ROTATION App - Cursor AI Implementation Guide

This guide provides step-by-step instructions and prompts for using Cursor AI to build the ROTATION mobile app.

Table of Contents

1. Project Setup
2. Backend Development
3. Mobile App Development
4. Integration & Testing
5. Deployment

Project Setup

Step 1: Initialize Projects

Backend Setup

Bash

```
# Create backend directory
mkdir rotation-backend && cd rotation-backend

# Initialize Node.js project
npm init -y

# Install dependencies
npm install express cors dotenv
npm install socket.io pg redis ioredis
npm install jsonwebtoken bcryptjs
npm install stripe firebase-admin
npm install @types/express @types/node typescript ts-node nodemon --save-dev

# Initialize TypeScript
npx tsc --init
```

Cursor AI Prompt:

Plain Text

```
Create a Node.js + Express + TypeScript backend project structure with:
- src/ folder with controllers, services, models, routes, middleware,
websocket, utils, config
- TypeScript configuration for Node.js
- ESLint and Prettier setup
- Environment variable configuration with dotenv
- Basic Express server setup with CORS
- Error handling middleware
- Request logging middleware
- Health check endpoint
```

Mobile App Setup

Bash

```
# Create React Native project
npx react-native init RotationApp --template react-native-template-typescript

cd RotationApp

# Install dependencies
npm install @react-navigation/native @react-navigation/stack
npm install react-native-screens react-native-safe-area-context
npm install socket.io-client axios
npm install @reduxjs/toolkit react-redux
npm install react-native-sound react-native-vibration
npm install @react-native-firebase/app @react-native-firebase/auth @react-
native-firebase/messaging
npm install react-native-paper react-native-vector-icons
npm install @stripe/stripe-react-native
```

Cursor AI Prompt:

Plain Text

```
Set up a React Native TypeScript project with:
- Navigation structure using React Navigation (Stack Navigator)
- Redux Toolkit for state management with slices for auth, session, rotation
- Folder structure: components/, screens/, services/, store/, utils/,
hooks/, assets/
- TypeScript interfaces for all data models
- API service layer with Axios
```

```
- WebSocket service layer with Socket.io-client
- Environment configuration for dev/staging/prod
```

Backend Development

Step 2: Database Setup

Cursor AI Prompt:

Plain Text

```
Create PostgreSQL database schema with Sequelize ORM for:

Tables:
1. users (id, email, username, password_hash, phone_number, created_at,
last_login, is_active, auth_provider)
2. user_profiles (id, user_id FK, display_name, avatar_url, preferences
JSON, total_sessions, total_rotations, created_at)
3. sessions (id, master_blunt_agent_id FK, session_name, session_code,
status ENUM, default_duration_seconds, started_at, ended_at, settings JSON,
created_at)
4. session_participants (id, session_id FK, user_id FK, join_order,
is_active, joined_at, left_at)
5. rotations (id, session_id FK, rotation_number, status ENUM,
duration_seconds, custom_sound_id, pass_phrase, current_turn_user_id FK,
started_at, created_at)
6. rotation_turns (id, rotation_id FK, user_id FK, turn_order,
turn_started_at, turn_ended_at, duration_seconds, skipped, timed_out)
7. rotation_history (id, rotation_id FK, user_id FK, action_type, metadata
JSON, created_at)
8. subscriptions (id, user_id FK, tier ENUM, status ENUM, started_at,
expires_at, payment_provider_id, amount, created_at)
9. custom_sounds (id, name, description, file_url, category ENUM,
is_premium, is_default, usage_count, created_at)
10. user_sounds (id, user_id FK, sound_id FK, is_favorite, added_at)

Include:
- Sequelize models with associations
- Migration files
- Seed data for custom_sounds table (10 default sounds)
- Database connection configuration
```

Step 3: Authentication Service

Cursor AI Prompt:

