⚔️ BOSS BATTLE GRADER ⚔️

Product Specification v2.1

Agent-Native Hybrid Architecture

A Gamified Grading Interface for Franklin School

Canvas LMS · Claude Agent Loop · Atomic Tools · Retro Pixel Aesthetic

Author: Jaymes Dec, Director of Innovation
Franklin School · Dwight Schools Network
February 2026

1. Project Overview

1.1 Vision

The Boss Battle Grader transforms the grading experience into an engaging, game-like session. Each course is a level, each assignment is a sub-level, and each student submission is an encounter. The teacher earns points for speed, consistency, and thoroughness while building up student character profiles that evolve across the semester. The interface uses a retro pixel-art RPG aesthetic to create an energizing, fun environment for what is traditionally one of teaching's most tedious tasks.

1.2 Hybrid Architecture: Why Agent-Native

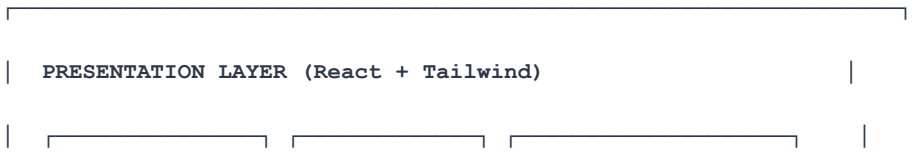
This application uses a hybrid architecture with two distinct layers, each handling what it does best:

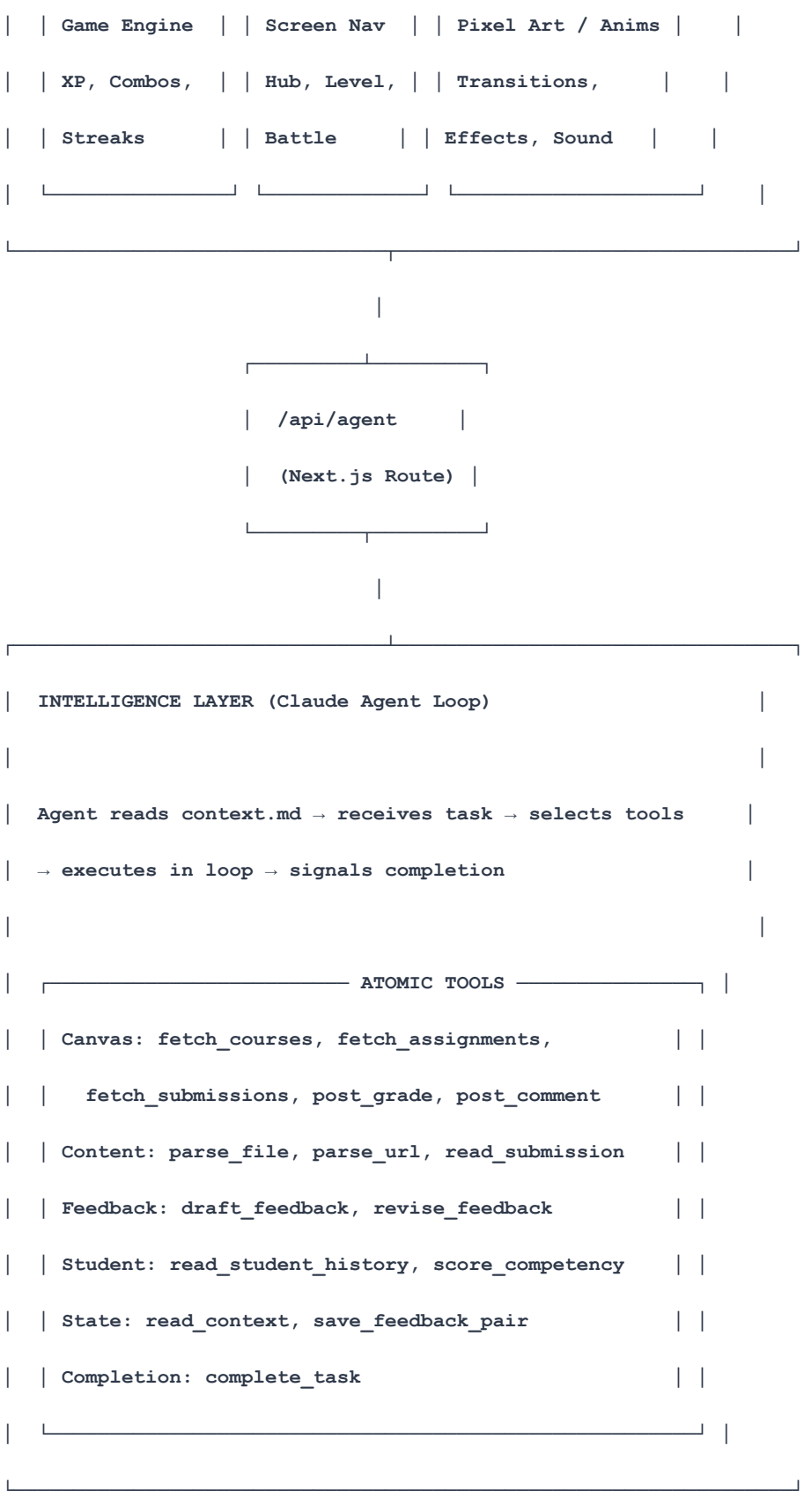
- **Presentation Layer (Traditional React):** Handles all deterministic, latency-sensitive concerns — game mechanics (points, combos, streaks), screen navigation, pixel-art animations, UI state, and rendering. No AI judgment needed here; it's pure math and state machines.
- **Intelligence Layer (Agent-Native):** Handles everything involving judgment — reading and interpreting student submissions, generating feedback, deciding what context to pull in, analyzing student growth patterns, and composing Canvas API calls. The agent operates in a loop with atomic tools, enabling emergent capabilities the developer didn't explicitly build.

This split follows four agent-native design principles:

- **Parity:** Whatever the teacher can do through the UI, the agent can accomplish through tools. The agent can fetch courses, read submissions, score competencies, draft feedback, and post grades — the same actions available in the interface.
- **Granularity:** Tools are atomic primitives (read_submission, draft_feedback, post_grade). Complex behaviors like "generate personalized feedback considering student history and rubric criteria" are achieved by the agent composing primitives in a loop, not by a monolithic function.
- **Composability:** New features = new prompts. Want the agent to flag struggling students? Write a prompt. Want it to generate parent-friendly summaries? Write a prompt. No new code required when the tools are properly atomic.
- **Emergent Capability:** The agent can handle requests the developer didn't anticipate. "Compare this student's collaboration scores across all courses" works without a dedicated feature because the agent can compose read_student_history + read_submission tools in a loop.

1.3 Architecture Diagram





1.4 Tech Stack

Layer	Technology
Frontend	Next.js 16 + React 19 + TypeScript 5 + Tailwind CSS 4
Agent Runtime	Anthropic Claude Sonnet 4 via Anthropic SDK with tool-use loop
Agent Tools	Next.js API routes wrapping Canvas API + file parsers + feedback logic
LMS Integration	Canvas REST API (bearer token auth, automatic pagination, per_page=100)
Speech-to-Text	Browser Web Speech API (built-in) or user's preferred STT tool
State	React hooks (game state) + context.md file (agent state) + localStorage (persistence)
Hosting	Vercel (or local dev via npm run dev on localhost:3000)

1.5 Environment Variables

Variable	Description
CANVAS_BASE_URL	Canvas LMS instance URL (e.g. https://franklin.instructure.com)
CANVAS_API_TOKEN	Canvas API bearer token (teacher account with grading permissions)
ANTHROPIC_API_KEY	Anthropic API key for Claude Sonnet 4 agent loop
SYSTEM_PROMPT	(Optional) Custom system prompt override for the feedback agent

2. Game Mechanics & Incentive Design

The game layer is handled by the Presentation Layer (deterministic React state for points, combos, and UI), with one critical exception: the Specificity category relies on the Intelligence Layer to verify whether teacher notes reference concrete submission content and rubric criteria. This section defines the five behavior categories the game incentivizes, the mechanics for tracking them, and the anti-patterns the system gently discourages.

