NINJA HACK GIVEAWAY GUIDE

Build Your Own Team Check-In Platform

A beginner-friendly, production-grade guide to building a weekly employee check-in and leadership insights platform

Full product description
Human-facing harness setup
Companion AI upload file
Testing and launch checklists

Presented by Brian Walker

Shop Marketing Pros

July 2026

Limited-Use License Agreement

Effective date: July 2026

This Limited-Use License Agreement (the "Agreement") governs access to and use of this guide, including its product description, prompts, specifications, checklists, examples, and supporting content (collectively, the "Materials"). The Materials are provided by Brian Walker and Shop Marketing Pros (collectively, the "Owner") exclusively to eligible members of Seven Figure Agency.

> **LICENSE NOTICE:** By accessing, downloading, copying, or using any part of the Materials, you agree to this Agreement. If you do not agree, do not use the Materials and delete every copy in your possession or control.

1. Definitions

Licensee means a current Seven Figure Agency member who lawfully receives the Materials directly through Seven Figure Agency or from the Owner.

Internal Tool means software that a Licensee creates and operates solely for the internal business operations of the Licensee's own organization. An Internal Tool is not offered to clients, customers, other agencies, or the public.

Built Product means any software, application, template, workflow, database, documentation, or related deliverable created using or substantially based on the Materials.

2. Limited License Grant

Subject to continued compliance with this Agreement, the Owner grants the Licensee a limited, non-exclusive, non-transferable, non-sublicensable, revocable, and royalty-free license to:

- Access and use the Materials for the Licensee's own learning and planning.
- Modify the prompts and specifications for the Licensee's own organization.
- Build, operate, maintain, and improve one or more Internal Tools for the Licensee's own internal business use.
- Make a reasonable number of private working or backup copies solely for those permitted purposes.

No right is granted except the rights expressly stated in this Agreement.

3. Seven Figure Agency Distribution Restriction

The Materials are for Seven Figure Agency members only. The Licensee may not distribute, publish, post, transmit, disclose, lend, sublicense, or otherwise provide the Materials, in whole or in part, to any person or organization outside Seven Figure Agency.

This restriction includes, without limitation:

- Posting the Materials or prompts on a public website, social network, forum, course platform, marketplace, or document-sharing site.
- Placing the Materials in a public repository or in a private repository accessible to an unauthorized person.
- Forwarding the guide to clients, prospects, employees, contractors, consultants, peer groups, other communities, or friends who are not authorized Seven Figure Agency members.
- Publishing summaries, prompt packs, templates, recordings, screenshots, or derivative instructions that reproduce a substantial portion of the Materials or allow another person to reconstruct them.
- Using public links, shared folders, or other access methods that do not limit access to authorized recipients.

The Licensee may show the resulting Internal Tool to people within the Licensee's organization as needed for normal internal use, but may not provide those people with the Materials unless they are independently authorized Seven Figure Agency members.

4. Limited AI Coding Service Exception

The Licensee may submit only the portions of the Materials reasonably necessary to a reputable AI coding service solely to build or maintain the Licensee's Internal Tool. This exception does not permit publishing the Materials, enabling public task or conversation links, using the Materials to train a public model, or sharing the Materials with human users outside Seven Figure Agency.

The Licensee is responsible for reviewing the service's privacy, retention, sharing, and training settings; using the most private settings reasonably available; and avoiding the submission of confidential, regulated, employee, client, or credential data that is not necessary for the build.

5. No Sale Or Commercialization

The Materials and every Built Product are limited to the Licensee's own internal business use. The Licensee may not, directly or indirectly:

- Sell, resell, license, sublicense, rent, lease, assign, transfer, or distribute a Built Product.
- Charge any person for access to or use of a Built Product.
- Offer a Built Product as software as a service, a hosted service, a managed service, a white-label product, or a client deliverable.
- Build or customize the product for a client, affiliate, portfolio company, franchisee, licensee, or other third party.
- Bundle a Built Product or the Materials with paid consulting, coaching, training, membership, agency services, or another commercial offering.
- Use the Materials as the basis for a commercial template, starter kit, prompt pack, course, software product, or competing product.

- Sell or transfer a business asset in a manner that separately values, licenses, or distributes the Built Product without the Owner's prior written permission.

Use of an Internal Tool to support the Licensee's ordinary internal operations does not violate this section merely because the Licensee operates a for-profit business. The restriction is on commercializing, distributing, or providing the Materials or Built Product to others.

6. Ownership And Branding

The Owner retains all right, title, and interest in the Materials, including all copyrights and other intellectual-property rights. The Materials are licensed, not sold. No ownership interest in the Materials transfers to the Licensee.

Subject to this Agreement and any rights belonging to third-party platforms, libraries, or service providers, the Licensee may own the original code and organization-specific content the Licensee independently creates for its Internal Tool. That ownership does not remove or reduce the internal-use, non-distribution, and no-commercialization restrictions in this Agreement.

No trademark or branding license is granted. The Licensee must choose an original product name and original branding. The Licensee may not name a Built Product PRO Pulse, ProPulse, PRO-Pulse, or any confusingly similar variation. The Licensee may not use Shop Marketing Pros, the Owner's logos, or other Owner branding without the Owner's prior written permission.

The name PRO Pulse may appear in these Materials because it identifies the Owner's source product and Ninja Hack presentation. It may not appear as the name, package name, repository name, domain, title, logo, interface copy, email sender identity, application metadata, or public description of a Built Product. Before launch, the Licensee must remove unauthorized source-product branding from the Built Product and its repository.

7. Security And Compliance

The Licensee is solely responsible for the design, security, privacy, accessibility, legal compliance, employment practices, data handling, operation, and results of its Internal Tool. The Licensee must protect employee and company information, use appropriate access controls, review third-party terms, and comply with laws and policies applicable to its organization.

The Licensee must use reasonable safeguards to prevent unauthorized access to the Materials and promptly notify the Owner if the Licensee becomes aware of unauthorized disclosure or distribution.

8. No Warranties

THE MATERIALS ARE PROVIDED "AS IS" AND "AS AVAILABLE," WITHOUT WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, NON-INFRINGEMENT, ACCURACY, SECURITY, OR ERROR-FREE OPERATION. THE OWNER

DOES NOT WARRANT THAT THE MATERIALS OR ANY BUILT PRODUCT WILL MEET THE LICENSEE'S REQUIREMENTS OR COMPLY WITH EVERY LAW, POLICY, PLATFORM RULE, OR THIRD-PARTY SERVICE TERM.

9. Limitation Of Liability

TO THE MAXIMUM EXTENT PERMITTED BY LAW, THE OWNER WILL NOT BE LIABLE FOR ANY INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES; LOSS OF PROFITS, REVENUE, DATA, GOODWILL, OR BUSINESS OPPORTUNITY; EMPLOYMENT OR PRIVACY CLAIMS; SECURITY INCIDENTS; OR THIRD-PARTY SERVICE FAILURES ARISING FROM OR RELATED TO THE MATERIALS OR ANY BUILT PRODUCT.

TO THE MAXIMUM EXTENT PERMITTED BY LAW, THE OWNER'S TOTAL LIABILITY ARISING FROM OR RELATED TO THE MATERIALS OR THIS AGREEMENT WILL NOT EXCEED THE AMOUNT, IF ANY, THE LICENSEE PAID DIRECTLY TO THE OWNER SPECIFICALLY FOR THE MATERIALS.

10. Termination And Remedies

This license terminates automatically if the Licensee breaches this Agreement. Upon termination, the Licensee must immediately stop using the Materials, delete all copies within its possession or control, remove unauthorized postings or shares, and stop any use or distribution of a Built Product that is not expressly permitted in writing by the Owner.

Unauthorized distribution or commercialization may cause harm that cannot be fully remedied by money alone. The Owner may seek any relief available under law or equity, in addition to terminating the license. The Owner may provide a written opportunity to cure a breach, but is not required to do so.

11. General Terms

The Licensee may not assign or transfer this Agreement. Any attempted assignment is void. If any provision is found unenforceable, the remaining provisions remain in effect and the affected provision will be enforced to the greatest extent permitted. A failure to enforce a provision is not a waiver. This Agreement is the entire agreement concerning the Materials and may be changed only in a writing authorized by the Owner.

This Agreement is governed by the laws applicable at the Owner's principal place of business, without regard to conflict-of-law rules. Any permitted dispute must be brought in a court with jurisdiction where the Owner's principal place of business is located, unless the parties agree otherwise in writing.

Requests for broader distribution, client use, transfer, or commercial rights require the Owner's advance written permission. Contact Brian Walker or Shop Marketing Pros for permission before engaging in any use not expressly allowed above.

Welcome

PRO Pulse is a private weekly employee check-in and leadership conversation platform. It gives each team member a short, repeatable way to share how work feels, what went well, what was difficult, and where support may be needed. It gives leaders a consistent view of their people without replacing the human conversations that make a healthy company work.

The product is intentionally simple for employees and intentionally useful for leaders. A team member can complete a check-in in a few minutes. A manager can quickly see who has submitted, who has not, who may need attention, and which conversations deserve follow-up. Over time, the same data becomes a clear picture of morale, stress, workload, participation, and recurring themes.

This guide explains the complete product, the decisions to make before building it, the recommended technical architecture, and the prompts needed to build it with an AI coding partner. It is written for a person who has never vibe-coded an application before.

> **KEY IDEA:** The goal is not to build a survey. The goal is to build a dependable weekly operating rhythm: employee reflection, leader awareness, conversation, follow-through, and long-term learning.

What You Will Build

- A secure login and account system for one company.
- Three roles: employee, manager, and administrator.
- A weekly check-in built around configurable 1-to-5 face ratings.
- Required weekly high and weekly low questions, plus optional comments.
- Draft saving, controlled submission periods, and historically accurate records.
- A personal dashboard and check-in history for each employee.
- A manager dashboard with filters, reminders, review status, and attention signals.
- Two-way conversations attached to each submitted check-in.
- Company and team insights, score distributions, trends, and a 12-week pulse grid.
- Email and in-app notifications with preferences and delivery logs.
- Administrative tools for people, teams, categories, assignments, reminders, email copy, and announcements.
- Optional profile photos, image and GIF support in conversations, and AI-generated leadership summaries.
- A production deployment with security policies, tests, backups, logging, and a launch checklist.

How To Use This Guide

1. Read Part 1 so you understand the product before asking an AI to build it.
2. Choose an original product name in Part 2. Do not use PRO Pulse or a similar variation.
3. Complete the remaining decisions in Part 2. These are business decisions, not coding decisions.

