Product Requirements Document

JidouNavi 自販機ナビ

Discover Japan, one vending machine at a time.

Document Name	JidouNavi: Crowdsourced Vending Machine Discovery App
Version	1.0 (Initial Draft)
Status	TBD
Product Owners	Leandro Trabucco & Matias Fernandez
Target Release	Q1 2026 (Beta Launch)
Platform	iOS & Android

1. Goal and Objectives (The "Why")

1.1 Objective Statement

To become the definitive discovery platform for Japan's unique vending machines by providing a gamified, crowdsourced map experience that helps tourists, collectors, and locals find, document, and share rare vending machines — while rewarding their contributions through badges, missions, and leaderboards that turn everyday exploration into an adventure.

1.2 Problem Statement

Japan has over 5 million vending machines, but the rare, themed, or bizarre ones are nearly impossible to find intentionally. Tourists discover them through TikTok and Instagram, but location info is buried in comments, impossible to search, and often outdated. There's no way to see what's actually near you right now. Existing solutions like Sagasoda.com catalog drinks but lack the discovery and gamification that would make exploration feel rewarding.

1.3 Target Users

- **Tourists**: Want unique content for TikTok/Instagram, something different from temples and ramen shops. The "I found the weirdest vending machine in Japan" content performs.
- **Long-term expats/foreign residents**: Already past the tourist phase, looking for new ways to explore. More likely to contribute content since they're here long-term and speak some Japanese.
- **Gachapon hunters:** But they care about *what's inside*, not the machine itself. Might use the app to find specific gachapon spots.
- **Japanese locals**: Probably smaller segment unless the app goes viral domestically. Most Japanese people walk past vending machines without thinking twice.

1.4 Business Value & Strategic Positioning

- **Portfolio Value:** Full-stack mobile app (React Native + Maps + Backend) demonstrates shippable product skills. Downloadable by recruiters — not just another GitHub repo.
- **Market Context:** Japan tourism is booming (31.9M visitors in 2023). "Weird Japan" content performs well on social media. Vending machines are a known tourist curiosity with no dominant discovery app.
- **Differentiation:** Sagasoda.com exists but focuses only on drinks and feels like a database. JidouNavi focuses on discovery and the weird/rare finds, with gamification to drive engagement. "Pokemon Go for vending machines" is a memorable pitch.
- **Competitive Advantage:** Crowdsourced data creates a moat over time — more users = more machines = better product. First mover in a small but real niche.
- **Realistic Monetization (if traction happens):** Start with free app, maybe small ads or tip jar. Premium features only if user base justifies it. Brand/tourism partnerships are year 2+ conversations, not launch plans.
- **Downside Protection:** Even if the app doesn't take off, we walk away with: shipped product, management artifacts, and collaboration experience. The project pays off either way.

2. Success Metrics (The "What")

The success of JidouNavi will be measured by the following Key Performance Indicators (KPIs):

Metric	Target	Notes
Seeded Content (Pre-Launch)	50-100 vending machines documented with photos, locations, and tags before beta launch.	Critical for overcoming chicken-egg problem.
User Acquisition (Beta)	>500 downloads within first 30 days of beta launch.	Measures initial market interest. Focus on Tokyo-based tourists and expat communities.
Content Contribution Rate	>2% of WAU add at least 1 new machine per week	Contributions are rare and valuable, don't expect most users to add, most will visit.
Visit/Stamp Rate	>50% of MAU "collect" (visit + verify) at least 1 machine per month	Core engagement loop. This is what keeps users coming back.
Verification Activity	>30% of machines verified (still exists) within 90 days of submission	Measures data freshness and community trust.
Badge Unlock Rate	Average user unlocks 2+ badges within first week	Measures gamification engagement. Are people playing the game?
Session Frequency	3+ sessions per active week	Are they opening the app multiple times during their exploration period?
Week 2 Retention	>40% of users active in week 1 return in week 2	For tourists, this is their whole trip. Did they keep using it?
Share Rate	>10% of visits result in share/screenshot	Organic growth. Are people posting finds?