Plain Text

```
Create a complete authentication system with:

1. Auth Controller (src/controllers/authController.ts):
   - POST /api/v1/auth/register - Register new user
   - POST /api/v1/auth/login - Login with email/password
   - POST /api/v1/auth/logout - Logout user
   - POST /api/v1/auth/refresh - Refresh JWT token
   - POST /api/v1/auth/forgot-password - Send password reset email
   - POST /api/v1/auth/reset-password - Reset password with token

2. Auth Service (src/services/authService.ts):
   - User registration with bcrypt password hashing
   - User login with JWT token generation
   - Token refresh logic
   - Password reset token generation and validation

3. Auth Middleware (src/middleware/authMiddleware.ts):
   - JWT token verification
   - User authentication check
   - Role-based authorization (User, MBA, Admin)

4. Validation:
   - Email format validation
   - Password strength validation (min 8 chars, 1 uppercase, 1 number)
   - Input sanitization

Include proper error handling and TypeScript types.
```

Step 4: Session Management Service

Cursor AI Prompt:

Plain Text

```
Create session management system with:

1. Session Controller (src/controllers/sessionController.ts):
   - POST /api/v1/sessions - Create new session (MBA)
   - GET /api/v1/sessions/:id - Get session details
   - PUT /api/v1/sessions/:id - Update session settings (MBA only)
   - DELETE /api/v1/sessions/:id - Delete session (MBA only)
   - POST /api/v1/sessions/join - Join session with code
   - POST /api/v1/sessions/:id/leave - Leave session
```

```
   - GET /api/v1/sessions/:id/participants - Get all participants
   - POST /api/v1/sessions/:id/participants - Add participant (MBA only)
   - DELETE /api/v1/sessions/:id/participants/:userId - Remove participant
(MBA only)

2. Session Service (src/services/sessionService.ts):
   - Generate unique 6-character session codes
   - Validate session codes
   - Check if user is MBA
   - Add/remove participants
   - Update session settings
   - Session lifecycle management (active, paused, ended)

3. Middleware:
   - Check if user is session participant
   - Check if user is MBA of session
   - Validate session exists and is active

Include Redis caching for active sessions.
```

Step 5: Rotation Service

Cursor AI Prompt:

Plain Text

```
Create rotation management system with:

1. Rotation Controller (src/controllers/rotationController.ts):
   - POST /api/v1/sessions/:sessionId/rotations - Create rotation
   - GET /api/v1/rotations/:id - Get rotation details
   - PUT /api/v1/rotations/:id - Update rotation settings (MBA only)
   - POST /api/v1/rotations/:id/start - Start rotation (MBA only)
   - POST /api/v1/rotations/:id/pause - Pause rotation (MBA only)
   - POST /api/v1/rotations/:id/end - End rotation (MBA only)
   - POST /api/v1/rotations/:id/pass - Pass turn to next person
   - GET /api/v1/rotations/:id/turns - Get turn history
   - GET /api/v1/rotations/:id/history - Get rotation action history

2. Rotation Service (src/services/rotationService.ts):
   - Create rotation with participant order
   - Timer logic (start, pause, reset)
   - Turn management (current turn, next turn, skip turn)
   - Calculate turn duration
   - Record turn history
   - Handle multiple rotations per session
```

```
3. Business Logic:
   - Circular rotation queue (after last person, go back to first)
   - Automatic turn advancement
   - Turn timeout handling
   - Skip functionality
   - Statistics calculation (average turn time, total rounds)

Include Redis for real-time rotation state caching.
```

Step 6: WebSocket Server

Cursor AI Prompt:

Plain Text

```
Create WebSocket server with Socket.io for real-time synchronization:

1. WebSocket Server (src/websocket/server.ts):
   - Initialize Socket.io server
   - Authentication middleware for socket connections
   - Session room management

2. Event Handlers (src/websocket/handlers/):
   - sessionHandler.ts:
     * join_session - User joins session room
     * leave_session - User leaves session room
     * session_updated - Broadcast session changes
     * participant_joined - Broadcast new participant
     * participant_left - Broadcast participant departure

   - rotationHandler.ts:
     * start_rotation - MBA starts rotation
     * rotation_started - Broadcast rotation start
     * turn_changed - Broadcast turn change
     * pass_turn - User passes turn
     * timer_alert - Broadcast timer alerts
     * rotation_ended - Broadcast rotation end

3. Features:
   - Room-based messaging (one room per session)
   - Broadcast to all participants except sender
   - Emit to specific user
   - Connection/disconnection handling
   - Reconnection with state recovery
   - Error handling and logging

4. Integration:
```

```
   - Connect WebSocket events to Rotation Service
   - Update Redis cache on state changes
   - Trigger push notifications via Notification Service
```