4. Install and use a coding harness described in Part 3. A normal website chat window is not enough.
5. Download the companion AI-facing Markdown specification and upload it to your coding harness.
6. Paste the Project Constitution into your coding task before any feature prompt.
7. Send one build prompt at a time, in order, inside the same coding task.
8. Do not advance until the acceptance checks for the current milestone pass.
9. Keep every working milestone in Git so you always have a clean recovery point.
10. Use the final review prompts before inviting your whole team.

> **IMPORTANT:** Companion AI file: Download `give-this-to-your-ai-of-choice.md`. Save that file with your private project and upload it to your coding harness before beginning the build.
>
> **IMPORTANT:** Do not paste the entire prompt pack into an AI at once. Large one-shot builds hide mistakes. Build in milestones, test each milestone, and let the existing code teach the AI what should happen next.
>
> **IMPORTANT:** This guide teaches you how to build a product inspired by the PRO Pulse workflow. It does not grant permission to copy the PRO Pulse name, Shop Marketing Pros branding, logos, or a confusingly similar identity. Your product must have its own name and branding before implementation begins.

Part 1: The Product

The Problem PRO Pulse Solves

Leaders often learn about employee frustration too late. A weekly meeting may be canceled, a quiet team member may not volunteer a concern, and a manager may remember the loudest conversation instead of the broader pattern. Traditional employee surveys are usually too infrequent to guide weekly leadership behavior. Chat messages are immediate but scattered. Spreadsheets collect data but do not create accountability, privacy, or a conversation trail.

PRO Pulse creates a lightweight signal every week. It helps a leader answer five practical questions:

- Who completed a check-in for the selected week?
- How is each person doing across the categories we care about?
- Is this an isolated difficult week or part of a downward pattern?
- What did the person say was their high, their low, or an important concern?
- Has a leader read the check-in, responded, and followed up?

It also helps each employee reflect on the week and preserve a personal history of wins, challenges, comments, and leadership responses.

The Product Philosophy

Short for employees

The weekly action should take only a few minutes. The form uses large face choices rather than tiny radio buttons. Prompts are direct. Drafts prevent lost work. The employee sees exactly which week is being submitted.

Useful for leaders

The manager view favors scanning and action. Leaders can filter for missing, unreviewed, or potentially concerning check-ins, then expand a person only when more context is needed.

A signal, not a diagnosis

Scores never claim to diagnose an employee or explain the whole story. Attention flags are deliberately gentle. They point a leader toward a conversation; they do not make a personnel decision.

Conversation belongs beside the data

A manager reply is attached to the exact check-in that prompted it. The employee can reply. Both parties can return to the thread later. This turns the product from a reporting tool into a leadership habit.

History must remain truthful

If an administrator renames a category or changes face labels, old submissions keep the labels that existed when they were submitted. Historical records should never silently change meaning.

Privacy is a product feature

Employees see their own information. Managers see only the people assigned to them or deliberately shared with them. Administrators have company-wide access. These rules are enforced by the server and database, not merely hidden in the interface.

The Weekly Rhythm

PRO Pulse uses a Sunday-through-Saturday reporting period. The stored period key is the Sunday that begins the week. The company timezone controls when a week rolls over.

Employees can submit for the current period or one of the two most recent periods. This allows someone returning from travel or time off to complete a missed check-in without opening unlimited historical editing.

The typical rhythm is:

1. A scheduled reminder tells employees that a check-in is due.

2. The employee opens the form, chooses a face for each required category, writes a weekly high and low, and optionally adds context.
3. The employee can save a draft or submit.
4. Submission notifies the direct manager and any additional designated leadership viewers.
5. The manager reviews the submission, marks it reviewed, and replies when helpful.
6. The employee receives an email and in-app notification when a leader replies.
7. Managers and administrators use trends and insights to notice recurring themes.

Roles And Visibility

Role	What the person can do	What the person can see
Employee	Manage an account, complete check-ins, save drafts, submit, view history, reply to comments, set notification preferences	Their own dashboard, history, profile, check-ins, and conversations
Manager	Everything an employee can do, plus review people, send reminders, reply, mark reviewed, and view team insights	Their own information, direct reports, and people assigned as additional leadership visibility
Administrator	Configure and operate the system, manage users and teams, view company insights, inspect email logs, and control settings	All active and historical company data allowed by policy

Every employee may have one direct manager. An employee may also have several additional leadership viewers. This supports situations such as an owner, department head, or people leader who needs visibility without replacing the direct manager relationship.

> **SECURITY:** A hidden menu item is not authorization. Every protected page, server action, database query, and storage operation must re-check the current user's permission.

Employee Experience

Login and account recovery

Employees sign in with email and password. The product includes forgot-password and reset-password flows. A deactivated user cannot continue using an old session. An employee can update their name, password, profile photo, and notification preferences from My Account.

Personal dashboard

The dashboard greets the employee and shows the current reporting period. It clearly displays one of three states: not started, draft in progress, or submitted. A primary action starts or continues the check-in.

The dashboard also shows a recent trend summary and a 12-week category trend chart. A new employee sees a useful empty state instead of an empty graph.

Weekly check-in form

Each active category appears with its label, explanatory prompt, required or optional status, and five large face choices. The default scale is:

- 1: Very Low
- 2: Low
- 3: Okay
- 4: Good
- 5: Great

An administrator may customize those five labels for each category. That matters because a workload prompt may need wording such as Unsustainable, Heavy, Manageable, Comfortable, and Great Capacity while still preserving the rule that a higher score is healthier.

The initial recommended categories are:

- Morale: How are you feeling about work overall?
- Stress Level: How manageable was your stress this week?
- Workload: How sustainable was your workload?

The second half of the form asks:

- What was your high this week?
- What was your low this week?
- Anything else you would like to share with your leader?

The high and low are required at submission. Additional comments are optional. Each category may also have an optional comment.

The period selector appears near the submit controls so the employee confirms the correct week before submitting. Changing the period keeps the answers already entered instead of wiping the form.

Draft and submission rules

A draft can be saved and edited. Draft content is private and does not notify leadership. A submitted check-in is treated as a historical record and is locked against casual rewriting. If the company later chooses to support corrections, implement a deliberate amendment workflow with an audit trail rather than silently changing the original submission.

Only one check-in may exist for an employee and reporting period. The database enforces this rule with a unique constraint, and the server validates it again.

My history

The history screen shows a 12-week category chart and a running list of past submitted check-ins. Each item shows the date range, overall score label, whether the thread contains replies, the original category responses and snapshots, the weekly high and low, and the full conversation.

Manager Experience

Team dashboard

The manager dashboard opens to a selected reporting period, usually the most recently completed week. At the top, leaders see summary counts for submitted, not submitted, awaiting review, and needing attention.

Filters make the page actionable:

- All
- Submitted
- Missing
- Unreviewed
- Needs attention

Each employee row shows name, profile photo, email, recent login information, submission status, category faces at a glance, attention indicators, and a control to expand the full check-in. A missing employee can receive an individual reminder without leaving the page.

Reviewing and replying

When expanded, the manager sees the submitted timestamp, category scores and comments, overall average, high, low, additional comments, and existing conversation. The manager can mark the check-in reviewed and write a reply.

Replies support a restrained rich-text subset such as paragraphs, bold, italic, and bullets. The system can optionally support approved GIFs and image uploads. Editing or deleting a comment must be limited to its author. Edited comments display an Edited indicator, and prior versions are retained in comment edit history.

The employee sees the conversation in My History and may reply. New replies create in-app notifications and, according to preferences, email notifications.

Attention signals

The manager dashboard may flag a person when one or more of these conditions are true:

- The latest overall average is below a configured threshold.
- The employee has two or more consecutive struggling weeks.
- The recent average is meaningfully lower than the preceding comparison window.

Missing weeks break a struggling streak rather than being treated as a low score. Trend logic must be pure, documented, and unit-tested. The interface should say Needs attention or Trending down, never make a clinical or disciplinary claim.

Insights Experience

The Insights area helps managers and administrators move from individual check-ins to patterns. It supports configurable periods such as one week, four weeks, eight weeks, twelve weeks, or a defined date range.

For each permitted company or team scope, show:

- Number of submitted check-ins, not number of category response rows.
- Overall average for the period.
- Average by category.
- Comparison with the previous equal-length period.
- Distribution of scores from 1 through 5.
- A 12-week pulse grid with one row per person and one cell per week.
- Empty states when the selected period has no submissions.

Managers see only their permitted team scope. Administrators see company-wide and team-level results. Clicking a team opens a focused drill-down.

Optional AI leadership summary

An optional AI feature can turn the selected period's numbers and free-text responses into a concise leadership overview containing:

- A short summary.
- A trend interpretation grounded in current and previous averages.
- Recurring themes.
- Watch-outs.
- Wins.

The output must use structured JSON, be stored with the model name and generation time, and never invent facts. When sample size is small, the summary should say so. The default privacy setting should remove employee names and send only counts, themes, and de-identified text. Revealing names should require an explicit administrator setting and a clear policy decision.

AI output is advisory. Leaders must still read the underlying data and use judgment.

Administrator Experience

People

Administrators can invite or create users, edit names and email addresses, set roles, assign teams, trigger password resets, and deactivate or reactivate users. Deactivation is preferred to deletion so historical check-ins, comments, and reports stay intact.

Teams

Administrators can create, rename, reorder, activate, and deactivate teams. A person may belong to one operational team for reporting. Team changes should affect future group reporting while old individual submissions remain intact.

Leadership assignments

Administrators select one direct manager for each employee and may add several additional leadership viewers. The interface must prevent self-management and duplicate assignments. Permissions change immediately when assignments change.

Categories

Administrators can create categories, edit the label and prompt, customize five face labels, set required or optional, reorder, hide, and reactivate. Historical submissions retain category and score label snapshots.

Reminders

Administrators can create multiple reminder schedules. Each schedule includes a name, enabled state, local weekday and hour, whether it targets the current or previous week, email subject, and introductory copy. A Send now control supports testing and one-time reminders.

Scheduled jobs must be protected by a secret, use the company timezone, avoid duplicate sends, skip employees who already submitted for the target period, honor personal preferences, and record every delivery attempt.

Email content and delivery logs

Administrators can edit approved portions of welcome, manager alert, reply, and reminder emails without editing code. Templates support a documented set of safe tokens. The product records sent, failed, and simulated outcomes, provider message IDs, errors, and timestamps.