3. Product Scope and Requirements (The "How")

3.1 Minimum Viable Product (MVP) Scope

The MVP will focus on core discovery and contribution functionality to validate market fit before adding advanced gamification.

IN SCOPE (MVP)	OUT OF SCOPE (Future)
• Interactive map with vending machine pins • Add machine: photo, GPS location, tags, description • Category filters (Food, Drinks, Gachapon, Weird, Retro, etc.) • User authentication (email + social) • Basic user profiles with contribution count • Search by location, category, or keyword • Machine detail view with photos and info • "Nearby" discovery based on current location • Badges and achievement system • Verification prompt ("Still there?" Yes/No) • Save/bookmark machines • "I visited this" check-in/stamp functionality	• Leveling/XP system • "Weirdest this week" trending section • Social features (follow users, likes) • AI-powered scavenger hunts/quests • AR camera overlay for directions • Brand partnership integrations • Machine status updates (restocked, broken)

3.2 Functional Requirements (User Stories)

These are the core user flows and capabilities required for the MVP:

ID	User Story	Acceptance Criteria
FR-01	As a user, I want to see a map of vending machines near me so I can discover interesting machines in my area.	Map loads with user's current location centered. Machine pins are visible within the viewport. Tapping a pin shows a preview card.
FR-02	As a user, I want to filter machines by category (Food, Drinks, Gachapon, Weird, Retro) so I can find specific types.	Filter UI is accessible from map view. Selecting a filter updates visible pins in real-time. Multiple filters can be combined.
FR-03	As a user, I want to add a new vending machine I've discovered so I can share it with the community.	"Add Machine" button triggers camera/gallery. Location auto-detects via GPS. User can add tags and description. Submission creates new pin on map.
FR-04	As a user, I want to view detailed information about a machine so I can decide if I want to visit it.	Detail view shows: all photos, exact address, category tags, description, contributor info, distance from current location.
FR-05	As a user, I want to search for machines by keyword or location so I can find specific machines.	Search bar accepts text input. Results show matching machines by name, tag, or location. Tapping a result navigates to that machine.
FR-06	As a user, I want to create an account so I can track my contributions and save favorites.	Sign up with email or social auth (Google, Apple). Profile page shows username, contribution count, and saved machines.
FR-07	As a user, I want to save/bookmark machines so I can remember ones I want to visit.	"Save" button on machine detail view. Saved machines appear in profile under "My Saved" section.
FR-08	As a user, I want to get directions to a machine so I can navigate there easily.	"Get Directions" button opens native maps app (Google Maps/Apple Maps) with destination coordinates.
FR-09	As a user, I want to check in when I visit a machine so I can collect it and track my discoveries.	"I visited" button on machine detail view. Check-in records timestamp and adds to user's "Visited" collection. Machine shows total visit count.
FR-10	As a user, I want to verify if a machine still exists so I can help keep data accurate.	After check-in, prompt asks "Is this machine still here? Yes/No". Verification updates "last verified" timestamp. Multiple "No" responses flag machine for review.
FR-11	As a user, I want to earn badges for my activity so I feel rewarded for exploring and contributing.	Badges unlock automatically based on triggers (first check-in, 5 visits, first contribution, etc.). Badge popup notification on unlock. Badges display on profile.

3.3 Non-Functional Requirements (NFRs)

Requirement Type	Description
Performance	Map must load within 3 seconds on standard mobile connection (4G). Machine pins within viewport must render within 500ms of map movement.
Reliability	App doesn't crash on common actions. Photo upload retries once on failure. Basic error messages when something breaks.
Usability	Adding a machine: max 5 taps. Check-in: max 2 taps. Works on iPhone and Android.
Localization	English only for MVP. Unicode support for machine names/descriptions (so Japanese text doesn't break).
Security	User authentication via secure OAuth 2.0. All API calls over HTTPS. User location data not shared with third parties without consent.
Scalability	Works for 100-500 machines and ~50 concurrent users. Supabase free tier limits are the ceiling for now.
Image Handling	Compress photos before upload (max 1-2MB). Saves storage costs, faster loads.
Analytics	Basic tracking: app opens, machines added, check-ins. Know if anyone's using it. (Supabase or free tier Mixpanel)
App Store Compliance	Privacy policy page (required for Stores). App doesn't crash on reviewer's device.

