Reliance Meeting Scheduler - Comprehensive Technical Documentation

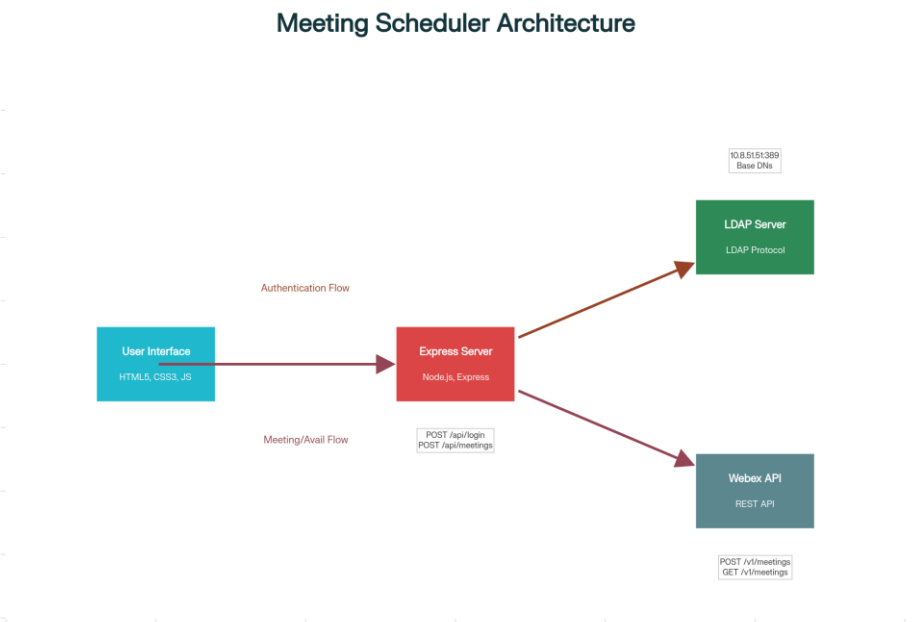
Executive Summary

The Reliance Meeting Scheduler is an enterprise-grade web application that integrates **LDAP authentication** with **Webex meeting scheduling** capabilities. After two months of development, the project delivers a production-ready single-page application built with vanilla JavaScript frontend and Node.js/Express backend, featuring real-time availability checking, JWT-based session management, and comprehensive user validation systems.

The application serves as a unified interface for Reliance employees to authenticate against corporate LDAP directory and seamlessly schedule Webex meetings through automated API integration, eliminating the need for direct Webex credentials while maintaining enterprise security standards.

System Architecture Overview

The application follows a **four-tier architecture** consisting of presentation layer (vanilla JavaScript SPA), application layer (Express.js API), authentication layer (LDAP integration), and external service layer (Webex API). The system implements stateless JWT authentication with 1-hour token expiry and comprehensive error handling across all integration points.

