Voice-Optimized PRD Template for Genesis/Claude Code V2

Designed for Wispr Flow dictation - speak naturally for ~10 minutes per project TRUE METHOD: Manual fresh-session iteration for zero context rot

Phase 1: VOICE CAPTURE (Your Part - ~10 min dictation)

Start Recording and Speak Through These Prompts:

PROJECT IDENTITY

- "The project name is..."
- "This is for..." (business/personal/client)
- "The main reason this exists is..."
- "Success looks like..."

THE PROBLEM & CONTEXT

- "The current situation is..."
- "The pain point we're solving is..."
- "Without this, what happens is..."
- "Users/stakeholders are currently..."

THE SOLUTION (High Level)

- "What we're building is..."
- "The core features are..."
- "Users will be able to..."
- "The expected workflow is..."

TECHNICAL LANDSCAPE

- "The stack/frameworks I want are..."
- "Existing codebase context..." (if applicable)
- "Integration points are..."
- "Dependencies include..."
- "File structure should be..."

EDGE CASES & CONSTRAINTS

- "What happens when..."
- "We need to handle..."
- "It should NOT do..."
- "Performance requirements..."
- "Scale considerations..."

USER INTERACTIONS

- "The main user journey is..."
- "Error scenarios look like..."
- "Mobile/desktop considerations..."
- "Authentication/permissions..."

SUCCESS METRICS

- "We know it's done when..."
- "Testing criteria include..."
- "Quality bars are..."

Phase 2: CLARIFICATION QUESTIONS (Claude's Part)

Claude will systematically extract these from your voice capture and ask what's missing

ARCHITECTURAL CLARITY

```
1. Repository Structure
   - Where does this code live?
   - New repo or existing? If existing, which path?
   - What's the file/folder naming convention?

2. Dependencies & Environment
   - Package manager? (npm, pip, etc.)
   - Environment variables needed?
   - API keys/credentials required?
   - Database connections?

3. External Integrations
   - Third-party services?
   - API endpoints to consume?
   - Webhooks to configure?
```

IMPLEMENTATION SPECIFICS

```
4. Data Flow
   - Where does data come from?
   - Where does it go?
   - What's the transformation logic?
   - Caching strategy?

5. UI/UX Details (if applicable)
   - Component library? (Shadcn, etc.)
   - Styling approach? (Tailwind, CSS modules, etc.)
   - Responsive breakpoints?
   - Accessibility requirements?

6. Error Handling
   - What happens on failure?
   - Retry logic?
   - User feedback mechanisms?
   - Logging strategy?
```

OPERATIONAL REQUIREMENTS

```
7. Testing Strategy
   - Unit tests required?
   - Integration test scenarios?
   - Manual testing checklist?
```

```
8. Documentation
   - README updates needed?
   - API docs required?
   - Comments level expected?

9. Deployment
   - How will this be deployed?
   - Build steps?
   - Environment-specific configs?
```

CRITICAL GAPS

```
10. Assumptions Check
    - "You mentioned X, but didn't specify Y. Should I assume...?"

11. Scope Boundaries
    - "Is [feature] in scope for this PRD or future?"

12. Priority Order
    - "If these conflict, which takes priority: performance vs features vs timeline?"
```

Phase 3: AGENT-READY OUTPUT FORMAT (ATOMIC STORIES FOR TRUE METHOD)

Claude transforms the above into this structure for Genesis/Claude Code CRITICAL: Each user story MUST complete in ONE fresh session

```json
{
  "project": {
    "name": "string",
    "type": "new|enhancement",
    "businessContext": "string",
    "primaryGoal": "string"
  },

  "technicalSpec": {
    "targetPath": "string",
    "stack": ["array of technologies"],
    "dependencies": {
      "required": ["must-haves"],
      "optional": ["nice-to-haves"]
    },
    "environment": {
      "variables": ["ENV_VAR_NAME"],
      "credentials": ["SERVICE_API_KEY"]
    }
  },

  "userStories": [
    {
      "id": "US-001",
      "title": "ATOMIC user story (5-15 min in fresh session)",
      "description": "As a [user], I want [goal] so that [benefit]",
      "acceptanceCriteria": [
        "GIVEN [context] WHEN [action] THEN [result]",
        "AND [additional criteria]"
      ],
      "technicalNotes": "Implementation hints for fresh context",
      "estimatedSessionTime": "5-15 minutes",
      "dependencies": ["US-002"],
      "fileTargets": ["/path/to/file.ts"]
    }
  ],

  "dataFlow": {
    "inputs": ["description of data sources"],
    "transformations": ["what happens to data"],
    "outputs": ["where data goes"],
    "storage": "persistence strategy"
  },
  "errorHandling": {
    "scenarios": [
      {
        "condition": "what could go wrong",
        "response": "how to handle it",
        "userFeedback": "what user sees"
      }
    ],
    "logging": "strategy description"
  },

  "testing": {
    "unitTests": ["test scenarios"],
    "integrationTests": ["end-to-end flows"],
    "manualChecklist": ["human verification steps"]
  },

  "successCriteria": {
    "functional": ["feature works as described"],
    "technical": ["no linter errors", "builds successfully"],
    "quality": ["performance benchmarks", "code standards"],
    "documentation": ["README updated", "comments added"]
  },

  "constraints": {
    "mustNot": ["things to avoid"],
    "performance": ["response time", "memory limits"],
    "scale": ["concurrent users", "data volume"]
  },

  "safetyLimits": {
    "maxStoriesPerSession": 1,
    "sessionTimeWarning": "human review needed if story exceeds 15 min",
    "totalEstimatedSessions": "calculated from story count"
  }
}
```