4. User Experience (UX) and Design

4.1 Key User Flows

Flow A: Discover a Machine

1. User opens app (lands on Map view by default)
2. User sees pins representing nearby vending machines
3. User taps a pin to see preview card (photo thumbnail, name, distance)
4. User taps preview to open full detail view
5. User taps "Get Directions" to navigate

Flow B: Add a New Machine

1. User taps "+" floating action button
2. Camera opens (or gallery picker)
3. User takes photo of vending machine
4. App auto-detects GPS location (user can adjust pin manually)
5. User selects category tags from predefined list
6. User adds optional description
7. User taps "Submit" — machine appears on map

Flow C: Check-in / Visit a Machine

1. User is at a vending machine location
2. User opens machine detail view
3. User taps "I visited this" button
4. App confirms GPS proximity (optional, or trust-based for MVP)
5. Check-in recorded, badge popup if unlocked
6. Prompt: "Is this machine still here? Yes / No"

Flow D: Collect a Badge

1. User completes trigger action (first visit, 5th check-in, first contribution, etc.)
2. Badge popup appears with animation
3. User taps to dismiss or "View all badges"
4. Badge now visible on profile

4.2 Design Direction

The visual style should be "modern functionality with retro-pixel accents" — keeping core UX clean and accessible while using pixel art for branding, icons, badges, and empty states to reinforce the playful, game-like discovery experience.

Color Palette

Role	Color	Usage
Primary	#FF4B4B	Vending machine red — CTAs, key actions, branding
Secondary	#3C91E6	Retro screen blue — links, secondary actions
Accent	#FFB7CE	Candy pink — highlights, badges, playful elements
Background	#FDF3E7	Creamy vintage white — main background
Dark	#2B2B2B	Softer black — text, headers
Highlight	#FFD966	Warm yellow — coin icons, achievements

Typography

- **Logo/Headers:** Press Start 2P or VT323 (pixel/arcade style)
- **Body/UI:** Inter (clean, readable)

Navigation Structure

Bottom tab navigation with 3-4 primary destinations:

- **Map** — explore, search, filters
- **Profile** — visited, saved, badges, stats, settings
- **(+) FAB** — add machine (floating button, always visible)

5. Technical Specification

5.1 Mobile Framework

5.1.1 Options Overview

Framework	Language	Codebase	Learning Curve	Community
React Native + Expo	JavaScript / TypeScript	Single	Low	Very Large
Flutter	Dart	Single	Medium	Large
Native (Kotlin + Swift)	Kotlin / Swift	Two (separate)	High	Very Large
Hotwire Native	Ruby / HTML	Hybrid	Low	Small

5.1.2 React Native + Expo

Overview: JavaScript-based framework maintained by Meta. Expo is a managed workflow that simplifies builds, deployments, and native API access without requiring Xcode or Android Studio.

Advantages:
- JavaScript/TypeScript foundation aligns with bootcamp training
- Single codebase deploys to both iOS and Android
- Expo provides OTA updates, push notifications, simplified app store submissions
- Largest ecosystem of third-party libraries and tutorials
- Hot reload enables rapid iteration
- Strong job market relevance

Disadvantages:
- Performance slightly below true native for complex animations
- Some native modules require "ejecting" from Expo
- Debugging native issues can be challenging
- App bundle size larger than native

Map/GPS Support: Excellent. react-native-maps and react-native-mapbox-gl are mature libraries.

Risk: LOW

5.1.3 Flutter

Overview: Google's UI toolkit using Dart. Compiles to native ARM code. Customizable UI.

Advantages:
- Excellent performance, near-native speed
- Consistent UI across platforms
- Strong documentation
- Hot reload

Disadvantages:
- Dart is a new language (neither team member knows it)
- Smaller job market in Japan
- Google's abandonment history creates uncertainty
- Fewer third-party libraries

Map/GPS Support: Good. google_maps_flutter and flutter_map available.

