PlanLock - Complete Cursor Build Guide

8 Prompts to Production-Ready MVP

February 2026

PLANLOCK - CURSOR VIBECODING PROMPTS

Complete App Build in One Session

Instructions: Copy each prompt below and paste into Cursor Composer (Cmd/Ctrl + I) one at a time. Cursor will generate all the code.

SETUP CHECKLIST

Before starting, ensure you have: - ✅ Supabase project created - ✅ Database schema created (from earlier) - ✅ Sample events added - ✅ Expo project created: `npx create-expo-app@latest planlock --template blank-typescript` - ✅ Dependencies installed (see below)

Run these in PowerShell:

```
cd planlock
npm install --legacy-peer-deps @supabase/supabase-js @react-native-async-storage/async-storage react-native-url-polyfill expo-router zustand react-native-paper react-native-safe-area-context react-native-screens expo-notifications date-fns @react-native-community/datetimepicker react-native-maps expo-image-picker @expo/vector-icons
```

PROMPT 1: Project Configuration & Supabase Setup

```
Set up the complete project configuration for PlanLock:

1. Update app.json with:
   - App name: "PlanLock"
   - Scheme: "planlock"
   - Add expo-router plugin
   - Add expo-notifications plugin
   - Set orientation to portrait
   - Configure iOS bundle ID and Android package

2. Create lib/supabase.ts with:
   - Import AsyncStorage, createClient from @supabase/supabase-js
   - Import react-native-url-polyfill/auto
   - Create Supabase client with:
     - URL: process.env.EXPO_PUBLIC_SUPABASE_URL
     - Key: process.env.EXPO_PUBLIC_SUPABASE_ANON_KEY
     - Auth config: AsyncStorage, autoRefreshToken, persistSession
   - Export supabase client

3. Create .env file with placeholders:
   EXPO_PUBLIC_SUPABASE_URL=your_url_here
   EXPO_PUBLIC_SUPABASE_ANON_KEY=your_key_here

4. Create types/database.ts with TypeScript interfaces for:
   - Profile, Group, GroupMember, Venue, Event, Plan, PlanParticipant, PlanMessage

Use TypeScript strictly. Include proper imports.
```

PROMPT 2: Auth Store & Helpers

```
Create authentication and helper utilities:

1. Create store/auth.ts with Zustand store:
   - State: session, user (Profile type), loading, initialized
   - Actions:
     - initialize(): Check session, fetch profile, setup auth listener
     - setSession, setUser
     - checkProfile(userId): Check if profile exists
     - signOut(): Clear auth
     - updatePushToken(token)
   - Use supabase from '../lib/supabase'
   - Export useAuthStore

2. Create lib/helpers.ts with functions:
   - calculateDistance(lat1, lng1, lat2, lng2): Haversine formula, return miles
   - formatDistance(miles): Return "2.3 mi" string
   - formatEventDate(dateString): Return "Today at 9:00 PM" or "Feb 7 at 9:00 PM"
   - formatRelativeTime(dateString): Return "2m ago" or "Yesterday"
   - generateUsername(fullName): Lowercase, no spaces, add random number
   - validatePhoneNumber(phone): Must start with +, min 10 digits
   - formatPhoneNumber(phone): Format as "+1 (555) 123-4567"
   - getCommitmentScoreColor(score): green/orange/red based on score
   - getCommitmentScoreLabel(score): "Highly Reliable" etc
   - generateShortCode(): Random 8-char alphanumeric
   - debounce(func, wait): Standard debounce

3. Create lib/commitment.ts with functions:
   - updateCommitmentScore(userId, outcome): Calculate score change, update profile
   - markPlanComplete(planId): Update all participants' scores
   - getGroupLeaderboard(groupId): Return sorted array with ranks
   - checkInToPlan(planId, userId): Mark checked in

4. Create lib/notifications.ts with functions:
   - registerForPushNotifications(): Get Expo push token
   - sendPlanInviteNotification(planId, recipientIds)
   - sendPlanConfirmedNotification(planId)
   - handleNotificationResponse(response): Navigate based on type
   - setupNotificationListeners()

Use date-fns for date formatting. Use TypeScript. Add proper error handling.
```