Announcement banner

An administrator may publish a reusable in-app announcement with optional start and expiration times. Users may dismiss it. Expiration behavior is deterministic, and the administrator can confirm what is active without guessing.

Product Boundaries

PRO Pulse should not become a performance review scorecard, medical diagnostic system, anonymous complaint channel, or employee surveillance product. It is a weekly reflection and conversation tool.

Leaders should not rank employees by emotional scores. Access to individual text should be limited. Retention, consent, and acceptable-use policies should be explained before launch. If the company operates in a regulated environment, obtain appropriate legal, privacy, and human-resources review.

Part 2: Decisions Before Coding

The AI can write code, but it should not silently choose your people policies. Complete this section first. Short, clear answers are enough.

Product Identity Worksheet

> **IMPORTANT:** Choosing an original product name is required. Do not use PRO Pulse, ProPulse, PRO-Pulse, or a name that could reasonably imply that your product is PRO Pulse or is affiliated with Shop Marketing Pros. The AI-facing Markdown file repeats this rule so your coding harness enforces it throughout the build.

- Original product name: [YOUR ORIGINAL PRODUCT NAME - NOT PRO PULSE OR A SIMILAR VARIATION]
- Company name: [YOUR COMPANY NAME]
- One-sentence purpose: [WHY THIS EXISTS]
- Primary brand color: [HEX COLOR]
- Supporting accent color: [HEX COLOR]
- Company timezone: [EXAMPLE: America/Chicago]
- Reporting week: [RECOMMENDED: Sunday through Saturday]
- Application URL or planned subdomain: [EXAMPLE: pulse.yourcompany.com]

Participation Worksheet

- Who completes a check-in: [ALL EMPLOYEES OR SELECT GROUPS]
- When reminders should be sent: [DAY AND LOCAL HOUR]
- Whether late submissions are allowed: [RECOMMENDED: CURRENT PLUS TWO PRIOR PERIODS]
- Who may see an employee's check-in: [DIRECT MANAGER, ADDITIONAL VIEWERS, ADMIN]
- Whether submitted check-ins are locked: [RECOMMENDED: YES]
- Whether leaders are expected to mark every check-in reviewed: [YES OR NO]
- Expected response time for a concerning check-in: [YOUR INTERNAL EXPECTATION]

Category Worksheet

For each category, decide the label, prompt, required status, and five labels from least healthy to most healthy.

Recommended starting set

Morale

- Prompt: How are you feeling about work overall?
- Required: Yes
- Labels: Very Low, Low, Okay, Good, Great

Stress Level

- Prompt: How manageable was your stress this week?
- Required: Yes
- Labels: Overwhelmed, Very Stressed, Managing, Comfortable, Calm

Workload

- Prompt: How sustainable was your workload this week?
- Required: Yes
- Labels: Unsustainable, Too Heavy, Manageable, Comfortable, Great Capacity

> **DESIGN RULE:** Every category must use the same direction: 1 is the least healthy state and 5 is the healthiest state. Do not create one category where a higher number means worse.

Communication Worksheet

- From name and email for system messages: [NAME AND ADDRESS]
- Reminder subject and opening sentence: [COPY]
- Manager alert subject and opening sentence: [COPY]
- Reply notification subject and opening sentence: [COPY]
- Welcome email subject and opening sentence: [COPY]
- Whether employees may disable each email type: [YES OR NO]
- Who reviews failed email logs: [OWNER]

Privacy Worksheet

- Written explanation employees will receive about the purpose of the tool: [COPY]
- People allowed to read individual comments: [ROLES]
- Data retention period: [POLICY]
- Whether AI summaries are enabled: [YES OR NO]
- If enabled, whether names may be sent to the provider: [RECOMMENDED: NO]
- How concerns requiring urgent support should be handled: [PROCESS OUTSIDE THE APP]

Launch Owner Worksheet

- Product owner: [NAME]
- Technical owner: [NAME]
- People-policy owner: [NAME]
- Pilot group: [NAMES OR TEAM]
- Pilot start date: [DATE]
- Company launch date: [DATE]
- Weekly person responsible for monitoring reminders and failures: [NAME]

> **IMPORTANT:** Decide who owns the product after launch. An internal tool with no operational owner slowly becomes untrustworthy even when the code is excellent.

Part 3: Human-Facing Getting Started Instructions

What Vibe Coding Means Here

Vibe coding means describing the desired product and constraints to an AI coding partner, allowing it to inspect and edit the project, and reviewing the working result in short cycles. You do not need to know how to write every line of code. You do need to make business decisions, protect credentials, test what the AI builds, and refuse shortcuts that weaken the product.

Treat the AI like a capable engineering partner, not an automatic website generator. Give it context, let it inspect before editing, ask it to verify its work, and keep a version history.

Use A Coding Harness, Not A Chat Window

This project requires a coding harness: an AI-powered application, command-line tool, or code editor that can work directly inside a real project. The harness must be able to read and edit repository files, run terminal commands, start the application, run tests, inspect browser behavior, use Git, and help deploy the verified code.

Good options include:

- Codex, when it is connected to your local project or repository and has terminal and browser-testing capabilities.
- Claude Code, running in the project directory with access to the repository and terminal.
- Cursor, opened on the project folder with its coding agent and terminal available.
- Another reputable coding agent that provides the same repository, terminal, testing, and browser capabilities.

A normal conversation at chatgpt.com or claude.ai is not a coding harness. Those chat windows can help explain a concept, but simply pasting this guide into a general chat does not give the AI dependable access to your files, terminal, development server, test suite, database migrations, Git history, or deployment environment. Do not attempt to complete this build in an ordinary browser chat.

> **IMPORTANT:** Before you begin, ask the AI to list the project files, identify the repository root, and state which terminal, testing, browser, Git, and deployment actions it can perform. If it cannot directly edit the repository and run the verification commands, stop and move the project to a real coding harness.

Accounts To Create

Create these accounts before the build reaches deployment:

- A coding harness such as Codex, Claude Code, or Cursor that can read files, edit code, run terminal commands, test a browser, and work with Git.
- GitHub, for private source control and recovery history.
- Supabase, for Postgres, authentication, and file storage.

- Vercel, for hosting the Next.js application.
- Resend, for transactional email.
- Optional: an AI API provider for generated leadership summaries.
- Optional: GIPHY Developer access for GIF search in check-in conversations.

Use company-owned accounts when possible. Turn on two-factor authentication. Do not create a critical production service under a temporary contractor's personal account.

Install The Local Tools

The easiest beginner setup is:

- GitHub Desktop for visual source control.
- A current supported Node.js LTS release.
- A coding harness with direct access to the project folder and terminal.
- A modern browser.
- Optional: Supabase CLI for a faithful local database and migration testing.

If your AI environment can create the project, run the development server, and connect to GitHub directly, it may handle much of this setup. You should still understand where the project folder and repository live.

Start With The AI-Facing Markdown File

Download both files from the Ninja Hack landing page:

- The human-facing PDF, which you are reading now.
- The AI-facing Markdown build specification, `give-this-to-your-ai-of-choice.md`, which is designed to be uploaded to your coding harness.

Create a new coding task in your chosen harness, attach or place the Markdown file inside the project, and send this kickoff message:

```
Read the complete AI build specification before changing any files. Confirm that you can
inspect and edit this repository, run terminal commands, start and test the application
in a browser, use Git, and help deploy it. Treat the specification's naming, licensing,
privacy, security, architecture, and verification rules as mandatory.

We must choose an original product name. Do not name or brand this product PRO Pulse,
ProPulse, PRO-Pulse, Shop Marketing Pros, or any confusingly similar variation. If I
have not supplied an original product name, ask me for one before creating package
metadata, visible branding, domains, or email identities.

Inspect the repository and then begin only with the discovery and implementation-plan
milestone. Do not attempt the full build in one pass.
```

Keep the Markdown file in the private repository so a new coding task can read the same specification later. Do not place it in a public repository or share it outside Seven Figure Agency.

Learn Five Words

Repository: The tracked project folder and its history.

Commit: A named recovery point containing a coherent set of changes.

Environment variable: A secret or deployment-specific setting stored outside the source code.

Migration: A versioned database change that can be applied in order.

Deployment: A published version of the application that people can use.

Create A Safe Working Setup

1. Create a private GitHub repository with a clear name.
2. Create or clone a local project folder.
3. Keep the production branch named main.
4. Make sure `.env.local` is listed in `.gitignore` before adding secrets.
5. Store the Supabase service-role key only in server-side environment variables.
6. Never paste passwords, private keys, tokens, or the contents of `.env.local` into a public chat or screenshot.
7. Use separate development and production Supabase projects if the application will hold real employee information.

> **SECURITY:** The public Supabase browser key may appear in client configuration. The service-role key is different: it bypasses database policies and must never reach browser code, logs, screenshots, or Git.

Recommended Build Stack

Use a boring, well-supported stack so the AI can rely on mature documentation and libraries:

- Next.js App Router with TypeScript.
- React and Tailwind CSS.
- A restrained component system based on accessible primitives.
- Supabase Postgres, Auth, Storage, and Row-Level Security.
- Zod or an equivalent schema validator for server inputs.
- Recharts or another mature chart library.
- Resend for transactional email.
- Vitest for unit tests and Playwright for critical browser workflows.
- Vercel for deployment and scheduled functions, or another scheduler when plan limits require it.

Ask the AI to consult the current official documentation when package setup or API syntax may have changed.

Create Service Projects

Supabase

1. Create a development Supabase project.
2. Record the project URL and publishable key in `.env.local`.
3. Record the service-role key in `.env.local`, clearly marked server-only.
4. Configure the allowed application URLs for authentication redirects.
5. Do not create production tables by clicking around in the dashboard. Ask the AI to create versioned SQL migrations.
6. When ready for production, create a separate production project and apply the same migrations.

Resend

1. Create an API key for development.
2. Verify a sending domain before company launch.
3. Choose a recognizable From name and monitored Reply-To address.
4. Put the API key and From address in environment variables.
5. Start in simulated mode or send only to pilot addresses until the email copy is approved.

Vercel

1. Connect the private GitHub repository.
2. Import the Next.js project.
3. Add production environment variables in Vercel rather than uploading `.env.local`.
4. Set the application URL and a strong cron secret.
5. Apply database migrations before inviting users into a deployment that expects the new schema.

