**REST API Methods - Professional PowerPoint Presentation Content**

**SLIDE 1: Title Slide**

=====================================

REST API ARCHITECTURE

Web Services Communication Flow

=====================================

Lighttpd ↔ FastCGI ↔ amx-fcgi

[Company Logo]

Technical Documentation

Version 1.0 | 2025

**SLIDE 2: Agenda**

=====================================

AGENDA

=====================================

1. REST API Overview

2. Authentication - /session API

3. Service Management - /serviceElements API

4. Command Execution - /commands API

5. File Operations - /download API

6. HTTP Methods Comparison

7. FastCGI Communication Architecture

8. Security & Best Practices

**SLIDE 3: REST API Overview**

=====================================

REST API OVERVIEW

=====================================

What is REST?

• REpresentational State Transfer

• Architectural style for web services

• Stateless client-server communication

• Resource-based URLs

Our Implementation Stack:

┌─────────────────┐

│ Web Browser │ ← JavaScript/AJAX

├─────────────────┤

│ Lighttpd Server│ ← Web Server (Port 80/443)

├─────────────────┤

│ FastCGI │ ← Protocol Bridge

├─────────────────┤

│ amx-fcgi │ ← Application Logic

└─────────────────┘

Key Benefits:

✓ Scalable architecture

✓ Platform independent

✓ Cacheable responses

✓ Uniform interface

**SLIDE 4: Session API - Authentication**

=====================================

/session - AUTHENTICATION API

=====================================

Endpoint: /session

Purpose: User authentication and session management

┌──────────────────────────────────────┐

│ METHOD: POST │

├──────────────────────────────────────┤

│ Description: │

│ • Authenticates user credentials │

│ • Creates new session │

│ • Generates JWT token │

│ │

│ Request: │

│ POST /session HTTP/1.1 │

│ Content-Type: application/json │

│ { │

│ "username": "admin", │

│ "password": "secure\_password" │

│ } │

│ │

│ Response (200 OK): │

│ { │

│ "sessionId": "abc123-def456", │

│ "token": "eyJhbGciOiJ...", │

│ "expires": 3600 │

│ } │

│ │

│ Security Features: │

│ • Password hashing (bcrypt) │

│ • JWT token generation │

│ • Session timeout management │

│ • Rate limiting │

└──────────────────────────────────────┘

**SLIDE 5: Service Elements API - Overview**

=====================================

/serviceElements - SERVICE MANAGEMENT

=====================================

Endpoint: /serviceElements

Purpose: Complete CRUD operations for service configuration

Available Methods:

┌────────┬───────────────────────────────┐

│ Method │ Operation │

├────────┼───────────────────────────────┤

│ GET │ Retrieve service(s) │

│ POST │ Create new service │

│ PUT │ Update existing service │

│ DELETE │ Remove service │

└────────┴───────────────────────────────┘

Resource Patterns:

• Collection: /serviceElements

• Specific: /serviceElements/{id}

Key Features:

