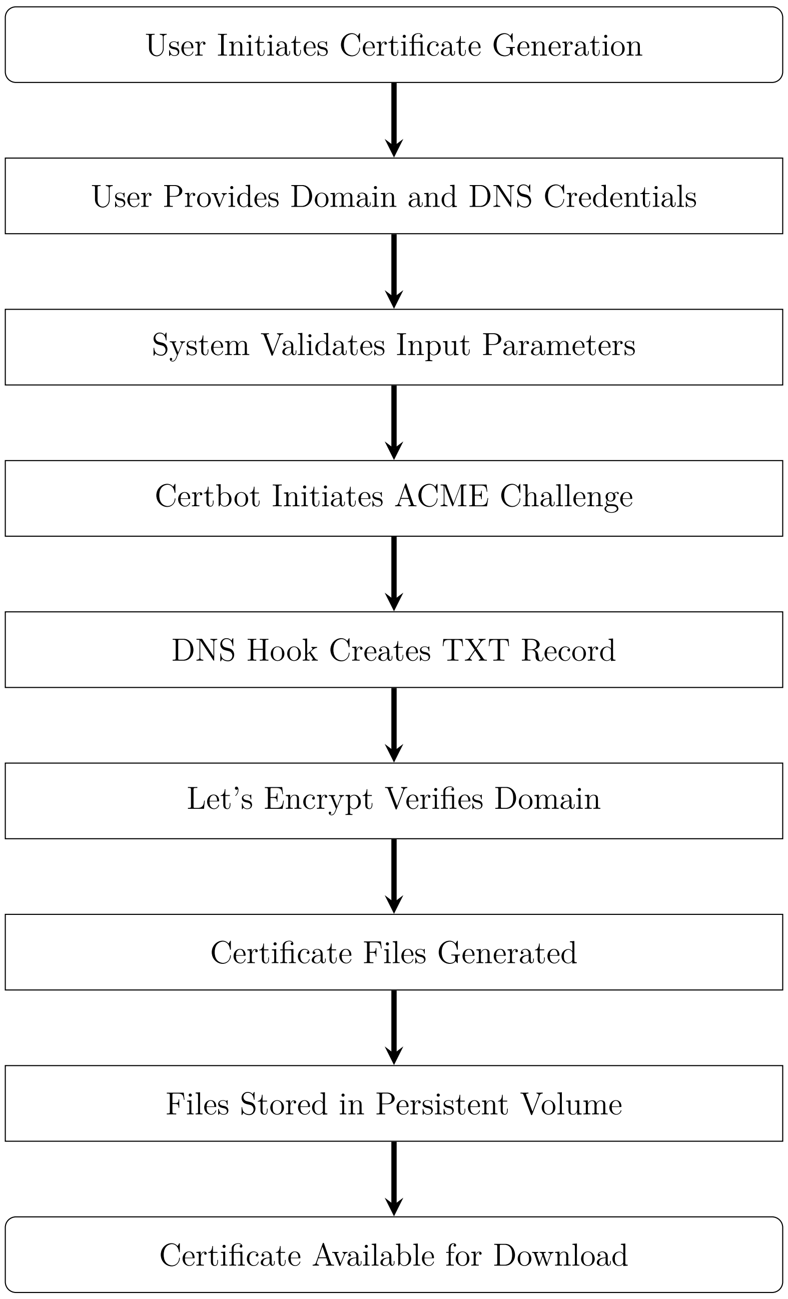
User Initiates Certificate Generation
User Provides Domain and DNS Credentials
System Validates Input Parameters
Certbot Initiates ACME Challenge
DNS Hook Creates TXT Record
Let's Encrypt Verifies Domain
Certificate Files Generated
Files Stored in Persistent Volume
Certificate Available for Download

FIGURE 4.1: Certificate Generation Process Flow

4.3.2 Error Handling Workflow

The system implements comprehensive error handling at each stage:

1. **Input Validation Errors**: Form validation provides immediate feedback
2. **DNS API Errors**: Credential validation and API response handling

