Meeting Attendance System - MERN Stack Architecture Blueprint

Project Overview

A centralized attendance tracking system that integrates with multiple meeting platforms (Zoom, Google Meet, Microsoft Teams) to automatically capture and manage attendance records.

How The System Actually Works

Two Main Approaches

Approach 1: Automatic Attendance via Platform APIs (Recommended)

User Journey:

1. **Initial Setup (One-time)**
   - Instructor logs into your system
   - Connects their Zoom/Google Meet/Teams account via OAuth
   - System gets permission to access their meeting data
2. **Before Meeting**
   - Instructor creates/schedules meeting on Zoom/Meet/Teams as usual
   - Your system automatically syncs upcoming meetings (polls API every 15 min)
   - OR instructor manually imports meeting into your system using meeting ID/link
3. **During Meeting**
   - Students join meeting normally through Zoom/Meet/Teams
   - Platform tracks who joins and when (you don't see this yet)
   - Instructor teaches as normal - no extra steps needed
4. **After Meeting**
   - Your system calls platform API: "Give me participant list for meeting XYZ"
   - Platform returns: participant names, join times, leave times
   - System automatically marks attendance based on rules:
     - Present: Joined within 10 min of start, stayed 80%+ of duration
     - Late: Joined after 10 min threshold
     - Absent: Never joined or < 20% duration

- Instructor reviews auto-generated attendance in your dashboard
- Can manually override if needed (e.g., technical issues)

Technical Flow:

```
Meeting Ends → System polls Zoom API → Fetches participant report →
Matches participants to your users (by email) → Auto-marks attendance →
Instructor reviews & approves
```

Limitations:

- Only works for platforms with public APIs
- Requires OAuth permissions from instructors
- Participant emails must match your system's user records
- Some platforms have API rate limits

Approach 2: Manual Attendance with Smart Features

If API integration is too complex or platform doesn't support it:

User Journey:

1. **Before/During Meeting**
   - Instructor creates meeting record in your system
   - Generates unique attendance code/link
   - Shares code with students during meeting
2. **Students Mark Themselves**
   - Student opens your web app on phone/laptop
   - Enters attendance code OR clicks shared link
   - System captures: timestamp, location (optional), device info
   - Student marked present
3. **Smart Verification (to prevent fraud)**
   - Time-based codes that expire
   - Geolocation verification (must be within radius)
   - IP address tracking
   - Limit one check-in per device

- Random code changes every 5 minutes

4. **Instructor Dashboard**
    - Real-time view of who has checked in
    - Can manually add/remove attendees
    - Export attendance records

Hybrid Approach (Best of Both Worlds)

Combine both methods:

- Use API integration when available (Zoom works great)
- Fall back to manual check-in for unsupported platforms
- Instructor chooses method per meeting

Detailed User Workflows

Workflow 1: Automatic Attendance (Zoom Example)

Instructor Side:

```
Day 1: Setup
├─ Login to your attendance system
├─ Click "Connect Zoom Account"
├─ OAuth flow → Grant permissions
└─ System now has access to meeting data

Day 2: Schedule Meeting
├─ Create meeting in Zoom (as normal)
├─ Your system auto-discovers it OR
└─ Instructor manually syncs by clicking "Import Meetings"

Day 3: Meeting Day
├─ Instructor starts Zoom meeting normally
├─ Students join via regular Zoom link
├─ Teaching happens (no interaction with your system)
└─ Meeting ends

Day 3: After Meeting (30 min later)
├─ System automatically calls Zoom API
├─ Fetches participant report
├─ Auto-marks attendance
├─ Instructor gets notification
├─ Opens your dashboard to review
└─ Approves or makes manual adjustments
```

Student Side:

```
├─ Receives Zoom link from instructor (email/LMS)
├─ Joins meeting via Zoom app
├─ Attends class
└─ Done! (Student does nothing extra)

Later:
└─ Can view their own attendance record in your system
```

Key Point: Students don't interact with your system during the meeting. They just join Zoom normally.