✓ RESTful resource management

✓ JSON request/response format

✓ Atomic operations

✓ Validation & error handling

**SLIDE 6: Service Elements - GET Method**

=====================================

/serviceElements - GET METHOD

=====================================

Purpose: Retrieve service information

Safe: Yes | Idempotent: Yes

┌──────────────────────────────────────┐

│ Use Cases: │

│ • List all services │

│ • Get specific service details │

│ • Filter by status/type │

│ • Pagination support │

├──────────────────────────────────────┤

│ Examples: │

│ │

│ 1. Get all services: │

│ GET /serviceElements │

│ │

│ 2. Get specific service: │

│ GET /serviceElements/123 │

│ │

│ 3. Filter active services: │

│ GET /serviceElements?status=active│

│ │

│ Response Format: │

│ [ │

│ { │

│ "id": 1, │

│ "name": "web-service", │

│ "status": "active", │

│ "port": 8080, │

│ "config": {...} │

│ } │

│ ] │

└──────────────────────────────────────┘

**SLIDE 7: Service Elements - POST Method**

=====================================

/serviceElements - POST METHOD

=====================================

Purpose: Create new service instance

Safe: No | Idempotent: No

┌──────────────────────────────────────┐

│ Workflow: │

│ 1. Validate input data │

│ 2. Check for duplicates │

│ 3. Generate unique ID │

│ 4. Create configuration │

│ 5. Start service process │

│ 6. Return created resource │

├──────────────────────────────────────┤

│ Request: │

│ POST /serviceElements │

│ Content-Type: application/json │

│ { │

│ "name": "api-gateway", │

│ "type": "reverse-proxy", │

│ "port": 8080, │

│ "config": { │

│ "upstream": "localhost:3000", │

│ "timeout": 30 │

│ } │

│ } │

│ │

│ Response (201 Created): │

│ Location: /serviceElements/124 │

│ { │

│ "id": 124, │

│ "status": "created", │

│ "message": "Service created" │

│ } │

└──────────────────────────────────────┘

**SLIDE 8: Service Elements - PUT Method**

=====================================

/serviceElements - PUT METHOD

=====================================

Purpose: Update or replace existing service

Safe: No | Idempotent: Yes

┌──────────────────────────────────────┐

│ Characteristics: │

│ • Full resource replacement │

│ • Creates if doesn't exist │

│ • Preserves resource ID │

│ • Atomic operation │

├──────────────────────────────────────┤

│ Request: │

│ PUT /serviceElements/123 │

│ Content-Type: application/json │

│ { │

│ "name": "updated-service", │

│ "type": "web", │

│ "port": 8081, │

│ "status": "active", │

│ "config": { │

│ "workers": 4, │

│ "timeout": 60 │

│ } │

│ } │

│ │

│ Response (200 OK): │

│ { │

│ "id": 123, │

│ "status": "updated", │

│ "changes": ["port", "config"] │

│ } │

└──────────────────────────────────────┘

**SLIDE 9: Service Elements - DELETE Method**

=====================================

/serviceElements - DELETE METHOD

=====================================

Purpose: Remove service and cleanup resources

Safe: No | Idempotent: Yes

┌──────────────────────────────────────┐

│ Deletion Process: │

│ 1. Verify service exists │

│ 2. Check dependencies │

│ 3. Stop running processes │

│ 4. Remove from database │

│ 5. Clean configuration files │

│ 6. Clear logs and temp files │

├──────────────────────────────────────┤

│ Request: │

│ DELETE /serviceElements/123 │

│ Authorization: Bearer {token} │

│ │

│ Response Options: │

│ │

│ Success (204 No Content): │

│ • No response body │

│ • Resource deleted │

│ │

│ Success (200 OK): │

│ { │

│ "status": "deleted", │

│ "id": 123, │

│ "message": "Service removed" │

│ } │

│ │

│ Error (409 Conflict): │

│ { │

│ "error": "Service has dependencies"│

│ } │

└──────────────────────────────────────┘

**SLIDE 10: Commands API**

=====================================

/commands - COMMAND EXECUTION

=====================================

Endpoint: /commands

Purpose: Execute system commands with authentication

┌──────────────────────────────────────┐

│ METHOD: POST │

├──────────────────────────────────────┤

│ Security Layers: │

│ • JWT authentication required │

│ • Command whitelist validation │

│ • Permission-based execution │

│ • Audit logging │

│ │

│ Request: │

│ POST /commands │

│ Authorization: Bearer {token} │

│ { │

│ "command": "restart", │

│ "target": "web-service", │

│ "params": { │

│ "force": true, │

│ "timeout": 30 │

│ } │

│ } │

│ │

│ Response (200 OK): │

│ { │

│ "status": "success", │

│ "output": "Service restarted", │

│ "executionTime": 1.247, │

