🎯 Project Goals & Overview

Primary Objectives:

1. Seamless Google Calendar Integration - Create a meeting scheduling system that directly integrates with users' Google Calendars
2. Real-time Collaboration - Enable instant notifications and updates for meeting invitations and status changes
3. Universal Accessibility - Support both registered and unregistered users with differentiated experiences
4. Enterprise-grade Security - Implement OAuth 2.0 with JWT tokens and secure session management

Core Value Proposition:

Your application functions as a sophisticated meeting orchestrator that bridges the gap between web-based scheduling and native Google Calendar functionality, providing a Calendly-like experience with enhanced real-time features.

🗄️ Database Architecture & Design

Schema Analysis

Your PostgreSQL schema demonstrates a well-structured relational design:

Core Tables:

`users` Table - Central identity management

- id: UUID primary key (using uuid-ossp extension)
- email: Unique identifier for authentication
- google_id: OAuth identity linking
- google_refresh_token: Persistent API access (critical for calendar operations)
- avatar_url: User profile imagery from Google

`meetings` Table - Event management hub
- host_id: Links to user who creates the meeting
- google_event_id: Bidirectional sync with Google Calendar
- start_ts/end_ts: Timezone-aware scheduling (TIMESTAMPTZ)

`invitations` Table - RSVP state management
- Uses ENUM type for status: ('pending', 'accepted', 'declined')
- Links meetings to invitees with cascade deletion

`notifications` Table - Real-time messaging system
- invite_id: Links to specific invitation
- recipient_id: Direct user targeting
- is_read: State tracking for UI

Advanced Features:

- UUID Generation: Uses `uuid-ossp` extension for globally unique identifiers
- Automatic Timestamps: Triggers update `updated_at` columns automatically
- Session Storage: `sessions` table for express-session persistence
- Performance Optimization: Strategic indexes on foreign keys and frequently queried columns

🔧 Backend Infrastructure & Architecture

Entry Point Analysis

Your `index.js` follows a well-structured Express.js architecture:

Middleware Stack (Order Matters):

1. CORS (credentials: true) - Cross-origin cookie sharing
2. Webhook routes - Before authentication (external Google callbacks)
3. Cookie parser - JWT token extraction
4. Session management - PostgreSQL-backed sessions
5. Passport initialization - OAuth middleware
6. JSON parsing - Request body handling
7. Route mounting - API endpoint organization

Security Implementation:

- CORS Configuration: Explicitly allows `localhost:3000` with credentials
- Session Security: HTTP-only cookies with 7-day expiration
- Route Protection: Authentication middleware on sensitive endpoints

Authentication Architecture

Your Passport.js configuration implements sophisticated OAuth 2.0 flows:

Google Strategy Configuration:
- Scopes: ['profile', 'email', 'calendar'] - Comprehensive permissions
- authorizationParams override: Ensures 'offline' access and 'consent' prompts
- Smart token handling: Only updates refresh tokens when new ones are received

Session Management:

- Serialization: Stores only user ID in session (lightweight)
- Deserialization: Fetches fresh user data on each request
- Token Persistence: Protects existing refresh tokens from null overwrites

Meeting Business Logic

Your `createMeeting` controller demonstrates enterprise-level transaction handling:

Transaction Flow:

1. Validation Layer: Email format, required fields, array structure
2. User Resolution: Distinguishes registered vs. unregistered invitees
3. Google Calendar Integration: Creates events with Meet links
4. Database Persistence: Atomic transactions with rollback capability
5. Real-time Notifications: Socket.IO emission to connected users

Error Handling Strategy:

- Transaction rollback on Google API failures
- Graceful degradation when refresh tokens missing
- Comprehensive logging for debugging

Google API Integration

Your `google.js` service encapsulates Google Calendar API complexity:
OAuth2 Client Management:

- Token Validation: Ensures refresh tokens exist before API calls
- Environment Validation: Checks required Google credentials
- Error Propagation: Detailed error logging with context

Calendar Operations:

- Event Creation: Supports Google Meet conference data
- Webhook Registration: Push notification setup for external changes
- Update Strategy: `sendUpdates: 'all'` ensures attendee notifications

Authentication Routes

Your auth router implements secure OAuth flows:

OAuth Callback Logic:

1. Passport authentication with failure handling
2. Refresh token persistence (only when new)
3. Webhook registration for calendar monitoring
4. JWT cookie generation with environment-specific security
5. Frontend redirection with error parameters

⚛️ Frontend Architecture & State Management

Application Structure

Your React application follows modern architectural patterns:

Context Providers Hierarchy:

AuthProvider (outermost) → SocketProvider (conditional) → Router → Routes

Route Protection Strategy:

- ProtectedRoute Component: Wrapper for authenticated-only pages
- Conditional Socket Context: Only provides real-time features to authenticated users
- Smart Redirects: Automatic navigation based on authentication state

Authentication State Management

Your `AuthContext` implements robust authentication patterns:

Reducer Pattern:

- LOGIN_START: Loading state management
- LOGIN_SUCCESS: User data persistence
- LOGIN_ERROR: Error state handling
- LOGOUT: Complete state cleanup
- CLEAR_ERROR: UI error management

Network Resilience:

- Retry Logic: Automatic retry for network errors with exponential backoff
- Token Validation: Continuous auth status checking via `/auth/me`
- Error Boundaries: Graceful handling of authentication failures