Core design principle: **The game rewards quality grading behaviors — reading carefully, providing rich context, personalizing feedback, staying timely, and finishing what you start. It explicitly does not reward speed-per-submission. Grading 10 submissions in 10 minutes is not a goal; grading 10 submissions thoroughly within a week of the due date is.**

2.1 Navigation Hierarchy

Layer	Game Term	Description
Course	Level / World	Each Canvas course the teacher is enrolled in. Displayed as a pixel-art world map tile.
Assignment	Sub-Level / Dungeon	Each assignment with student submissions. Shows submission count, due date, grading progress.
Submission	Encounter / Battle	Individual student submission. The core grading interface.

2.2 Behavior Categories & Points

Points are organized into five behavior categories. Each category incentivizes a specific quality-grading habit. The Session Summary screen breaks down XP by category so the teacher can see where their strengths and growth areas are — a meta-level application of the same competency-based feedback model used for students.

👁️ Engagement: Did You Actually Read the Work?

This category ensures the teacher engages with the student's submission before grading. The core mechanic is scroll-to-unlock: the "✨ Synthesize Feedback" button is hidden until the teacher has scrolled through the submission content.

Scroll-to-Unlock Mechanic:

- **Tracking:** The Submission Viewer panel tracks scroll position relative to total scrollable height. Once the teacher has scrolled through at least 90% of the content, the "✨ Synthesize Feedback" button fades in with a pixel-art unlock animation (treasure chest opening or lock clicking open).
- **Short submissions:** If the submission content fits within a single viewport (no scrollbar), the button is available immediately — there's nothing to scroll through.
- **No timer gate:** The system does not require a minimum time on the submission. Scroll position is the sole unlock criterion. A fast reader who scrolls through everything quickly still unlocks the button.

- **Visual cue:** Before unlock, the button area shows a dimmed placeholder: "🔒 Read the submission to unlock AI feedback" with a subtle progress ring that fills as you scroll.

Action	XP	Details
Scroll through submission	+75	Awarded when 90% scroll threshold is reached and AI button unlocks
Expand student history	+50	"Lore Master" bonus: clicked to view the student's prior competency grades and trends before scoring
View assignment rubric	+25	Opened the rubric/assignment description reference panel on the right

🎯 Specificity: Did You Engage Deeply?

This category rewards the teacher for referencing concrete details from the student's submission and from the rubric criteria in their voice/text notes. The agent analyzes the teacher's notes during the feedback synthesis step and detects references to specific content. This is the one category where the Intelligence Layer participates in scoring.

How It Works: When the teacher clicks "✨ Synthesize Feedback," the agent receives the teacher's raw notes alongside the submission text and rubric criteria. As part of the synthesis loop, the agent identifies:

- **Submission references:** Did the teacher mention specific elements from the student's work? (e.g., "your user persona for the elderly demographic", "the prototype's navigation flow", "your cost analysis in slide 3")
- **Rubric/competency references:** Did the teacher mention specific TD competencies or rubric criteria by name? (e.g., "this shows strong systems thinking", "your reflexivity could go deeper here")

Specificity Rating: The Feedback Composer displays a "Specificity" meter after synthesis that shows how many concrete references the agent detected. This is informational and encouraging, not punitive.

Action	XP	Details
Reference submission content	+40 each	Per specific reference to student's work detected by agent (max 5 = 200 XP)
Reference rubric criteria	+40 each	Per specific competency or rubric criterion referenced in notes (max 5 = 200 XP)
Score all 9 competencies	+300	"Full Spectrum" bonus: scored every TD competency, not just the obvious 3–4

✍️ Personalization: Did You Make It Yours?

This category rewards the teacher for providing rich input to the AI and for editing the AI's output to reflect their own voice. The system tracks edit distance between the AI draft and the final posted version, and rewards voice note quality.

Edit Distance Tracking: The system computes the edit distance (by sentence) between the AI-generated feedback and the teacher's final version. Edits are weighted:

- **Adding new sentences:** Highest weight. Indicates the teacher contributed original thought.
- **Modifying existing sentences:** Medium weight. Indicates the teacher refined the AI's language.
- **Deleting sentences:** Low weight. Indicates editorial judgment but less original contribution.

Personal Touch Meter: A visual meter in the Feedback Composer shows the current personalization level based on edit distance. Three tiers:

- ⚫ **Untouched (0% change):** No personalization points. The AI draft was posted as-is.
- 🟡 **Reviewed (1–20% change):** Minor tweaks. Partial personalization points.
- 🟢 **Personalized (>20% change):** Substantial edits. Full personalization points.

Action	XP	Details
Provide voice/text notes	+100	"Mentor's Voice" bonus: entered any feedback notes before synthesis (not blank)
Rich voice notes	+100	Notes exceed 50 words (encourages substantive input, not just "good job")
Edit AI feedback (Reviewed)	+75	1–20% edit distance: teacher reviewed and tweaked the draft
Edit AI feedback (Personalized)	+200	>20% edit distance: teacher substantially rewrote or added to the draft
Post to Canvas	+100	Actually posted the grade and feedback to Canvas LMS

⏰ Timeliness: Are You Keeping Up?

This category rewards grading submissions promptly after the assignment due date. It replaces the speed-per-submission bonuses from v1 with a days-since-deadline model. The goal is not to grade fast within a session, but to grade consistently so students get timely feedback.

How It Works: When grading a submission, the system calculates the number of days between the assignment's due_at date and now. This determines a timeliness multiplier applied to all points earned on that submission.

Days Since Due Date	Multiplier	Label
0–7 days	1.5x	⚡ Swift Justice — graded within a week
8–14 days	1.25x	✅ Timely — graded within two weeks
15–21 days	1.0x	⏳ Standard — no bonus, no penalty
22–28 days	0.75x	🟡 Overdue — students have been waiting

29+ days	0.5x	🔴 Critical — feedback may no longer be actionable

Design note: **The multiplier never drops to zero. Even very late grading earns some XP. The point is encouragement, not punishment. The Level Select screen shows a color-coded urgency indicator on each assignment based on how many days have passed since its due date, naturally surfacing what needs attention first.**

🏆 Completeness: Did You Finish?

This category awards large bonuses for completing entire batches of grading, encouraging follow-through rather than partial sessions.

Achievement	XP	Details
🚪 Dungeon Cleared	+500	All submissions for an assignment graded. The big milestone for each batch.
🌍 World Complete	+1000	All assignments in a course fully graded. Major achievement.
⚔️ Balanced Adventurer	+300	Weekly bonus: graded submissions in 2+ different courses this week. Prevents neglecting one class.
📤 Canvas Synced	+250	All graded submissions in a session were posted to Canvas (not just saved locally).

2.3 Combo System (Revised)

The combo system rewards sustained grading sessions without incentivizing speed. The combo counter increments per submission graded, and the idle timeout is generous to allow thoughtful reading.

- **Combo counter:** Increments by 1 for each submission graded (after clicking "⚔️ Submit Grade"). Resets if the teacher is idle for more than 15 minutes between submissions. The timeout is intentionally long — reading a complex submission, recording thoughtful voice notes, and editing AI feedback should not break the combo.
- **Combo multiplier:** 1.0x base + 0.05x per combo level (max 1.5x at combo 10). Intentionally lower than v1 (was 2.0x) because the main incentive should come from behavior categories, not from the combo. Applied to Engagement and Personalization points only (not Timeliness or Completeness, which have their own multipliers).
- **Combo labels:** Combo 3 = "✨ ROLLING", Combo 5 = "⚡ FLOW STATE", Combo 8 = "🔥 ON FIRE", Combo 10 = "💥 UNSTOPPABLE"
- **Visual feedback:** Animated streak bar with pixel particles. Color changes and screen-edge glow at higher combos. Subtle pixel flames at "On Fire" and above.
- **No speed bonus:** There is deliberately no bonus for grading a submission quickly. The combo rewards consistency (keep going) not velocity (go fast).

2.4 Anti-Pattern Nudges

The system gently surfaces prompts when it detects patterns that suggest disengagement. These never take away points — they withhold bonus multipliers and offer a friendly nudge. The tone is light and game-appropriate, delivered through a pixel-art companion character.