Step 7: Payment & Subscription Service

Cursor AI Prompt:

Plain Text

```
Create payment and subscription system with Stripe:

1. Subscription Controller (src/controllers/subscriptionController.ts):
   - GET /api/v1/subscriptions/plans - Get available plans
   - POST /api/v1/subscriptions/subscribe - Create subscription
   - GET /api/v1/subscriptions/me - Get user's subscription
   - POST /api/v1/subscriptions/cancel - Cancel subscription
   - POST /api/v1/subscriptions/webhook - Stripe webhook handler

2. Subscription Service (src/services/subscriptionService.ts):
   - Create Stripe customer
   - Create subscription with Stripe
   - Handle subscription lifecycle (active, canceled, expired)
   - Check subscription status and tier
   - Grant/revoke premium features
   - Handle payment failures

3. Subscription Plans:
   - Free: Basic features, ads
   - Premium Monthly: $2.99/month
   - Premium Yearly: $19.99/year (save 44%)
   - Lifetime: $49.99 one-time

4. Features:
   - Stripe webhook signature verification
   - Subscription status sync
   - Trial period support (7 days)
   - Promo code support
   - Invoice generation

Include middleware to check subscription tier for premium features.
```

Step 8: Content & Sound Service

Cursor AI Prompt:

Plain Text

```
Create content management system for custom sounds:

1. Sound Controller (src/controllers/soundController.ts):
   - GET /api/v1/sounds - Get all sounds (filtered by free/premium)
   - GET /api/v1/sounds/:id - Get sound details
   - POST /api/v1/sounds/upload - Upload custom sound (premium only)
   - GET /api/v1/users/me/sounds - Get user's sounds
   - POST /api/v1/users/me/sounds/:id/favorite - Favorite a sound
   - DELETE /api/v1/sounds/:id - Delete custom sound

2. Sound Service (src/services/soundService.ts):
   - Upload audio file to S3/CDN
   - Validate audio format (mp3, wav, ogg)
   - Validate audio duration (max 5 seconds)
   - Generate audio thumbnail/waveform
   - Track sound usage statistics
   - Content moderation (check for inappropriate content)

3. Default Sounds Library:
   - Beep (free, default)
   - Ding (free)
   - Chime (free)
   - Bell (free)
   - Buzzer (free)
   - Air Horn (premium)
   - DJ Scratch (premium)
   - "Pass the blunt" voice (premium)
   - "Your turn" voice (premium)
   - Custom uploaded sounds (premium)

4. Features:
   - CDN integration for fast audio delivery
   - Audio file compression
   - Lazy loading for sound library
   - Search and filter sounds by category
```

Mobile App Development

Step 9: Authentication Screens

Cursor AI Prompt:

```
Plain Text
```

```
Create authentication flow for React Native:

1. Screens (src/screens/auth/):
   - SplashScreen.tsx - App loading screen with logo animation
   - LoginScreen.tsx - Email/password login form
   - SignupScreen.tsx - Registration form
   - ForgotPasswordScreen.tsx - Password reset request
   - OnboardingScreen.tsx - First-time user tutorial (3 slides)

2. Components (src/components/auth/):
   - AuthInput.tsx - Styled text input with validation
   - AuthButton.tsx - Primary action button
   - SocialAuthButtons.tsx - Google/Apple sign-in buttons
   - PasswordStrengthIndicator.tsx - Visual password strength

3. Redux Slice (src/store/slices/authSlice.ts):
   - State: user, token, isAuthenticated, loading, error
   - Actions: login, signup, logout, refreshToken
   - Thunks: loginAsync, signupAsync, logoutAsync

4. API Service (src/services/authService.ts):
   - login(email, password)
   - signup(email, username, password)
   - logout()
   - refreshToken()
   - forgotPassword(email)
   - resetPassword(token, newPassword)

5. Features:
   - Form validation with error messages
   - Loading states during API calls
   - Token storage with AsyncStorage
   - Auto-login on app launch
   - Biometric authentication (Face ID/Touch ID)

Use React Native Paper for UI components and React Hook Form for form
handling.
```