Real-time Communication

Your `SocketContext` manages WebSocket connections efficiently:

Connection Management:
- Namespace Resolution: Strips /api to connect to root namespace
- User Association: Emits 'init' with userId for server-side tracking
- Cleanup Handling: Proper disconnection on component unmount

State Synchronization:

- Connection Status: Real-time connectivity feedback
- Automatic Reconnection: Socket.IO built-in resilience
- User Scoping: Server associates sockets with specific users

🔄 Integration Workflows & Data Flow

Meeting Creation Workflow:

1. Frontend Initiation (`NewMeeting.js`):

   User Input → Validation → API Call → Loading State

2. Backend Processing (`createMeeting`):
   Authentication → Transaction Start → User Resolution → Google Calendar API → Database Persistence → Socket Emission
3. Real-time Propagation:

   Socket.IO → Connected Invitees → UI Updates → Notification Display

Authentication Flow:

1. OAuth Initiation:

   Login Button → /api/auth/google → Google Consent → Callback Processing

2. Token Management:
   Refresh Token Storage → JWT Generation → HTTP-Only Cookie → Frontend State Update
3. Session Persistence:

   Page Reload → /auth/me Endpoint → Token Validation → Context Update

Google Calendar Synchronization:

1. Event Creation:

   Meeting Creation → Google API Call → Event ID Storage → Webhook Registration

2. External Changes:

   Google Calendar Webhook → RSVP Detection → Database Update → Socket Notification

🎯 Functionalities & Implementation Logic

1. User Authentication & Management

Approach:

- OAuth 2.0 with Google as the sole identity provider
- JWT tokens for stateless authentication
- Refresh token persistence for long-term Google API access

Logic:

```
// Smart refresh token handling prevents overwriting valid tokens
if (refreshToken && refreshToken !== user.google_refresh_token) {
  // Only update when new token received
}
```

Outcomes:

- Seamless single sign-on experience
- Persistent Google Calendar access
- Secure session management

2. Meeting Scheduling & Calendar Integration

Approach:

- Transaction-based database operations
- Atomic Google Calendar event creation
- Rollback capability for consistency

Logic:

```
// Google Calendar event structure
const eventData = {
  summary, description, start, end, attendees,
  conferenceData: { // Google Meet integration
    createRequest: {
      conferenceSolutionKey: { type: 'hangoutsMeet' }
    }
  }
};
```

Outcomes:

- Automatic Google Meet link generation
- Bidirectional calendar synchronization
- Consistent data across platforms

3. Real-time Notifications

Approach:

- Socket.IO for WebSocket management
- User-specific room joining
- Event-driven notification system

Logic:

```
// Server-side user association

socket.on('init', ({ userId }) => {

  activeUsers.set(socket.id, userId);

  socket.join(`user:${userId}`);

});

// Targeted notifications

getIO().to(`user:${inviteeId}`).emit('notification', payload);
```

Outcomes:

- Instant invitation notifications
- Real-time RSVP updates
- Live connection status

4. Invitation Management

Approach:

- Differential handling for registered/unregistered users
- Email domain detection for enhanced features
- State synchronization across platforms

Logic:

```
// User classification
const foundEmails = usersResult.rows.map(u => u.email.toLowerCase());
const notFoundEmails = emails.filter(e => !foundEmails.includes(e));

// Gmail detection for unregistered users
const isGmail = email.toLowerCase().endsWith('@gmail.com');
```

Outcomes:

- Universal invitation capability
- Enhanced features for Gmail users
- Seamless RSVP tracking

5. Webhook Integration

Approach:

- Google Calendar push notifications
- Secure webhook validation
- Automated status synchronization

Logic:

```
// Webhook security
const incomingToken = req.query.token;
if (incomingToken !== process.env.WEBHOOK_SECRET) {
  return res.status(403).send('Forbidden');
}

// RSVP status synchronization
if (responseStatus === 'accepted' && invite.status !== 'accepted') {
  // Update database and notify users
}
```

Outcomes:

- Automatic RSVP detection
- Cross-platform consistency
- Reduced manual synchronization

🏗️ Technical Infrastructure

Backend Stack:

- Runtime: Node.js with ES6 modules
- Framework: Express.js with middleware architecture
- Database: PostgreSQL with connection pooling
- Authentication: Passport.js with Google OAuth 2.0
- Real-time: Socket.IO with room-based targeting
- API Integration: Google Calendar API v3

Frontend Stack:

- Framework: React 18 with functional components
- State Management: Context API with useReducer
- Routing: React Router v6 with protected routes
- HTTP Client: Axios with interceptors
- Real-time: Socket.IO client with connection management

Development Environment:

- Backend Port: 5000 (configurable via environment)
- Frontend Port: 3000 (Create React App default)
- Database: PostgreSQL with UUID extensions
- Session Storage: PostgreSQL-backed sessions

📊 Outcomes & Achievements

Functional Achievements:

1. Complete Google Integration: Seamless calendar synchronization with automatic event creation
2. Real-time Collaboration: Instant notifications and status updates
3. Universal Access: Support for both registered and unregistered users
4. Robust Authentication: Secure OAuth 2.0 implementation with refresh token management
5. Enterprise Security: HTTP-only cookies, CORS configuration, and secure session management