USAGE WORKFLOW (TRUE METHOD)

Step 1: Dictate (You)

```
1. Open Wispr Flow
2. Start recording
3. Talk through Phase 1 prompts naturally for ~10 minutes
4. Don't overthink - just brain dump everything
5. Stop recording
```

Step 2: Initial Processing (Claude)

```
Paste your voice transcription and say:

"Use the Voice-Optimized PRD Template V2 (TRUE Method). Parse my voice capture,
extract all details into the structured format with ATOMIC user stories
(each completable in ONE fresh session), then ask me ONLY the clarifying
questions where information is missing or ambiguous."
```

Step 3: Rapid-Fire Q&A (You + Claude)

```
Claude asks targeted questions one by one.
You answer briefly - voice or text.
Claude updates the PRD in real-time.
Claude ensures stories are atomic enough for ONE fresh session each.
```

Step 4: Final PRD Review (Claude)

```
Claude presents:
- Complete agent-ready JSON
- Summary of ATOMIC user stories
- Dependency graph
- Estimated session count
- Risk flags

You review and approve or iterate.
```

Step 5: Manual Execution with TRUE Method

```
For each user story (in dependency order):

1. Open FRESH Claude Code session (maximum context window)
2. Paste user story + technical context
3. Execute to completion (5-15 min)
4. Validate acceptance criteria met
5. Close session
6. Update HANDOFF.md with completion status
7. Open NEW fresh session for next story
```

```
NO PLUGIN. NO AUTOMATION. FRESH CONTEXT EVERY STORY.
Zero context rot = optimal performance throughout.
```

ANTI-PATTERNS TO AVOID

❌ **Don't dictate like you're writing**

- Avoid: "The system shall implement a RESTful API..."
- Do: "We need an API that handles GET and POST requests for user data..."

❌ **Don't skip edge cases**

- Avoid: Assuming happy path only
- Do: "What if user has no internet? What if API is down?"

❌ **Don't say "something like X"**

- Avoid: "Use something like Tailwind or whatever"
- Do: "Use Tailwind for styling, specifically version 3.x"

❌ **Don't leave authentication vague**

- Avoid: "Handle login somehow"
- Do: "OAuth with Google, store JWT in httpOnly cookie"

❌ **Don't create multi-session stories** (NEW - TRUE METHOD)

- Avoid: "Build entire authentication system" (too broad)
- Do: Split into atomic stories:
  - "Setup OAuth provider configuration"
  - "Create OAuth callback endpoint"
  - "Implement JWT token generation"
  - "Add protected route middleware"

POWER TIPS FOR ONE-SHOT EXCELLENCE

During Dictation:

1. **Start with "Why" not "What"** - Context drives better requirements
2. **Use user perspective** - "A user clicks X and sees Y"
3. **Say constraints out loud** - "We can't use package Z because..."
4. **Mention failure modes** - "If this fails, then..."
5. **Reference existing patterns** - "Like we did in project Y..."

For Claude's Questions:

1. **Prioritize by risk** - Ask about integration points first
2. **Group related questions** - Bundle auth questions together
3. **Provide context with questions** - "You mentioned API but didn't specify..."
4. **Offer smart defaults** - "Should I assume REST unless you say GraphQL?"
5. **Confirm scope boundaries** - "Is X in this PRD or separate?"
6. **Validate atomicity** - "Can this story complete in ONE fresh 15-min session?"

