HANGFUL

Product Requirements Document (PRD)

&

Technical Requirements Document (TRD)

MVP Specification

Version 1.0

January 2026

CONFIDENTIAL - DO NOT SHARE

PART 1: PRODUCT REQUIREMENTS DOCUMENT

1. Executive Summary

Hangful is a social utility app designed to make hanging out with close friends effortless. Unlike event-focused platforms (Partiful, Eventbrite) or passive social media (Instagram, TikTok), Hangful occupies the unique space of daily, close-friend, plan-making that no current app adequately serves.

The MVP consists of two components: a Brand Portal (web) for businesses to create promotional deals, and a User App (iOS/Swift) for consumers to discover deals, claim them, and coordinate hangouts with friends.

1.1 Problem Statement

Gen Z is the loneliest generation, with 79% reporting loneliness and IRL time with friends down 45%. Social media apps, originally marketed for connection, now compete for attention rather than facilitating real-world interaction. Hanging out has become logistically difficult, especially post-college when schedules become complex and coordination feels formal.

1.2 Solution

Hangful reduces friction in planning casual hangouts by combining deal discovery with social coordination. Brands sponsor hangouts through promotional deals, users claim deals and invite friends, creating a flywheel where real-world connection is incentivized and facilitated.

1.3 Target Users

- Primary: Gen Z and young millennials (18-28) who want to hang out more but find coordination difficult
- Secondary: Local brands and businesses seeking authentic Gen Z engagement

2. Product Vision & Strategy

2.1 Core Value Proposition

For Users: Discover exclusive deals that give you reasons to hang out, coordinate effortlessly with friends, and get rewarded for spending time together IRL.

For Brands: Connect with Gen Z consumers through authentic, real-world experiences rather than traditional advertising.

2.2 MVP Scope

The MVP focuses on validating the core loop: Brand creates deal → User claims deal → User hangs out with friends → User uploads proof → User receives reward → User invites more friends.

2.3 Drop-Based Model

Events/deals are released in scheduled drops rather than continuous availability. Between drops, the app displays a countdown timer, creating anticipation and urgency. This model allows for controlled scaling and creates viral moments.

3. User Stories & Requirements

3.1 Brand Portal (Web)

User Stories

1. As a brand manager, I want to create promotional deals so that I can attract Gen Z customers to my business.
2. As a brand manager, I want to see how many users have redeemed my deals so that I can measure ROI.
3. As a brand manager, I want to approve/reject proof submissions so that I can ensure authentic engagement.

Functional Requirements

ID	Requirement	Priority
BR-001	Brand authentication and secure login	P0 (Must Have)
BR-002	Create deal with title, description, reward type, quantity, expiration	P0 (Must Have)
BR-003	Dashboard showing total claims, redemptions, pending approvals	P0 (Must Have)
BR-004	Review and approve/reject proof photo submissions	P0 (Must Have)
BR-005	Generate unique redemption codes for approved deals	P1 (Should Have)

3.2 User App (iOS/Swift)

User Stories

4. As a user, I want to browse available deals so that I can find reasons to hang out with friends.
5. As a user, I want to claim a deal so that I can redeem it when I hang out.
6. As a user, I want to upload proof of my hangout so that I can receive my reward.
7. As a user, I want to invite friends via a magic link so that they can join Hangful.
8. As a user, I want to receive SMS reminders so that I don't forget about claimed deals.
9. As a user, I want to request brands I want to see on the platform.

Functional Requirements

ID	Requirement	Priority
UR-001	User authentication (phone number + OTP)	P0 (Must Have)

UR-002	Display available deals in card-based feed	P0 (Must Have)
UR-003	Claim deal with single tap (limited quantity)	P0 (Must Have)
UR-004	Upload proof photo from camera or gallery	P0 (Must Have)
UR-005	View redemption code/reward after proof approval	P0 (Must Have)
UR-006	Generate and share referral magic link (unlocked after first hangout)	P0 (Must Have)
UR-007	Receive SMS reminders for claimed deals	P1 (Should Have)
UR-008	Display countdown timer between drops	P0 (Must Have)
UR-009	Request a brand feature (submit brand suggestions)	P1 (Should Have)
UR-010	Scratch-off lottery ticket animation for one-time reward codes	P2 (Nice to Have)