The Milestone Rhythm

For every build prompt, use this rhythm:

1. Let the AI inspect the repository and restate the milestone.
2. Let it implement the complete milestone.
3. Ask it to run formatting, lint, typecheck, unit tests, and a production build.
4. Open the application and test the acceptance checks yourself.
5. Tell the AI exactly what looked wrong, using the visible wording and screen name.
6. When the milestone passes, ask for a focused commit with a clear message.
7. Move to the next prompt.

Do not ask for a cosmetic redesign in the middle of a database or permission milestone. Finish one concern at a time.

How To Report A Problem To The AI

Weak report: The page is broken.

Useful report: On the Team page, I selected the week of June 7. The Submitted count says 4/7, but I can see five submitted employee cards. Refreshing does not change it. Please reproduce this, find the root cause, fix it, add a regression test, and rerun the full verification suite.

Include the page, action, expected result, actual result, and whether the problem survives refresh. A screenshot is useful for visual issues.

What Not To Approve

- Hardcoded employee IDs, emails, team names, or dates.
- Data access that is protected only by hidden buttons.
- A service-role key used in browser code.
- Arrays or comma-separated text used instead of real relationship tables.
- Destructive edits to migrations that have already run.
- Email sending with no logs or duplicate protection.
- A chart whose calculations have no unit tests.
- Historical labels that change when an administrator edits a category.
- A production launch with sample accounts or shared passwords.
- A claim that something is fixed when typecheck, tests, or build still fail.

Your First Pilot

Before company-wide use, create at least these test users:

- One administrator.
- One manager.
- Two employees assigned to that manager.
- One additional leadership viewer.
- One deactivated employee with historical data.

Use them to test every permission boundary. Sign in as each role rather than trusting the administrator view.

Part 4: Product And Data Blueprint

Recommended Application Areas

Area	Primary audience	Purpose
Dashboard	Everyone	Current status, personal trend, and next action
Check-in	Employees	Draft and submit a weekly pulse
My History	Employees	Review prior submissions and conversations
Team	Managers and administrators	Review participation, read check-ins, remind, reply, and mark reviewed
Insights	Managers and administrators	View team and company patterns over selected periods
My Account	Everyone	Profile, photo, password, and notification preferences
Admin	Administrators	People, teams, categories, assignments, reminders, email, announcements, and logs

Recommended Database Model

Use UUID primary keys, `created_at` and `updated_at` timestamps, foreign keys, check constraints, purposeful indexes, and Row-Level Security on every exposed table.

profiles

One row per authenticated user. Store full name, email mirror, role, active status, team, avatar path, and last sign-in time. Authentication credentials remain in the authentication system. Deactivation blocks access while preserving history.

teams

Stores operational teams used for assignment and reporting. Names should be unique within the company. Prefer deactivation to deletion after a team has history.

manager_assignments

Stores one direct manager per employee. Enforce `unique(employee_id)`, prevent self-management, and index the manager ID.

visibility_assignments

Stores additional leadership viewers. Enforce `unique(employee_id, viewer_id)` and prevent self-assignment.

checkin_categories

Stores key, label, prompt, sort order, active status, required status, and five optional face labels. Keys are stable identifiers. Labels are editable display copy.

weekly_checkins

Stores employee, week start date, draft or submitted status, weekly high, weekly low, additional comments, submission time, and timestamps. Enforce one row per employee and week.

checkin_responses

Stores the response for one category within one check-in. Include category reference, stable category key, category label snapshot, 1-to-5 score, score label snapshot, and optional category comment. Enforce one response per check-in and category.

checkin_comments

Stores the two-way conversation attached to a check-in. Include author, safe rich-text body, created time, edited time, and optional soft-deletion metadata. Permission depends on both check-in visibility and authorship.

comment_edit_history

Stores the prior body and edit metadata whenever a comment changes. The current comment stays easy to query while history remains auditable.

checkin_reviews

Stores that a particular leader reviewed a particular check-in and when. Enforce one review record per check-in and reviewer.

notifications

Stores in-app notifications with recipient, type, title, message, destination path, read time, and created time. Row policies must restrict every operation to the recipient.

notification_preferences

Stores each user's enabled state by notification type. Enforce one row per user and notification type. Define documented defaults when a preference row does not yet exist.

reminder_schedules

Stores schedule name, weekday, local hour, target period, enabled state, email subject, intro copy, last attempted time, and last successful time.

email_templates

Stores administrator-editable subject and body sections for supported message types. Validate allowed tokens. Keep system-owned legal and action content outside freeform fields when appropriate.

email_logs

Stores message type, intended recipient, related user or check-in, outcome, provider ID, error summary, attempt count, idempotency key, and timestamps. Avoid storing secrets or unnecessary sensitive body content.

announcement_banners and banner_dismissals

Store the active announcement, optional schedule, creator, and one dismissal row per user. The banner query must account for activation and expiration in the company timezone.

weekly_analyses

Optional. Stores the scope, period key, structured summary JSON, model identifier, privacy mode, generation time, and source-data fingerprint. A fingerprint helps determine whether saved analysis is stale after new check-ins arrive.

audit_logs

Stores important administrative mutations such as role changes, deactivation, assignments, category changes, reminder changes, and email-template updates. Record actor, action, entity type, entity ID, safe metadata, and time.

Historical Snapshot Rules

At submission, copy the category label and selected face label into the response row. Do not rely on a future join to the editable category record for historical display.

Store the reporting week explicitly. Do not recompute a historical week from `created_at` because late submissions are valid.

Store the submitted timestamp separately from created and updated times. A draft may exist for days before submission.

Store comment edits in an append-only history table. Editing the current body should not erase what was previously said.

Authorization Rules

Employee

- Read and update their own profile fields allowed by policy.
- Read their own check-ins, responses, conversations, and notifications.

- Create or edit their own open draft.
- Submit only for an allowed reporting period.
- Add a reply only to their own check-in.
- Edit or delete only comments they authored.

Manager

- All employee permissions for their own account.
- Read people assigned as direct reports or additional visibility.
- Read those employees' submitted check-ins and conversations.
- Add replies and review marks to those check-ins.
- Send a reminder only to people they may manage or view according to the product policy.
- View only permitted team-level insights.

Administrator

- Perform company administration after an explicit server-side role check.
- Use privileged database access only in narrow server-only modules.
- Never expose service-role credentials to the browser.

Analytics Definitions

Weekly overall average: Mean of the available category scores for one submitted check-in.

Category average: Mean of submitted scores for that category in the selected scope and period.

Submission count: Count of distinct submitted check-ins, not response rows.

Consecutive struggling weeks: Number of trailing submitted weeks at or below the threshold. A missing week stops the streak.

Trend: Difference between the average of a recent window and an equally sized prior window. Define a minimum difference so tiny changes remain Flat.

Needs attention: A transparent combination of low latest average, struggling streak, or meaningful downward trend. Keep component reasons available so the interface can explain the flag.

Required Quality Characteristics

- Responsive layouts that work on a phone and desktop.
- Keyboard-accessible forms, menus, dialogs, and charts with text alternatives.
- Visible loading, success, error, empty, and disabled states.
- Server validation for every mutation.
- Database constraints that back up important business rules.
- Row-Level Security and matching application checks.
- Bounded queries and useful indexes.

- Forward-only migrations.
- Retry and idempotency for external side effects.
- Structured logs without secrets.
- Unit, integration, and browser tests proportional to risk.
- Production build verification before deployment.

Part 5: The Project Constitution

Paste the following prompt into your AI coding task before Prompt 1. Ask the AI to save the rules in a project-level `AGENTS.md` or equivalent instruction file so they remain active throughout the build.

```
You are my senior engineering partner for a production-grade employee check-in platform.
We are building for one company. Treat employee check-in data as private and sensitive.

Identity and intellectual-property standard:
- The new product must have an original name and original branding selected by the user.
- Never name or brand the product PRO Pulse, ProPulse, PRO-Pulse, Shop Marketing Pros,
or any confusingly similar variation.
- Do not use prohibited source branding in package names, repository names, domains,
page titles, navigation, interface copy, email identities, metadata, seed data,
documentation, logos, or launch materials.
- If the user has not supplied an original product name, ask for one before creating
branded surfaces or production identifiers.
- Before every release, search the repository case-insensitively for prohibited source
branding. It may remain only inside the licensed source specification or license notice,
never in the Built Product.

Engineering standard:
- Build durable production software. Do not use temporary shortcuts, placeholder
architecture, or "good enough for an internal tool" decisions.
- Inspect the existing repository before proposing or editing code. Follow established
patterns unless a change is required for correctness.
- Keep modules focused and use explicit domain types. Avoid giant mixed-purpose files.
- Prefer mature libraries for authentication, validation, charts, date/time handling,
rich text, and testing.
- Use TypeScript strictly. Do not use `any` to escape a design problem.

Data standard:
- Model real relationships with normalized tables and foreign keys.
- Add constraints, indexes, and Row-Level Security policies with each schema change.
- Use forward-only, versioned migrations. Never rewrite a migration that may already
have run.
- Preserve historical truth with category-label and score-label snapshots.
- Use soft deactivation when deleting a record would damage history.
- Store explicit reporting periods. Centralize week and timezone calculations.

Security standard:
- The server is the trust boundary. Client-side hiding is never authorization.
- Every privileged operation requires an explicit server-side permission check.
- Enable Row-Level Security on every exposed table and storage bucket.
- Keep service-role keys and provider secrets server-only.
- Validate and sanitize every server input.
- Validate uploads by MIME type, size, ownership, path, and access policy.
- Never log secrets, passwords, tokens, or unnecessary private employee text.

Product standard:
```

```
- Build complete workflows, including loading, empty, pending, success, failure, and
retry states.
- Keep the weekly check-in fast and mobile-friendly.
- Make attention signals explanatory and non-diagnostic.
- Make user-facing email copy configurable where appropriate.
- Log external delivery outcomes and make failures understandable to an administrator.
- Meet basic accessibility expectations: labels, keyboard behavior, focus visibility,
contrast, semantic headings, and screen-reader text for visual scores.

Testing and delivery standard:
- Add tests for reporting-week math, check-in state transitions, permissions, analytics,
notification gating, and other business-critical logic.
- Add browser tests for login, employee submission, manager review/reply, and admin
configuration.
- Before declaring a milestone complete, run formatting, lint, typecheck, unit tests,
relevant browser tests, and a production build.
- Fix root causes. Add a regression test for every meaningful bug.
- Summarize changed files, migrations, test results, remaining risks, and any manual
setup I must perform.
- Do not claim success while required checks fail.

Collaboration standard:
- Ask a question only when the answer changes the architecture, security, or product
policy and cannot be discovered from the repository or this guide.
- Otherwise make a conservative production-grade decision, implement it, and document
it.
- Make focused commits after verified milestones so we always have a reliable recovery
point.
```

> **QUALITY BAR:** Keep this constitution in the repository. When an AI proposes a shortcut later, point it back to these rules and ask for the durable implementation.