Step 10: Home & Navigation

Cursor AI Prompt:

Plain Text

```
Create main navigation and home screen:

1. Navigation (src/navigation/):
```

```
   - AppNavigator.tsx - Root navigator with auth check
   - AuthNavigator.tsx - Stack navigator for auth screens
   - MainNavigator.tsx - Stack navigator for main app screens
   - TabNavigator.tsx - Bottom tab navigator (Home, Profile, Premium)

2. Home Screen (src/screens/HomeScreen.tsx):
   - Header with user avatar and settings button
   - "Create Session" button (large, prominent)
   - "Join Session" button
   - Recent sessions list (last 5 sessions)
   - Quick stats card (total sessions, total rotations)
   - Premium upsell banner (if free user)

3. Components:
   - SessionCard.tsx - Display session info in list
   - StatsCard.tsx - Display user statistics
   - PremiumBanner.tsx - Upsell banner for premium features

4. Features:
   - Pull-to-refresh for recent sessions
   - Navigation to create/join session screens
   - Deep linking support for session codes
   - Push notification handling

Use React Native Paper for UI and React Navigation for navigation.
```

Step 11: Session Creation & Setup

Cursor AI Prompt:

Plain Text

```
Create session creation and setup flow:

1. Screens:
   - CreateSessionScreen.tsx - Initial session creation
   - SessionSetupScreen.tsx - Configure session settings (MBA only)
   - WaitingRoomScreen.tsx - Pre-rotation participant gathering

2. CreateSessionScreen.tsx:
   - Session name input (optional)
   - "Create Session" button
   - Loading state while generating session code
   - Navigate to SessionSetupScreen on success

3. SessionSetupScreen.tsx:
   - Display generated session code (large, shareable)
```

```
   - "Share Code" button (share via SMS, WhatsApp, etc.)
   - Duration picker (15s, 30s, 45s, 60s, 90s, 120s, custom)
   - Sound selector (dropdown with preview)
   - Pass phrase input (optional)
   - "Continue to Waiting Room" button

4. WaitingRoomScreen.tsx:
   - Session code display at top
   - Participant list with avatars
   - "Add Participant" button (MBA only)
   - "Remove Participant" button next to each user (MBA only)
   - "Start Rotation" button (MBA only, enabled when 2+ participants)
   - Real-time participant updates via WebSocket

5. Components:
   - SessionCodeDisplay.tsx - Large, copyable session code
   - DurationPicker.tsx - Custom picker for duration
   - SoundSelector.tsx - Dropdown with sound preview
   - ParticipantList.tsx - List of participants with avatars
   - ParticipantItem.tsx - Single participant row

6. Redux Slice (src/store/slices/sessionSlice.ts):
   - State: currentSession, participants, loading, error
   - Actions: createSession, updateSession, addParticipant, removeParticipant
   - Thunks: createSessionAsync, joinSessionAsync, leaveSessionAsync

7. WebSocket Integration:
   - Connect to session room on WaitingRoomScreen mount
   - Listen for participant_joined and participant_left events
   - Update participant list in real-time
   - Disconnect on screen unmount

Use React Native Share for code sharing and React Native Modal for dialogs.
```

Step 12: Join Session Flow

Cursor AI Prompt:

Plain Text

```
Create join session flow:

1. Screen:
   - JoinSessionScreen.tsx - Enter session code and join

2. JoinSessionScreen.tsx:
   - Session code input (6 characters, auto-uppercase)
```

```
   - "Join Session" button
   - Loading state during validation
   - Error message for invalid codes
   - Navigate to WaitingRoomScreen on success

3. Features:
   - Auto-format session code (XXX-XXX)
   - QR code scanner option (scan QR from partner products)
   - Recent session codes (last 3 joined)
   - Deep link handling (rotation://join/ABC123)

4. Components:
   - SessionCodeInput.tsx - Styled code input with auto-format
   - QRScanner.tsx - Camera-based QR code scanner
   - RecentSessionsList.tsx - List of recent session codes

5. API Integration:
   - Validate session code via API
   - Join session and get participant list
   - Connect to WebSocket session room

Use react-native-camera for QR scanning and react-native-masked-text for
code formatting.
```