Workflow 2: Manual Check-in System

Instructor Side:

Before Meeting:
├─ Creates meeting in your system
├─ System generates unique 6-digit code (e.g., 7K3M9P)
└─ Display code on screen during meeting

During Meeting:
├─ Share code verbally or on screen share
├─ Code changes every 5 minutes (prevents sharing)
├─ View real-time dashboard of who checked in
└─ Can manually mark students who have tech issues

After Meeting:
├─ Close attendance window
└─ System finalizes records

Student Side:

During Meeting:
├─ Open your system on phone/laptop
├─ Navigate to "Mark Attendance"
├─ Enter code shown by instructor
├─ Click "Submit"
├─ See confirmation message
└─ Continue attending meeting

Workflow 3: QR Code Check-in (For Physical/Hybrid Classes)

Instructor Side:

├─ Creates meeting in system
├─ System generates unique QR code
├─ Display QR on classroom screen OR
└─ Print and place at entrance

During Class:
└─ Monitor who has scanned in

Student Side:

├─ Opens your app
├─ Tap "Scan QR Code"
├─ Camera opens, scan code
└─ Marked present

Workflow 4: Link-based Check-in

Instructor Side:

```
├─ Creates meeting
├─ System generates unique link
├─ Share in Zoom chat/WhatsApp/Email
└─ Students click link during meeting time
```

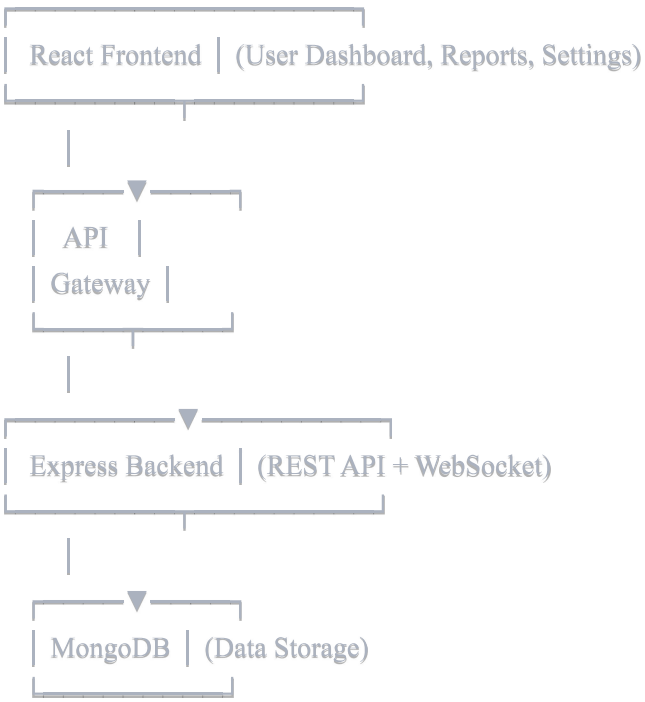
Student Side:

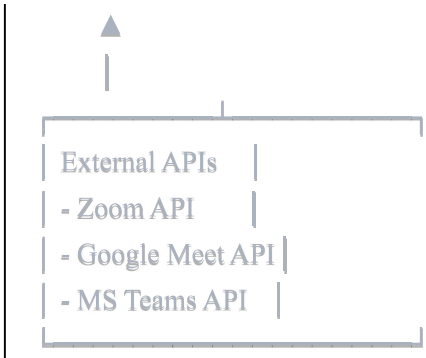
```
├─ Click link from instructor
├─ Link opens your web app
├─ Automatically marks attendance (if within time window)
└─ Shows confirmation
```

Security:

- Link only works during meeting time ±15 min buffer
- Link expires after meeting ends
- Can't be reused by same student
- Tracks IP/device to prevent multiple check-ins

High-Level Architecture





Technology Stack Details

Frontend (React)

- **React 18+** with Hooks
- **React Router** for navigation
- **Redux Toolkit** or **Zustand** for state management
- **Axios** for API calls
- **TailwindCSS** or **Material-UI** for styling
- **Chart.js** or **Recharts** for attendance analytics
- **Socket.io-client** for real-time updates

Backend (Node.js + Express)

- **Express.js** framework
- **Socket.io** for real-time communication
- **JWT** for authentication
- **Passport.js** for OAuth integrations
- **Node-cron** for scheduled tasks
- **Bull** or **Agenda** for job queuing
- **Winston** for logging

Database (MongoDB)

- **Mongoose** ODM
- **MongoDB Atlas** for cloud hosting (recommended)
- Indexes on frequently queried fields

Additional Tools

- **Redis** for caching and session management
- **Docker** for containerization
- **Nginx** as reverse proxy
- **PM2** for process management

Database Schema Design

1. Users Collection

```javascript
{
  _id: ObjectId,
  email: String (unique, indexed),
  password: String (hashed),
  role: String (enum: ['admin', 'instructor', 'student']),
  firstName: String,
  lastName: String,
  profilePicture: String,
  organizationId: ObjectId (ref: Organization),
  externalIds: {
    zoom: String,
    googleMeet: String,
    msTeams: String
  },
  createdAt: Date,
  updatedAt: Date
}
```