For Manual Execution (TRUE METHOD):

1. **Keep stories atomic** - If >15 min, split into smaller stories
2. **Fresh session per story** - No context accumulation between stories
3. **Validate before next** - Confirm acceptance criteria met
4. **Update HANDOFF.md** - Track progress for Genesis integration
5. **Include technical context** - Each story has implementation hints
6. **Set realistic expectations** - More sessions, but optimal performance

---

EXAMPLE VOICE DICTATION (GOOD)

"Okay so the project name is AgileAdapt Client Dashboard. This is for my AI automation business targeting Australian tradies. The main reason this exists is I need a way for clients to view their chatbot analytics and manage their AI voice agent settings without calling me every time.

The current situation is I'm manually sending screenshots of analytics from multiple platforms and it's taking hours per week. The pain point is clients can't self-serve and I'm becoming a bottleneck.

What we're building is a Next.js dashboard, using Tailwind and Shadcn UI components, that pulls data from Stammer AI's API and 10web analytics. Users will be able to log in with their email, see call volume charts, chatbot conversation counts, and toggle voice agent hours.

The stack is Next.js 14 app router, TypeScript, Tailwind, Shadcn, and we'll use Supabase for auth and storing user preferences. The existing codebase is at github.com/AgileAdapt/client-portal. Integration points are Stammer AI REST API and 10web webhooks.

What happens when Stammer API is down? Show cached data with a warning banner. We need to handle rate limiting from 10web - max 100 requests per hour so we'll cache for 15 minutes. It should NOT allow users to modify pricing plans - that's admin only. Performance requirement is dashboard loads under 2 seconds.

The main user journey is: user gets email invite with magic link, clicks it, sees dashboard immediately with their three clients (if they're an agency) or their single business (if they're a sole tradie). They can switch between clients with a dropdown. Error scenarios - if API returns 401, log them out and show re-auth button.

We know it's done when: user can log in, see real data from both APIs, switch clients if multi-client, and their toggled settings persist and actually update the voice agent hours in Stammer. Testing criteria include manual login test, API mock tests for failure scenarios, and visual regression test for the charts.

Also need to handle mobile - should work on iPhone since tradies use phones a lot. Use Shadcn's responsive variants. Authentication needs rate limiting on magic link sends - max 3 per hour per email to prevent abuse. The dashboard should auto-refresh data every 5 minutes without page reload. And create an agents.md file in the /dashboard folder explaining the API integration patterns for future features."

EXAMPLE CLARIFICATION Q&A (GOOD)

Claude: "You mentioned Stammer AI REST API but didn't specify authentication. What auth method does Stammer use? Do you have API keys?"

You: "Yeah Stammer uses API key in header as Bearer token. I have keys in my password manager."

Claude: "Got it. Should I set up environment variables for staging and production keys separately?"

You: "Yes - STAMMER_API_KEY_STAGING and STAMMER_API_KEY_PROD."

Claude: "You mentioned 'three clients if agency' - where does this client list come from? Supabase table?"

You: "Yes, there's a 'clients' table that links to the user. Schema is user_id, client_id, client_name, stammer_bot_id, tenWeb_site_id."

Claude: "Perfect. For the magic link email, should I build that or use a service?"

You: "Use Resend for emails. I have an account and API key."

Claude: "Clear. Last question - you said 'toggle voice agent hours' but what's the actual interface? Dropdown? Time picker?"

You: "Time range picker - start time and end time in client's local timezone. Restrict to business hours 6am-10pm."

Claude: "I'll split the 'authentication system' into atomic stories for TRUE method execution. Each will complete in one fresh session. Sound good?"

You: "Perfect. Keep each story under 15 minutes execution time."

EXAMPLE ATOMIC USER STORIES (TRUE METHOD)

❌ **BAD (Too Broad - Would Cause Context Rot):**

```
{
  "id": "US-001",
  "title": "Build complete authentication system",
  "acceptanceCriteria": [
    "Users can sign up, log in, reset password, and manage profile"
  ]
}
```