3. **ACME Challenge Errors**: Retry mechanisms with exponential backoff
4. **Certificate Generation Errors**: Detailed error reporting and recovery guidance

4.3.3 System Architecture Flow

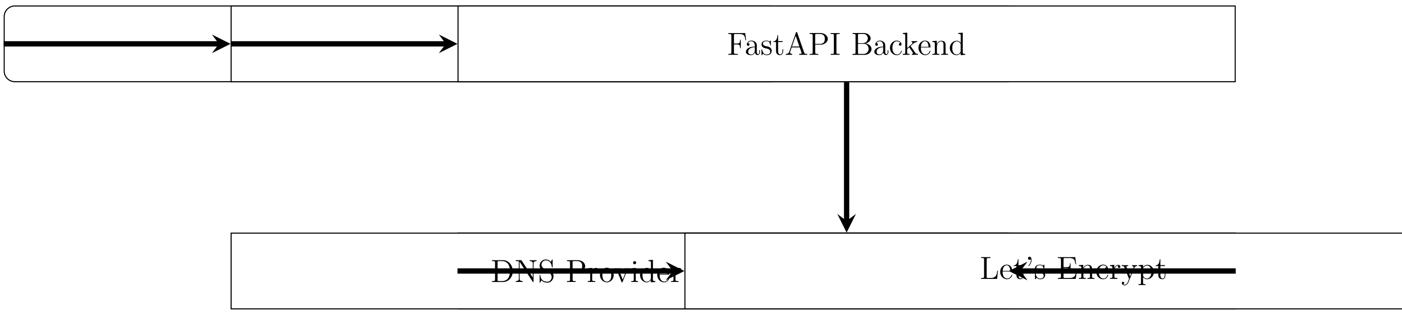

Figure 4.2: System Architecture Data Flow

4.4 Drawbacks and Limitations

4.4.1 Current System Limitations

1. **DNS Provider Support**
   - Currently limited to GoDaddy DNS provider only
   - No support for other major providers (Cloudflare, AWS Route53, etc.)
   - Manual configuration required for each DNS provider
2. **Authentication and Security**
   - No built-in user authentication system
   - API credentials handled client-side only
   - No role-based access control implementation
3. **Certificate Management**

- No automatic certificate renewal scheduling
- No certificate revocation capabilities
- Limited certificate metadata tracking

4. **Scalability Constraints**
    - Single-instance deployment architecture
    - No database integration for certificate metadata
    - Limited concurrent request handling

4.4.2 Technical Limitations

1. **Dependency on External Services**
    - Reliance on Let's Encrypt service availability
    - DNS provider API rate limiting
    - Internet connectivity requirements
2. **Error Recovery**
    - Limited automated error recovery mechanisms
    - Manual intervention required for complex failures
    - No automatic retry for transient failures

4.5 Proposed Enhancements

4.5.1 Short-term Improvements

1. **Multi-DNS Provider Support**

- Implement Cloudflare DNS integration
- Add AWS Route53 support
- Create abstract DNS provider interface

2. **Enhanced Security**

   - Implement JWT-based authentication
   - Add API key management system
   - Implement request rate limiting

3. **Improved User Experience**

   - Add certificate renewal notifications
   - Implement bulk certificate operations
   - Enhanced error messaging and recovery guidance

4.5.2 Long-term Enhancements

1. **Enterprise Features**

   - Multi-tenant architecture
   - Advanced monitoring and analytics
   - Integration with CI/CD pipelines

2. **Automation Improvements**

   - Automatic certificate renewal scheduling
   - Certificate lifecycle automation
   - Integration with certificate deployment tools

3. **Scalability Enhancements**

- Database integration for metadata management
- Microservices architecture implementation
- Horizontal scaling capabilities

4.6 Conclusions

The SSL Automation Tool successfully addresses the primary challenge of automating SSL certificate generation and management through Let's Encrypt integration. The system provides:

- Streamlined certificate generation with DNS-01 validation
- User-friendly web interface with comprehensive documentation
- Robust backend API with proper error handling
- Container-based deployment for easy scaling
- Extensible architecture for future enhancements

The project demonstrates effective integration of modern web technologies (React 19, Next.js 15, FastAPI) with established certificate management tools (Certbot, Let's Encrypt) to create a practical solution for SSL certificate automation.