Step 13: Active Rotation Screen

Cursor AI Prompt:

Plain Text

```
Create the core active rotation screen:

1. Screen:
   - ActiveRotationScreen.tsx - Main rotation screen with timer

2. Layout:
   - Top: Session info (name, code, participant count)
   - Center: Large circular timer with progress ring
   - Below timer: Current user's name and avatar
   - Bottom: "Pass" button (large, prominent)
   - Participant list on side (showing turn order)

3. Timer Component (src/components/rotation/Timer.tsx):
   - Circular progress ring (animated)
   - Countdown display (MM:SS)
   - Color changes: Green (100%-80%), Yellow (80%-20%), Red (20%-0%)
   - Pulse animation when time is low
```

```
   - Alert trigger at 80% and 100%

4. States:
   - YOUR_TURN: Timer active, "Pass" button enabled
   - WAITING: Timer shows other user's time, "Pass" button disabled
   - PAUSED: Timer paused (MBA only)
   - ENDED: Session ended, show summary

5. Features:
   - Real-time timer synchronization via WebSocket
   - Sound playback at 80% and 100%
   - Vibration at alerts
   - "Pass" button triggers turn change
   - Background timer (continue when app is backgrounded)
   - Foreground service notification (Android)

6. Components:
   - CircularTimer.tsx - Animated circular progress timer
   - TurnIndicator.tsx - Show whose turn it is
   - ParticipantQueue.tsx - Show turn order
   - PassButton.tsx - Large action button
   - SessionControls.tsx - MBA controls (pause, end, skip)

7. Redux Slice (src/store/slices/rotationSlice.ts):
   - State: rotation, currentTurn, timerState, turnHistory
   - Actions: startRotation, passTurn, endRotation, updateTimer
   - Thunks: startRotationAsync, passTurnAsync, endRotationAsync

8. WebSocket Integration:
   - Listen for turn_changed events
   - Listen for timer_alert events
   - Listen for rotation_ended events
   - Emit pass_turn when user passes
   - Sync timer state across all devices

9. Audio Integration:
   - Load custom sound from API
   - Play sound at alerts
   - Volume control
   - Fallback to default beep if custom sound fails

Use react-native-sound for audio, react-native-vibration for haptics, and
react-native-svg for circular progress.
```

Step 14: Timer Logic & Hooks

Cursor AI Prompt:

Plain Text

```
Create custom React hooks for timer logic:

1. useRotationTimer Hook (src/hooks/useRotationTimer.ts):
   - Parameters: duration (seconds), onAlert (callback), onComplete
(callback)
   - State: timeRemaining, isRunning, isPaused, progress (0-1)
   - Functions: start(), pause(), resume(), reset(), stop()
   - Effects:
     * Countdown every second when running
     * Trigger onAlert at 80% (warning) and 100% (time up)
     * Trigger onComplete when timer reaches 0
     * Continue in background using react-native-background-timer
   - Return: { timeRemaining, progress, isRunning, isPaused, start, pause,
resume, reset, stop }

2. useSound Hook (src/hooks/useSound.ts):
   - Parameters: soundUrl (string)
   - Functions: play(), stop(), setVolume(volume)
   - State: isLoaded, isPlaying, error
   - Effects:
     * Load sound file on mount
     * Unload on unmount
     * Handle loading errors
   - Return: { play, stop, setVolume, isLoaded, isPlaying, error }

3. useVibration Hook (src/hooks/useVibration.ts):
   - Functions: vibrate(pattern), vibrateOnce(), cancel()
   - Patterns: short (100ms), medium (200ms), long (500ms), pattern ([100,
200, 100])
   - Return: { vibrate, vibrateOnce, cancel }

4. useWebSocket Hook (src/hooks/useWebSocket.ts):
   - Parameters: sessionId (string)
   - Functions: connect(), disconnect(), emit(event, data), on(event,
handler)
   - State: isConnected, error
   - Effects:
     * Auto-connect on mount
     * Auto-reconnect on disconnect
     * Disconnect on unmount
   - Return: { isConnected, connect, disconnect, emit, on, error }

5. Background Timer:
   - Use react-native-background-timer for accurate timing
   - Continue timer when app is backgrounded
   - Show foreground service notification (Android)
```

```
   - Update notification with remaining time

Include proper cleanup and error handling in all hooks.
```