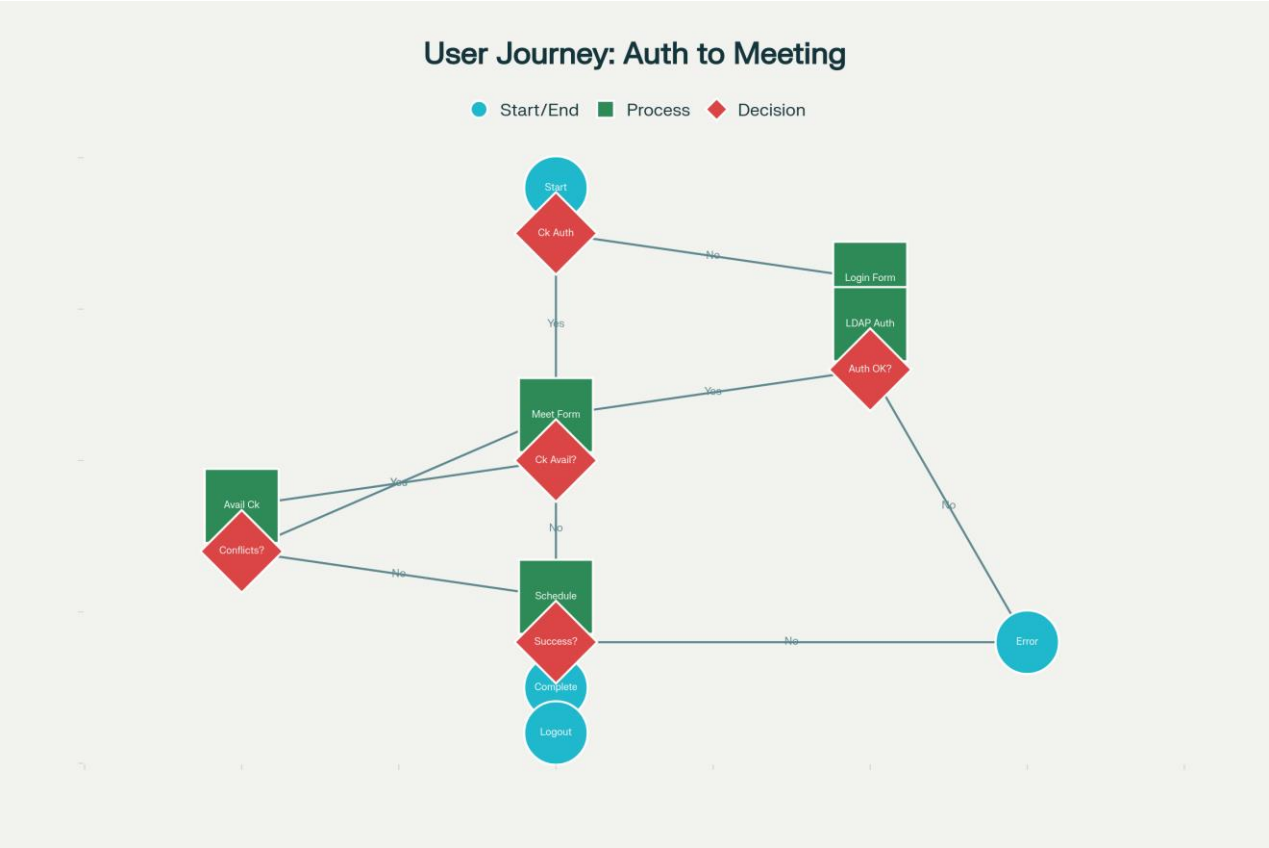


System Architecture and Data Flow Diagram for Reliance Meeting Scheduler

Complete User Lifecycle Analysis

User Journey Flow

The user experience encompasses **eight distinct phases**: initial access, authentication verification, login process, meeting form interaction, optional availability checking, meeting scheduling, result handling, and session termination. Each phase includes comprehensive validation, error handling, and state management to ensure smooth user experience.



Complete User Journey Flowchart for Reliance Meeting Scheduler

Detailed User States and Transitions

Phase 1: Application Bootstrap

- User accesses application URL
- `checkAuthenticationStatus()` examines localStorage for valid JWT
- System determines initial UI state (login vs scheduler)
- Background processes initialize event listeners and DOM references

Phase 2: Authentication Flow

- Login form displays with corporate branding and validation
- Real-time field validation provides immediate feedback
- LDAP authentication occurs server-side with multiple Base DN searches
- JWT token generation and storage upon successful authentication
- User profile pre-population from LDAP attributes

Phase 3: Meeting Management

- Authenticated users access scheduler interface with pre-filled host information
- Form validation occurs in real-time with 15-minute minimum duration enforcement
- Optional availability checking analyzes conflicts and suggests alternatives
- Meeting submission triggers Webex API integration through backend proxy

Phase 4: Session Management

- JWT tokens automatically expire after 1 hour
- Logout functionality clears all local storage and returns to login state
- Session persistence allows users to resume work without re-authentication

Authentication System Implementation

LDAP Integration Architecture

The authentication system implements **dual Base DN searching** across `o=relianceada` and `o=reliancegroup` organizational units. The backend performs sequential searches using the filter `(cn=username)` and extracts user attributes including distinguished name, email, common name, and display name for profile population.