Future development should focus on expanding DNS provider support, implementing authentication mechanisms, and adding advanced monitoring capabilities to create a comprehensive enterprise-grade certificate management platform.

Appendix A

User Interface Screens

This appendix contains screenshots and descriptions of the user interface screens of the SSL Automation Tool.

A.1 Main Dashboard

The SSL Certificate Dashboard serves as the central hub for certificate management operations. It provides an overview of all certificates, their status, and quick access to essential functions.

A.1.1 Dashboard Overview

The main dashboard features:

- Statistics cards showing certificate counts and status
- Certificate list with sortable columns
- Quick action buttons for certificate operations

- Responsive design for mobile and desktop use

A.1.2 Statistics Section

The statistics cards display:

- **Total Certificates**: Aggregate count of all managed certificates
- **Valid Certificates**: Number of currently valid certificates
- **Expiring Soon**: Certificates expiring within 30 days (yellow warning)
- **Invalid Certificates**: Expired or problematic certificates (red alert)

A.2 Certificate Generation Dialog

The certificate generation interface provides a comprehensive form for creating new SSL certificates with proper validation and user guidance.

A.2.1 Basic Information Fields

- **Domain**: Target domain for certificate generation
- **Email**: Contact email for Let's Encrypt registration
- **DNS Provider**: Currently supporting GoDaddy with dropdown selection

A.2.2 DNS Configuration Section

- **API Key**: Secure input field for DNS provider API key

- **API Secret**: Password-masked field for API secret
- **Base Domain**: Base domain for DNS record management

A.2.3 Advanced Settings (Expandable)

- **TTL Setting**: Configurable DNS record time-to-live (default: 600)
- **Propagation Delay**: DNS propagation wait time (default: 60 seconds)

A.2.4 Progress Indicators

During certificate generation, the interface shows:

- Real-time progress updates
- Step-by-step process indicators
- Success/error notifications
- Loading animations for user feedback

A.3 Certificate List View

The certificate list provides comprehensive management capabilities with sortable columns and action buttons.

A.3.1 Column Information

- **Domain**: Certificate domain name with support for wildcards

- **Status Badge**: Color-coded status indicator
  - Green: Valid certificate (>30 days remaining)
  - Yellow: Expiring soon (7-30 days remaining)
  - Red: Expired or invalid (<7 days or expired)
- **Issued Date**: Certificate generation timestamp
- **Expires On**: Certificate expiration date
- **Actions**: Download and management options

A.3.2 Action Buttons

Each certificate row provides:

- **Download Fullchain**: Downloads fullchain.pem file
- **Download Private Key**: Downloads privkey.pem file
- **Download ZIP**: Downloads complete certificate bundle
- **View Details**: Shows detailed certificate information

A.4 Documentation Interface

The documentation system provides comprehensive guides for server configuration and certificate deployment.

A.4.1 Tab Navigation

The documentation is organized into tabs:

- **Overview**: General information about SSL certificates
- **Nginx**: Complete Nginx SSL configuration guide
- **Apache**: Apache HTTP server SSL setup
- **Docker**: Container-based deployment strategies
- **Troubleshooting**: Common issues and solutions

A.4.2 Code Examples

Each documentation section includes:

- Syntax-highlighted configuration examples
- Copy-to-clipboard functionality
- Step-by-step implementation guides
- Best practices and security recommendations

A.5 Dark Mode Interface

The application supports both light and dark themes with automatic detection and manual toggle capabilities.

A.5.1 Theme Features

- **Automatic Detection**: Respects system theme preferences
- **Manual Toggle**: Theme switcher in the header
- **Consistent Styling**: All components adapt to theme changes
- **Accessibility**: High contrast ratios for readability

A.6 Mobile Responsive Design

The interface adapts to various screen sizes and device orientations.