│ "timestamp": "2025-06-24T12:58:37Z"│

│ } │

│ │

│ Supported Commands: │

│ • restart, stop, start │

│ • status, reload, enable, disable │

└──────────────────────────────────────┘

**SLIDE 11: Download API**

=====================================

/download - FILE OPERATIONS

=====================================

Endpoint: /download

Purpose: Secure file download with streaming

┌──────────────────────────────────────┐

│ METHOD: GET │

├──────────────────────────────────────┤

│ Features: │

│ • Chunked transfer encoding │

│ • Resume support (Range headers) │

│ • MIME type detection │

│ • Path traversal protection │

│ │

│ Request: │

│ GET /download?file=backup.tar.gz │

│ Authorization: Bearer {token} │

│ Accept: application/octet-stream │

│ │

│ Response Headers: │

│ HTTP/1.1 200 OK │

│ Content-Type: application/gzip │

│ Content-Disposition: attachment; │

│ filename="backup.tar.gz" │

│ Content-Length: 10485760 │

│ Transfer-Encoding: chunked │

│ │

│ Streaming Process: │

│ ┌─────────────────────────┐ │

│ │ Read 8KB → Send chunk │ │

│ │ Read 8KB → Send chunk │ │

│ │ ... (repeat until EOF) │ │

│ └─────────────────────────┘ │

│ │

│ Security Validations: │

│ • File exists in allowed directory │

│ • User has download permission │

│ • File size limits enforced │

└──────────────────────────────────────┘

**SLIDE 12: HTTP Methods Comparison Matrix**

=====================================

HTTP METHODS COMPARISON MATRIX

=====================================

┌─────────┬────────┬────────┬────────┬─────────┐

│Property │ GET │ POST │ PUT │ DELETE │

├─────────┼────────┼────────┼────────┼─────────┤

│Purpose │Retrieve│Create │Update │Remove │

│ │ │ │Replace │ │

├─────────┼────────┼────────┼────────┼─────────┤

│Body │ None │Required│Required│ None │

│Required │ │ │ │ │

├─────────┼────────┼────────┼────────┼─────────┤

│Safe │ Yes │ No │ No │ No │

│ │ │ │ │ │

├─────────┼────────┼────────┼────────┼─────────┤

│Idempotent│ Yes │ No │ Yes │ Yes │

│ │ │ │ │ │

├─────────┼────────┼────────┼────────┼─────────┤

│Cacheable│ Yes │ No\* │ No │ No │

│ │ │ │ │ │

├─────────┼────────┼────────┼────────┼─────────┤

│Success │ 200 │ 201 │ 200 │ 204 │

│Status │ OK │Created │ OK │No Content│

├─────────┼────────┼────────┼────────┼─────────┤

│Database │ SELECT │ INSERT │ UPDATE │ DELETE │

│Operation│ │ │ │ │

└─────────┴────────┴────────┴────────┴─────────┘

\* POST can be cacheable with proper headers

**SLIDE 13: FastCGI Communication Protocol**

=====================================

FASTCGI COMMUNICATION PROTOCOL

=====================================

Protocol Flow:

┌──────────────────────────────────────┐

│ Lighttpd Server │

│ 1. Receive HTTP Request │

│ 2. Parse headers and body │

│ 3. Create FastCGI packet │

└────────────┬─────────────────────────┘

│

▼ FastCGI Records

┌──────────────────────────────────────┐

│ FastCGI Protocol Structure │

├──────────────────────────────────────┤

│ BEGIN\_REQUEST │

│ ├─ Request ID │

│ ├─ Role (RESPONDER) │

│ └─ Flags │

│ │

│ PARAMS │

│ ├─ REQUEST\_METHOD │

│ ├─ REQUEST\_URI │

│ ├─ CONTENT\_TYPE │

│ └─ Authorization headers │

│ │

│ STDIN │

│ └─ Request body (JSON) │

│ │

│ STDOUT (Response) │

│ ├─ HTTP headers │

│ └─ Response body │

│ │

│ END\_REQUEST │

│ └─ Request complete │

└──────────────────────────────────────┘

Connection Types:

• UNIX Socket: /tmp/amx-fcgi.sock

• TCP Socket: 127.0.0.1:9000

**SLIDE 14: Security Best Practices**

=====================================

SECURITY BEST PRACTICES

=====================================

Authentication & Authorization:

┌──────────────────────────────────────┐

│ 1. JWT Token Implementation │

│ • Short expiration times (1 hour) │

│ • Refresh token rotation │

│ • Secure storage (httpOnly) │

│ │

│ 2. Input Validation │

│ • Sanitize all inputs │

│ • Parameterized queries │

│ • Request size limits │

│ │

│ 3. Rate Limiting │

│ • Per-endpoint limits │

│ • User-based throttling │

│ • DDoS protection │

│ │

│ 4. HTTPS Enforcement │

│ • TLS 1.2+ only │

│ • Strong cipher suites │

│ • HSTS headers │

│ │

│ 5. Error Handling │

│ • Generic error messages │

│ • No stack traces in production │

│ • Comprehensive logging │

│ │

│ 6. CORS Configuration │

│ • Whitelist allowed origins │

│ • Restrict methods │

│ • Validate preflight requests │

└──────────────────────────────────────┘

**SLIDE 15: Response Status Codes**

=====================================

HTTP RESPONSE STATUS CODES

=====================================

Success Codes (2xx):