LDAP Connection Specifications:

- **Server:** 10.8.51.51:389 (LDAP v3 protocol)
- **Search Scope:** Subtree search across organizational units
- **Bind Method:** Simple bind authentication with discovered DN
- **Timeout Configuration:** 10 seconds for both connection and operation timeouts
- **Attribute Extraction:** dn, mail, cn, displayName for user profile creation

JWT Token Management

Token Structure and Security:

- **Algorithm:** HS256
- **Payload:** userId, email, name from LDAP attributes
- **Expiration:** 3600 seconds (1 hour) from issuance
- **Storage:** Frontend localStorage with automatic cleanup on expiry
- **Verification:** Middleware validates all protected endpoints before processing

API Specifications and Integration

Backend API Endpoints

1. Authentication Endpoint

```
POST /api/login
Authentication: None (public endpoint)
Content-Type: application/json

Request Body:
{
  "username": "string (LDAP common name)",
  "password": "string (plaintext, transmitted over LDAPS)"
}

Response (Success - 200):
{
  "message": "Login successful!",
  "user": {
    "name": "string (displayName or cn)",
    "email": "string (LDAP mail attribute)"
  },
  "token": "string (JWT with 1-hour expiry)"
}

Response (Error - 401/404):
{
  "message": "string (error description)"
}
```

2. Meeting Scheduling Endpoint

```
POST /api/meetings/schedule
Authentication: Bearer JWT token (required)
Content-Type: application/json
Request Body:
{
  "title": "string (meeting subject)",
  "startDateTime": "string (ISO 8601 format)",
  "endDateTime": "string (ISO 8601 format)",
  "inviteeEmails": "array of strings (email addresses)",
  "password": "string (optional meeting password)",
  "agenda": "string (optional meeting description)"
}
Response (Success - 200):
{
  "message": "Webex meeting scheduled successfully!",
  "meeting": {
    "webLink": "string (join URL)",
    "meetingNumber": "string (numeric ID)",
    "id": "string (Webex meeting UUID)",
    "hostEmail": "string (service account email)"
  }
}
```

3. Availability Check Endpoint

```
GET /api/meetings/availability
Authentication: Bearer JWT token (required)
Query Parameters: startDate, endDate (YYYY-MM-DD format)
Response (Success - 200):
{
  "meetings": [
    {
      "id": "string (Webex meeting ID)",
      "title": "string (meeting subject)",
      "start": "string (ISO 8601 datetime)",
      "end": "string (ISO 8601 datetime)"
    }
  ]
}
```

Webex API Integration

External API Configuration:

- **Base URL:** https://webexapis.com/v1/
- **Authentication:** Personal Access Token (Bearer authentication)
- **Timeout:** 30 seconds for all requests
- **Endpoints Used:** POST /meetings (creation), GET /meetings (availability)
- **Rate Limiting:** Standard Webex API limits apply
- **Error Handling:** Comprehensive HTTP status code processing

Frontend Implementation Architecture

The frontend implements a **single-class architecture** with the `RelianceMeetingScheduler` class managing all user interactions, API communications, and state management. The implementation uses vanilla JavaScript ES2021 features with no external frameworks, resulting in a lightweight 51KB script bundle.

Core Frontend Methodologies

State Management Implementation:

- **Authentication State:** `isAuthenticated` boolean with localStorage persistence
- **Meeting Data:** `currentMeetingData` object for form state and success display
- **Availability State:** `existingMeetings` array and `selectedTimeSlot` object
- **UI State:** Loading indicators, error states, and form validation status

Validation System Architecture:

- **Real-time Validation:** Input event listeners provide immediate feedback
- **Time Range Validation:** Enforces 15-minute minimum duration and logical time ordering
- **Email Validation:** Regular expression parsing for participant email lists
- **Form State Management:** Invalid field tracking with visual indicators

Availability Check Algorithm:

```
// Time conflict detection logic
const isSlotAvailable = (desiredSlot, existingMeetings) => {
    const newStart = timeToMinutes(desiredSlot.start);
    const newEnd = timeToMinutes(desiredSlot.end);

    return !existingMeetings.some(meeting => {
        const meetingStart = timeToMinutes(meeting.start);
        const meetingEnd = timeToMinutes(meeting.end);
        // Overlap detection: (startA < endB) && (endA > startB)
        return newStart < meetingEnd && newEnd > meetingStart;
    });
};
```

Backend Implementation Details

Express.js Server Architecture

The backend implements **middleware-based request processing** with CORS enablement, JSON parsing, static file serving, and JWT verification for protected endpoints. The server runs on port 80 with comprehensive error handling and logging throughout all operations.