Detected Pattern	Nudge
Skipping scroll	Teacher attempted to grade without scrolling through the submission (e.g., jumped straight to competency buttons). Companion says: "🤔 Hmm, looks like you haven't read through this one yet. Want to take a look?" The AI feedback button remains locked.
Empty voice notes	Teacher clicked Synthesize Feedback with a blank or very short (<10 words) notes field. Companion says: "💬 Your notes help me write better feedback! Even a sentence or two makes a big difference." AI button still works, but Personalization XP is reduced.
Unchanged AI feedback	Teacher posted AI feedback without any edits (0% edit distance) 3+ times in a row. Companion says: "✍️ I'm flattered, but students can tell when feedback is personal. Want to add your touch?" No points withheld, but the Personal Touch meter stays at ⚫ Untouched.
Rapid-fire grading	3+ submissions graded in under 90 seconds each without voice notes or AI feedback. Companion says: "🏃 Whoa, you're moving fast! Are you sure you want to skip feedback on these?" No combo break, but the Engagement scroll-unlock still applies per submission.

2.5 Session Summary & Rewards

Displayed after exiting a grading session. The summary breaks down XP by behavior category, showing the teacher where their strengths are and where they could improve — mirroring the competency-based feedback model used for students.

- **Category breakdown:** XP displayed per category (Engagement, Specificity, Personalization, Timeliness, Completeness) with bar charts and percentages. Highlights the teacher's strongest category.
- **Session stats:** Submissions graded, total session time, assignments touched, courses touched.
- **Achievement badges:** Pixel-art badges earned during the session. Examples:

Badge	Criteria
👁️ Deep Reader	Scrolled through and spent meaningful time on every submission
🎯 Sharpshooter	Average 3+ specific references per submission in voice notes

✍️ Personal Touch	Edited AI feedback on every submission (no “Untouched” posts)
🌟 Full Spectrum	Scored all 9 competencies on every submission
⚡ Swift Justice	All submissions graded within 7 days of due date
🚪 Dungeon Cleared	All submissions for the assignment graded and posted
💬 Mentor’s Voice	Provided voice/text notes on every submission in the session
⚔️ Balanced Adventurer	Graded across 2+ courses this week
🔍 Second Look	Went back and revised a grade/feedback after initial submission (rewards reflection)
🏆 Lore Master	Reviewed student history on every submission before grading

- **Student highlights:** Agent-surfaced patterns like “Sofia improved 2 levels in Systems Thinking since last assignment” or “3 students scored A+ in Agency.” The agent composes read_student_history calls to generate these — this is an emergent capability, not a hardcoded feature.
- **Personal best tracking:** Compared against your own past sessions. Shows whether this session was your highest XP, highest personalization rate, fastest turnaround time, etc.

3. Screen-by-Screen Specifications

3.1 Hub Screen (World Map)

Purpose

Landing screen. Displays all Canvas courses as explorable worlds/levels on a pixel-art world map.

Data Source

Agent tool: fetch_courses → Canvas GET /api/v1/courses with include[]=total_students&include[]=teachers&include[]=term&state[]=available. The agent filters to teacher enrollments.

UI Elements

- **World map background:** Pixel-art landscape (dark, atmospheric) with each course as a distinct biome/location tile.
- **Course tiles:** Name, pixel-art icon, student count, progress indicator (assignments fully graded / total).
- **Teacher avatar:** Pixel character on the map. Clicking a course "walks" the avatar there with a brief animation.
- **Session stats bar:** Top bar with lifetime XP, current session duration, last session's score.

3.2 Level Select Screen (Assignment List)

Purpose

Shows all assignments within the selected course as sub-levels/dungeons.

Data Source

Agent tool: fetch_assignments → Canvas GET /api/v1/courses/:course_id/assignments with include[]=submission_summary. The agent also fetches rubric data for assignments that have one.

UI Elements

- **Assignment list:** Vertical scrolling list styled as dungeon doors. Each shows name, due date, submission count (graded/total), status.
- **Status indicators:** 🔴 Ungraded, 🟡 In Progress, ✅ Cleared, ⚫ No Submissions.
- **Course info header:** Course name, student count, term, overall grading progress bar.

3.3 Battle Screen (Grading Interface)

This is the core of the application. It is the most complex screen and should receive the most development attention. The Battle Screen is where the Presentation Layer and Intelligence Layer meet: the UI renders the panels, game HUD, and animations, while the agent handles submission reading, feedback generation, and Canvas posting.

Layout: Three-Panel Design

Panel	Width	Contents

Left Panel	~25%	Student Character Card: pixel avatar, 9-axis radar chart, competency stat bars with trend arrows, student queue.
Center Panel	~50%	Submission viewer (scrollable student work) + Feedback Composer (voice/text input, AI generate button, editable preview).
Right Panel	~25%	TD Competency scoring grid: 9 competencies with A+ through F buttons. Assignment description and rubric for reference.

Top Bar: Streak & Stats HUD

Persistent heads-up display (Presentation Layer only):

- **Streak bar** with animated pixel particles and combo-level color changes
- **Points counter** with "slot machine" roll-up animation on point gains
- **Timer** for current submission (starts when submission loads)
- **Progress** "4 / 14 Graded" with pixel-art progress bar

Left Panel: Student Character Card

- **Student name and avatar:** Pixel-art avatar via DiceBear's pixel-art style API (student name as seed). Canvas display name.
- **Radar chart:** 9-point radar chart. Grade values: A+ = 100, A = 90, B = 80, C = 70, D = 55, F = 30. Updates in real time as teacher assigns grades.
- **Stat bars with trends:** All 9 competencies as horizontal bars with grade label and historical trend arrow (↑ → ↓). Trend data comes from the agent calling read_student_history.
- **Student queue:** Scrollable list of all students for this assignment. Name, graded/ungraded status, checkmark for completed.

Center Panel: Submission Viewer & Feedback Composer

Upper section — Submission Content: **Scrollable area showing the student's work. The agent uses read_submission + parse_file/parse_url tools to extract and prepare the content. For text submissions, render HTML body. For file attachments, show extracted text. For URLs, display link and attempt to fetch content. Tab to view submission metadata (submitted_at, attempt, late status).**

Lower section — Feedback Composer: **This is where the Intelligence Layer is most visible to the teacher:**

1. **Voice/Text Input Box:** Large text area for raw feedback. Mic icon activates browser Web Speech API as built-in fallback. Placeholder: "Speak or type your feedback notes..."
2. **"✨ Synthesize Feedback" Button:** Triggers the agent loop. The agent receives the teacher's raw notes, assignment description, rubric criteria, submission text, and competency grades. It composes tools (draft_feedback → revise_feedback) in a loop until satisfied. Pixel-art loading animation during generation.
3. **AI Feedback Preview:** Editable rich text area showing the agent's output. Teacher reviews, edits, and refines. Includes "Regenerate" and "Post to Canvas" buttons. Teacher edits are captured as before/after pairs via save_feedback_pair for the learning loop.

Right Panel: TD Competency Scoring

- **Competency cards:** Each of the 9 competencies with name, emoji icon, and grade selector buttons (A+ through F). Selected grade highlights with competency's color.

- **Rubric descriptors:** On hover/click, shows full descriptor text from the Design & Tech row for the selected grade level.
- **Quick reference:** Collapsible section showing assignment description and Canvas rubric.

Action Buttons

- **"⚔️ Submit Grade":** Saves locally, awards XP, triggers combo animation, advances to next student. Does NOT post to Canvas.
- **"📤 Post to Canvas":** Agent calls post_grade + post_comment tools to push to Canvas. Awards bonus XP.
- **"⏭️ Skip":** Next student without grading. Resets combo.
- **"🚪 Exit Dungeon":** Returns to Level Select. Triggers Session Summary if any grading was done.

3.4 Session Results Screen

- **XP breakdown** with animated counter
- **Time stats and submissions graded**
- **Achievement badges** (pixel-art icons)
- **Student highlights:** Agent-generated insights via emergent tool composition

4. The 9 Transdisciplinary Competencies

These competencies form the core of Franklin School's assessment philosophy. In the Boss Battle Grader, they serve as the 9 "stats" on each student's character card. The rubric descriptors below are from the Design & Tech row. The full rubric PDF contains subject-specific rows for all disciplines. The system should support switching subject rows in a future phase.