┌────────┬──────────────────────────────┐

│ 200 │ OK - Request succeeded │

│ 201 │ Created - New resource │

│ 204 │ No Content - Deleted │

└────────┴──────────────────────────────┘

Client Error Codes (4xx):

┌────────┬──────────────────────────────┐

│ 400 │ Bad Request - Invalid syntax │

│ 401 │ Unauthorized - Auth required │

│ 403 │ Forbidden - No permission │

│ 404 │ Not Found - Resource missing │

│ 405 │ Method Not Allowed │

│ 409 │ Conflict - Resource conflict │

│ 422 │ Unprocessable Entity │

│ 429 │ Too Many Requests │

└────────┴──────────────────────────────┘

Server Error Codes (5xx):

┌────────┬──────────────────────────────┐

│ 500 │ Internal Server Error │

│ 502 │ Bad Gateway - FastCGI error │

│ 503 │ Service Unavailable │

│ 504 │ Gateway Timeout │

└────────┴──────────────────────────────┘

**SLIDE 16: Performance Optimization**

=====================================

PERFORMANCE OPTIMIZATION

=====================================

Optimization Strategies:

1. Connection Pooling

• Reuse FastCGI connections

• Maintain persistent sockets

• Connection timeout: 30s

2. Caching Strategy

┌─────────────────────────┐

│ GET requests → Cache │

│ Cache-Control headers │

│ ETags for validation │

│ Redis/Memcached layer │

└─────────────────────────┘

3. Response Compression

• Enable gzip/deflate

• Compress JSON responses

• Minimum size threshold: 1KB

4. Database Optimization

• Connection pooling

• Prepared statements

• Index optimization

• Query result caching

5. Load Balancing

• Multiple amx-fcgi workers

• Round-robin distribution

• Health check endpoints

Performance Metrics:

• Response time: < 100ms

• Throughput: 1000 req/sec

• Concurrent connections: 500

**SLIDE 17: Error Handling Framework**

=====================================

ERROR HANDLING FRAMEWORK

=====================================

Error Response Structure:

┌──────────────────────────────────────┐

│ { │

│ "error": { │

│ "code": "ERR\_INVALID\_INPUT", │

│ "message": "Validation failed", │

│ "details": { │

│ "field": "port", │

│ "reason": "Out of range" │

│ }, │

│ "timestamp": "2025-06-24T12:00Z"│

│ "requestId": "req\_abc123" │

│ } │

│ } │

└──────────────────────────────────────┘

Error Categories:

┌──────────────────────────────────────┐

│ Validation Errors │

│ • Required field missing │

│ • Invalid data format │

│ • Out of range values │

├──────────────────────────────────────┤

│ Authentication Errors │

│ • Invalid credentials │

│ • Token expired │

│ • Insufficient permissions │

├──────────────────────────────────────┤

│ Resource Errors │

│ • Resource not found │

│ • Resource already exists │

│ • Resource locked │

├──────────────────────────────────────┤

│ System Errors │

│ • Database connection failed │

│ • File system error │

│ • External service timeout │

└──────────────────────────────────────┘

**SLIDE 18: Testing Strategy**

=====================================

API TESTING STRATEGY

=====================================

Testing Levels:

1. Unit Testing

• Individual function tests

• Mock FastCGI environment

• Code coverage > 80%

2. Integration Testing

┌─────────────────────────┐

│ Test Complete Flow: │

│ HTTP → FastCGI → DB │

│ Validate responses │

│ Check side effects │

└─────────────────────────┘

3. API Testing Tools

• Postman collections

• curl scripts

• Automated test suites

Example Test Case:

┌──────────────────────────────────────┐

│ Test: Create Service │

│ Method: POST /serviceElements │

│ │

│ Steps: │

│ 1. Authenticate user │

│ 2. Send POST with valid data │

│ 3. Verify 201 status │

│ 4. Check Location header │

│ 5. GET created resource │

│ 6. Validate response data │

│ 7. Cleanup test data │

│ │

│ Expected: Service created successfully│

│ Actual: ✓ PASS │

└──────────────────────────────────────┘

**SLIDE 19: Monitoring & Logging**

=====================================

MONITORING & LOGGING

=====================================

Logging Configuration:

┌──────────────────────────────────────┐

│ Log Levels: │

│ • ERROR - System errors │

│ • WARN - Potential issues │

│ • INFO - General information │

│ • DEBUG - Detailed debugging │

│ │

│ Log Format: │

│ [2025-06-24 12:58:37] [INFO] │

│ [amx-fcgi] Request: POST /session │

│ User: admin, IP: 192.168.1.100 │

│ Response: 200 OK, Time: 45ms │

└──────────────────────────────────────┘

Monitoring Metrics:

┌──────────────────────────────────────┐

│ Performance Metrics │

│ • Request rate (req/sec) │

│ • Response time (p50, p95, p99) │

│ • Error rate (4xx, 5xx) │

│ • Active connections │

│ │

│ Business Metrics │

│ • API usage by endpoint │

│ • User activity patterns │

│ • Most frequent operations │

│ • Resource utilization │

│ │

│ Alerting Thresholds │

│ • Response time > 500ms │

│ • Error rate > 1% │

│ • CPU usage > 80% │

│ • Memory usage > 90% │

└──────────────────────────────────────┘

**SLIDE 20: Summary & Key Takeaways**

=====================================

SUMMARY & KEY TAKEAWAYS

=====================================

REST API Implementation:

✓ 4 main endpoints implemented

✓ Full CRUD operations support

✓ JWT-based authentication

✓ FastCGI protocol integration

Technical Stack:

• Lighttpd → Web Server

• FastCGI → Protocol Bridge

• amx-fcgi → Application Logic

• JSON → Data Format

Best Practices Applied:

1. RESTful design principles

2. Secure authentication (JWT)

3. Input validation & sanitization

4. Comprehensive error handling

5. Performance optimization

6. Detailed logging & monitoring

Next Steps:

→ API versioning strategy

→ GraphQL implementation

→ WebSocket support

→ Microservices migration

→ Container deployment

Questions?

Contact: api-team@company.com

Documentation: docs.company.com/api

**SLIDE 21: Appendix - Quick Reference**

=====================================

APPENDIX - QUICK REFERENCE

=====================================

API Endpoint Summary:

┌────────────────────┬─────────┬──────────┐

│ Endpoint │ Methods │ Auth Req │

├────────────────────┼─────────┼──────────┤

│ /session │ POST │ No │

│ /serviceElements │ GET │ Yes │

│ │ POST │ Yes │

│ │ PUT │ Yes │

│ │ DELETE │ Yes │

│ /commands │ POST │ Yes │

│ /download │ GET │ Yes │

└────────────────────┴─────────┴──────────┘

Common Headers:

• Content-Type: application/json

• Authorization: Bearer {token}

• Accept: application/json

Tools & Resources:

• Postman Collection: /docs/api.postman

• OpenAPI Spec: /docs/openapi.yaml

• Test Environment: test-api.company.com

• Production: api.company.com

**Presentation Notes for Speaker:**

1. **Slide 4 (Session)**: Emphasize security - JWT tokens expire in 1 hour for security
2. **Slide 5-9 (Service Elements)**: Demonstrate the RESTful nature with live examples if possible
3. **Slide 10 (Commands)**: Highlight the security layers and audit trail
4. **Slide 11 (Download)**: Mention chunked transfer for large files
5. **Slide 12 (Methods Matrix)**: Key point - GET is safe and idempotent, POST creates new resources
6. **Slide 13 (FastCGI)**: Explain the bridge between web server and application
7. **Slide 14 (Security)**: Never skip security - it's crucial for production
8. **Slide 16 (Performance)**: Show real metrics if available
9. **Slide 19 (Monitoring)**: Demonstrate dashboard if available

This presentation provides comprehensive technical documentation suitable for:

* Development teams
* System architects
* Security auditors
* Technical stakeholders
* API consumers

Each slide is designed to be self-contained with clear, professional information that can be understood by technical audiences.