Request Processing Pipeline:

1. **CORS Middleware:** Enables cross-origin requests for frontend-backend communication
2. **JSON Parser:** Processes application/json request bodies
3. **Static File Server:** Serves HTML, CSS, JavaScript, and asset files
4. **JWT Verification:** Validates Bearer tokens on protected endpoints
5. **Route Handlers:** Process business logic and external API integration
6. **Error Handling:** Consistent error response formatting with appropriate HTTP status codes

LDAP Integration Implementation

Multi-Base DN Search Strategy:

```
const searchSequence = async (username) => {
    const baseDNs = ['o=relianceada', 'o=reliancegroup'];

    for (const baseDn of baseDNs) {
        const searchResult = await performLdapSearch(client, baseDn, username);
        if (searchResult) return searchResult; // First match wins
    }

    throw new Error('User not found in any organizational unit');
};
```

Connection Management:

- **Connection Pooling:** Single client per request with proper cleanup
- **Event Handling:** Comprehensive event listeners for connection states
- **Error Recovery:** Timeout handling and connection retry logic
- **Attribute Processing:** Extraction and validation of user profile data

Technical Specifications and Configuration

Security Implementation

JWT Security Measures:

- **Secret Management:** Currently hardcoded for development (requires environment variable for production)
- **Token Expiration:** 1-hour lifespan reduces replay attack windows
- **Payload Minimization:** Only essential user data included in token
- **HTTP-Only Recommendations:** Should implement HTTP-only cookies for enhanced security

LDAP Security Configuration:

- **Password Transmission:** Plaintext over internal network (LDAPS recommended for production)
- **Credential Storage:** No local password storage, authentication via LDAP bind
- **DN Discovery:** Dynamic user DN resolution prevents hardcoded user paths
- **Attribute Filtering:** Limited attribute extraction reduces data exposure

Performance Optimization Strategies

Frontend Performance:

- **Vanilla JavaScript:** No framework overhead, direct DOM manipulation
- **Event Delegation:** Efficient event handling for dynamic content
- **Lazy Loading:** Components initialized only when needed
- **CSS Variables:** Efficient theming and style calculations

Backend Performance:

- **Stateless Architecture:** No server-side session storage enables horizontal scaling
- **Connection Pooling:** Efficient LDAP connection management
- **Async Operations:** Non-blocking I/O for all external API calls
- **Error Caching:** Prevents repeated failed authentication attempts

Error Handling and Validation Framework

Comprehensive Error Categories

LDAP Authentication Errors:

- Connection timeouts and server unreachability
- Invalid credentials and user not found scenarios
- Ambiguous username resolution across multiple Base DNs
- LDAP server errors and protocol-level issues

Webex API Integration Errors:

- Rate limiting and quota exceeded responses
- Invalid meeting parameters and scheduling conflicts
- Network timeouts and service unavailability
- Authentication token expiry and renewal requirements

Frontend Validation Errors:

- Real-time field validation with immediate user feedback
- Time range validation ensuring logical meeting duration
- Email format validation for participant lists
- Form completeness checking before submission

Error Response Standardization

All API endpoints return consistent error response formats with appropriate HTTP status codes, descriptive error messages, and actionable user guidance. The frontend implements centralized error handling with user-friendly message translation and appropriate UI state management.

Deployment Requirements and Configuration

Backend Dependencies:

- Node.js runtime environment (v14+ recommended)
- NPM package manager with production dependency installation
- Network access to LDAP server (10.8.51.51:389)
- Outbound HTTPS access for Webex API integration

Frontend Requirements:

- Modern web browser with ES2021 support
- JavaScript enabled for application functionality
- Local storage support for JWT token persistence
- Network access to backend API server

This comprehensive technical documentation provides complete coverage of the Reliance Meeting Scheduler's architecture, implementation details, API specifications, and operational requirements, serving as a definitive reference for development, deployment, and maintenance activities.