This would require 30+ minutes, causing context degradation

✅ **GOOD (Atomic - Completable in One Fresh Session):**

```
[
  {
    "id": "US-001",
    "title": "Setup Supabase auth client and environment",
    "acceptanceCriteria": [
      "GIVEN fresh Next.js project WHEN Supabase client configured THEN auth helpers work",
      "AND environment variables documented in .env.example",
      "AND connection test passes"
    ],
    "technicalNotes": "Use @supabase/auth-helpers-nextjs, configure SUPABASE_URL and SUPABASE_ANON_KEY",
    "estimatedSessionTime": "5 minutes",
    "fileTargets": [".env.example", "lib/supabase.ts"]
  },
  {
    "id": "US-002",
    "title": "Create magic link authentication endpoint",
    "acceptanceCriteria": [
      "GIVEN user submits email WHEN POST /api/auth/magic-link THEN email sent via Resend",
      "AND rate limiting enforced (3 per hour per email)",
      "AND success/error responses returned"
    ],
    "technicalNotes": "Use Resend SDK, implement rate limiter with Redis or in-memory cache",
    "estimatedSessionTime": "10 minutes",
    "dependencies": ["US-001"],
    "fileTargets": ["app/api/auth/magic-link/route.ts", "lib/rate-limit.ts"]
  },
  {
    "id": "US-003",
    "title": "Build OAuth callback handler",
    "acceptanceCriteria": [
      "GIVEN OAuth flow completes WHEN callback hits THEN JWT stored in httpOnly cookie",
      "AND user redirected to /dashboard",
      "AND failed auth shows error and redirects to /login"
    ],
    "technicalNotes": "Handle Supabase OAuth callback, set secure cookie with 7-day expiry",
    "estimatedSessionTime": "8 minutes",
    "dependencies": ["US-001"],
    "fileTargets": ["app/api/auth/callback/route.ts", "middleware.ts"]
  },
  {
    "id": "US-004",
    "title": "Add protected route middleware",
    "acceptanceCriteria": [
      "GIVEN unauthenticated request to /dashboard WHEN middleware runs THEN redirect to /login",
      "AND authenticated request passes through",
      "AND JWT validation works correctly"
    ],
    "technicalNotes": "Next.js middleware validates JWT, protects /dashboard/* routes",
    "estimatedSessionTime": "7 minutes",
    "dependencies": ["US-002", "US-003"],
    "fileTargets": ["middleware.ts"]
  }
]
```

Each story:

- ✅ Completes in ONE fresh session (5-15 min)
- ✅ Has clear acceptance criteria
- ✅ Includes technical implementation hints
- ✅ Specifies exact file targets
- ✅ Calls out dependencies
- ✅ Prevents context rot through atomicity

TRUE METHOD EXECUTION EXAMPLE

Session 1:

```
1. Open fresh Claude Code
2. Paste: "Execute US-001: Setup Supabase auth client..."
3. Claude completes in ~5 min
4. Validate: Environment variables set, connection test passes
5. Close session
6. Update HANDOFF.md: "US-001 complete, files: .env.example, lib/supabase.ts"
```

Session 2:

```
1. Open fresh Claude Code (MAXIMUM context window)
2. Paste: "Execute US-002: Create magic link endpoint..." + technical context
3. Claude completes in ~10 min
4. Validate: Endpoint works, rate limiting functional
5. Close session
6. Update HANDOFF.md: "US-002 complete, files: app/api/auth/magic-link/route.ts, lib/rate-limit.ts"
```

Continue pattern for US-003, US-004, etc.

Result: ZERO context degradation across entire project execution.

SUCCESS METRICS FOR THIS TEMPLATE

You'll know this template is working when:

✅ Claude asks fewer than 10 clarifying questions ✅ Zero "wait, you didn't tell me X" during development ✅ Each user story completes in ONE fresh session (5-15 min) ✅ No story requires >15 min (context rot indicator) ✅ First iteration of each story produces runnable code ✅ You can dictate a new PRD in under 15 minutes ✅ Clarification Q&A takes under 10 minutes ✅ No scope creep during implementation ✅ Manual iteration overhead acceptable for quality gain

---

GENESIS INTEGRATION

After each successful story:

```markdown
# HANDOFF.md Update Pattern

## Last Completed Story
- ID: US-003
- Title: Build OAuth callback handler
- Files Modified: app/api/auth/callback/route.ts, middleware.ts
- Status: ✅ Complete
- Validation: OAuth flow tested, JWT stored correctly

## Next Story
- ID: US-004
- Title: Add protected route middleware
- Dependencies: US-002, US-003 (both complete)
- Ready to execute: YES

## Learnings
- Supabase OAuth requires specific redirect URL pattern
- httpOnly cookies need sameSite: 'lax' for local dev
- JWT validation library: jose works better than jsonwebtoken
```

This builds system memory while maintaining execution freshness.

NEXT: Save Your Best PRDs as Examples

After you've used this template 3-4 times, save your best voice dictation + final PRD pairs. Use them as few-shot examples when starting new projects:

"Here's how I previously dictated the ClientDashboard PRD [paste example]. Now I want to dictate a new project for [X] using TRUE Method with atomic stories..."

This trains Claude to understand YOUR communication style and business context, making each subsequent PRD even better.

TRUE METHOD PHILOSOPHY REMINDER

Plugin approach: Automate iteration → context rot degrades performance **TRUE Method:** Manual iteration with fresh context → optimal performance guaranteed

Trade-off: More human clicks per story **Benefit:** ZERO performance degradation, predictable outcomes

For Genesis autonomous agents, this pattern enables:

- Reliable execution quality
- Predictable resource usage
- Clear success/failure boundaries
- System memory accumulation without context pollution