2. Organizations Collection

```javascript
```

```
{
  _id: ObjectId,
  name: String,
  domain: String,
  subscription: {
    plan: String,
    expiresAt: Date
  },
  integrations: {
    zoom: {
      enabled: Boolean,
      clientId: String,
      clientSecret: String (encrypted)
    },
    googleMeet: { ... },
    msTeams: { ... }
  },
  createdAt: Date
}
```

3. Meetings Collection

javascript

```
{
  _id: ObjectId,
  title: String,
  description: String,
  organizationId: ObjectId (ref: Organization),
  instructorId: ObjectId (ref: User),
  platform: String (enum: ['zoom', 'google-meet', 'ms-teams']),
  externalMeetingId: String (indexed),
  scheduledStartTime: Date,
  scheduledEndTime: Date,
  actualStartTime: Date,
  actualEndTime: Date,
  meetingLink: String,
  status: String (enum: ['scheduled', 'ongoing', 'completed', 'cancelled']),
  attendanceConfig: {
    autoMark: Boolean,
    lateThresholdMinutes: Number,
    minimumDurationMinutes: Number
  },
  createdAt: Date,
  updatedAt: Date
}
```

4. Attendance Records Collection

```javascript
{
  _id: ObjectId,
  meetingId: ObjectId (ref: Meeting),
  userId: ObjectId (ref: User),
  status: String (enum: ['present', 'absent', 'late', 'excused']),
  joinTime: Date,
  leaveTime: Date,
  totalDurationMinutes: Number,
  isManualEntry: Boolean,
  markedBy: ObjectId (ref: User),
  notes: String,
  createdAt: Date,
  updatedAt: Date
}

// Compound index on (meetingId, userId)
```

5. Integration Tokens Collection

```javascript
{
  _id: ObjectId,
  userId: ObjectId (ref: User),
  platform: String (enum: ['zoom', 'google-meet', 'ms-teams']),
  accessToken: String (encrypted),
  refreshToken: String (encrypted),
  expiresAt: Date,
  createdAt: Date
}
```

6. Audit Logs Collection

```javascript
```

```
{
  _id: ObjectId,
  action: String,
  performedBy: ObjectId (ref: User),
  targetResource: String,
  targetId: ObjectId,
  metadata: Object,
  timestamp: Date
}
```

API Architecture

REST API Endpoints

Authentication

```
POST   /api/auth/register
POST   /api/auth/login
POST   /api/auth/logout
POST   /api/auth/refresh-token
GET    /api/auth/me
POST   /api/auth/forgot-password
POST   /api/auth/reset-password
```

OAuth Integration

```
GET    /api/oauth/:platform/authorize
GET    /api/oauth/:platform/callback
DELETE /api/oauth/:platform/disconnect
```

Users

```
GET    /api/users
GET    /api/users/:id
PUT    /api/users/:id
DELETE /api/users/:id
GET    /api/users/:id/attendance-summary
```

Meetings

```
GET    /api/meetings
POST   /api/meetings
GET    /api/meetings/:id
PUT    /api/meetings/:id
DELETE /api/meetings/:id
POST   /api/meetings/:id/sync
GET    /api/meetings/:id/attendance
POST   /api/meetings/:id/start
POST   /api/meetings/:id/end
```

Attendance

```
GET    /api/attendance
POST   /api/attendance
PUT    /api/attendance/:id
DELETE /api/attendance/:id
GET    /api/attendance/meeting/:meetingId
GET    /api/attendance/user/:userId
POST   /api/attendance/bulk-mark
GET    /api/attendance/export
```

Reports & Analytics

```
GET    /api/reports/attendance-summary
GET    /api/reports/user/:userId
GET    /api/reports/meeting/:meetingId
GET    /api/reports/date-range
POST   /api/reports/export
```

WebSocket Events

javascript

```
// Server to Client
'meeting:started'
'meeting:ended'
'attendance:updated'
'participant:joined'
'participant:left'

// Client to Server
'subscribe:meeting'
'unsubscribe:meeting'
```

Integration Strategy

Zoom Integration

1. **Setup**: Register app in Zoom Marketplace
2. **OAuth Flow**: Implement OAuth 2.0
3. **Webhooks**: Subscribe to meeting events
4. **API Usage**:
   - Get meeting participants
   - Get participant join/leave times
   - List scheduled meetings

```javascript
// Key Zoom API Endpoints
GET /meetings/{meetingId}
GET /metrics/meetings/{meetingId}/participants
GET /past_meetings/{meetingId}/participants
```