Risk: MEDIUM — Learning Dart adds 2-4 weeks.

5.1.4 Native (Kotlin + Swift)

Overview: Separate native apps for Android (Kotlin) and iOS (Swift). Industry standard at scale.

Advantages:

- Best possible performance and UX
- Full access to all platform APIs
- Highly valued on resumes
- Smaller app sizes

Disadvantages:

- Two separate codebases
- Two languages to learn
- Development time ~2x
- Requires Mac for iOS

Map/GPS Support: Excellent. First-party SDKs.

Risk: HIGH — Not recommended for MVP timeline.

5.1.5 Hotwire Native

Overview: 37signals framework that wraps Rails app in native shell. Minimal native code required.

Advantages:

- Leverages existing Rails knowledge
- Server-rendered, logic stays in Ruby
- Fast server-side iteration

Disadvantages:

- Very new, limited community
- Not designed for map-heavy apps
- Native features require bridging
- Small job market

Map/GPS Support: Poor. Requires native bridging or WebView maps.

Risk: HIGH — Framework immaturity + map requirements = bad fit.

5.1.6 Framework Decision Matrix

Criteria	Weight	React Native	Flutter	Native	Hotwire
Time to MVP	High	**Fast**	**Medium**	**Slow**	**Medium**
Team Experience	High	**JS known**	**Dart new**	**New langs**	**Rails known**
Map/GPS Support	High	**Excellent**	**Good**	**Excellent**	**Poor**
Community	Medium	**Very Large**	**Large**	**Large**	**Small**
Job Market	Medium	**High**	**Medium**	**Very High**	**Low**

Recommendation: React Native + Expo
Best balance of speed, skill alignment, and capability. Flutter is an acceptable alternative if the team prefers.

5.2 Maps Provider

Criteria	Mapbox	Google Maps
Free Tier	50,000 map loads/month	$200 credit/month (~28,000 loads)
After Free Tier	$5 per 1,000 loads	$7 per 1,000 loads
Customization	High (custom styles, pixel aesthetic)	Limited styling
React Native Support	Good (react-native-mapbox-gl)	Excellent (react-native-maps)
User Familiarity	Less familiar	Very familiar

Recommendation: Mapbox
Better free tier, cheaper at scale, customizable for pixel aesthetic. Google Maps is easy fallback if needed.

5.3 Backend / Database

Criteria	Supabase	Firebase
Database Type	PostgreSQL (SQL, relational)	Firestore (NoSQL, document)
Query Language	SQL (familiar from Rails)	Custom SDK queries (new)
Free Tier	500MB DB, 1GB storage	1GB storage, 10GB bandwidth
Auth	Built-in	Built-in (more options)
Open Source	Yes (can self-host)	No (Google lock-in)

Recommendation: Supabase
Data is relational (users → machines → photos → visits). SQL knowledge from bootcamp transfers directly.

5.4 Additional Decisions

5.4.1 Image Storage

Recommended: Supabase Storage / Simplicity, same dashboard. If need Cloudinary later

5.4.2 State Management

Recommendation: Zustand — Simple, clean, performant. Redux is overkill.

5.5 Recommended Stack Summary

Layer	Choice	Key Reason
Mobile Framework	React Native + Expo	JS skills, single codebase
Maps	Mapbox	Better free tier, customizable
Backend / DB	Supabase	PostgreSQL, SQL transfers
Auth	Supabase Auth	Built-in, one less service
Image Storage	Supabase Storage	Simplicity
State Management	Zustand	Simple, clean

Estimated Costs:

- Supabase: Free tier
- Mapbox: Free tier (50k loads/month)
- Expo: Free
- Apple Developer: $99/year
- Google Play: $25 one-time **Total upfront: ~$125**

5.6 Data Model (Core Entities) *— TBD These are pure reference and not definitive —*

Users Table

Field	Type	Description
id	UUID	Primary key
email	String	User email (unique)
username	String	Display name
avatar_url	String	Profile picture URL
contribution_count	Integer	Number of machines added
created_at	Timestamp	Account creation date