PROMPT 3: Root Layout & Navigation

```
Set up Expo Router navigation structure:

1. Delete App.tsx

2. Create app/_layout.tsx with:
   - Import Stack from expo-router
   - Import PaperProvider from react-native-paper
   - Import useAuthStore
   - Call useAuthStore.initialize() on mount
   - Set up notification listeners
   - Register for push notifications when logged in
   - Show loading screen while initializing
   - Protect routes: redirect to /index if not logged in
   - Wrap in PaperProvider and Stack
   - Hide headers

3. Create app/(tabs)/_layout.tsx with:
   - Import Tabs from expo-router
   - 4 tabs: feed, groups, plans, profile
   - Use MaterialCommunityIcons
   - Icons: calendar-multiple, account-group, check-circle, account-circle
   - Active color #6366f1, inactive #9ca3af
   - Hide headers

Use TypeScript. Add proper cleanup for listeners.
```

PROMPT 4: Authentication Screens

```
Create all authentication screens:

1. app/index.tsx (Phone auth):
   - Title "PlanLock" + subtitle
   - Phone number input with validation
   - "Send Code" button
   - On mount: check if logged in, redirect to /(tabs)/feed if yes
   - On submit: supabase.auth.signInWithOtp, navigate to /verify with phone param
   - Use React Native Paper components
   - Modern design with #6366f1 primary color

2. app/verify.tsx (OTP verification):
   - Get phone from route params
   - 6-digit code input
   - "Verify" button
   - "Resend" link with 30s countdown
   - On verify: supabase.auth.verifyOtp
   - If profile exists: navigate to /(tabs)/feed
   - If no profile: navigate to /profile-setup
   - Use Paper components

3. app/profile-setup.tsx (First-time setup):
   - "Welcome!" heading
   - Full name input (required)
   - Username input (optional, auto-suggest)
   - "Get Started" button
   - On submit: Insert into profiles table
   - Navigate to /(tabs)/feed with router.replace
   - Use Paper components

Use expo-router navigation. Handle errors gracefully. Use TypeScript.
```

PROMPT 5: Events Feed & Detail

```
Create event discovery screens:

1. app/(tabs)/feed.tsx:
   - Fetch events from Supabase:
     - Join with venues
     - Filter: start_time > now
     - Order by start_time ASC
     - Limit 20
   - FlatList with event cards showing:
     - Cover image (200px height)
     - Title, venue name
     - Date/time (use formatEventDate)
     - Price or "Free" badge
     - Distance (use fake location 40.7589, -73.9851)
   - Pull to refresh
   - Loading skeleton
   - Empty state
   - Tap to navigate to /event/[id]
   - Use Paper Card components

2. app/event/[id].tsx:
   - Get id from route params
   - Fetch event + venue details
   - ScrollView showing:
     - Hero cover image
     - Event title and description
     - Venue name and address
     - Date & time section
     - Price info
     - "Create Plan" button → navigate to /create-plan?eventId={id}
   - Back button in header
   - Use Paper components

Use date-fns for formatting. Use TypeScript. Add error states.
```

PROMPT 6: Groups Management

```
Create group screens:

1. app/(tabs)/groups.tsx:
   - Fetch user's groups:
     - Join group_members on user_id
     - Join groups
     - Count members
   - FlatList of group cards showing:
     - Group name
     - Member count
     - Tap to navigate to /group/[id]
   - FAB (Floating Action Button) → navigate to /create-group
   - Pull to refresh
   - Empty state with "Create your first group"
   - Use Paper components

2. app/create-group.tsx:
   - Group name input (required)
   - Search members by phone
   - List selected members with remove option
   - "Create Group" button
   - On submit:
     - Insert into groups
     - Insert creator as admin in group_members
     - Insert selected members
     - Navigate back to /groups
   - Use Paper components

3. app/group/[id].tsx:
   - Fetch group + members + recent plans
   - Show:
     - Member list with commitment scores
     - Group stats (total plans, success rate)
     - Recent plans (last 5)
   - "Create Plan" FAB
   - "Leave Group" button at bottom
   - Use Paper Card components

Use expo-router. Add real-time subscriptions for group_members. Use TypeScript.
```