Part 6: Copy-And-Paste Build Prompts

The companion AI-facing Markdown file contains the product specification, constitution, milestones, acceptance checks, and review prompts in a format designed for a coding harness. Upload that file first. This section is the human-readable copy of the milestone prompts so you can understand and supervise the build.

Replace bracketed placeholders before sending a prompt. Use the same coding task for the whole build so the AI retains context. At the beginning of every new task, tell the AI to read the companion specification, project constitution, and current repository documentation first.

Prompt 1: Discovery And Implementation Plan

Goal: *Turn your business decisions into a concrete build plan without writing feature code yet.*

```
Read the project constitution and inspect the entire current repository. We are building
a private weekly employee check-in platform for one company.

Business decisions:
- Original product name: [ORIGINAL PRODUCT NAME - MUST NOT BE PRO PULSE OR A CONFUSINGLY
SIMILAR VARIATION]
- Company name: [COMPANY NAME]
- Company timezone: [TIMEZONE]
```

```
- Reporting week: Sunday through Saturday
- Open submission periods: current week plus the two most recent prior weeks
- Default categories: Morale, Stress Level, Workload
- Roles: employee, manager, administrator
- Visibility: an employee has one direct manager and may have additional leadership
viewers
- Hosting: Vercel
- Database/auth/storage: Supabase
- Email: Resend
- Optional AI summaries: [ENABLED OR DISABLED]

Naming constraint: Treat the selected original product name as required input. Do not
use PRO Pulse, ProPulse, PRO-Pulse, Shop Marketing Pros, or any confusingly similar
source branding anywhere in the Built Product. If the placeholder has not been replaced
with an original name, stop and ask me to choose one before implementation.

Produce an implementation plan before editing feature code. Include:
1. Current repository assessment.
2. Target architecture and trust boundaries.
3. Route and screen map.
4. Database tables, relationships, constraints, indexes, snapshots, and RLS strategy.
5. Server-side authorization strategy.
6. Reporting-week and timezone strategy.
7. Email, in-app notification, scheduler, and idempotency strategy.
8. Testing pyramid and critical browser journeys.
9. Milestones in dependency order.
10. Environment variables and external setup, with every secret marked server-only or
public.
11. Risks and policy decisions that require my answer.

Do not build features in this prompt. You may create or update architecture and planning
documentation. Keep the design for one company and avoid unrelated product modules. Make
the plan specific enough that another engineer could execute it without guessing.
```

ACCEPTANCE CHECK: You can explain the product's roles, screens, core tables, and privacy model after reading the plan. Any unresolved question is a real business decision rather than a coding preference.

Prompt 2: Scaffold The Application

Goal: *Create a clean, runnable application shell and engineering baseline.*

```
Implement the application foundation described in the approved plan.

Requirements:
- Confirm that [PRODUCT NAME] is an original name and is not PRO Pulse or a confusingly
similar variation. Use only the approved original name in package metadata, repository
documentation, application metadata, routes, interface copy, and email placeholders.
- Use the current stable Next.js App Router, React, and TypeScript configuration
supported by official documentation.
- Use a `src` directory, Tailwind CSS, ESLint, strict TypeScript, absolute imports, and
a clear module structure.
- Add accessible UI primitives for buttons, inputs, textareas, selects, badges, tabs,
cards, dialogs, and form messages. Use a mature primitive library where useful.
- Create a quiet, work-focused visual system for [PRODUCT NAME]. It should be
responsive, readable, and professional. Avoid a marketing landing page; the first
protected screen will be the actual product.
- Add public routes for login, forgot password, reset password, and a
setup/configuration status screen.
- Add protected route groups for Dashboard, Check-in, My History, Team, Insights, My
Account, and Admin. Use temporary empty states only for route scaffolding, with no fake
data presented as real.
```

```
- Add environment validation that fails clearly for missing production settings but
allows an intentional local setup screen.
- Add scripts for dev, lint, typecheck, unit tests, browser tests, and production build.
- Add a README with local setup, environment variable names, test commands, and
deployment outline.
- Add CI that runs lint, typecheck, unit tests, and production build on pull requests
and main.

Do not implement the database domain yet. Do not put secrets in source control. Verify
the application at desktop and mobile widths. Run every available check, then create a
focused commit.
```

ACCEPTANCE CHECK: The app starts locally, public routes render, protected routes have a consistent shell, mobile navigation is usable, and CI configuration is present.

Prompt 3: Supabase Foundation And Core Schema

Goal: *Establish the database as a reliable source of truth before building forms.*

```
Implement the Supabase foundation and the first forward-only database migration.

Use current official Supabase server-side auth guidance for Next.js. Create separate
browser, cookie-based server, and narrowly scoped server-only admin clients. The
service-role client must never be imported into client code.

Create enums or constrained values for role and check-in status. Create these core
tables:
- profiles
- teams
- manager_assignments
- visibility_assignments
- checkin_categories
- weekly_checkins
- checkin_responses
- checkin_comments
- comment_edit_history
- checkin_reviews
- notifications
- notification_preferences
- reminder_schedules
- email_templates
- email_logs
- announcement_banners
- banner_dismissals
- audit_logs

Use the data blueprint in our architecture documentation. Add:
- UUID keys and timestamps.
- Foreign keys with intentional delete behavior.
- `unique(employee_id, week_start_date)` for weekly check-ins.
- One direct manager per employee.
- Duplicate and self-assignment prevention.
- Score constraint from 1 through 5.
- Category and score label snapshots in check-in responses.
- Comment edit history.
- Useful indexes for employee/week, manager, viewer, team, status, notification
recipient/read state, reminders, and email logs.
- An auth-user trigger that creates a profile safely and idempotently from approved
metadata.
- Default categories for Morale, Stress Level, and Workload with healthy score
direction.
```

```
- Updated-at triggers where useful.

Enable Row-Level Security on every exposed table. Implement and document helper
functions and policies for:
- self access;
- administrator access;
- direct-manager access;
- additional-viewer access;
- author-only comment editing;
- recipient-only notification access.

Avoid recursive RLS policies. Use security-definer helpers only when necessary, with
fixed search paths and least privilege. Generate TypeScript database types from the
schema.

Add migration verification and policy tests against a local or disposable Supabase
project. Run all checks and commit the verified migration and documentation.
```

> **ACCEPTANCE CHECK:** A database reset applies cleanly; forbidden cross-user reads fail at the database level; duplicate weekly check-ins and duplicate assignments are rejected.

Prompt 4: Authentication And Account Lifecycle

Goal: *Make sign-in, recovery, session refresh, and deactivation dependable.*

```
Implement the full authentication and account lifecycle using the existing Supabase
foundation.

Requirements:
- Email/password login with accessible errors and pending state.
- Forgot-password request with a non-enumerating success response.
- Reset-password page that handles expired or invalid recovery sessions.
- Server-side session refresh using the current official Supabase Next.js pattern.
- Protected-route middleware or proxy behavior that preserves safe return paths.
- `requireProfile` and `requireRole` server helpers.
- Immediate denial for deactivated profiles, even if the browser has an old session.
- Last sign-in tracking without creating a write loop on every request.
- My Account page for name, email change flow, password update, avatar placeholder, and
notification preferences placeholder.
- Logout from desktop and mobile navigation.
- Clear handling for configured, misconfigured, and unavailable backend states.

Add integration tests for login success/failure, protected redirects, password recovery
boundaries, and deactivated users. Do not leak whether an email address exists. Run all
checks and create a focused commit.
```

> **ACCEPTANCE CHECK:** Employee, manager, and administrator accounts can log in; a deactivated account cannot; password recovery works using the configured redirect URL.

Prompt 5: Roles, Navigation, And Permission Helpers

Goal: *Make every route and action use one coherent permission model.*

```
Implement the role-aware application shell and permission module.

Requirements:
- Employee navigation: Dashboard, Check-in, My History, My Account.
- Manager navigation adds Team and Insights.
- Administrator navigation adds the full Admin area and company-wide Insights.
- Responsive desktop and mobile navigation with visible current location and keyboard
```

```
support.
- A server-only permission module that can answer whether the current user may view an
employee, check-in, team scope, or admin surface.
- Permission behavior must match the database RLS model: self, administrator, direct
manager, or additional leadership viewer.
- Do not fetch all records and filter them in the browser.
- Reject unauthorized server actions even when called directly.
- Add a 403 or not-found experience that does not reveal private record existence.

Write table-driven unit tests for every role and relationship combination, including
self, direct report, additional viewer, unrelated employee, deactivated user, and
administrator. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** Manually changing a URL or submitting a crafted request cannot reveal another employee's data.

Prompt 6: Admin People, Teams, And Assignments

Goal: *Give an administrator safe tools to configure the company without editing the database manually.*

```
Build the Admin areas for People, Teams, and Leadership Assignments.

People requirements:
- Invite or create a user with full name, email, role, and optional team.
- Edit name, email, role, and team.
- Deactivate and reactivate; do not hard-delete people with history.
- Trigger a password-reset email.
- Separate active and deactivated lists.
- Show clear success, failure, pending, and empty states.

Team requirements:
- Create, rename, reorder, deactivate, and reactivate teams.
- Enforce normalized, case-insensitive unique names.
- Explain how team changes affect reporting.

Assignment requirements:
- Select exactly one direct manager per employee when assigned.
- Select zero or more additional leadership viewers.
- Prevent self-assignment and duplicates in both UI and database.
- Show the current saved value immediately after a change.
- Apply permission changes without requiring a redeploy.

Every mutation must require administrator permission on the server, use validated
inputs, write an audit log, and handle provider/database partial failure deliberately.
Add unit and browser tests for the core workflows. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** An administrator can configure a pilot organization from the UI; a manager cannot call the admin actions directly.