How Zoom Integration Actually Works:

```javascript
```

```
// After meeting ends, your system calls:
const response = await axios.get(
  `https://api.zoom.us/v2/past_meetings/${meetingId}/participants`,
  { headers: { Authorization: `Bearer ${accessToken}` } }
);

// Zoom returns something like:
{
  "participants": [
    {
      "name": "John Doe",
      "user_email": "john@university.edu",
      "join_time": "2025-01-15T10:02:00Z",
      "leave_time": "2025-01-15T11:58:00Z",
      "duration": 116  // minutes
    },
    {
      "name": "Jane Smith",
      "user_email": "jane@university.edu",
      "join_time": "2025-01-15T10:15:00Z",  // 13 min late
      "leave_time": "2025-01-15T11:55:00Z",
      "duration": 100
    }
  ]
}

// Your system then:
// 1. Matches emails to users in your database
// 2. Calculates if late/present/absent based on rules
// 3. Creates attendance records
// 4. Notifies instructor to review
```

Important Zoom Limitations:

- Participant report only available after meeting ends
- Free Zoom accounts: data available for 1 month
- Paid Zoom accounts: data available for 6 months
- Rate limit: 80 requests per second
- Must have "View usage reports" permission

Google Meet Integration

1. **Setup**: Create project in Google Cloud Console
2. **OAuth Flow**: Google OAuth 2.0

3. **API Usage**: Google Calendar API + Admin SDK

4. **Note**: Limited direct participant tracking, may need Chrome extension

```javascript
// Key Google APIs
Calendar API: Get meeting details
Admin SDK: Get participation reports (Workspace only)
```

How Google Meet Integration Actually Works:

Problem: Google Meet doesn't have a public API for participant tracking like Zoom does.

Solutions:

Option A: Google Workspace Admin SDK (Enterprise Only)

- Only works if your school/org uses Google Workspace
- Requires admin privileges
- Can access meeting attendance reports
- Limited to organizations with Enterprise licenses

```javascript
// Only works with Workspace Enterprise
GET https://admin.googleapis.com/admin/reports/v1/activity/users/all/applications/meet

// Returns basic meeting data, but participant details limited
```

Option B: Google Calendar API (Partial Solution)

- Can see who was invited to meeting
- Cannot see who actually attended
- Cannot see join/leave times
- Basically only scheduling info

Option C: Build a Chrome Extension (Most Practical)

```
Your Chrome Extension:
├─ Students/Instructors install extension
├─ When in Google Meet, extension detects it
├─ Extension reads participant list from DOM
├─ Sends data to your backend via API
└─ Your system records attendance
```

Option D: Manual Check-in (Fallback)

- Use QR code/link method described earlier
- Student marks themselves during Google Meet
- Most reliable for Google Meet

Recommendation: For Google Meet, use manual check-in system or require Chrome extension installation.

Microsoft Teams Integration

1. **Setup**: Register app in Azure AD
2. **OAuth Flow**: Microsoft Identity Platform
3. **API Usage**: Microsoft Graph API
4. **Webhooks**: Change notifications for events

```javascript
// Key Graph API Endpoints
GET /me/onlineMeetings
GET /communications/callRecords
```

Practical Implementation Recommendation

Start with This Approach:

Phase 1: Build for Zoom Only (API-based)

- Zoom has the best API support
- 80%+ of online classes use Zoom
- Fully automated attendance
- Most reliable data

Phase 2: Add Manual Check-in System

- Works for Google Meet, Teams, or any platform

- QR codes + time-based codes
- Fallback for when API fails
- Works for in-person classes too

Phase 3: Add Other Platforms (if needed)