A.6.1 Mobile Adaptations

- **Responsive Grid**: Statistics cards stack vertically on mobile
- **Collapsible Tables**: Certificate list optimized for mobile viewing
- **Touch-Friendly**: Buttons and interactions optimized for touch
- **Navigation**: Hamburger menu for mobile navigation

A.6.2 Tablet Interface

- **Hybrid Layout**: Combines desktop and mobile elements
- **Split View**: Side-by-side content where appropriate
- **Touch Navigation**: Optimized for tablet interaction patterns

A.7 Error and Success Notifications

The application provides comprehensive feedback through toast notifications and inline messaging.

A.7.1 Notification Types

- **Success Notifications**: Green toast for successful operations
- **Error Messages**: Red toast with detailed error information
- **Warning Alerts**: Yellow notifications for important information
- **Info Messages**: Blue notifications for general information

A.7.2 Error Handling Interface

- **Form Validation**: Real-time field validation with error highlighting
- **API Error Display**: Detailed error messages from backend operations
- **Retry Mechanisms**: Options to retry failed operations
- **Help Context**: Links to relevant documentation for error resolution

Appendix B

Sample Program Code

This appendix contains key program code samples that demonstrate the core functionality and implementation details of the SSL Automation Tool.

B.1 Frontend Code Samples

B.1.1 SSL Dashboard Component

The main dashboard component that manages certificate display and operations:

```
// SSLDashboard.tsx - Main certificate management interface
import React, { useState, useEffect } from 'react';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { SSLApiService } from '@/lib/api';
import { Certificate } from '@/types/ssl';

export function SSLDashboard() {
```

```
const [certificates, setCertificates] = useState<Certificate[]>([]);
const [loading, setLoading] = useState(true);

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

const loadCertificates = async () => {
  try {
    setLoading(true);
    const response = await SSLApiService.listCertificates();
    setCertificates(response.certificates);
  } catch (error) {
    console.error('Failed to load certificates:', error);
  } finally {
    setLoading(false);
  }
};

const downloadCertificate = async (domain: string, type: string) => {
  try {
    const blob = await SSLApiService.downloadCertificateFile(domain, type);
    SSLApiService.triggerDownload(blob, `${domain}-${type}.pem`);
  } catch (error) {
    console.error('Download failed:', error);
  }
};
```

```
const getStatusColor = (cert: Certificate) => {
  if (!cert.valid) return 'red';
  const daysUntilExpiry = Math.ceil(
    (new Date(cert.expires_on).getTime() - Date.now()) / (1000 * 60 * 60 * 24)
  );
  if (daysUntilExpiry < 7) return 'red';
  if (daysUntilExpiry < 30) return 'yellow';
  return 'green';
};

return (
  <div className="space-y-6">
    <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
      {/* Statistics Cards */}
      <Card>
        <CardHeader>
          <CardTitle>Total Certificates</CardTitle>
        </CardHeader>
        <CardContent>
          <div className="text-2xl font-bold">{certificates.length}</div>
        </CardContent>
      </Card>
      {/* Additional statistics cards... */}
    </div>

    {/* Certificate List */}
    <Card>
      <CardHeader>
        <CardTitle>Certificate Management</CardTitle>
      </CardHeader>
      <CardContent>
        {certificates.map((cert) => (
          <div key={cert.domain} className="border-b py-4">
            <div className="flex justify-between items-center">
              <div>
                <h3 className="font-medium">{cert.domain}</h3>
                <p className="text-sm text-gray-500">
                  Expires: {new Date(cert.expires_on).toLocaleDateString()}
                </p>
              </div>
              <div className="flex space-x-2">
                <Button
                  onClick={() => downloadCertificate(cert.domain, 'fullchain')}
                  size="sm"
                >
                  Download Fullchain
                </Button>
                <Button
                  onClick={() => downloadCertificate(cert.domain, 'privkey')}
                  size="sm"
                >
                  Download Private Key
                </Button>
              </div>
            </div>
          </div>
```

```
        ))}
      </CardContent>
    </Card>
  </div>
 );
}
```