4.1 🤝 Collaboration

Definition: **Works productively and respectfully with others to achieve shared goals.**

Grade	Design & Tech Descriptor
A+	Integrates diverse ideas to develop superior outcomes and facilitates inclusion.
A	Shares leadership, listens well, and improves collective work.
B	Contributes ideas and completes role cooperatively.
C	Relies on peers for direction; passive in group decisions.
D	Disengaged or struggles with group communication.
F	Refuses to participate or blocks group progress.

4.2 💬 Storytelling / Communication

Definition: **Communicates ideas clearly, creatively, and appropriately for audience and purpose.**

Grade	Design & Tech Descriptor
A+	Communicates design choices with impact, purpose, and user awareness.
A	Explains processes and reasoning effectively.
B	Presents basic structure and tools used.
C	Shares limited insight into choices.
D	Communicates process poorly or incompletely.
F	No communication or context provided.

4.3 🪞 Reflexivity

Definition: **Reflects critically on learning, decisions, and assumptions.**

Grade	Design & Tech Descriptor
A+	Critiques design choices and future improvements.
A	Recognizes user feedback and adapts design.
B	Comments on product development.
C	Gives surface-level review of product.
D	Fails to recognize iterative process.

F	No reflection on design process.

4.4 💛 Empathy / Perspective Taking

Definition: **Demonstrates understanding and respect for others' perspectives and experiences.**

Grade	Design & Tech Descriptor
A+	Centers design on user wellbeing, accessibility, and dignity.
A	Applies user feedback meaningfully.
B	Incorporates basic user-centered thinking.
C	Acknowledges user needs superficially.
D	Overlooks key user perspectives.
F	Design disregards or harms user interests.

4.5 📚 Knowledge-Based Reasoning

Definition: **Applies disciplinary and interdisciplinary knowledge to solve problems.**

Grade	Design & Tech Descriptor
A+	Bases design on deep research and interdisciplinary knowledge.
A	Makes informed decisions using data.
B	Uses basic knowledge to support ideas.
C	Design lacks rationale.
D	Unsupported or random design choices.
F	No clear thinking or evidence shown.

4.6 🔮 Futures Thinking

Definition: **Envisions and prepares for multiple and preferred futures.**

Grade	Design & Tech Descriptor
A+	Designs systems anticipating future user or planetary needs.
A	Creates sustainable or futuristic products thoughtfully.
B	Explores improvements with guidance.
C	Overlooks long-term consequences.
D	Short-term or impractical design.
F	Ignores future needs.

4.7 🕸️ Systems Thinking

Definition: **Identifies and understands interconnections within and across systems.**

Grade	Design & Tech Descriptor
A+	Designs with multiple systems and users in mind; maps effects.
A	Considers cause/effect and user interdependence.
B	Designs address some interactions or usability.
C	Limited systems view in planning.
D	Ignores system implications.
F	No integration of systems thinking.

4.8 🔄 Adaptability

Definition: **Responds constructively to change and ambiguity.**

Grade	Design & Tech Descriptor
A+	Responds to critique with inventive, positive revisions.
A	Updates plans based on results and testing.
B	Changes parts of design with prompting.
C	Resists changes or iterates minimally.
D	Ignores design problems.
F	Abandons work when revision needed.

4.9 🚀 Agency

Definition: **Takes initiative and ownership of learning and actions.**

Grade	Design & Tech Descriptor
A+	Proactively leads, iterates designs, and proposes innovative solutions.
A	Manages project stages effectively and demonstrates independent ideas.
B	Completes designs with moderate independence and creativity.
C	Follows guidance to complete basic designs.
D	Requires assistance to maintain progress.
F	Avoids or neglects assigned work.

5. Atomic Tools Catalog

Instead of a rigid API-route-per-action architecture, the Intelligence Layer exposes a set of atomic tools that the Claude agent composes in a loop. Each tool performs one conceptual action. Judgment — what to call, in what order, and how to interpret results — stays in the agent's prompt.

5.1 Canvas Tools

These tools wrap the Canvas REST API. All calls go through server-side Next.js routes to protect the bearer token.

Tool Name	Returns	Description
fetch_courses	Course[]	GET /api/v1/courses. Returns active courses where user has teacher enrollment. Includes student count, term, teachers.
fetch_assignments	Assignment[]	GET /api/v1/courses/:id/assignments. Returns assignments with rubric data, submission summaries, due dates. Input: course_id.
fetch_submissions	Submission[]	GET /api/v1/courses/:id/assignments/:id/submissions. Returns submissions with user data, comments, rubric assessments. Input: course_id, assignment_id.
post_grade	Result	PUT /api/v1/.../submissions/:user_id. Posts submission[posted_grade] and optionally rubric_assessment. Input: course_id, assignment_id, user_id, grade, rubric_map.
post_comment	Result	POST submission comment with text_comment. Input: course_id, assignment_id, user_id, comment_text.

5.2 Content Tools

Tool Name	Returns	Description
read_submission	string	Extracts readable text from a submission. Handles text entries (HTML body), file attachments, and URL submissions. Delegates to parse_file or parse_url as needed.
parse_file	string	Parses a file attachment from a Canvas URL. Supports DOCX (via mammoth), PDF (via pdf-parse-new), and plain text. Input: file_url, content_type.
parse_url	string	Extracts text from a URL submission. Handles Google Docs URLs and generic web pages. Input: url.

5.3 Feedback Tools

These are the judgment-heavy tools. Unlike Canvas tools (which are thin wrappers), these involve Claude's reasoning. The agent decides when and how to use them.

Tool Name	Returns	Description

draft_feedback	FeedbackDraft	Generates initial feedback based on: teacher_notes, assignment_description, rubric_criteria, submission_text, competency_grades, student_name. Returns { summary, strengths[], growthAreas[], nextSteps[], formattedFeedback }.
revise_feedback	FeedbackDraft	Takes an existing draft and revision instructions. The agent may call this multiple times in a loop until satisfied. Input: current_draft, revision_notes.
save_feedback_pair	void	Stores a before/after pair (AI draft vs. teacher-edited version) to the local feedback learning store. Used to improve future generations via few-shot injection.

5.4 Student Tools

Tool Name	Returns	Description
read_student_history	CompetencyStat[]	Fetches a student's grading history across assignments in a course. Returns per-competency grade history with dates and trend direction. Input: course_id, user_id.
score_competency	void	Records the teacher's grade for one competency on the current submission. This is the teacher's action passed to the agent for state tracking. Input: competency_id, grade.

5.5 State & Completion Tools

Tool Name	Returns	Description
read_context	ContextData	Reads the context.md file (see Section 6). Returns the agent's current knowledge: user preferences, recent activity, available data, guidelines.
read_preferences	Preferences	Reads the teacher's learned feedback preferences (distilled from feedback pairs). Returns style rules and recent examples for few-shot injection.
complete_task	void	Explicit completion signal. The agent calls this when done with the current task. Includes success/failure status and optional notes. Never use heuristics for completion detection.

5.6 Tool Design Principles

One conceptual action per tool. Judgment stays in prompts. There is no "analyze_and_grade" tool — the agent reads, reasons, and grades through separate tool calls.

Keep primitives available. Domain tools (draft_feedback, revise_feedback) are shortcuts, not gates. The agent always has access to the underlying read/write primitives.

CRUD completeness. For every entity (courses, assignments, submissions, grades, comments, feedback), verify the agent has Create, Read, Update, and Delete capabilities.

Dynamic capability discovery. The agent can call fetch_courses to discover what courses exist, then fetch_assignments to discover assignments, then fetch_submissions to discover students. It builds its understanding at runtime.

Explicit completion. The agent calls complete_task when done. No heuristic completion detection. The agent loop runs until this signal.