- Microsoft Teams has decent API
- Google Meet requires workarounds

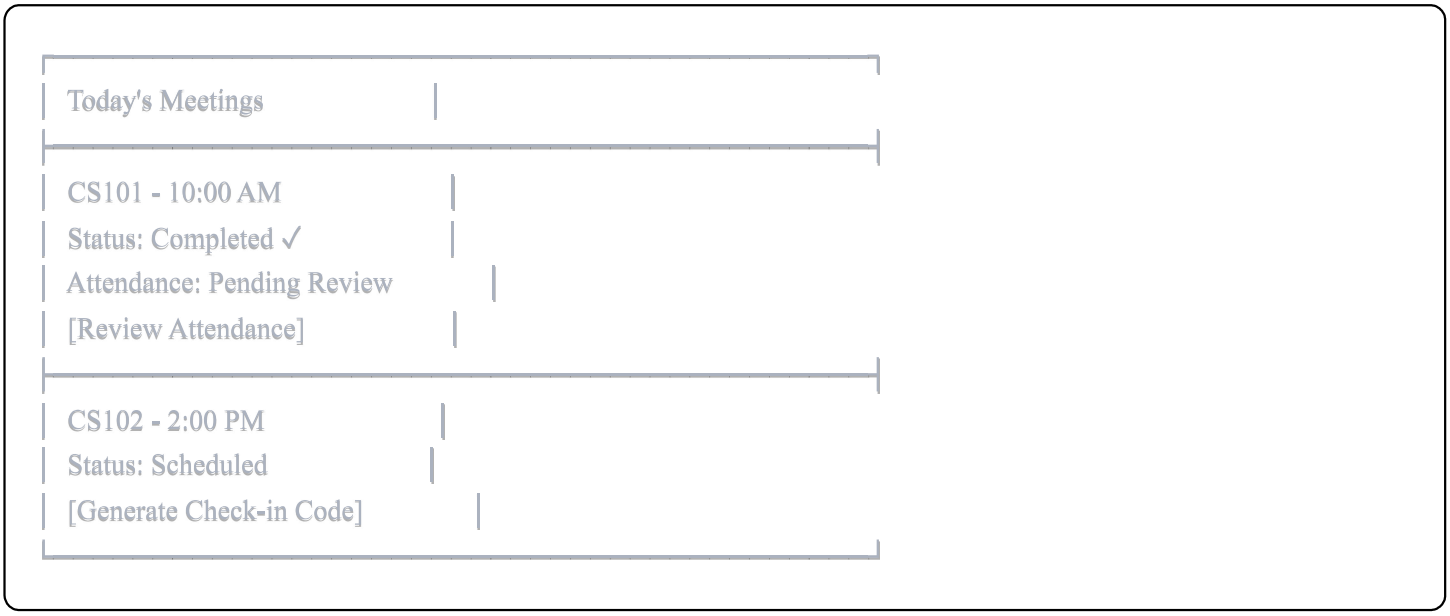
Real-World Example Flow

Scenario: CS101 class with 50 students

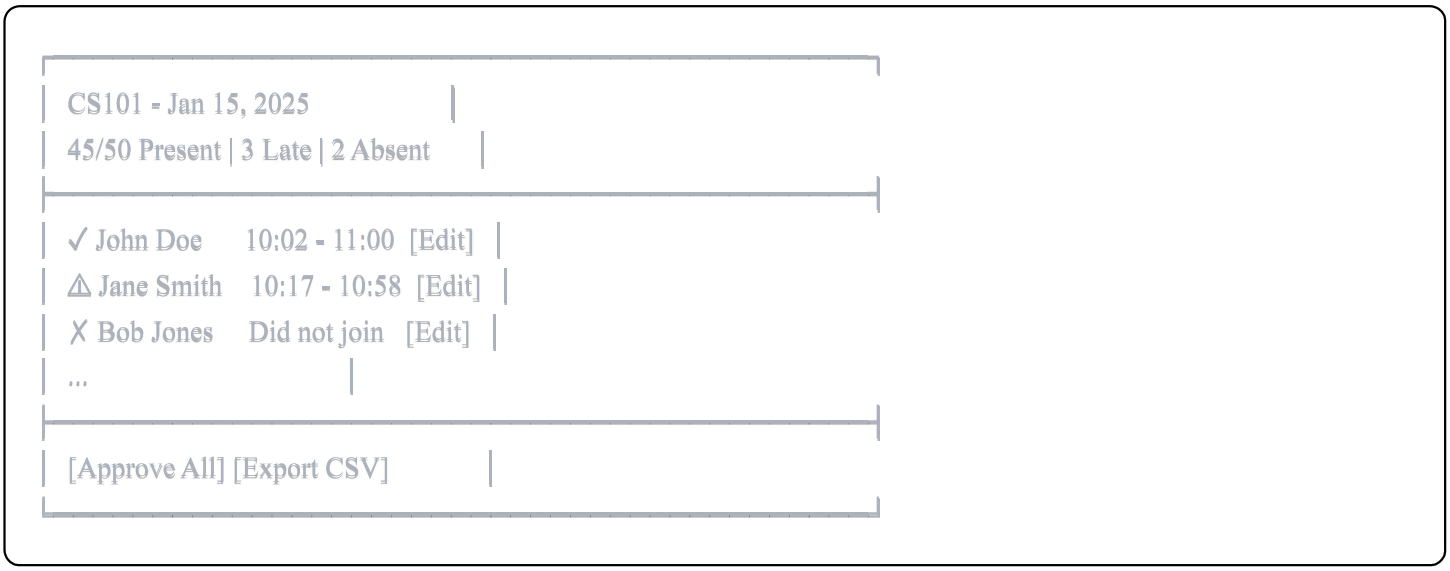
```
Professor Setup (Week 1):
1. Professor creates account on your system
2. Adds 50 students (imports CSV from student list)
3. Connects Zoom account via OAuth
4. System syncs scheduled meetings

Meeting Day:
1. 10:00 AM - Professor starts Zoom meeting
2. Students join normally via Zoom link
3. 10:05 AM - Late threshold passes
4. 11:00 AM - Meeting ends
5. 11:05 AM - Your system auto-fetches participant data
6. 11:05 AM - System marks attendance:
   - 45 present (joined before 10:05, stayed 80%+)
   - 3 late (joined after 10:05)
   - 2 absent (never joined or < 20% duration)

Professor Review (11:10 AM):
1. Gets notification: "Attendance ready for CS101"
2. Opens dashboard, reviews list
3. Sees "Jane Doe - Absent"
4. Remembers she emailed about internet issues
5. Manually changes to "Excused"
6. Clicks "Approve Attendance"
7. System saves final records

Student View:
1. Student logs in to your system
2. Sees attendance history
3. Sees: CS101 - Jan 15 - Present ✓
```