B.1.2 Certificate Generation Form

Form component for creating new SSL certificates:

```
// CertificateGenerationDialog.tsx - Certificate creation interface
import React from 'react';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import * as z from 'zod';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Label } from '@/components/ui/label';
import { SSLApiService } from '@/lib/api';

const formSchema = z.object({
  domain: z.string().min(1, 'Domain is required'),
  email: z.string().email('Valid email is required'),
  dns_provider: z.literal('godaddy'),
  api_key: z.string().min(1, 'API key is required'),
  api_secret: z.string().min(1, 'API secret is required'),
  base_domain: z.string().min(1, 'Base domain is required'),
  ttl: z.number().min(300).max(86400).optional(),
  propagation_delay: z.number().min(30).max(600).optional(),
});

type FormData = z.infer<typeof formSchema>;

export function CertificateGenerationDialog({ onSuccess, onClose }) {
  const [isGenerating, setIsGenerating] = useState(false);

  const form = useForm<FormData>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      dns_provider: 'godaddy',
      ttl: 600,
      propagation_delay: 60,
    },
  });

  const onSubmit = async (data: FormData) => {
    try {
      setIsGenerating(true);
      const response = await SSLApiService.generateCertificate(data);

      if (response.success) {
        toast.success('Certificate generated successfully!');
        onSuccess();
        onClose();
      } else {
```

```
      toast.error(‘Generation failed: ${response.error}‘);
    }
  } catch (error) {
    toast.error(’Failed to generate certificate’);
    console.error(error);
  } finally {
    setIsGenerating(false);
  }
};

return (
  <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">
    <div>
      <Label htmlFor="domain">Domain</Label>
      <Input
        id="domain"
        placeholder="example.com or *.example.com"
        {...form.register(’domain’)}
      />
      {form.formState.errors.domain && (
        <p className="text-red-500 text-sm">
          {form.formState.errors.domain.message}
        </p>
      )}
    </div>

    <div>
      <Label htmlFor="email">Email</Label>
      <Input
        id="email"
        type="email"
        placeholder="admin@example.com"
        {...form.register('email')}
      />
    </div>

    <div>
      <Label htmlFor="api_key">GoDaddy API Key</Label>
      <Input
        id="api_key"
        type="password"
        {...form.register('api_key')}
      />
    </div>

    <div>
      <Label htmlFor="api_secret">GoDaddy API Secret</Label>
      <Input
        id="api_secret"
        type="password"
        {...form.register('api_secret')}
      />
    </div>

    <Button
      type="submit"
```

```
        disabled={isGenerating}
        className="w-full"
      >
        {isGenerating ? 'Generating...' : 'Generate Certificate'}
      </Button>
    </form>
  );
}
```

B.2 Backend Code Samples

B.2.1 FastAPI Main Application

The core FastAPI application setup with CORS and routing:

```
# app/main.py - FastAPI application entry point
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.routes import certs

app = FastAPI(
    title="SSL Automation API",
    description="Automated SSL certificate generation using Let's Encrypt",
    version="1.0.0"
)

# CORS configuration for frontend integration
app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "http://localhost:3000",
        "http://localhost:3001",
        "http://127.0.0.1:3000",
        "http://127.0.0.1:3001"
    ],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include certificate management routes
app.include_router(certs.router, prefix="/api/v1")

@app.get("/")
async def root():
    return {"message": "SSL Automation API is running"}

@app.get("/health")
async def health_check():
    return {"status": "healthy", "service": "ssl-automation-api"}
```