Machines Table

Field	Type	Description
id	UUID	Primary key
latitude	Float	GPS latitude
longitude	Float	GPS longitude
name	String	Machine name/title (optional)
description	Text	User-provided description
category_tags	Array[String]	Tags food, drinks, gachapon, weird, retro etc.
contributor_id	UUID (FK)	Reference to Users table
verified	Boolean	Community verification status
visit_count	Integer	Total check-ins (denormalized for performance)
last_verified_at	Timestamp	Last time someone confirmed it exists
created_at	Timestamp	Submission date

Machine_Photos Table

Field	Type	Description
id	UUID	Primary key
machine_id	UUID (FK)	Reference to Machines table
photo_url	String	CDN URL for image
thumbnail_url	String	Resized thumbnail URL
uploaded_by	UUID (FK)	Reference to Users table
created_at	Timestamp	Upload date

Visits Table *(for check-ins/stamps)*

Field	Type	Description
id	UUID	Primary key
user_id	UUID (FK)	Reference to Users table
machine_id	UUID (FK)	Reference to Machines table
visited_at	Timestamp	When user checked in
still_exists	Boolean	User verified machine is still there (null if NA)

Saved_Machines Table *(bookmarks)*

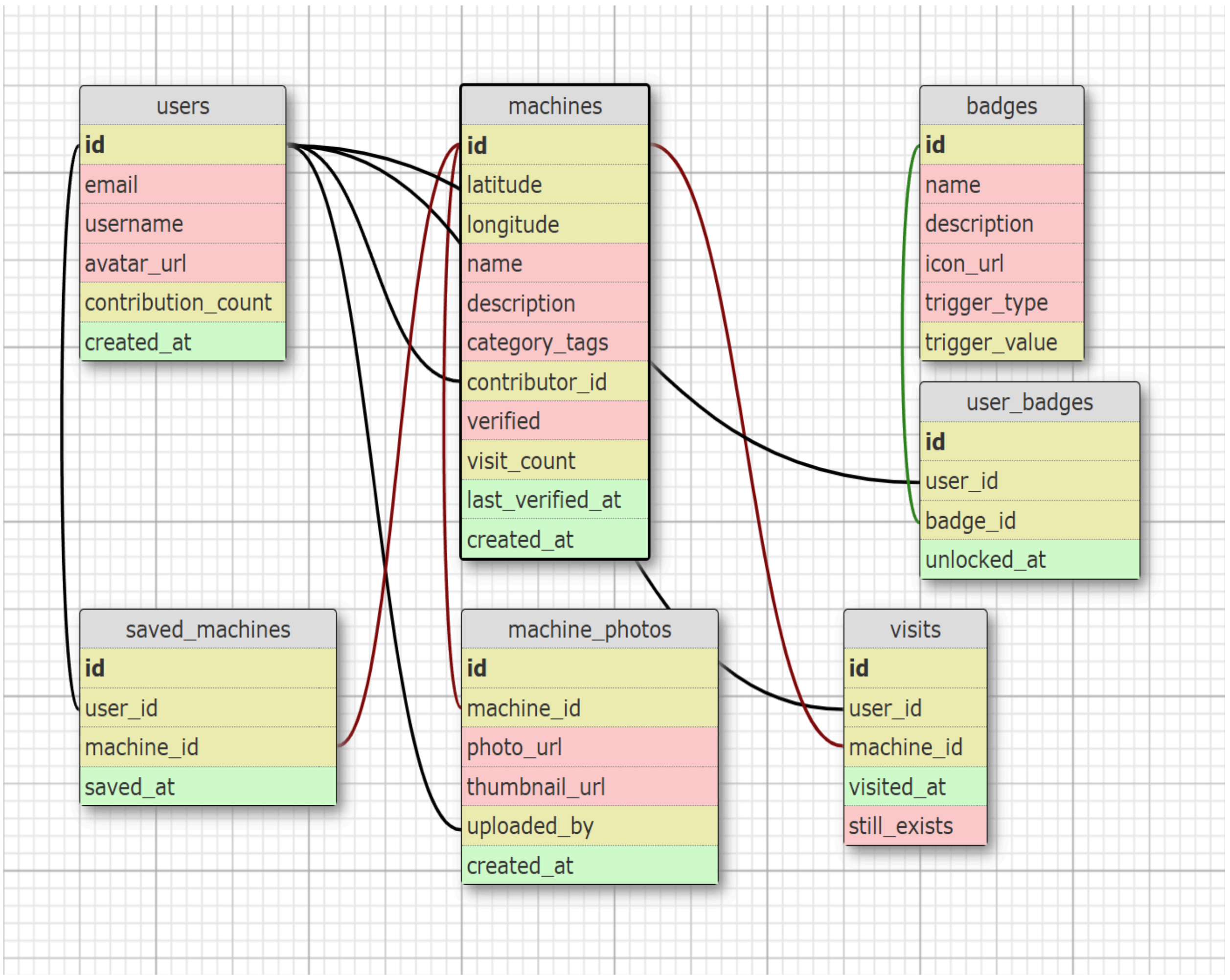
Field	Type	Description
id	UUID	Primary key
user_id	UUID (FK)	Reference to Users table
machine_id	UUID (FK)	Reference to Machines table
saved_at	Timestamp	When user bookmarked