6. The Agent Loop & context.md

6.1 The context.md Pattern

At the start of each agent invocation, the system prompt includes a dynamically generated context.md that gives the agent situational awareness. This replaces hardcoded context injection with a living document the agent reads every time.

context.md Template

```
# Boss Battle Grader — Agent Context

## Who I Am
Grading assistant for Franklin School. I help the teacher grade
student submissions by composing atomic tools in a loop.

## What I Know About This Teacher
- Name: {teacher_name}
- Courses: {course_list}
- Feedback style: {learned_preferences}
- Tends to: {distilled_patterns}

## Current Session
- Course: {current_course_name}
- Assignment: {current_assignment_name}
- Student: {current_student_name} ({graded_count}/{total_count})
- Competency grades so far: {grades_map}
- Teacher's voice notes: {raw_notes}

## What Exists
- {course_count} active courses
- {assignment_count} assignments in current course
- {submission_count} submissions for current assignment
- {feedback_pairs_count} stored feedback examples
```

```
## My Guidelines

- Always reference specific parts of the submission in feedback

- Feedback should be encouraging but honest

- Use the 9 TD competencies as the framework

- Match the teacher's voice (see learned preferences)

- When unsure about a grade, note the ambiguity


## Available Tools

Canvas: fetch_courses, fetch_assignments, fetch_submissions,

  post_grade, post_comment

Content: read_submission, parse_file, parse_url

Feedback: draft_feedback, revise_feedback, save_feedback_pair

Student: read_student_history, score_competency

State: read_context, read_preferences, complete_task
```

6.2 Agent Loop Flow

The agent operates in a tool-use loop managed by the Anthropic SDK. The frontend sends a task to the /api/agent route, which runs the loop server-side:

1. **Frontend sends task:** e.g., { task: "generate_feedback", context: { course_id, assignment_id, user_id, teacher_notes, competency_grades } }
2. **Server builds system prompt:** Injects the dynamically generated context.md + task-specific instructions.
3. **Agent loop begins:** Claude receives the prompt and makes its first tool call.
4. **Tool execution:** Server executes the tool (Canvas API call, file parse, etc.) and returns the result.
5. **Agent decides next step:** Based on the tool result, Claude either calls another tool or calls complete_task.
6. **Loop continues** until the agent signals completion. The result is streamed back to the frontend.

6.3 Example Agent Loops

Task: Generate Feedback for a Submission

The frontend triggers this when the teacher clicks "✨ Synthesize Feedback":

```
Agent receives: "Generate feedback for Alex Chen on MVP Prototype Pitch"


Loop:
```

```
1. read_context()          → loads teacher preferences, current state
2. read_submission(...)    → extracts Alex's submission text
3. read_student_history()  → gets Alex's prior competency grades
4. read_preferences()      → loads teacher's feedback style + examples
5. draft_feedback(...)     → generates initial feedback with all context
6. [Agent reviews draft internally, notices it's too generic]
7. revise_feedback(...)    → refines with specific submission references
8. complete_task(feedback)  → returns final feedback to frontend
```

Note: steps 6–7 are the agent's judgment. A rigid pipeline would skip this self-correction. The agent loop allows it to iterate until the feedback meets quality standards.

Task: Surface Student Highlights for Session Summary

After the teacher exits a grading session, the agent is asked to generate highlights:

```
Agent receives: "Analyze grading session and surface interesting patterns"

Loop:
  1. read_context()              → loads session data (who was graded)
  2. read_student_history(Sofia)  → checks Sofia's trend
  3. read_student_history(Kai)    → checks Kai's trend
  4. read_student_history(Maya)   → checks Maya's trend
  5. [Agent identifies patterns across students]
  6. complete_task(highlights)     → returns insights to frontend
```

This is an emergent capability — the student highlights feature was never explicitly coded. The agent composes existing tools (read_student_history) in a loop to produce insights. Adding new kinds of highlights requires changing the prompt, not the code.

Task: Unexpected Request

A future scenario demonstrating composability:

```
Teacher asks: "How has the class's collaboration skills changed
               since the beginning of the semester?"
```

```
Loop:
  1. fetch_assignments(course_id)   → gets all assignments
  2. fetch_submissions(assign_1)    → gets first assignment submissions
  3. [Loops through students, reads history for collaboration]
  4. [Aggregates trends across the class]
  5. complete_task(analysis)         → returns class-wide trend report
```

No new feature was built. The agent composed existing tools to answer a question the developer never anticipated. This is the ultimate test of agent-native architecture.

6.4 Feedback Learning Loop

When the teacher edits AI-generated feedback before posting, the system captures the learning:

Capture: When the teacher modifies the AI draft and clicks “Post to Canvas,” the frontend calls save_feedback_pair with the original AI output and the teacher’s edited version.

Store: Pairs are saved to a local JSON file (feedback-pairs.json) with assignment context, rubric, and timestamp.

Inject: When the agent calls read_preferences, it gets the most relevant past examples (matched by assignment type and competencies) for few-shot injection into draft_feedback.

Distill: Periodically (e.g., every 20 pairs), the agent analyzes patterns across edits and distills explicit style rules (“softens critical language,” “adds specific next steps,” “references class discussions”) that are added to context.md.

7. Visual Design: Retro Pixel-Art Aesthetic

The entire application uses a retro pixel-art RPG aesthetic inspired by 16-bit era games (SNES/Genesis era Final Fantasy, Chrono Trigger, Earthbound). This is the core identity, not a skin.

7.1 Color Palette

Role	Hex	Usage
Background (deep)	#0A0A1A	Main background, dungeon floors
Background (mid)	#1A1A2E	Panel backgrounds, card backgrounds
Surface	#2D2D44	Buttons, input fields, borders
Accent (primary)	#00FFAA	XP gains, success states, positive actions
Accent (gold)	#FFD93D	Points counter, streak highlights, achievements
Accent (danger)	#FF6B6B	Low grades, combo resets, warnings
Text (primary)	#E8E8F0	Body text on dark backgrounds
Text (muted)	#8888AA	Secondary text, labels, timestamps

7.2 Typography

Display / headings: Press Start 2P (Google Fonts) — pixel-perfect retro game font.

Body / UI: JetBrains Mono or IBM Plex Mono — clean monospace for readable text.

7.3 Animations & Effects

Screen transitions: Pixel-dissolve or scanline wipe between screens.

Point gains: Floating “+200 XP” text like RPG damage numbers.

Combo effects: Screen edge glow intensifies with combo. Pixel flames at “On Fire.”

Grade selection: Satisfying “chip” click with real-time radar chart update.

Student transition: Brief “encounter!” splash with name and avatar.

Sound effects (optional): 8-bit sounds via Tone.js. Toggleable.

7.4 Pixel Art Assets Needed

World map background (tileable, dark atmospheric)

Course biome tiles (6–8 variants)

Teacher avatar sprite (idle, walking, celebrating)

Student avatars (DiceBear pixel-art API with name seed)

Achievement badges (8–12 pixel-art icons)

UI elements (pixel buttons, panels, borders, progress bars)

Dungeon door sprites (for assignment entries)

8. Data Models (TypeScript Interfaces)

Game State (Presentation Layer)

```
interface GameState {
  currentScreen: 'hub' | 'level' | 'battle' | 'results';
  selectedCourseId: number | null;
  selectedAssignmentId: number | null;
  sessionXP: number;
  combo: number;
  streak: number;
  lastGradeTimestamp: number | null;
  sessionStartTime: number;
  gradedSubmissionIds: string[];
}
```

Student Character

```
interface StudentCharacter {
  canvasUserId: number;
  displayName: string;
  avatarUrl: string;
  competencyStats: Record<CompetencyId, CompetencyStat>;
}


interface CompetencyStat {
  currentGrade: Grade | null;
  history: { assignmentId: number; grade: Grade; date: string }[];
  trend: 'improving' | 'steady' | 'declining' | 'new';
}


type Grade = 'A+' | 'A' | 'B' | 'C' | 'D' | 'F';
type CompetencyId = 'collaboration' | 'communication' | 'reflexivity'
  | 'empathy' | 'knowledge' | 'futures' | 'systems'
  | 'adaptability' | 'agency';
```