Step 15: Profile & Statistics

Cursor AI Prompt:

Plain Text

```
Create profile and statistics screens:

1. ProfileScreen.tsx:
   - Header with avatar and display name
   - Edit profile button
   - Statistics section:
     * Total sessions participated
     * Total rotations
     * Average turn time
     * Fastest pass time
     * Longest session
   - Settings section:
     * Notifications toggle
     * Sound volume slider
     * Vibration toggle
     * Theme selector (light/dark)
   - Premium section:
     * Current plan display
     * "Upgrade to Premium" button (if free)
     * "Manage Subscription" button (if premium)
   - Logout button

2. EditProfileScreen.tsx:
   - Avatar picker (camera or gallery)
   - Display name input
   - Email display (read-only)
   - Username input
   - Save button

3. StatisticsScreen.tsx:
   - Session history list (last 30 days)
   - Charts:
     * Sessions per week (bar chart)
     * Average turn time trend (line chart)
     * Turn time distribution (pie chart)
   - Achievements section:
     * Badges earned
```

```
     * Progress to next badge
   - Leaderboard (compare with friends)

4. Components:
   - StatCard.tsx - Display single statistic
   - SessionHistoryItem.tsx - Single session in history
   - AchievementBadge.tsx - Badge display
   - Chart components using react-native-chart-kit

5. Redux Slice (src/store/slices/profileSlice.ts):
   - State: profile, stats, achievements, loading, error
   - Actions: updateProfile, loadStats, loadAchievements
   - Thunks: updateProfileAsync, loadStatsAsync

Use react-native-image-picker for avatar selection and react-native-chart-
kit for charts.
```

Step 16: Premium Features & Subscription

Cursor AI Prompt:

Plain Text

```
Create premium features and subscription flow:

1. PremiumScreen.tsx:
   - Hero section with premium benefits
   - Subscription plans:
     * Free: Basic features, ads
     * Premium Monthly: $2.99/month
     * Premium Yearly: $19.99/year (save 44%)
     * Lifetime: $49.99 one-time
   - Feature comparison table
   - "Subscribe Now" buttons
   - "Restore Purchases" button (iOS)

2. Premium Benefits:
   - ✓ Custom alert sounds library (50+ sounds)
   - ✓ Upload personal audio files
   - ✓ Custom timer durations
   - ✓ Unlimited participants per session
   - ✓ Advanced statistics and history
   - ✓ Ad-free experience
   - ✓ Priority support
   - ✓ Early access to new features

3. CustomSoundsScreen.tsx:
```

```
   - Sound library with categories:
     * Default (free)
     * Voice Alerts (premium)
     * Music Clips (premium)
     * Sound Effects (premium)
     * My Uploads (premium)
   - Sound preview button
   - Download/favorite button
   - Upload custom sound button (premium)
   - Search and filter

4. UploadSoundScreen.tsx (Premium only):
   - File picker (audio files only)
   - Audio preview player
   - Trim audio tool (max 5 seconds)
   - Name and description inputs
   - Upload button

5. Payment Integration:
   - Stripe payment sheet
   - Apple Pay / Google Pay support
   - Payment confirmation
   - Subscription activation
   - Receipt email

6. Components:
   - PlanCard.tsx - Subscription plan display
   - FeatureComparisonTable.tsx - Feature comparison
   - SoundLibraryItem.tsx - Sound in library
   - AudioTrimmer.tsx - Trim audio file

7. Redux Slice (src/store/slices/subscriptionSlice.ts):
   - State: subscription, plans, loading, error
   - Actions: loadPlans, subscribe, cancelSubscription
   - Thunks: loadPlansAsync, subscribeAsync, cancelSubscriptionAsync

8. Subscription Checks:
   - Middleware to check subscription tier
   - Lock premium features for free users
   - Show upgrade prompts
   - Handle subscription expiration

Use @stripe/stripe-react-native for payments and react-native-document-
picker for file selection.
```