Prompt 7: Category Administration

Goal: *Make the weekly pulse configurable while preserving history.*

```
Build Admin > Categories.

For each category support:
- Stable machine key generated once.
- Editable label and explanatory prompt.
- Five optional face labels ordered from least healthy to most healthy.
- Required or optional state.
```

```
- Sort order.
- Active or hidden state.

Seed Morale, Stress Level, and Workload using the approved prompts and labels. Use a
five-face visual preview in the editor.

Rules:
- Hiding or deactivating affects future forms only.
- Past responses display stored category and score label snapshots.
- Do not delete response history when a category is hidden.
- Prevent duplicate keys and confusing duplicate active labels.
- Require at least one active category before a check-in can be submitted.
- Every action is administrator-only, validated, audited, and covered by tests.

Run a migration if the existing schema needs changes. Never edit an already-applied
migration. Verify the full suite and commit.
```

ACCEPTANCE CHECK: Rename a category after submitting sample data; the old check-in keeps the original wording while the new form uses the new wording.

Prompt 8: Reporting Period Engine

Goal: *Centralize every date decision before building the check-in workflow.*

```
Implement a pure reporting-period module and make it the only source of week
calculations.

Business rules:
- Reporting periods run Sunday through Saturday.
- Period identity is the local calendar date of the Sunday start, stored as `YYYY-MM-
DD`.
- The company timezone is [TIMEZONE]. Do not let UTC midnight roll the period early or
late.
- Employees may file the current period and the two most recent prior periods.
- Managers may review historical periods and default to the most recently completed
period.
- Period labels display the full local date range.

Provide pure functions for current period, previous period, last N periods, open
employee periods, period end, range labels, and server validation of a submitted period
value. Use an established timezone-aware date library if needed.

Write boundary tests for Saturday-to-Sunday rollover, daylight-saving transitions, year
changes, late-night UTC offsets, malformed period values, and closed periods. Replace
any scattered date arithmetic with this module. Run all checks and commit.
```

ACCEPTANCE CHECK: Changing the machine timezone or running near midnight does not change the company reporting period incorrectly.

Prompt 9: Weekly Check-In Workflow

Goal: *Build the core employee action end to end.*

```
Build the weekly check-in page and server actions using the category and reporting-
period systems.

Form requirements:
- One accessible section per active category.
- Large five-face choice control with visible category-specific labels.
- Required marker only when the category is required.
```

```
- Optional per-category comment.
- Required weekly high and weekly low.
- Optional Anything else field.
- Period selector near the save/submit controls.
- Changing the selected period preserves answers already entered in the browser.
- Save Draft and Submit as distinct actions.
- Visible saved-draft timestamp and pending state.
- Useful validation summary and field-level errors.
- Mobile-friendly stable layout with no shifting controls.

Server rules:
- Re-authenticate and authorize.
- Validate the period against the reporting-period module.
- Validate every active required category and score range.
- Require high and low only on final submission.
- Upsert the one draft for employee and week in a transaction or safe equivalent.
- Snapshot category label and selected face label.
- Enforce one check-in per employee and period.
- Lock submitted check-ins from normal editing.
- Create downstream notifications only after a successful final submission.
- Revalidate affected pages.

Isolate and unit-test the draft/submission state machine. Add browser tests for save,
resume, period switch, submit, duplicate submit, and locked history. Run all checks and
commit.
```

> **ACCEPTANCE CHECK:** Start a draft, leave, return, switch periods without losing browser input, submit, and confirm the record cannot be silently changed afterward.

Prompt 10: Employee Dashboard And My History

Goal: *Give employees immediate value from their own data.*

```
Build the personal Dashboard and My History experiences.

Dashboard:
- Greeting and current reporting-period range.
- Current check-in state: Not started, Draft in progress, or Submitted.
- Primary action that starts, continues, or views the current check-in.
- Recent trend label based on tested analytics, with a clear no-data state.
- 12-week category trend chart using submitted check-ins only.
- Accessible text equivalent for chart information.

My History:
- Load a bounded initial history window with a deliberate way to reach older records.
- 12-week category trend chart.
- Expandable submitted check-in list ordered newest first.
- Date range, overall score label, category faces and snapshot labels, category
comments, weekly high, weekly low, additional comments, submitted timestamp, and
conversation.
- Deep-link support that opens a specific check-in from a notification.
- Useful empty state for a new employee.

Do not expose another person's data. Avoid unbounded queries. Add analytics unit tests
and browser tests for empty, draft, submitted, and historical states. Run all checks and
commit.
```

> **ACCEPTANCE CHECK:** A new employee sees guidance rather than broken charts; an established employee can understand their last 12 weeks and open any recent check-in.

Prompt 11: Manager Team Dashboard

Goal: *Turn submissions into a clear leadership work queue.*

```
Build the Team dashboard for managers and administrators.

Requirements:
- Period picker that defaults to the most recently completed reporting period.
- Managers see only direct reports and additional-viewer assignments.
- Administrators may switch between their own assigned people and all active employees.
- Summary counts: Submitted, Not submitted, Awaiting your review, Needs attention.
- Filters: All, Submitted, Missing, Unreviewed, Needs attention.
- Employee rows show photo, name, email, last sign-in, status, category faces at a
glance, and relevant attention reason badges.
- A missing row has an individual Send reminder action with confirmation and result
feedback.
- Expanding a submitted row shows the full immutable check-in, conversation, review
control, and tested 12-week trend context.
- Empty filters and no-team assignments have deliberate empty states.

All counts must be derived from the same scoped dataset shown on screen. Do not use
response-row count as submission count. Queries must be bounded and indexed. Add browser
tests for each filter, permission scope, period selection, and individual reminder
state. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** Every summary number matches the visible people, and a manager cannot discover or load an unrelated employee.

Prompt 12: Review State And Two-Way Conversations

Goal: *Keep leadership follow-up attached to the check-in that caused it.*

```
Implement review marks and two-way check-in conversations.

Requirements:
- A permitted leader can mark or unmark their own review state for a submitted check-in.
- Review state is per reviewer, not a single shared boolean.
- A permitted leader can reply to a submitted check-in.
- The employee who owns the check-in can reply in the same thread.
- Messages render oldest to newest with author name, profile photo, local timestamp, and
You label for the current author.
- Authors may edit or delete only their own messages.
- Editing stores the prior body in append-only edit history and displays an Edited
indicator.
- Deletion behavior is explicit: use a tombstone when removing the row would make the
conversation misleading.
- Use a safe, limited rich-text format and render through one shared parser in the app
and email.
- Sanitize content and reject empty messages.

Authorization must be enforced by server code and RLS. Notify the other party only after
a successful new reply. Do not let notification failure roll back the saved comment. Add
permission, edit-history, ordering, and browser tests. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** Manager and employee can hold a chronological conversation; unrelated users cannot read or write it; edits remain auditable.

Prompt 13: In-App Notifications

Goal: *Make important events visible even when email is missed.*

```
Build the in-app notification system.

Events:
- A manager or additional viewer receives a notification when an assigned employee
submits.
- An employee receives a notification when a leader replies.
- A conversation participant receives a notification for a later reply, excluding the
author.
- An administrator may receive an operational notification when an announcement expires
or a scheduled job repeatedly fails.

Interface:
- Notification bell with unread count.
- Recent list with title, short message, local timestamp, unread state, and safe
destination.
- Mark one read, mark all read, and dismiss.
- Polling or a lightweight realtime approach that does not leak data or create excessive
load.
- Deep links open the correct check-in and conversation.

Rules:
- Every notification belongs to exactly one recipient.
- RLS permits only that recipient to read, update, or delete it.
- Event creation is deduplicated using a deterministic event key where retries are
possible.
- Keep notification creation in a domain service, not scattered through components.
- Notification failure must be logged and must not destroy a successfully saved check-in
or comment.

Add RLS, domain, and browser tests. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** Notification counts are private, deep links work, and refreshing or retrying an action does not create duplicates.

Prompt 14: Transactional Email And Preferences

Goal: *Add dependable email without coupling product success to provider success.*

```
Integrate Resend for transactional email using a server-only adapter.

Email types:
- Welcome or account invitation.
- Manager alert after a submitted check-in.
- Employee alert after a leader reply.
- Scheduled or individual reminder.

Requirements:
- Responsive HTML and useful plain-text fallback.
- Recognizable From name, configured Reply-To, and application deep links.
- Administrator-editable subject and intro copy using a documented allowlist of template
tokens.
- Per-user preferences by email type, with explicit defaults.
- No self-notification for actions a person performs on their own record.
- Deduplicate manager recipients across direct and additional assignments.
- Simulated mode when email is intentionally unconfigured in local development.
- Email log for every sent, failed, skipped, and simulated attempt.
- Retry only transient failures with a small bounded policy.
```

```
- Idempotency keys for retryable sends.
- Redact secrets and unnecessary private text from logs.
- Email failure never rolls back a submitted check-in or saved reply.

Build Admin > Email Content and Admin > Email Logs. Add provider-adapter tests, template
tests, preference tests, deduplication tests, and a manual test-send workflow. Run all
checks and commit.
```

> **ACCEPTANCE CHECK:** Turn a preference off, trigger the event, and confirm the email is skipped and logged; re-enable it and confirm one message is delivered and deep-linked correctly.

Prompt 15: Reminder Scheduling

Goal: *Automate participation reminders safely in local company time.*

```
Build reminder scheduling and Admin > Reminders.

Each schedule needs:
- Name.
- Enabled state.
- Local weekday and hour.
- Target period: current or most recently completed.
- Subject and intro copy.
- Last attempted and last successful run.

Execution rules:
- One protected route handler may be invoked by Vercel Cron or an external scheduler.
- Require a strong bearer secret and fail closed when missing.
- Evaluate schedules in the company timezone.
- The scheduler may run hourly; each schedule sends only once for its intended local
window.
- Target only active employees who have not submitted for the target period.
- Honor user preferences.
- Use deterministic idempotency keys so retries do not duplicate emails.
- Isolate failures per recipient and summarize the run.
- Record counts for eligible, sent, skipped, failed, and already processed.

Admin experience:
- Create, edit, enable, disable, and delete unused schedules.
- Send reminders now for testing.
- Explain which reporting period each schedule targets.
- Show simulated mode and recent outcomes.

Add timezone, daylight-saving, duplicate-run, preference, and partial-failure tests.
Document Vercel plan limitations and the external-scheduler alternative without
weakening endpoint security. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** Invoke the job twice for the same window; the second run sends no duplicate messages and reports why.