Agent Tool Types (Intelligence Layer)

```
// Tool input/output schemas
```

```
interface AgentTask {
  task: 'generate_feedback' | 'surface_highlights' | 'post_grades'
    | 'analyze_trends' | 'custom';
  context: Record<string, any>;
  prompt?: string;  // For custom/composable tasks
}

interface AgentToolResult {
  success: boolean;
  output: string;
  shouldContinue: boolean;  // false = agent is done
}

interface FeedbackDraft {
  summary: string;
  strengths: string[];
  growthAreas: string[];
  nextSteps: string[];
  formattedFeedback: string;  // Ready for Canvas posting
}

interface FeedbackPair {
  id: string;
  assignmentId: number;
  studentId: number;
  originalDraft: string;
  teacherEdited: string;
  competencyGrades: Record<CompetencyId, Grade>;
  timestamp: string;
}
```

Grading Submission

```
interface GradingSubmission {
  submissionId: number;
```

```
  student: StudentCharacter;
  submissionContent: string;
  submissionType: 'text' | 'url' | 'file' | 'none';
  attachments: { filename: string; url: string; contentType: string }[];
  submittedAt: string;
  late: boolean;
  attempt: number;
  competencyGrades: Record<CompetencyId, Grade>;
  teacherNotes: string;
  aiFeedback: FeedbackDraft | null;
  postedToCanvas: boolean;
  gradedAt: number | null;
  timeSpentSeconds: number;
}
```

9. Multi-Teacher Architecture

The Boss Battle Grader is designed to scale from a single-teacher prototype to a school-wide platform. This section covers authentication, database infrastructure, the faculty leaderboard, and the cross-class student competency system. The deployment target is Replit, which provides hosting, a built-in PostgreSQL database (via Neon), and a secrets manager for environment variables in a single workspace.

9.1 Deployment: Replit

The app runs as a Next.js project on Replit with a public URL shareable with colleagues. Replit provides:

Hosting: Next.js runs directly in the Replit workspace. Replit Deployments provides a persistent public URL (e.g., boss-battle-grader.replit.app) with automatic HTTPS.

PostgreSQL: Replit's built-in PostgreSQL integration (powered by Neon) provisions a database directly from the workspace. Connection string is automatically available as a Replit Secret. No separate database provider needed.

Secrets manager: ANTHROPIC_API_KEY, GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, CANVAS_BASE_URL, DATABASE_URL, NEXTAUTH_SECRET, and TOKEN_ENCRYPTION_KEY are stored in Replit Secrets (equivalent to environment variables). Never committed to code.

ORM: Prisma for type-safe database queries. Prisma generates TypeScript types from the schema, matching the project's existing type system. Migrations run via prisma migrate deploy.

9.2 Authentication: Google OAuth + Canvas Token

Teachers sign in with their Google accounts (which they already use daily for Google Workspace at Franklin) and then do a one-time Canvas API token setup. This approach requires zero Canvas admin configuration — no Developer Key, no OAuth app registration — and gets colleagues up and running immediately.

Login Flow

1. **Teacher clicks "Sign In with Google":** App redirects to Google's OAuth2 consent screen via NextAuth.js. Teacher selects their Franklin Google Workspace account.
2. **Session created:** Google returns the teacher's profile (name, email, avatar). NextAuth.js creates a database-backed session. The teacher is now logged in.
3. **Canvas setup prompt (first login only):** If no Canvas token is stored for this teacher, the app shows a setup screen with step-by-step instructions for generating a Canvas API token.
4. **Teacher generates Canvas token:** In Canvas, go to Account → Settings → scroll to Approved Integrations → click "+ New Access Token" → name it "Boss Battle Grader" → copy the token.
5. **Teacher pastes token:** Back in the Boss Battle Grader, the teacher pastes their token into the setup form. The app validates it (test GET /api/v1/users/self) and stores it encrypted in the database tied to their Google account.
6. **Done:** On subsequent logins, the teacher just clicks "Sign In with Google" and their stored Canvas token is loaded automatically. No re-pasting.

Canvas Token Setup Screen

This screen appears on first login (and is accessible from Settings for token rotation). It should be friendly and pixel-art-themed to match the game aesthetic:

Step-by-step wizard: Visual guide with numbered steps and screenshots showing exactly where to click in Canvas. Written for non-technical teachers.

Token validation: On paste, the app immediately calls Canvas GET /api/v1/users/self with the token. If valid, show a green checkmark with the teacher's Canvas name ("✅ Connected as Jaymes Dec"). If invalid, show a clear error and retry.

Token expiry handling: Canvas personal access tokens don't expire unless the teacher sets an expiration date or revokes them. The setup wizard recommends leaving the expiration blank. If a token stops working (Canvas returns 401), the app shows a "Reconnect Canvas" prompt.

Token Security

Encryption at rest: Canvas tokens are encrypted with AES-256-GCM before storage in the database. The encryption key (TOKEN_ENCRYPTION_KEY) is stored as a Replit Secret, never in code.

Per-teacher scoping: Each teacher's Canvas token only has access to their own courses and students, as determined by Canvas's built-in permissions. The app cannot access more than the teacher themselves could in Canvas.

Token never sent to client: Canvas tokens are decrypted server-side only. The frontend never sees the raw token — all Canvas API calls go through the app's server-side agent and tool routes.

Auth Library: NextAuth.js

NextAuth.js manages the Google OAuth flow, session lifecycle, and database persistence. It uses the Prisma adapter to store sessions, users, and accounts in PostgreSQL. Configuration lives in src/app/api/auth/[...nextauth]/route.ts with the built-in Google provider.

Google Cloud Console Setup

A one-time setup by Jaymes to create Google OAuth credentials:

Project: Create a project in Google Cloud Console (e.g., "Boss Battle Grader").

OAuth consent screen: Internal (Google Workspace) or External. App name, support email, authorized domains.

Credentials: Create OAuth 2.0 Client ID. Authorized redirect URI: https://boss-battle-grader.replit.app/api/auth/callback/google

Secrets: Store GOOGLE_CLIENT_ID and GOOGLE_CLIENT_SECRET as Replit Secrets.

Future Upgrade Path: Canvas OAuth2

If the app grows beyond Franklin or if token management becomes burdensome, the auth system can be upgraded to Canvas OAuth2 (where teachers click "Sign In with Canvas" and the app receives tokens automatically). This would require a Canvas Developer Key from the admin. The database schema already supports this — the canvas_api_token_encrypted column would be replaced with OAuth access/refresh tokens managed by NextAuth.

9.3 Database Schema

The database uses PostgreSQL (Replit/Neon) with Prisma ORM. The schema extends the existing TypeScript interfaces with relational tables for multi-teacher persistence. Canvas user IDs serve as the foreign key linking teachers and students across the system.

Entity Relationship Overview

```
teachers ──┬── grading_sessions (game XP, badges)
```

```
        |
        ├── competency_scores (per student per assignment)
        |          |
        |          └── students (cross-class profiles)
        |
        ├── feedback_pairs (AI draft vs. teacher edit)
        |
        └── teacher_preferences (learned feedback style)
```

Core Tables

`teachers`

Teacher identity comes from Google OAuth. Canvas API token is stored encrypted and linked to the Google account.

Column	Type	Description
id	UUID (PK)	Internal primary key
google_email	TEXT	Google Workspace email (unique, from OAuth)
display_name	TEXT	Teacher's name from Google profile
avatar_url	TEXT	Google profile photo (or generated pixel art)
canvas_api_token_encrypted	TEXT	AES-256-GCM encrypted Canvas personal access token (null until setup)
canvas_user_id	INTEGER	Canvas user ID (fetched on token validation via GET /api/v1/users/self)
canvas_token_valid	BOOLEAN	Whether the stored Canvas token last validated successfully
lifetime_xp	INTEGER	Aggregated total XP across all sessions
level	INTEGER	Teacher's game level (derived from XP thresholds)
created_at	TIMESTAMP	First Google login
last_active_at	TIMESTAMP	Most recent session

`students`

Shared student records. A student appears once in this table regardless of how many teachers grade them. Canvas user ID is the unique key.

Column	Type	Description
id	UUID (PK)	Internal primary key
canvas_user_id	INTEGER	Canvas user ID (unique)
display_name	TEXT	Student's name from Canvas
avatar_url	TEXT	Pixel avatar URL (DiceBear seed = name)

first_seen_at	TIMESTAMP	First time any teacher graded this student

competency_scores

The central table for cross-class student tracking. Every time a teacher scores a competency through the Boss Battle Grader, a row is written here AND the grade is posted to Canvas. This table is the source of truth for holistic student profiles.