Step 17: Notifications & Push

Cursor AI Prompt:

Plain Text

```
Create notification system:

1. Notification Service (src/services/notificationService.ts):
   - Initialize Firebase Cloud Messaging
   - Request notification permissions
   - Get FCM token
   - Handle foreground notifications
   - Handle background notifications
   - Handle notification taps

2. Notification Types:
   - YOUR_TURN: "It's your turn! Pass the rotation."
   - TIME_WARNING: "10 seconds left on your turn!"
   - TIME_UP: "Time's up! Pass it along."
   - SESSION_STARTED: "Session ABC123 has started!"
   - SESSION_ENDED: "Session ABC123 has ended."
   - PARTICIPANT_JOINED: "John joined the session."
   - PARTICIPANT_LEFT: "Jane left the session."

3. Local Notifications:
   - Use react-native-push-notification
   - Schedule local notifications for timer alerts
   - Custom sound support
   - Vibration patterns

4. Push Notifications:
   - Firebase Cloud Messaging setup
   - Handle notification permissions
   - Token registration with backend
   - Deep linking from notifications

5. Notification Settings:
   - Enable/disable push notifications
   - Enable/disable sound
   - Enable/disable vibration
   - Notification sound selection

6. Components:
   - NotificationPermissionModal.tsx - Request permission
   - NotificationSettingsScreen.tsx - Manage settings

Use @react-native-firebase/messaging for push notifications.
```

Integration & Testing

Step 18: API Integration

Cursor AI Prompt:

Plain Text

```
Create complete API service layer:

1. API Client (src/services/apiClient.ts):
   - Axios instance with base URL
   - Request interceptor (add auth token)
   - Response interceptor (handle errors)
   - Token refresh logic
   - Retry failed requests

2. API Services:
   - authApi.ts - Authentication endpoints
   - sessionApi.ts - Session management endpoints
   - rotationApi.ts - Rotation endpoints
   - soundApi.ts - Sound library endpoints
   - subscriptionApi.ts - Payment endpoints
   - userApi.ts - User profile endpoints

3. Error Handling:
   - Network errors
   - Authentication errors (401)
   - Authorization errors (403)
   - Validation errors (400)
   - Server errors (500)
   - Timeout errors

4. Request/Response Types:
   - TypeScript interfaces for all API requests
   - TypeScript interfaces for all API responses
   - Generic API response wrapper

5. Features:
   - Request cancellation
   - Request debouncing
   - Offline queue (store requests when offline)
   - Request caching
   - Loading states

Include comprehensive error messages and retry logic.
```

Step 19: WebSocket Integration

Cursor AI Prompt:

Plain Text

```
Create WebSocket service for real-time features:

1. WebSocket Client (src/services/websocketClient.ts):
   - Socket.io client initialization
   - Connection with auth token
   - Auto-reconnection logic
   - Connection state management
   - Event emitter pattern

2. Event Handlers:
   - session_updated
   - participant_joined
   - participant_left
   - rotation_started
   - turn_changed
   - timer_alert
   - rotation_ended
   - session_ended

3. Integration with Redux:
   - Dispatch Redux actions on WebSocket events
   - Update store state in real-time
   - Sync local state with server state

4. Connection Management:
   - Connect on app foreground
   - Disconnect on app background
   - Reconnect on network change
   - Handle connection errors

5. Features:
   - Room-based messaging
   - Acknowledgment callbacks
   - Event queue when disconnected
   - Heartbeat/ping-pong

Include proper cleanup and memory leak prevention.
```

Step 20: Testing

Cursor AI Prompt:

Plain Text

```
Create test suite for ROTATION app:

1. Backend Tests:
   - Unit tests for services (Jest)
   - Integration tests for API endpoints (Supertest)
   - WebSocket tests
   - Database tests

2. Mobile App Tests:
   - Unit tests for components (Jest + React Native Testing Library)
   - Unit tests for Redux slices
   - Unit tests for hooks
   - Integration tests for screens
   - E2E tests (Detox)

3. Test Cases:
   - Authentication flow
   - Session creation and joining
   - Rotation timer logic
   - Turn passing
   - WebSocket synchronization
   - Payment processing
   - Subscription management

4. Mock Data:
   - Mock API responses
   - Mock WebSocket events
   - Mock user data
   - Mock session data

5. Test Coverage:
   - Aim for 80%+ coverage
   - Critical paths: 100% coverage
   - Edge cases and error handling

Generate test files for all major components and services.
```

Deployment

Step 21: Backend Deployment

Cursor AI Prompt:

Plain Text

```
Create deployment configuration for backend:

1. Docker Setup:
   - Dockerfile for Node.js app
   - docker-compose.yml with services:
     * Node.js app
     * PostgreSQL
     * Redis
     * Nginx (reverse proxy)

2. Environment Configuration:
   - .env.example file
   - Environment variables for dev/staging/prod
   - Secrets management

3. CI/CD Pipeline (GitHub Actions):
   - Run tests on push
   - Build Docker image
   - Push to container registry
   - Deploy to server
   - Run database migrations
   - Health check after deployment

4. Deployment Scripts:
   - deploy.sh - Deploy to production
   - rollback.sh - Rollback to previous version
   - migrate.sh - Run database migrations

5. Monitoring:
   - PM2 for process management
   - Winston for logging
   - Sentry for error tracking
   - Prometheus + Grafana for metrics

Include deployment instructions and troubleshooting guide.
```

Step 22: Mobile App Deployment

Cursor AI Prompt:

Plain Text

```
Create deployment configuration for mobile app:

1. iOS Deployment:
```

```
   - Xcode project configuration
   - App Store Connect setup
   - TestFlight beta testing
   - App Store submission checklist
   - Screenshots and app preview video

2. Android Deployment:
   - Build configuration (release variant)
   - Signing configuration
   - Google Play Console setup
   - Internal/beta/production tracks
   - Play Store listing

3. Fastlane Setup:
   - Fastfile for automated builds
   - iOS lane: build, test, upload to TestFlight
   - Android lane: build, test, upload to Play Store
   - Screenshot automation

4. Version Management:
   - Semantic versioning (1.0.0)
   - Build number increment
   - Changelog generation

5. Release Checklist:
   - [ ] Update version number
   - [ ] Run all tests
   - [ ] Build release version
   - [ ] Test on physical devices
   - [ ] Generate screenshots
   - [ ] Write release notes
   - [ ] Submit to app stores
   - [ ] Monitor crash reports

Include step-by-step deployment instructions.
```

Additional Resources

Useful Cursor AI Prompts

Code Refactoring

Plain Text

```
Refactor this component to:
- Use TypeScript strict mode
- Follow React best practices
- Optimize performance (useMemo, useCallback)
- Add proper error handling
- Add loading states
- Add accessibility labels
```

Bug Fixing

Plain Text

```
Debug this issue:
[Describe the bug]

Expected behavior: [What should happen]
Actual behavior: [What is happening]
Error message: [If any]

Suggest fixes and explain the root cause.
```

Performance Optimization

Plain Text

```
Optimize this code for:
- Faster rendering
- Reduced memory usage
- Better battery efficiency
- Smaller bundle size

Suggest specific improvements with code examples.
```

Documentation

Plain Text

```
Generate comprehensive documentation for this module including:
- Purpose and overview
- Function/component descriptions
- Parameters and return values
- Usage examples
- Edge cases and error handling
```

Next Steps

1. **Start with Backend**: Build API and database first
2. **Test with Postman**: Verify all endpoints work
3. **Build Mobile UI**: Create screens and components
4. **Integrate API**: Connect mobile app to backend
5. **Add WebSocket**: Implement real-time features
6. **Test End-to-End**: Test complete user flows
7. **Add Payments**: Integrate Stripe
8. **Polish UI/UX**: Refine design and animations
9. **Beta Testing**: Release to test users
10. **Launch**: Submit to app stores

Support

For questions or issues during development:

- Check the Technical Specification document
- Review architecture diagrams
- Consult API documentation
- Test with Postman/Insomnia

Good luck building ROTATION! 🚀