Prompt 16: Analytics And Attention Signals

Goal: *Build trustworthy calculations before visualizing them.*

```
Implement a pure analytics module with no database or UI dependencies.

Functions must support:
- Weekly overall averages.
- Category series aligned to an explicit list of reporting weeks.
- Missing weeks represented as missing, not zero.
```

```
- Score distributions from 1 through 5.
- Current-period versus previous equal-length comparison.
- Consecutive struggling weeks, broken by a missing week.
- Trend detection with documented recent/prior window sizes and a minimum meaningful
delta.
- Transparent attention reasons: low latest average, struggling streak, downward trend.
- Safe handling of empty and small samples.

Use submitted check-ins only. Preserve category label snapshots when displaying
historical data. Add exhaustive table-driven tests for empty data, one week, missing
weeks, mixed categories, boundary thresholds, and trend direction.

Document every formula in plain language so a leader can understand what the system
means. Do not hide business logic inside chart components or SQL strings. Run all checks
and commit.
```

ACCEPTANCE CHECK: A nontechnical reviewer can read the definitions and independently reproduce sample results from the test fixtures.

Prompt 17: Insights And Pulse Grid

Goal: *Create leader-level pattern recognition without losing scope or context.*

```
Build the Insights area on top of the tested analytics module.

Period controls:
- 1, 4, 8, and 12 reporting weeks.
- Previous equal-length comparison.
- Clear local date-range labels.

Administrator scope:
- Company overview.
- One card per active team.
- Optional Unassigned group only when it contains data.
- Click-through team detail.

Manager scope:
- Only the team or people the manager is authorized to view.
- Helpful explanation when the manager has no assigned scope.

Each group shows:
- Distinct check-in count.
- Overall average.
- Category averages and change from the previous period.
- 1-to-5 distribution.
- Clear empty and small-sample states.

Add a 12-week pulse grid with one row per active person and one cell per reporting week.
A cell shows the weekly average using a consistent red-to-green scale and a neutral no-
data state. Include an accessible legend and text labels. Clicking a person opens the
permitted employee history.

Use bounded, indexed queries and avoid N+1 reads. Ensure every query is scoped before
aggregation. Add permissions, aggregation, empty-state, and browser tests. Run all
checks and commit.
```

ACCEPTANCE CHECK: Company, team, and manager totals match known fixture data; no unauthorized employee contributes to a manager's aggregates.

Prompt 18: Optional AI Leadership Summaries

Goal: *Add evidence-based narrative summaries without turning AI into the source of truth.*

```
Implement optional AI-generated summaries for a selected Insights scope and period.

Requirements:
- Place the provider behind a server-only interface so it can be replaced.
- Use the current provider's supported structured-output or JSON-schema feature.
- Output exactly: summary, trend, themes, watchouts, and wins.
- Input includes aggregate counts, category averages, previous-period averages,
distributions, and bounded free-text excerpts.
- Default to de-identified input. Do not send profile names, emails, IDs, or unnecessary
metadata.
- A separate administrator setting may allow names only after an explicit policy
decision.
- Instruct the model to cite counts approximately, acknowledge small samples, avoid
diagnoses, and never invent facts.
- Validate provider output against a schema before storage or display.
- Store scope, period, source-data fingerprint, model, privacy mode, structured output,
and generated time.
- Mark a saved summary stale when the underlying source fingerprint changes.
- Generation is manual by default, permission-gated, rate-limited, and visibly pending.
- Provider failure leaves numeric Insights fully usable.

Add tests using a fake provider for prompt construction, redaction, structured
validation, stale detection, permissions, and failure handling. Do not call the real
provider in automated tests. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** Inspect the exact provider payload from a test fixture and confirm it contains no prohibited identity fields in the default privacy mode.

Prompt 19: Profile Photos, Rich Text, Images, And GIFs

Goal: *Add personality to conversations without weakening security or performance.*

```
Complete the My Account profile photo flow and optional conversation media.

Profile photos:
- Supabase Storage bucket with explicit access policy.
- Validate image MIME type, extension, decoded type where practical, dimensions, and
size.
- Use deterministic user-owned paths and safe replacement behavior.
- Store the path rather than trusting an arbitrary external URL.
- Show initials fallback and optimized display throughout the app.

Conversation images:
- Use a private bucket or signed URLs appropriate for private check-in data.
- Permit upload only to a check-in the current user may access.
- Restrict type and size, generate non-guessable paths, and enforce ownership and
visibility.
- Ensure delete/edit behavior does not orphan files indefinitely; add a cleanup
strategy.

GIF search:
- Proxy GIPHY through a server route so the API key remains private.
- Use a workplace-appropriate content rating and bounded results.
- Store only approved HTTPS media tokens or normalized metadata.
- Render the same safe rich-text and media representation in app and email.
- Gracefully hide GIF search when no API key is configured.
```

```
Add upload-policy, URL allowlist, size, MIME, permission, and rendering tests. Verify
layout on mobile and slow-loading media. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** An unrelated user cannot fetch a private conversation image by guessing its path; oversized and disguised files are rejected.

Prompt 20: Account Preferences And Announcement Banner

Goal: *Finish everyday account controls and company communication.*

```
Finish My Account and build the administrator-controlled announcement banner.

My Account:
- Update full name.
- Guided email-change flow consistent with auth provider behavior.
- Update password after recent authentication or recovery.
- Profile photo upload/remove.
- Separate notification preference toggles for reminders, manager alerts when
applicable, replies, and announcements.
- Clear success and error feedback without echoing secrets.

Announcement banner:
- Admin creates or edits one active company announcement with safe text, optional link,
start time, and expiration time.
- Display on protected pages only while active in the company timezone.
- Each user may dismiss the current announcement.
- Editing to a materially new announcement resets dismissal through a version or new
record, not by deleting history blindly.
- Admin can preview, publish, unpublish, and see scheduled/active/expired state.
- Expiration is deterministic even without a background worker.

Require server permission checks, RLS, validation, and audit logs. Add account and
banner lifecycle tests. Run all checks and commit.
```

> **ACCEPTANCE CHECK:** One employee's dismissal does not hide the banner for another, and a newly published version appears even if the prior version was dismissed.

Prompt 21: Seed Data And Demo Environment

Goal: *Make the product easy to evaluate without contaminating production.*

```
Create a development-only seed and demo guide.

Seed:
- One administrator.
- Two managers.
- At least five employees across two teams.
- Direct-manager and additional-viewer assignments.
- Twelve weeks of realistic submitted, missing, and struggling patterns.
- Draft and submitted examples.
- Review states and conversation examples.
- One deactivated person with historical records.
- Reminder schedules, preferences, logs, and one announcement example.

Rules:
- Use obviously fake names and non-deliverable example email domains unless local auth
requires a controlled address.
- Never run the demo seed automatically in production.
- Make reset and reseed deterministic.
- Include patterns that prove trend, streak, missing-data, team aggregation, and
```

```
permission behavior.
- Document demo accounts and local-only passwords in a development guide, not production
UI.

Add a safety check that refuses to seed a production-marked environment. Reset, apply
all migrations, seed, and run the integration and browser suites. Commit.
```

> **ACCEPTANCE CHECK:** A new developer can reset the development database and reproduce the same dashboards and test scenarios.

Prompt 22: Security And Privacy Review

Goal: *Put the finished product under an adversarial review before launch.*

```
Perform a security and privacy review of the complete application. Start with findings,
ordered by severity, and include file or migration references. Then implement every
confirmed in-scope fix and add regression tests.

Review at least:
- Every route and server action for authentication and authorization.
- Every RLS policy for cross-user leakage, recursion, missing write checks, and
deactivated-user behavior.
- Every service-role use for an explicit prior permission check.
- Browser bundles for accidental secrets or privileged imports.
- Environment validation and fail-closed behavior.
- Password recovery and redirect validation.
- CSRF-relevant mutations and origin assumptions.
- Input validation, stored content rendering, rich-text sanitization, and XSS.
- Storage bucket policies, MIME and size validation, object ownership, signed URL
lifetime, and cleanup.
- IDOR risk on employee, check-in, comment, notification, team, and admin IDs.
- Email header/template injection, recipient scoping, duplicate sends, and log
redaction.
- Cron authentication, replay/idempotency, and timezone boundaries.
- Query bounds, rate limits, and denial-of-service hotspots.
- Dependency vulnerabilities and supported framework versions.
- Audit coverage for administrative changes.
- Data retention and export/delete implications.

After fixes, run migration tests, RLS tests, unit tests, browser tests, dependency
audit, and production build. Document residual risks and operational controls. Do not
describe the product as launch-ready if a high-severity issue remains.
```

> **ACCEPTANCE CHECK:** Test an unrelated employee, unrelated manager, deactivated user, anonymous browser, and forged record ID against every sensitive workflow.

Prompt 23: User Experience And Accessibility Review

Goal: *Make the application feel calm, complete, and obvious to a weekly user.*

```
Perform a full product-design and accessibility review at mobile, tablet, and desktop
widths. Start with findings, then implement confirmed fixes.

Review:
- Login and recovery clarity.
- Employee time-to-complete for a weekly check-in.
- Face controls, labels, required states, keyboard selection, and screen-reader text.
- Draft, submit, locked, error, and success states.
- Period-selector clarity and protection against accidental submission.
- Dashboard and history empty states.
```

```
- Team counts, filters, collapsed-card scanability, expansion, reminder, review, and
reply workflows.
- Chart labels, legends, contrast, color-independent meaning, and text alternatives.
- Notification focus behavior and deep links.
- Admin form confirmation, destructive-action wording, and saved-state feedback.
- Mobile navigation, touch targets, text wrapping, loading states, and layout stability.
- Heading hierarchy, labels, focus order, focus visibility, reduced motion, and
contrast.

Do not turn the application into a marketing site or decorate operational pages with
unnecessary cards. Keep information dense enough for repeated leadership use while
preserving breathing room. Add browser accessibility checks and screenshot regression
coverage for critical screens. Run all checks and commit.
```

ACCEPTANCE CHECK: Complete the main employee and manager journeys using only a keyboard, then repeat on a phone-sized viewport without horizontal scrolling or overlapping text.

Prompt 24: Performance And Reliability Review

Goal: *Prepare the application for normal company growth and external-service failures.*

```
Perform a performance and reliability review of the complete application. Start with
measurable findings, implement fixes, and add regression coverage.

Review:
- Unbounded list and history queries.
- N+1 reads on Team, History, Insights, notifications, and comments.
- Missing indexes confirmed with query patterns.
- Server/client component boundaries and excessive browser JavaScript.
- Chart payload sizes and unnecessary repeated calculations.
- Avatar and conversation image optimization.
- Notification polling frequency and visibility behavior.
- Slow or failed email, AI, GIF, and storage providers.
- Transaction boundaries and partial failures.
- Duplicate form submissions, scheduler retries, and idempotency.
- Cache and revalidation correctness for private data.
- Structured logging, request correlation, and administrator-visible failures.
- Database backup, migration recovery, and restore documentation.

Add realistic load fixtures for the expected company size plus a sensible growth margin.
Keep the solution simple enough to operate, but do not accept silent failure or data
corruption. Run all checks and commit.
```

ACCEPTANCE CHECK: The core check-in submission succeeds even if email is unavailable; leaders see a useful error when an optional provider fails; large histories remain bounded and responsive.