B.2.2 Certificate Service Implementation

Core business logic for certificate operations:

```
# app/services/cert_service.py - Certificate file operations
```

```
import os

import zipfile

from pathlib import Path

from typing import Optional, Dict, List

from app.utils.cert_info import get_cert_info


class CertService:

    @staticmethod

    def get_cert_paths(domain: str) -> Optional[Dict[str, str]]:

        """Get paths to certificate files for a domain."""

        cert_dir = Path(f"/etc/letsencrypt/live/{domain}")


        if not cert_dir.exists():

            return None


        fullchain_path = cert_dir / "fullchain.pem"

        privkey_path = cert_dir / "privkey.pem"


        if fullchain_path.exists() and privkey_path.exists():

            return {

                "fullchain": str(fullchain_path),

                "privkey": str(privkey_path)

            }


        return None


    @staticmethod

    def create_cert_zip(domain: str) -> Optional[str]:
        """Create a ZIP archive containing all certificate files."""
        cert_paths = CertService.get_cert_paths(domain)
        if not cert_paths:
            return None

        zip_path = f"/tmp/{domain}-certificates.zip"

        with zipfile.ZipFile(zip_path, 'w') as zipf:
            for file_type, file_path in cert_paths.items():
                if os.path.exists(file_path):
                    zipf.write(file_path, f"{file_type}.pem")

        return zip_path if os.path.exists(zip_path) else None

    @staticmethod
    def list_all_certificates() -> List[Dict]:
        """List all managed certificates with their information."""
        certificates = []
        le_dir = Path("/etc/letsencrypt/live")

        if not le_dir.exists():
            return certificates

        for domain_dir in le_dir.iterdir():
            if domain_dir.is_dir():
                cert_info = get_cert_info(domain_dir.name)
                if cert_info:
                    certificates.append(cert_info)
        return certificates
```

B.2.3 Certbot Runner Service

Service for executing Certbot commands with DNS validation:

```
# app/services/certbot_runner.py - Certbot execution wrapper
import subprocess
import os
import logging
from typing import Dict, Tuple

logger = logging.getLogger(__name__)

class CertbotRunner:
    @staticmethod
    def generate_certificate(
        domain: str,
        email: str,
        dns_provider: str,
        api_key: str,
        api_secret: str,
        base_domain: str,
        ttl: int = 600,
        propagation_delay: int = 60
    ) -> Tuple[bool, str]:
        """Execute Certbot to generate SSL certificate."""
```

```
# Set environment variables for DNS hook
env = os.environ.copy()
env.update({
    'GODADDY_API_KEY': api_key,
    'GODADDY_API_SECRET': api_secret,
    'BASE_DOMAIN': base_domain,
    'DNS_TTL': str(ttl),
    'PROPAGATION_DELAY': str(propagation_delay)
})

# Construct Certbot command
cmd = [
    'certbot', 'certonly',
    '--manual',
    '--preferred-challenges=dns',
    '--manual-auth-hook', '/app/scripts/godaddy_auth_hook.py',
    '--manual-cleanup-hook', '/app/scripts/godaddy_cleanup_hook.py',
    '--domain', domain,
    '--email', email,
    '--agree-tos',
    '--non-interactive',
    '--manual-public-ip-logging-ok'
]

try:
    logger.info(f"Starting certificate generation for {domain}")
```

```
        result = subprocess.run(
            cmd,
            env=env,
            capture_output=True,
            text=True,
            timeout=300  # 5 minute timeout
        )

        if result.returncode == 0:
            logger.info(f"Certificate generated successfully for {domain}")
            return True, result.stdout
        else:
            logger.error(f"Certbot failed for {domain}: {result.stderr}")
            return False, result.stderr

    except subprocess.TimeoutExpired:
        logger.error(f"Certbot timeout for {domain}")
        return False, "Certificate generation timed out"
    except Exception as e:
        logger.error(f"Certbot execution failed for {domain}: {str(e)}")
        return False, f"Execution failed: {str(e)}"
```