PROMPT 7: Plans & Voting

```
Create plan screens:

1. app/(tabs)/plans.tsx:
   - Tabs: "Upcoming" and "Past"
   - Fetch user's plans:
     - Join with events, venues, plan_participants
     - Filter by user in group
     - Upcoming: planned_date >= today, status != completed
     - Past: opposite
   - FlatList of plan cards showing:
     - Event thumbnail
     - Title, date, venue
     - Status badge (color-coded)
     - Participant count "4 of 6 voted"
     - Tap to navigate to /plan/[id]
   - Use Paper Chip for status badges

2. app/create-plan.tsx:
   - Get eventId from route params
   - Fetch event details (preview card at top)
   - Select group dropdown
   - Optional description input
   - Date/time pickers (default to event date/time)
   - Min attendees slider (2-10, default 3)
   - "Create Plan" button
   - On submit:
     - Insert into plans
     - Fetch group members
     - Insert all as plan_participants (status: pending)
     - Navigate to /plan/[planId]
   - Use Paper and DateTimePicker

3. app/plan/[id].tsx (MOST IMPORTANT SCREEN):
   - Fetch plan + event + venue + participants + messages
   - Sections:
     - Event info with cover image
     - Status badge
     - Voting buttons (Yes/Maybe/No) if status = proposed
       - Update plan_participants on vote
       - Auto-confirm if yes_count >= min_attendees
     - Confirmed banner if status = confirmed
     - Participant list showing votes and commitment scores
     - Chat section (FlatList of messages + input)
   - Real-time subscriptions:
     - plan_participants changes
     - plan_messages changes
     - plans changes
   - Use Supabase Realtime
   - Use Paper components with vote button colors:
     - Yes: #10b981
     - Maybe: #f59e0b
     - No: #ef4444

Use KeyboardAvoidingView for chat. Use TypeScript. Add animations for status changes.
```

PROMPT 8: Components & Polish

```
Create reusable components and add polish:

1. components/EventCard.tsx:
   - Reusable event card component
   - Props: event, onPress
   - Show image, title, venue, date, price
   - Use Paper Card

2. components/PlanCard.tsx:
   - Reusable plan card
   - Props: plan, onPress
   - Show status, date, venue, participants
   - Use Paper Card and Chip

3. components/ParticipantListItem.tsx:
   - Show participant with avatar, name, vote badge, score
   - Props: participant
   - Use Paper List.Item

4. components/VoteButton.tsx:
   - Large colored button for voting
   - Props: type ('yes'|'maybe'|'no'), selected, onPress
   - Show icon + label
   - Highlight if selected

5. components/LoadingSkeleton.tsx:
   - EventCardSkeleton
   - PlanCardSkeleton
   - Shimmer animation effect

6. Update app/(tabs)/profile.tsx:
   - Show user name, phone, avatar
   - Commitment score with ProgressBar
   - Total stats (attended, flaked)
   - Score label and color
   - "Sign Out" button → useAuthStore.signOut()

Use TypeScript. Use Animated API for shimmer effect. Use Paper components.
```

USAGE INSTRUCTIONS

1. **Start with Prompt 1** - Paste into Cursor Composer (Cmd/Ctrl + I)
2. Let Cursor generate all the files
3. **Fill in your Supabase credentials** in .env file
4. **Continue with Prompts 2-8** in order
5. After each prompt, let Cursor finish before pasting the next one
6. **Test frequently**: Run `npx expo start` after every 2-3 prompts

TESTING CHECKLIST

After all prompts are complete:

```
npx expo start -c
```

Test flow: 1. ✅ Sign up with phone (+15555551234, code: 123456) 2. ✅ Create profile 3. ✅ Browse events 4. ✅ Create group 5. ✅ Create plan from event 6. ✅ Vote on plan (should auto-confirm) 7. ✅ Send chat message 8. ✅ Check profile tab 9. ✅ Sign out

---

DEPLOYMENT (After Testing)

```
# Install EAS CLI
npm install -g eas-cli

# Login
eas login

# Configure
eas build:configure

# Build iOS
eas build --platform ios --profile preview

# Build Android APK
eas build --platform android --profile preview
```

TROUBLESHOOTING

Build errors: Run `npx expo start -c` to clear cache

RLS errors: Run the policy fixes from earlier in Supabase SQL Editor

Phone auth not working: Enable Phone provider in Supabase, use test code 123456

Real-time not updating: Enable Realtime in Supabase Database settings

---

That's it! You now have all 8 prompts to build the complete MVP. Start pasting them into Cursor one by one! 🚀