User Interface Mockups

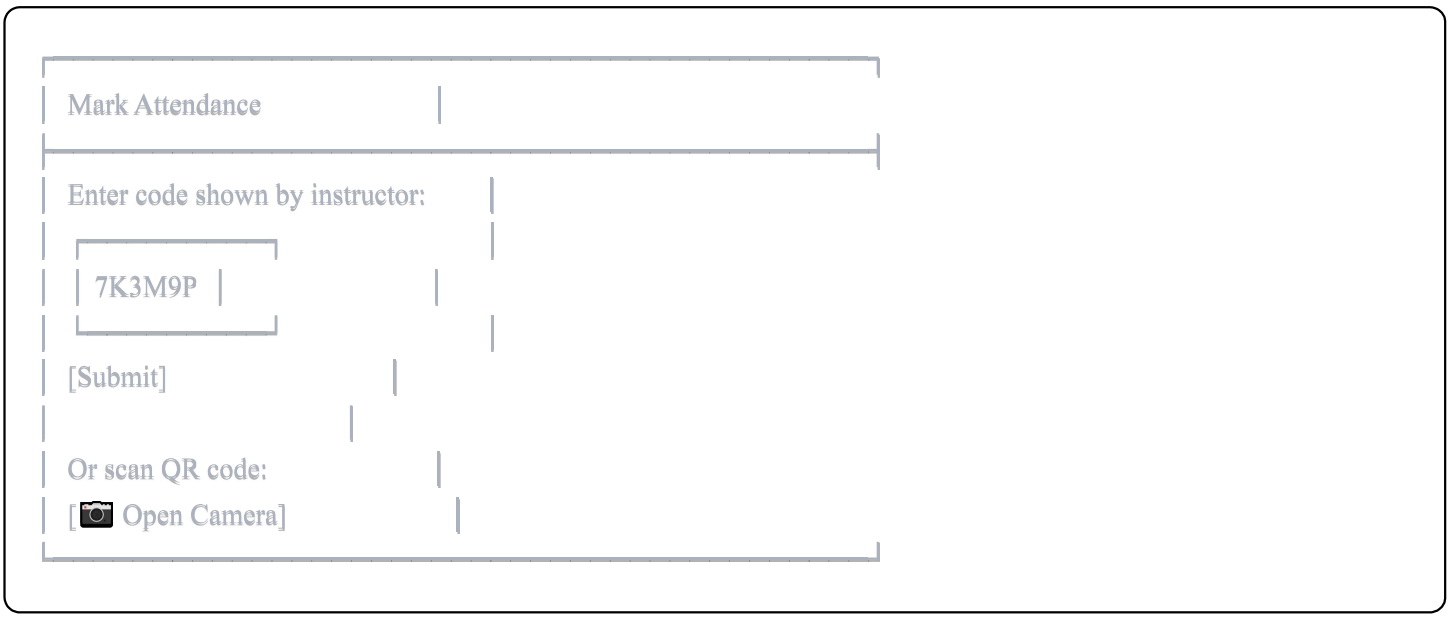
Instructor Dashboard



Attendance Review Screen



Student Self Check-in



Backend Implementation Flow

1. Meeting Sync Process

```
javascript

// Scheduled job (runs every 15 minutes)
1. Fetch upcoming meetings from each platform
2. Check if meeting exists in database
3. If new, create meeting record
4. If exists, update details if changed
5. Store sync timestamp
```