Column	Type	Description
id	UUID (PK)	Internal primary key
student_id	UUID (FK)	References students.id
teacher_id	UUID (FK)	References teachers.id — who assigned this grade
competency_id	TEXT	One of the 9 TD competency IDs (e.g., 'systems', 'agency')
grade	TEXT	A+, A, B, C, D, or F
canvas_course_id	INTEGER	Which Canvas course this score came from
canvas_assignment_id	INTEGER	Which Canvas assignment
subject_area	TEXT	e.g., 'Design & Tech', 'English' — for rubric row context
graded_at	TIMESTAMP	When this grade was assigned

grading_sessions

One row per grading session (from entering the Battle Screen to exiting/completing). This is the leaderboard source data.

Column	Type	Description
id	UUID (PK)	Internal primary key
teacher_id	UUID (FK)	References teachers.id
canvas_course_id	INTEGER	Which course was graded
canvas_assignment_id	INTEGER	Which assignment was graded
submissions_graded	INTEGER	How many submissions were graded in this session
xp_engagement	INTEGER	XP earned in the Engagement category
xp_specificity	INTEGER	XP earned in the Specificity category
xp_personalization	INTEGER	XP earned in the Personalization category
xp_timeliness	INTEGER	XP earned in the Timeliness category
xp_completeness	INTEGER	XP earned in the Completeness category
xp_total	INTEGER	Sum of all category XP for this session
max_combo	INTEGER	Highest combo reached during the session
badges_earned	TEXT[]	Array of badge IDs earned (e.g., ['deep_reader', 'swift_justice'])
duration_seconds	INTEGER	Total session duration
started_at	TIMESTAMP	Session start time

ended_at	TIMESTAMP	Session end time

`feedback_pairs`

Stores AI draft vs. teacher-edited pairs for the feedback learning loop. Per-teacher — each teacher's editing patterns train their own feedback preferences.

Column	Type	Description
id	UUID (PK)	Internal primary key
teacher_id	UUID (FK)	References teachers.id
canvas_assignment_id	INTEGER	Assignment context
canvas_student_id	INTEGER	Student context
original_draft	TEXT	AI-generated feedback
teacher_edited	TEXT	Teacher's final version after edits
edit_distance	FLOAT	Computed edit distance (0.0 to 1.0)
competency_grades	JSONB	The grades assigned on this submission
created_at	TIMESTAMP	When the pair was stored

`teacher_preferences`

Distilled feedback style rules derived from a teacher's feedback pairs. Updated periodically as new pairs accumulate.

Column	Type	Description
id	UUID (PK)	Internal primary key
teacher_id	UUID (FK)	References teachers.id (unique — one row per teacher)
style_rules	JSONB	Distilled rules (e.g., {"softens_criticism": true, "includes_next_steps": true})
recent_examples	JSONB	Best few-shot examples for injection into draft_feedback prompts
pairs_analyzed	INTEGER	How many feedback pairs have been analyzed so far
updated_at	TIMESTAMP	Last distillation run

9.4 Faculty Leaderboard

The leaderboard aggregates grading_sessions data across all teachers. It's designed to celebrate quality grading behaviors, not create toxic competition. The RPG theme helps frame it as collaborative progression rather than a zero-sum ranking.

Leaderboard Design Principles

Normalize for fairness: Teachers with more students or courses naturally accumulate more raw XP. The leaderboard shows XP per submission graded as the primary ranking metric, so a teacher with 12 students is on equal footing with one who has 40.

Category spotlights: Instead of one monolithic ranking, show category-specific leaders: highest Personalization rate, best Timeliness, most consistent Engagement. This mirrors the "everyone has strengths" philosophy of TD competency assessment.

Collaborative tone: Frame as a guild/party, not a competition. "Franklin Faculty Guild — Level 12" with a collective XP bar that fills as all teachers grade. Individual stats visible but framed as contributions to the collective.

Time-scoped views: This week, this month, this semester, all time. Prevents early adopters from permanently dominating.

Leaderboard Views

View	Description
🏆 Overall Ranking	Teachers ranked by XP per submission graded (normalized). Shows teacher avatar, name, level, total submissions graded, and XP/submission. Filterable by time period.
👁️ Engagement Leaders	Ranked by average Engagement XP per submission. Who reads most carefully?
🎯 Specificity Leaders	Ranked by average Specificity XP. Who provides the most concrete, evidence-based feedback?
✍️ Personalization Leaders	Ranked by average edit distance and voice note usage. Who adds the most personal touch?
⏰ Timeliness Leaders	Ranked by average days-to-grade after due date. Who returns feedback fastest?
🏆 Completeness Leaders	Ranked by percentage of assignments fully cleared. Who finishes what they start?
🏰 Faculty Guild	Collective view: total submissions graded school-wide, collective XP, guild level, school-wide streaks ("Franklin graded 200 submissions this week!").

Teacher Profiles

Each teacher has a public-within-the-school profile card (pixel-art character card, mirroring the student character card design):

Pixel-art avatar (DiceBear with teacher name as seed, or uploaded)

Level and XP bar (lifetime progression)

5-axis radar chart of average scores across the 5 behavior categories (Engagement, Specificity, Personalization, Timeliness, Completeness)

Badge collection (all earned badges displayed)

Recent activity ("Cleared Business Proposal Assignment — 3 days ago")

9.5 Cross-Class Student Competency Tracking

This is the most pedagogically valuable feature of the multi-teacher system. When multiple teachers score the same student's 9 TD competencies across different courses, the system builds a holistic student character card that no single teacher could see on their own.

How It Works

1. **Dual write on grade:** When a teacher grades a competency and posts to Canvas, the app simultaneously writes a row to the competency_scores table. Canvas remains the source of truth for official grades; the database is the source of truth for the cross-class competency picture.

2. **Student lookup by Canvas ID:** Students are identified by their Canvas user ID, which is consistent across all courses. When Teacher A grades student #4521 in Design & Tech and Teacher B grades student #4521 in English, both rows are linked to the same student record.
3. **Holistic character card:** The Student Character Card (left panel of Battle Screen) queries all competency_scores for this student, not just the current teacher's course. The radar chart shows aggregated scores with a recency weight — recent grades count more than older ones.
4. **Per-course drill-down:** Teachers can expand the character card to see how a student scored in other classes. "Collaboration: A in English (Ms. Park), B in Design & Tech (Mr. Dec), C in Science (Mr. Torres)." This reveals patterns invisible to any single teacher.

Privacy & Access Scoping

Enrollment-based access: Teachers can only view holistic character cards for students enrolled in their current Canvas courses. This is enforced by checking the teacher's Canvas course enrollments against the student's.

Teacher attribution: Each competency score shows which teacher assigned it. Teachers see their colleagues' names and courses alongside the grades, fostering cross-departmental visibility.

No student self-access (initially): Students do not log into the Boss Battle Grader. Their character cards are internal tools for faculty. A future phase could expose read-only student dashboards.

Aggregate Views

The agent can compose cross-class queries as emergent capabilities:

"Show me all students whose Adaptability has declined across 2+ courses" — the agent queries competency_scores, groups by student and competency, and identifies declining trends.

"Which competencies are weakest school-wide this semester?" — aggregate across all teachers and students. Could reveal that Futures Thinking is consistently the lowest-scored competency.

"Compare this student's self-assessment in Reflexivity with their teacher scores" — future integration with student self-assessment data.

Design note: These queries are not pre-built features. They emerge from the agent composing database read tools with its reasoning capabilities. This is the ultimate demonstration of agent-native architecture at the school level — faculty can ask questions about student growth that nobody anticipated, and the agent composes the answer from atomic tools.

9.6 Infrastructure Summary

Component	Technology
Hosting	Replit Deployments (Next.js, public URL with HTTPS)
Database	PostgreSQL via Replit (Neon-powered), ~6 tables + NextAuth tables
ORM	Prisma (type-safe queries, migrations, schema management)
Identity	Google OAuth2 via NextAuth.js (Google Workspace login)
Canvas Access	Per-teacher personal access tokens, encrypted at rest (AES-256-GCM), one-time paste setup
Session Storage	Database-backed sessions via NextAuth Prisma adapter
Secrets	Replit Secrets: ANTHROPIC_API_KEY, GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, CANVAS_BASE_URL, DATABASE_URL, NEXTAUTH_SECRET, TOKEN_ENCRYPTION_KEY
AI	Anthropic Claude Sonnet 4 (shared API key, server-side, per-teacher context)

10. Development Roadmap

10.1 Phase 1: Atomic Tools & Agent Loop (Foundation)

Build the Intelligence Layer foundation. Single-teacher mode with environment variable token.