Badges Table *(badge definitions)*

Field	Type	Description
id	UUID	Primary key
name	String	Badge name (e.g., "First Find")
description	String	How to earn it
icon_url	String	Badge image
trigger_type	String	What unlocks it (first_visit, visit_count, first_contribution, etc.)
trigger_value	String	Threshold (e.g., 5 for "visit 5 machines")

User_Badges Table *(earned badges)*

Field	Type	Description
id	UUID	Primary key
user_id	UUID (FK)	Reference to Users table
badge_id	UUID (FK)	Reference to Badges table
unlocked_at	Timestamp	When earned



5.7 API Endpoints (Core)

Method	Endpoint	Description
GET	/machines	Get machines within bounding box (lat/lng params). Supports category filter.
GET	/machines/:id	Get single machine details with photos.
POST	/machines	Create new machine entry. Requires auth.
POST	/machines/:id/photos	Upload additional photo to existing machine.
POST	/machines/:id/visit	Check in to a machine. Records visit, prompts verification.
GET	/users/:id	Get user profile and contribution stats.
GET	/users/:id/visits	Get user's visited machines (stamp collection).
POST	/users/:id/saved	Save/bookmark a machine.
DELETE	/users/:id/saved/:machine_id	Remove bookmark.
GET	/users/:id/saved	Get user's saved machines.
GET	/users/:id/badges	Get user's earned badges.
GET	/badges	List all available badges (for "locked" display).
GET	/search	Full-text search across machine names, descriptions, tags.

6. Assumptions, Dependencies, and Risks

6.1 Key Assumptions

- **Content Seeding:** Founders will manually seed 30+ machines in one area before launch.
- **Map API Costs:** Mapbox free tier (50,000 loads/month) sufficient for beta.
- **User Motivation:** Users will contribute for intrinsic rewards before gamification.
- **React Native Capability:** Both team members can ramp up. JS experience transfers.
- **No Moderation at MVP:** Community will be small and trusted.

6.2 Dependencies

- **Design Assets:** Pixel art icons and badges (itch.io, Creative Commons).
- **Map Provider:** Final decision on Mapbox vs Google Maps before dev starts.
- **App Store Accounts:** Apple Developer ($99/year), Google Play ($25 one-time).
- **Seeding Trips:** 2-3 photography trips to Akihabara + Sagamihara retro park.

6.3 Critical Risks

Risk	Description	Mitigation
Cold Start / Low Density	100 pins spread across Tokyo = feels empty. User opens app in Shinjuku, nearest pin is in Ueno → they close app.	HIGH-DENSITY CLUSTERS. Seed 30+ machines in ONE area (Akihabara). User can hit 5 spots in 15 minutes.
Data Staleness	Machines change inventory rapidly. "Weird" machine today → Coca-Cola next month.	"Still there?" prompt on check-in. Rely on community verification.