4. Success Metrics

- User Activation Rate: % of signups who complete first hangout within 7 days (Target: 40%)
- Referral Rate: % of users who send at least one invite link (Target: 60%)
- Claim-to-Redemption Rate: % of claimed deals that result in proof submission (Target: 50%)
- Brand Satisfaction: NPS score from partner brands (Target: 50+)
- Weekly Active Users (WAU) Growth: Week-over-week growth (Target: 15%)

PART 2: TECHNICAL REQUIREMENTS DOCUMENT

5. System Architecture

5.1 High-Level Architecture

The system follows a client-server architecture with the following components:

- iOS User App (Swift/SwiftUI) - Native mobile application
- Brand Portal (React/Next.js) - Web-based admin dashboard
- Backend API (Node.js/Express or Python/FastAPI) - RESTful API server
- Database (PostgreSQL) - Primary data store
- Object Storage (AWS S3/Cloudflare R2) - Image storage for proof uploads
- SMS Service (Twilio) - OTP and reminder notifications
- Push Notifications (APNs) - iOS push notifications

5.2 Technology Stack

Layer	Technology	Rationale
iOS App	Swift 5.9, SwiftUI, Combine	Modern iOS, reactive patterns
Brand Portal	Next.js 14, React, TailwindCSS	SSR, fast iteration, modern DX
Backend API	Node.js/Express or FastAPI	Rapid development, async support
Database	PostgreSQL (Supabase/Neon)	Relational integrity, JSON support
Authentication	JWT + Twilio Verify	Secure, phone-based auth
Image Storage	AWS S3 / Cloudflare R2	Scalable, cost-effective
Hosting	Vercel (web) + Railway/Render (API)	Simple deployment, auto-scaling

6. Data Model

6.1 Entity Relationship Diagram (Conceptual)

The core entities and their relationships are:

Users Table

- id (UUID, PK)
- phone_number (VARCHAR, unique, indexed)
- display_name (VARCHAR)
- referral_code (VARCHAR, unique)
- referred_by (UUID, FK to users)

- first_hangout_completed (BOOLEAN, default false)
- created_at, updated_at (TIMESTAMP)

Brands Table

- id (UUID, PK)
- name (VARCHAR)
- email (VARCHAR, unique)
- password_hash (VARCHAR)
- logo_url (VARCHAR)
- created_at, updated_at (TIMESTAMP)

Deals Table

- id (UUID, PK)
- brand_id (UUID, FK to brands)
- title (VARCHAR)
- description (TEXT)
- reward_type (ENUM: discount, freebie, credit)
- reward_value (VARCHAR)
- total_quantity (INT)
- claimed_quantity (INT, default 0)
- drop_id (UUID, FK to drops)
- expires_at (TIMESTAMP)
- created_at, updated_at (TIMESTAMP)

Drops Table

- id (UUID, PK)
- name (VARCHAR)
- starts_at (TIMESTAMP)
- ends_at (TIMESTAMP)
- is_active (BOOLEAN)

Claims Table

- id (UUID, PK)
- user_id (UUID, FK to users)
- deal_id (UUID, FK to deals)
- status (ENUM: claimed, proof_submitted, approved, rejected, redeemed)
- proof_image_url (VARCHAR, nullable)
- redemption_code (VARCHAR, nullable)
- claimed_at, submitted_at, approved_at, redeemed_at (TIMESTAMP)

Brand Requests Table

- id (UUID, PK)
- user_id (UUID, FK to users)
- brand_name (VARCHAR)
- created_at (TIMESTAMP)

7. API Specification

7.1 Authentication Endpoints

POST /api/auth/request-otp

Request body: { phone_number: string }

Response: { success: boolean, message: string }

Sends OTP via Twilio Verify to the provided phone number.

POST /api/auth/verify-otp

Request body: { phone_number: string, code: string }

Response: { access_token: string, refresh_token: string, user: User }

Verifies OTP and returns JWT tokens. Creates user if new.

7.2 Deal Endpoints

GET /api/deals

Headers: Authorization: Bearer <token>

Response: { deals: Deal[], next_drop: Drop | null }

Returns active deals for current drop, or next drop countdown if between drops.

POST /api/deals/:id/claim

Headers: Authorization: Bearer <token>

Response: { claim: Claim }

Claims a deal for the authenticated user. Returns error if sold out or already claimed.

7.3 Proof Endpoints

POST /api/claims/:id/proof

Headers: Authorization: Bearer <token>, Content-Type: multipart/form-data

Body: image file

Response: { claim: Claim }

Uploads proof image to S3/R2, updates claim status to proof_submitted.