1. Implement all Canvas tools (fetch_courses, fetch_assignments, fetch_submissions, post_grade, post_comment) as Next.js API route handlers.
2. Implement content tools (read_submission, parse_file, parse_url) with DOCX/PDF/URL support.
3. Build the /api/agent route: system prompt injection, tool-use loop, context.md generation, completion signal handling.
4. Implement feedback tools (draft_feedback, revise_feedback) with Claude Sonnet 4.
5. Implement student tools (read_student_history, score_competency).
6. Test the agent loop end-to-end: fetch a submission, read it, generate feedback, post to Canvas — all through the agent composing tools.

10.2 Phase 2: Battle Screen (Core Grading UI)

Build the Presentation Layer for the core grading experience:

1. Three-panel Battle Screen layout with submission viewer, feedback composer, and competency scoring grid.
2. Student Character Card with radar chart and stat bars.
3. TD competency grade buttons with real-time radar chart updates.
4. Voice/text feedback input with Web Speech API integration.
5. “Synthesize Feedback” button wired to the agent loop with streaming response display.
6. Editable AI feedback preview with “Post to Canvas” button.
7. Scroll-to-unlock mechanic on submission viewer.

10.3 Phase 3: Game Layer

1. Five behavior category points system (Engagement, Specificity, Personalization, Timeliness, Completeness).
2. Combo mechanics with 15-minute idle timeout.
3. Anti-pattern nudge system with companion character.
4. Hub screen (world map) with course tiles.
5. Level select screen with timeliness urgency indicators.
6. Session results screen with category breakdown and badge awards.

10.4 Phase 4: Pixel Art & Polish

1. Retro color palette and typography (Press Start 2P + JetBrains Mono).
2. Screen transition animations.
3. Point gain and combo visual effects.
4. Student encounter splash screens.
5. Pixel-art UI elements and asset integration.

6. Optional 8-bit sound effects via Tone.js.

10.5 Phase 5: Multi-Teacher & Database

This phase transforms the app from a single-teacher prototype to a shared school-wide platform:

1. Set up PostgreSQL on Replit with Prisma schema and migrations.
2. Set up Google OAuth2 via NextAuth.js (Google Cloud Console project, credentials, Replit Secrets).
3. Build Canvas token setup screen: step-by-step wizard, token paste, validation via GET /api/v1/users/self, AES-256-GCM encryption, and storage.
4. Refactor Canvas tools to pull per-teacher encrypted tokens from database instead of environment variable.
5. Migrate game state from localStorage to database-backed grading_sessions.
6. Implement dual-write for competency scores (Canvas + database).
7. Build faculty leaderboard with normalized rankings and category spotlights.
8. Build teacher profile cards with 5-axis radar chart.
9. Build cross-class student character cards with enrollment-scoped access.
10. Migrate feedback_pairs and teacher_preferences from local JSON to database.

10.6 Phase 6: Advanced & Emergent

1. Faculty Guild collective view with school-wide stats.
2. Agent-powered cross-class queries (“show declining students,” “weakest competencies school-wide”).
3. Subject-specific rubric row switching (Design & Tech, English, Science, etc.).
4. Upgrade to Canvas OAuth2 (eliminates token paste, requires Canvas Developer Key from admin).
5. Google Drive OAuth for Google Docs submission parsing.
6. Student self-assessment integration.
7. Read-only student dashboard for viewing their own character card.
8. Custom agent prompts: teacher types ad-hoc questions and the agent composes tools to answer.

11. File Structure

The file structure reflects the three-layer architecture: Intelligence Layer (agent and tools), Presentation Layer (React components), and the new Data Layer (Prisma schema, auth, and database queries).

File	Purpose
INTELLIGENCE LAYER	
`src/app/api/agent/route.ts`	Main agent endpoint: builds system prompt, runs tool-use loop, streams response
`src/lib/agent/context.ts`	Generates context.md dynamically from current state + database
`src/lib/agent/loop.ts`	Agent loop runner: Anthropic SDK tool-use loop with completion signal
`src/lib/agent/prompts.ts`	System prompt templates for different task types
`src/lib/tools/canvas.ts`	Canvas tools: fetch_courses, fetch_assignments, fetch_submissions, post_grade, post_comment
`src/lib/tools/content.ts`	Content tools: read_submission, parse_file, parse_url
`src/lib/tools/feedback.ts`	Feedback tools: draft_feedback, revise_feedback, save_feedback_pair
`src/lib/tools/student.ts`	Student tools: read_student_history, score_competency (now dual-writes to DB)
`src/lib/tools/state.ts`	State tools: read_context, read_preferences, complete_task
`src/lib/tools/registry.ts`	Tool registry: maps tool names to implementations for the agent loop
DATA LAYER (NEW)	
`prisma/schema.prisma`	Prisma schema: teachers, students, competency_scores, grading_sessions, feedback_pairs, teacher_preferences + NextAuth tables
`src/app/api/auth/[...nextauth]/route.ts`	NextAuth.js config: Google OAuth provider, Prisma adapter, session callbacks
`src/app/api/canvas-token/route.ts`	Canvas token setup endpoint: validate token, encrypt (AES-256-GCM), store in DB
`src/lib/db.ts`	Prisma client singleton (connection pooling for Replit/Neon)
`src/lib/auth.ts`	Auth helpers: getServerSession, requireAuth middleware, Canvas token decryption and retrieval
`src/lib/crypto.ts`	Token encryption/decryption utilities (AES-256-GCM with TOKEN_ENCRYPTION_KEY)
`src/lib/leaderboard.ts`	Leaderboard queries: normalized rankings, category spotlights, guild stats
`src/lib/student-profiles.ts`	Cross-class student queries: holistic character card, enrollment-scoped access
PRESENTATION LAYER	

`src/app/page.tsx`	Main app: screen routing, game state management (useReducer)
`src/app/login/page.tsx`	Login screen: “Sign In with Google” button (pixel-art themed)
`src/app/setup/page.tsx`	Canvas token setup wizard: step-by-step instructions, token paste, validation feedback
`src/components/HubScreen.tsx`	World map with course tiles
`src/components/LevelSelect.tsx`	Assignment list for selected course (with timeliness indicators)
`src/components/BattleScreen.tsx`	Core grading interface (3-panel layout)
`src/components/ResultsScreen.tsx`	Session summary with category breakdown and agent highlights
`src/components/Leaderboard.tsx`	Faculty leaderboard with tabs for category views and guild stats
`src/components/TeacherProfile.tsx`	Teacher character card with 5-axis radar chart and badges
`src/components/CharacterCard.tsx`	Student profile with cross-class radar chart and per-course drill-down
`src/components/RadarChart.tsx`	Reusable 9-axis (student) or 5-axis (teacher) radar chart (SVG)
`src/components/CompetencyScorer.tsx`	Grade selector buttons for 9 competencies
`src/components/FeedbackComposer.tsx`	Voice/text input + AI generate + editable preview + Personal Touch meter
`src/components/StreakBar.tsx`	Combo/streak HUD component
`src/components/SubmissionViewer.tsx`	Renders student submission content (with scroll-to-unlock tracking)
`src/components/StudentQueue.tsx`	Scrollable student list for assignment
SHARED	
`src/lib/game.ts`	Points, combos, streak logic, behavior category scoring (pure functions)
`src/lib/competencies.ts`	9 TD competency definitions and rubric descriptors
`src/lib/cache.ts`	Client-side caching utilities with TTL
`src/types.ts`	Shared TypeScript interfaces (GameState, StudentCharacter, LeaderboardEntry, etc.)
`context.md`	Agent context template (see Section 6)

End of Specification v2.1

Agent-native where it matters. Pixel-perfect where it counts. Faculty-wide where it helps. ⚔️