B.3 DNS Integration Code

B.3.1 GoDaddy DNS Hook Script

Python script for automated DNS challenge handling:

```
#!/usr/bin/env python3

# app/scripts/godaddy_auth_hook.py - GoDaddy DNS authentication hook


import os

import sys

import requests

import time

import logging


logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(__name__)


def create_txt_record():

    """Create TXT record for DNS challenge validation."""


    # Get environment variables set by Certbot

    domain = os.environ.get('CERTBOT_DOMAIN')

    validation = os.environ.get('CERTBOT_VALIDATION')


    # Get GoDaddy API credentials

    api_key = os.environ.get('GODADDY_API_KEY')

    api_secret = os.environ.get('GODADDY_API_SECRET')

    base_domain = os.environ.get('BASE_DOMAIN')

    ttl = int(os.environ.get('DNS_TTL', 600))

    delay = int(os.environ.get('PROPAGATION_DELAY', 60))


    if not all([domain, validation, api_key, api_secret, base_domain]):

        logger.error("Missing required environment variables")
        sys.exit(1)

    # Construct record name
    record_name = f"_acme-challenge.{domain}"
    if domain.startswith('*.'):
        record_name = f"_acme-challenge.{domain[2:]}"

    # Remove base domain from record name for GoDaddy API
    if record_name.endswith(f".{base_domain}"):
        record_name = record_name[:-len(f".{base_domain}")]

    # GoDaddy API headers
    headers = {
        'Authorization': f'sso-key {api_key}:{api_secret}',
        'Content-Type': 'application/json'
    }

    # Record data
    record_data = [{
        'data': validation,
        'ttl': ttl
    }]

    # API endpoint
    url = f"https://api.godaddy.com/v1/domains/{base_domain}/records/TXT/{record_name

    try:
        logger.info(f"Creating TXT record: {record_name} = {validation}")
```

```
        response = requests.put(url, json=record_data, headers=headers)
        response.raise_for_status()

        logger.info(f"TXT record created successfully")
        logger.info(f"Waiting {delay} seconds for DNS propagation...")

        time.sleep(delay)

        logger.info("DNS propagation wait complete")

    except requests.exceptions.RequestException as e:
        logger.error(f"Failed to create TXT record: {e}")
        sys.exit(1)

if __name__ == "__main__":
    create_txt_record()
```

B.4 API Models and Types

B.4.1 Pydantic Models

Data validation models for API requests and responses:

```
# app/models/models.py - Pydantic data models
from pydantic import BaseModel, EmailStr
from typing import Optional, List
from datetime import datetime
```

```
class CertRequest(BaseModel):
    """Certificate generation request model."""
    domain: str
    email: EmailStr
    dns_provider: str = "godaddy"
    api_key: str
    api_secret: str
    base_domain: str
    ttl: Optional[int] = 600
    propagation_delay: Optional[int] = 60

class CertFetchRequest(BaseModel):
    """Certificate retrieval request model."""
    domain: str

class Certificate(BaseModel):
    """Certificate information model."""
    domain: str
    issued_on: str
    expires_on: str
    valid: bool
    fullchain_path: Optional[str] = None
    privkey_path: Optional[str] = None

class CertificateListResponse(BaseModel):
    """Certificate list response model."""
    certificates: List[Certificate]
    total: int


class CertificateGenerationResponse(BaseModel):
    """Certificate generation response model."""
    success: bool
    message: str
    domain: Optional[str] = None
    error: Optional[str] = None
    output: Optional[str] = None
```

B.5 Configuration Files

B.5.1 Docker Configuration

Docker Compose setup for containerized deployment:

```
# docker-compose.yml - Container orchestration
version: '3.8'


services:
  ssl-automation-backend:
    build: .
    ports:
      - "80:8000"
    volumes:
      - .:/app
      - cert-data:/etc/letsencrypt
    environment:
      - PYTHONPATH=/app
    working_dir: /app
    command: uv run uvicorn app.main:app --host 0.0.0.0 --port 8000

volumes:
  cert-data:
    driver: local
```

B.5.2 Python Project Configuration

Modern Python project setup with UV package manager:

```
# pyproject.toml - Python project configuration
[project]
name = "ssl-automation-backend"
version = "0.1.0"
description = "Automated SSL certificate generation using Let’s Encrypt"
readme = "README.md"
requires-python = ">=3.13"

dependencies = [
    "fastapi>=0.115.12",
    "uvicorn>=0.34.3",
    "pydantic>=2.5.0",
    "pydantic[email]>=2.5.0",
    "certbot>=4.1.1",
    "cryptography>=45.0.4",
    "requests>=2.31.0",
    "python-multipart>=0.0.6"
]

[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[tool.uv]
dev-dependencies = [
    "pytest>=7.0.0",
    "pytest-asyncio>=0.21.0",
    "httpx>=0.24.0"
]
```

Appendix C

Additional Documentation

This appendix contains additional technical documentation, API references, and deployment guides that support the SSL Automation Tool implementation.