2. Attendance Capture Process

```
javascript

// Real-time via webhooks
1. Platform sends webhook notification
2. Verify webhook signature
3. Parse participant data
4. Create/update attendance records
5. Emit WebSocket event to connected clients
6. Calculate attendance status based on rules
```

3. Manual Attendance Flow

```
javascript

1. Instructor views meeting attendance
2. Can manually mark/override attendance
3. System logs manual changes
4. Updates attendance record with marker info
```

Frontend Architecture

Page Structure

```
/
├── /auth
│   ├── /login
│   ├── /register
│   └── /forgot-password
├── /dashboard
│   ├── Overview (stats, upcoming meetings)
```

```
│   └── Quick actions
├── /meetings
│   ├── /list (all meetings)
│   ├── /create
│   ├── /:id (meeting details)
│   └── /:id/attendance
├── /attendance
│   ├── /records (all records)
│   └── /mark (manual marking)
├── /reports
│   ├── /overview
│   ├── /users
│   └── /meetings
├── /settings
│   ├── /profile
│   ├── /integrations
│   └── /organization
└── /admin (admin only)
    ├── /users
    └── /system
```

State Management Structure

```javascript
// Redux slices or Zustand stores
- authSlice: user, token, isAuthenticated
- meetingsSlice: meetings, selectedMeeting, filters
- attendanceSlice: records, statistics
- integrationsSlice: connectedPlatforms, status
- uiSlice: loading, notifications, modals
```

Component Hierarchy

```
App
├── AuthProvider
├── Router
│   ├── PublicRoute
│   │   └── LoginPage
│   └── PrivateRoute
│       └── Layout
│           ├── Navbar
│           ├── Sidebar
│           ├── MainContent
│           │   ├── Dashboard
│           │   ├── MeetingList
```

Security Considerations

Authentication & Authorization

- JWT-based authentication with refresh tokens
- Role-based access control (RBAC)
- Password hashing with bcrypt (10+ rounds)
- Rate limiting on auth endpoints
- CSRF protection
- Secure cookie settings (httpOnly, secure, sameSite)

Data Protection

- Encrypt sensitive data at rest (API keys, tokens)
- Use environment variables for secrets
- HTTPS only in production
- Input validation and sanitization
- SQL injection protection (Mongoose helps with this)
- XSS prevention

API Security

- API key rotation
- Webhook signature verification
- OAuth token refresh handling
- Request timeout handling
- CORS configuration

Development Phases

Phase 1: Foundation (2-3 weeks)

- [ ] Setup project structure (frontend + backend)
- [ ] Configure MongoDB and create schemas

- [ ] Implement authentication system
- [ ] Build basic CRUD for users and meetings
- [ ] Create frontend layout and routing

Phase 2: Platform Integration (3-4 weeks)

- [ ] Implement Zoom OAuth and API integration
- [ ] Setup webhook handlers for Zoom
- [ ] Implement Google Meet integration
- [ ] Test attendance capture from platforms
- [ ] Build integration management UI

Phase 3: Attendance Features (2-3 weeks)

- [ ] Automatic attendance marking logic
- [ ] Manual attendance interface
- [ ] Attendance rules engine (late, absent, etc.)
- [ ] Real-time updates via WebSocket
- [ ] Bulk operations

Phase 4: Reporting & Analytics (2 weeks)

- [ ] Build attendance reports
- [ ] Create data visualization dashboards
- [ ] Export functionality (CSV, PDF)
- [ ] Email notifications
- [ ] Attendance statistics