GET /api/claims/:id/status

Headers: Authorization: Bearer <token>

Response: { claim: Claim }

Returns current claim status and redemption code if approved.

7.4 Referral Endpoints

GET /api/users/referral-link

Headers: Authorization: Bearer <token>

Response: { referral_link: string | null, eligible: boolean }

Returns referral link if user has completed first hangout.

POST /api/users/apply-referral

Request body: { referral_code: string }

Response: { success: boolean }

Applies referral code during onboarding.

7.5 Brand Portal Endpoints

POST /api/brand/deals

Creates a new deal (brand auth required).

GET /api/brand/deals/:id/claims

Returns all claims for a deal with proof images.

POST /api/brand/claims/:id/approve

Approves a proof submission, generates redemption code.

POST /api/brand/claims/:id/reject

Rejects a proof submission with optional reason.

8. iOS App Architecture

8.1 Project Structure (MVVM)

The iOS app follows MVVM architecture with the following structure:

- Models/ - Data models (User, Deal, Claim, Drop)
- Views/ - SwiftUI views organized by feature
- ViewModels/ - ObservableObject classes for state management
- Services/ - API client, auth manager, image uploader
- Utilities/ - Extensions, helpers, constants

8.2 Key Views

- AuthView - Phone number entry and OTP verification
- DealsView - Card-based feed of available deals or countdown timer
- DealDetailView - Full deal info with claim button
- MyClaimsView - List of user's claimed deals and their status
- ProofUploadView - Camera/gallery picker with preview
- RedemptionView - Shows redemption code after approval
- ReferralView - Share magic link with friends
- RequestBrandView - Form to suggest brands
- CountdownView - Timer displayed between drops
- ScratchCardView - Lottery ticket animation for rewards (P2)

8.3 Key Dependencies

- Alamofire or URLSession - Network requests
- KeychainSwift - Secure token storage
- SDWebImageSwiftUI - Async image loading

9. Security Considerations

- All API endpoints require JWT authentication except auth routes
- Tokens stored in iOS Keychain, not UserDefaults
- Rate limiting on OTP requests (max 3 per phone per 10 min)
- Image uploads scanned for inappropriate content (future: AWS Rekognition)
- Redemption codes are one-time use and cryptographically random
- Brand passwords hashed with bcrypt (cost factor 12)
- HTTPS enforced on all endpoints

PART 3: LMU CMSI CURRICULUM MAPPING

10. Technical Concepts & Related LMU Courses

This section maps the technical components of Hangful to courses in the LMU Computer Science (CMSI) curriculum.

10.1 Database Systems

Direct application to Hangful:

- Relational schema design (Users, Brands, Deals, Claims, Drops tables)
- Foreign key relationships and referential integrity
- Indexing strategies (phone_number, referral_code for fast lookups)
- Transaction handling for atomic claim operations (prevent overselling)
- Query optimization for deal feeds and analytics dashboards
- PostgreSQL-specific features (ENUM types, JSONB for flexible metadata)

10.2 Programming Languages

Direct application to Hangful:

- Swift language features (optionals, closures, protocols)
- TypeScript for type-safe frontend development
- Async/await patterns in both Swift and JavaScript/TypeScript
- Functional programming concepts (map, filter, reduce in data transforms)

10.3 Web Application Development

Direct application to Hangful:

- React component architecture for Brand Portal
- Next.js SSR and API routes
- RESTful API design and implementation
- State management (React hooks, context)
- Responsive design with TailwindCSS

10.4 Computer Networks

Direct application to Hangful:

- HTTP/HTTPS protocols for API communication
- JWT token transmission and security
- CDN usage for image delivery (S3 + CloudFront)
- DNS configuration for custom domains
- SSL/TLS certificate management

10.5 Algorithms

Direct application to Hangful:

- Efficient data structures for deal sorting and filtering

- Hash-based lookup for redemption code validation
- Time complexity analysis for API endpoints

10.6 Data Structures

Direct application to Hangful:

- Arrays and dictionaries for API responses
- Queue structures for SMS reminder scheduling
- Tree structures for referral chain tracking (future analytics)

10.7 Operating Systems

Direct application to Hangful:

- Concurrency handling for simultaneous deal claims
- Process/thread management in Node.js event loop
- Memory management in iOS app lifecycle
- File system interactions for image handling