C.1 API Documentation

C.1.1 REST API Endpoints

The SSL Automation Tool provides a comprehensive REST API for certificate management operations.

C.1.1.1 Certificate Generation

Endpoint: `POST /api/v1/generate-cert`

Description: Initiates SSL certificate generation using Let's Encrypt with DNS validation.

Request Body:

```
{
  "domain": "example.com",
  "email": "admin@example.com",
  "dns_provider": "godaddy",
  "api_key": "your_godaddy_api_key",
  "api_secret": "your_godaddy_api_secret",
  "base_domain": "example.com",
  "ttl": 600,
  "propagation_delay": 60
}
```

Response:

```
{
  "success": true,
  "message": "Certificate generated successfully",
  "domain": "example.com",
  "output": "Certbot execution output..."
}
```

C.2 Environment Configuration

C.2.1 Required Environment Variables

The following environment variables must be configured for proper system operation:

C.2.1.1 Frontend Configuration

```
# .env.local - Frontend environment variables
```

```
NEXT_PUBLIC_API_URL=http://localhost:80
NEXT_PUBLIC_ENV=development
```

C.2.1.2 Backend Configuration

Environment variables are passed at runtime through Docker or system configuration:

```
# Runtime environment variables
GODADDY_API_KEY=your_api_key_here
GODADDY_API_SECRET=your_api_secret_here
BASE_DOMAIN=yourdomain.com
DNS_TTL=600
PROPAGATION_DELAY=60
PYTHONPATH=/app
```

C.3 Security Considerations

C.3.1 Security Best Practices

C.3.1.1 Credential Management

- Never store DNS provider credentials in configuration files
- Use environment variables or secrets management systems
- Rotate API keys regularly (quarterly recommended)
- Monitor API key usage for unauthorized access
- Implement IP whitelisting where possible

C.3.1.2 Certificate Security

- Set proper file permissions (600) for private keys
- Use encrypted storage for certificate backups
- Implement certificate pinning for critical applications
- Monitor certificate expiration proactively
- Have emergency certificate renewal procedures

Publications and Presentations

1. SSL Automation Tool Development: Presented at StartMySafari Innovations Private Limited internal technical review, June 2025.

2. "Automated SSL Certificate Management using Let's Encrypt and DNS Validation" - Technical report submitted as part of MCA project requirements, Dr. Babasaheb Ambedkar Marathwada University, June 2025.

Acknowledgements

I would like to express my sincere gratitude to all those who contributed to the successful completion of this SSL Automation Tool project.

First and foremost, I extend my heartfelt thanks to **Dr. Kavari Lad**, my project guide, for his invaluable guidance, continuous support, and expert advice throughout the development of this project. His insights into modern web technologies and security practices were instrumental in shaping this work.

I am deeply grateful to **StartMySafari Innovations Private Limited (smsipl)** for providing the opportunity to work on this practical and industry-relevant project. The real-world exposure and access to modern development tools and technologies significantly enhanced my learning experience.

Special thanks to **Prof. Abhijeet Shelke**, Director, for his administrative support and for creating an environment conducive to learning and innovation.

I would like to acknowledge the faculty members of the Department of Management Science at **Dr. Babasaheb Ambedkar Marathwada University, Chhatrapati Sambhajinagar**, for their academic support and for providing the foundational knowledge that enabled me to undertake this project.

My appreciation extends to the open-source community, particularly the developers of React, Next.js, FastAPI, and Let's Encrypt, whose technologies formed the backbone of this SSL automation solution.

I am thankful to my colleagues and peers who provided feedback and suggestions during the development and testing phases of the project.

Finally, I express my gratitude to my family and friends for their constant encouragement and support throughout my academic journey.

This project has been an enriching experience that has significantly enhanced my understanding of web development, SSL certificate management, and modern software engineering practices.

Dipak Nivrutti Rathod

June 2025