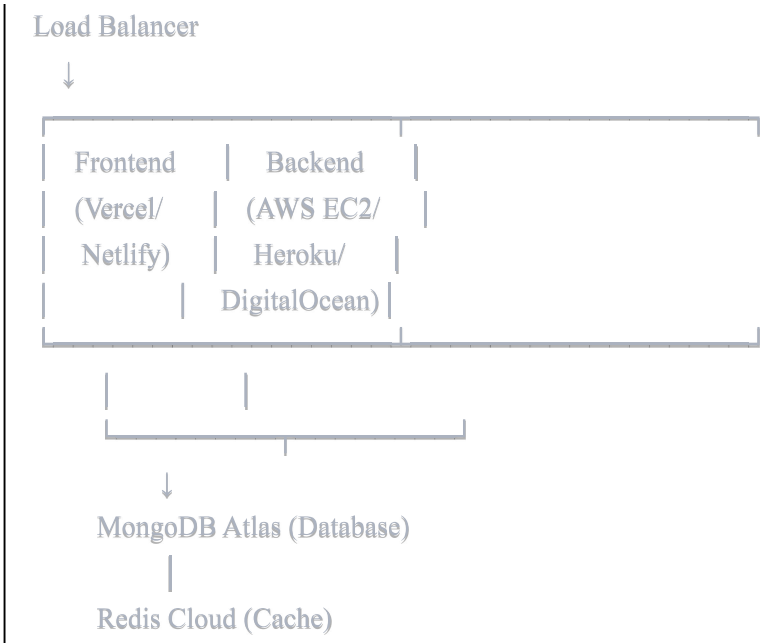
Phase 5: Polish & Deploy (1-2 weeks)

- [ ] Error handling and edge cases
- [ ] Performance optimization
- [ ] Security audit
- [ ] Documentation
- [ ] Deployment setup
- [ ] Testing (unit, integration, e2e)

Deployment Architecture

Recommended Setup

```
CloudFlare (CDN)
    ↓
Nginx (Reverse Proxy, SSL)
    ↓
```



Environment Variables

bash

```
# Backend
NODE_ENV=production
PORT=5000
MONGODB_URI=mongodb+srv://...
JWT_SECRET=...
JWT_REFRESH_SECRET=...
REDIS_URL=...

# Zoom
ZOOM_CLIENT_ID=...
ZOOM_CLIENT_SECRET=...
ZOOM_WEBHOOK_SECRET=...

# Google
GOOGLE_CLIENT_ID=...
GOOGLE_CLIENT_SECRET=...

# Microsoft
AZURE_CLIENT_ID=...
AZURE_CLIENT_SECRET=...
AZURE_TENANT_ID=...

# Frontend
REACT_APP_API_URL=https://api.yourdomain.com
REACT_APP_WS_URL=wss://api.yourdomain.com
```

Testing Strategy

Backend Testing

- **Unit Tests**: Jest for individual functions
- **Integration Tests**: Supertest for API endpoints
- **Mock External APIs**: Use nock or MSW

Frontend Testing

- **Unit Tests**: Jest + React Testing Library
- **Component Tests**: Test user interactions
- **E2E Tests**: Cypress or Playwright

Coverage Goals

- Backend: 80%+ coverage
- Frontend: 70%+ coverage
- Critical paths: 100% coverage

Monitoring & Maintenance

Logging

- Application logs (Winston)
- Error tracking (Sentry)
- API request logs
- Webhook logs

Monitoring

- Uptime monitoring (UptimeRobot)
- Performance monitoring (New Relic/Datadog)
- Database performance (MongoDB Atlas monitoring)

Backup Strategy

- Automated daily database backups
- Store backups in separate region
- Test restore procedures monthly

Future Enhancements

1. **Mobile App**: React Native companion app
2. **Additional Platforms**: Webex, GoToMeeting
3. **Face Recognition**: For in-person attendance
4. **AI Insights**: Attendance pattern analysis
5. **Geolocation**: Verify physical attendance
6. **Integration with LMS**: Canvas, Moodle, Blackboard
7. **Advanced Analytics**: Predictive attendance
8. **Multi-language Support**: i18n implementation

Resources & Documentation

Official Documentation

- Zoom API Docs
- Google Meet API
- Microsoft Graph API
- MongoDB Manual
- Express.js Guide
- React Documentation

Useful Libraries

- `zoomapis/zoom-api-js` - Zoom SDK
- `googleapis` - Google APIs Node.js Client
- `@microsoft/microsoft-graph-client` - Graph SDK
- `socket.io` - Real-time communication
- `axios` - HTTP client
- `joi` or `yup` - Validation
- `helmet` - Security headers

Getting Started Checklist

- [ ] Create GitHub repository

- [ ] Setup MongoDB Atlas account
- [ ] Register apps on Zoom/Google/Microsoft
- [ ] Setup local development environment
- [ ] Create project documentation
- [ ] Setup CI/CD pipeline
- [ ] Configure environment variables
- [ ] Initialize frontend (create-react-app or Vite)
- [ ] Initialize backend (Express generator)
- [ ] Create initial database schemas

Next Steps: Start with Phase 1 and build the foundation. Focus on getting authentication and basic CRUD working before moving to platform integrations.