Prompt 25: Production Deployment

Goal: *Publish a controlled production system without moving development shortcuts into live use.*

```
Prepare and execute the production deployment to Vercel and the production Supabase
project.

Before deployment:
- Confirm the working tree and intended commit.
- Run formatting, lint, typecheck, unit tests, RLS/migration tests, critical browser
tests, and production build.
- Verify every production environment variable and classify it as public or server-only.
- Confirm `.env.local`, seed passwords, test tokens, and service keys are absent from
Git history and client bundles.
```

```
- Apply forward-only migrations to a tested clone or staging project first.
- Confirm production email domain, redirect URLs, application URL, cron secret, and
scheduler configuration.
- Confirm production has no demo users or seed data.
- Document backup and remediation steps for the migration.

Deploy:
- Push the verified commit to the production branch.
- Apply production migrations in the approved order.
- Deploy the exact commit.
- Run post-deploy smoke tests for login, employee submission, manager review/reply,
notifications, email logging, Insights permissions, Admin permissions, and mobile
layout.
- Verify logs contain no secrets or unexpected private text.
- Record deployment URL, commit SHA, migration version, smoke-test result, and any
residual risk.

If any required check fails, stop and fix the root cause before calling production
complete.
```

ACCEPTANCE CHECK: Two pilot employees and one manager complete the real workflow in production, and the administrator can explain exactly which version and schema are live.

Prompt 26: Final Launch Audit

Goal: *Obtain a candid release decision rather than a celebratory summary.*

```
Act as the final release reviewer for [PRODUCT NAME]. Do not make assumptions from prior
claims. Inspect the current code, database migrations, policies, tests, configuration,
and deployed behavior.

Provide:
1. Findings ordered by severity, with evidence.
2. Go or No-Go recommendation.
3. Required fixes before company launch.
4. Important but non-blocking follow-ups with owners.
5. Test and deployment evidence.
6. Security and privacy residual risks.
7. Operations runbook gaps.
8. A concise launch-day checklist.

As part of the release audit, search the complete repository case-insensitively for PRO
Pulse, ProPulse, PRO-Pulse, and Shop Marketing Pros. The terms may remain only inside
the licensed AI specification or license notice. Treat any occurrence in application
code, package metadata, repository naming, domains, interface copy, email identity, seed
data, product documentation, or launch materials as a release-blocking defect.

Re-run the full verification suite and critical browser journeys. Check production
configuration without exposing secret values. If you find a launch-blocking issue,
implement the durable fix, add a regression test, redeploy the verified commit, and
repeat the audit. Do not call it ready until the evidence supports that conclusion.
```

ACCEPTANCE CHECK: The final report is specific, evidence-based, and willing to say No-Go.

Part 7: Reusable Review Prompts

Use these after launch whenever the product changes.

Root-Cause Bug Prompt

```
Reproduce this problem before editing code:
[PAGE, ACTION, EXPECTED RESULT, ACTUAL RESULT, AND ANY SCREENSHOT DETAILS]

Trace the behavior through UI, server action or route, permission checks, database
query, RLS, and external side effects as applicable. Identify the root cause. Implement
the durable fix, add a regression test that fails before the fix and passes after it,
run the full relevant verification suite, and summarize the evidence. Do not patch only
the visible symptom.
```

Database Migration Review Prompt

```
Review the new database migration as if it will run on a production database with
existing employee history.

Check forward-only safety, locks, backfill behavior, null transitions, constraints,
indexes, foreign keys, RLS, function search paths, idempotent remediation, historical
truth, application compatibility during deployment, and verification against a realistic
clone. Start with findings. Fix confirmed issues and add migration tests. Do not edit a
migration that has already been applied; add a new corrective migration.
```

Permission Review Prompt

```
Review this feature's complete authorization matrix for anonymous, employee self, direct
manager, additional viewer, unrelated manager, administrator, and deactivated user.
Check page loads, server actions, route handlers, storage, database RLS, exports,
notifications, and aggregate queries. Start with any leaks or mismatches. Implement
fixes and table-driven regression tests.
```

Analytics Review Prompt

```
Review the business definition and implementation of this metric: [METRIC]. State the
formula in plain language, list inclusion and exclusion rules, identify missing-data
behavior, and reproduce the result from fixtures. Check that UI labels match the formula
and that scope permissions are applied before aggregation. Fix discrepancies and add
boundary tests.
```

Email And Notification Review Prompt

```
Review the complete event flow for [EVENT]: successful domain action, recipient
resolution, preference gating, deduplication, content rendering, deep link, provider
call, retry behavior, email log, in-app notification, and failure isolation. Confirm no
self-notification or cross-user leakage. Reproduce failures with a fake provider,
implement fixes, and add tests.
```

Pre-Deploy Prompt

```
Prepare this change for production. Inspect the diff and list its behavioral surface.
Run formatting, lint, typecheck, unit tests, migration/RLS tests when relevant, critical
browser tests, and production build. Smoke-test affected desktop and mobile screens.
Confirm no secret or unrelated file is included. If everything passes, create a focused
commit, push the intended branch, deploy the exact commit, and report the URL, commit
```

```
SHA, checks, and any residual risk.
```

Part 8: Testing And Launch Checklists

Employee Workflow Checklist

- ☐ Login succeeds with correct credentials and fails safely with incorrect credentials.
- ☐ Forgot-password response does not reveal whether an address exists.
- ☐ Dashboard shows the correct local week and status.
- ☐ Required category faces can be selected by keyboard and touch.
- ☐ Custom face labels are visible and announced accessibly.
- ☐ Draft saves and resumes.
- ☐ Changing the period preserves current browser input.
- ☐ Closed or malformed periods are rejected by the server.
- ☐ Weekly high and low are required only for final submission.
- ☐ Duplicate submission is prevented.
- ☐ Submitted history keeps category and score label snapshots.
- ☐ Employee can read and reply to their conversation.
- ☐ Employee cannot inspect another employee by changing a URL.

Manager Workflow Checklist

- ☐ Team scope contains only permitted people.
- ☐ Summary counts equal visible rows.
- ☐ Period defaults to the intended completed week.
- ☐ All, Submitted, Missing, Unreviewed, and Needs attention filters are correct.
- ☐ Missing employee can receive one reminder with visible outcome.
- ☐ Expanded check-in shows the correct immutable data.
- ☐ Review mark is per reviewer.
- ☐ Reply creates one employee notification and respects email preferences.
- ☐ Edited comment retains history.
- ☐ Unrelated manager cannot load, aggregate, or search the employee.

Administrator Workflow Checklist

- ☐ User invitation, edit, role change, deactivation, and reactivation work.
- ☐ Deactivated user is denied immediately and history remains.
- ☐ Team create, rename, order, deactivate, and reactivate work.

- ☐ Direct-manager and additional-viewer assignment rules are enforced.
- ☐ Category rename changes future forms but not historical snapshots.
- ☐ Reminder schedule observes local day/hour and target period.
- ☐ Repeated scheduler invocation sends no duplicate.
- ☐ Email templates validate tokens and log outcomes.
- ☐ Announcement scheduling and per-user dismissal work.
- ☐ Audit logs identify the administrator and change.

Insights Checklist

- ☐ Submission counts count check-ins, not response rows.
- ☐ Category averages match fixture calculations.
- ☐ Missing weeks remain missing rather than zero.
- ☐ Previous-period comparison uses an equal-length window.
- ☐ Struggling streak stops on missing data.
- ☐ Attention reasons are visible and understandable.
- ☐ Manager aggregates exclude unauthorized people before calculation.
- ☐ Pulse-grid colors have labels and a neutral no-data state.
- ☐ Small and empty samples have honest wording.
- ☐ Optional AI input is de-identified by default and output is schema-validated.

Production Checklist

- ☐ Production database is separate from development.
- ☐ All migrations were tested on a representative clone or staging project.
- ☐ No demo seed ran in production.
- ☐ Production redirect URLs and application URL are correct.
- ☐ Service-role and provider secrets are server-only.
- ☐ Email sending domain is verified.
- ☐ Cron endpoint requires a strong secret and fails closed.
- ☐ Database backups and restore instructions exist.
- ☐ Logging and email logs are monitored by a named owner.
- ☐ Two-factor authentication is enabled on critical service accounts.
- ☐ Employee purpose, privacy, and use guidance has been communicated.
- ☐ The product has an original name, and prohibited source branding appears nowhere in the Built Product.

- ☐ Pilot users completed the full workflow.
- ☐ Rollback or remediation steps are documented.

First 30 Days

Week 1: Watch login, reminder, email, and permission errors daily. Ask pilot users where the workflow feels confusing.

Week 2: Review submission rates, reminder timing, and manager follow-through. Fix wording before adding features.

Week 3: Review category usefulness. Do not change labels casually; explain changes and rely on snapshots for history.

Week 4: Hold an operational review. Confirm ownership, backups, incident contacts, unresolved bugs, and the next small improvement.

> **OPERATING RULE:** A weekly check-in product earns trust through consistency. Reliable reminders, correct permissions, thoughtful responses, and honest analytics matter more than adding a long feature list.

Part 9: Official Reference Links

Ask your AI coding partner to consult current official documentation whenever setup details differ from this guide.

- Next.js App Router
- Next.js installation
- Supabase Auth with Next.js
- Supabase Row-Level Security
- Supabase Storage access control
- GitHub getting started
- GitHub Desktop and Git basics
- Vercel for Next.js
- Vercel Cron quickstart
- Resend with Next.js
- GIPHY Developer documentation

Final Advice

Build the smallest complete workflow, not the smallest amount of code. A complete workflow includes the employee action, the leader response, the privacy rule, the failure state, the test, and the operational owner.

Take one milestone at a time. Keep the working version working. Ask the AI to prove its claims. Test as each role. Protect employee trust. The result can be every bit as thoughtful and dependable as software built by a traditional product team, because the quality comes from the decisions and discipline around the code, not from how fast the first screen appeared.

Build the workflow, not a copy of the source brand. Your internal tool should reflect your own company, your own product name, and your own visual identity.