Tourist Churn	Primary users have 1-2 week lifecycle. Can't rely on long-term retention.	Focus on sharing/virality. Shareable badge cards drive acquisition to replace churned users.
Abuse / Bad Data	Duplicate machines, joke submissions, wrong locations.	Distance threshold for submissions. Soft duplicate detection (±50m = warning).
Image Uploads	Mobile photos are 5MB+	Compress client-side BEFORE upload (expo-image-manipulator) Max 1200px, <500KB Supabase limits
Gamification Trap	Don't let users chase XP instead of exploring	3-5 badges max. World is the reward not the UI

7. Development Tickets

7.1 Ticket Overview

ID	Ticket Name	Priority	Type	Source
JN-001	Project Setup: React Native + Expo Init	Critical	Setup	—
JN-002	Setup Supabase Project (DB, Auth, Storage)	Critical	Setup	—
JN-003	Implement Map View with Mapbox/Google Maps	Critical	Feature	FR-01
JN-004	Display Machine Pins on Map	Critical	Feature	FR-01
JN-005	Implement Machine Preview Card (on pin tap)	High	Feature	FR-01
JN-006	Build Machine Detail Screen	High	Feature	FR-04
JN-007	Implement Category Filter UI	High	Feature	FR-02
JN-008	Build Add Machine Flow (Camera + Form)	Critical	Feature	FR-03
JN-009	Implement GPS Auto-Detection for New Machines	High	Feature	FR-03
JN-010	Create Machines API Endpoint (GET /machines)	Critical	Backend	FR-01
JN-011	Create Machine Submission API (POST /machines)	Critical	Backend	FR-03
JN-012	Implement Image Upload to Supabase Storage	High	Backend	FR-03
JN-013	Implement User Authentication (Email + OAuth)	High	Feature	FR-06
JN-014	Build User Profile Screen	Medium	Feature	FR-06
JN-015	Implement Search Functionality	Medium	Feature	FR-05
JN-016	Implement Save/Bookmark Feature	Medium	Feature	FR-07
JN-017	Implement "Get Directions" (Open Native Maps)	Medium	Feature	FR-08
JN-018	Build Bottom Tab Navigation	High	Feature	UX
JN-019	Performance Audit: Map Load Time < 3s	High	Tech	NFR-01
JN-020	Seed Database with Initial 50+ Machines	Critical	Content	—
JN-021	Implement Check-in / "I Visited" Flow	High	Feature	FR-09

JN-022	Create Visits API (POST /machines/:id/visit)	High	Backend	FR-09
JN-023	Implement Verification Prompt ("Still there?")	Medium	Feature	FR-10
JN-024	Build Badges System (definitions + unlock logic)	High	Feature	FR-11
JN-025	Create Badges API Endpoints	Medium	Backend	FR-11
JN-026	Display Badges on Profile	Medium	Feature	FR-11
JN-027	Badge Unlock Popup Animation	Low	Feature	FR-11

8. Proposed Timeline

Estimated timeline assuming 2 developers working part-time (~20 hrs/week each).

Phase	Duration	Deliverables
Phase 0	Week 1	Tech stack finalized, repo setup, Supabase project, Expo init.
Phase 1	Weeks 2-4	Core map view, pins displaying, machine detail, basic navigation.
Phase 2	Weeks 5-7	Add machine flow, image upload, category filters, search.
Phase 3	Weeks 8-9	User auth, profiles, saved machines, "Get Directions".
Phase 4	Weeks 10-11	Check-in flow, verification prompt, badges system, profile badges.
Phase 5	Week 12	Polish, bug fixes, performance optimization, UI refinement.
Seeding	Ongoing	Photography trips to seed 50-100 machines (parallel with dev).
Beta Launch	Week 13	TestFlight (iOS) + Internal Testing (Android). Invite beta users.

— End of Document —