LARA MVP1 Implementation Plan

Complete Technical Specification

Version: 1.0
Date: January 2026
Status: Final Specification

Table of Contents

---

1. Executive Summary

1.1 Project Purpose

LARA (Learning Assessment & Response Assistant) is a formative feedback engine designed to support classroom-based writing assessment. The MVP1 version focuses on a single, clear learning loop where students submit written responses, receive structured feedback aligned with research-based principles, and complete one revision cycle.

1.2 Core Value Proposition

LARA's "Three-Question Guarantee":
Every feedback screen must answer:
- Where am I going? (goal/success criteria)
- How am I going? (current performance)
- Where to next? (specific next steps)

This is the fundamental product differentiator and must be enforced in every feedback interaction.

1.3 Key Design Constraints

Session-Based Architecture:
- No persistent student accounts
- All student data is session-scoped and ephemeral
- Teacher maintains external ID-to-name mapping
- Sessions are temporary classroom activities (like Kahoot or Mentimeter)

Teacher Oversight:
- Teacher reviews and approves all feedback before students see it
- Teacher can edit feedback with guided suggestions for improvement
- Teacher controls session lifecycle (start/end)

Pedagogical Integrity:
- All feedback must be specific, descriptive, and improvement-focused
- Forbidden language detection (vague praise, ability judgments, peer comparison)
- Balanced feedback types (task, process, self-regulation)
- Mandatory next step selection before student can proceed

---

2. System Overview

2.1 System Type

LARA is a web-based, real-time classroom activity platform for formative writing assessment.

Think of it as: Kahoot or Mentimeter, but for writing feedback instead of quizzes or polls.

Not: Video conferencing, homework platform, or persistent learning management system.

2.2 Primary Users

Teachers:
- Create writing tasks with success criteria
- Start classroom sessions with shareable class codes
- Monitor live student submissions
- Review, edit, and approve AI-generated feedback
- View class-level patterns and insights
- End sessions when activity concludes

Students:
- Join sessions using 6-digit class codes (no account required)
- Submit written responses to assigned tasks
- Receive structured feedback (after teacher approval)
- Choose one next step for improvement
- Submit one revision
- Complete session (no data retained afterward)

2.3 Core Workflow

```
Teacher Creates Task → Starts Session → Generates Class Code
                                ↓
Students Join → Write Response → Submit → Wait for Feedback
                                ↓
AI Generates Feedback → Teacher Reviews/Approves → Student Receives
                                ↓
Student Selects Next Step → Revises Response → Completes
                                ↓
Teacher Ends Session → Data Cleared (except summary for teacher)
```

---

3. Core Principles & Non-Negotiable Requirements

3.1 LARA's 10 (+1) Feedback Principles

These principles (from "LARA Feedback Principles" document) guide every design decision:

Principle 1: Formative by design, not just comments

Feedback must exist within a learning loop: clarifying goals, checking evidence of learning, and informing next actions. Not just tidying up finished work.

Implementation Impact:
- Goal must be visible before feedback appears
- Feedback must include actionable next steps
- Revision opportunity must be immediate

Principle 2: Always answer three questions

Every feedback output must clearly address:
- Where am I going? (goal/success criteria)
- How am I going? (current performance)
- Where to next? (specific next steps)

Implementation Impact:
- Non-collapsible goal box at top of feedback screen
- Structured strengths and growth areas
- Mandatory next steps with selection requirement

Principle 3: Focus on task, process, and self-regulation

Prioritize feedback about the work itself, strategies used, and how students manage learning. Avoid fixed-ability judgments.

Implementation Impact:
- Each feedback comment must be tagged with type (task/process/self_reg)
- Validation requires at least one of each type
- Forbidden language filter blocks ability praise

Principle 4: Be specific, descriptive, and improvement-focused

No vague "good job" or "add more detail". Every critique must point to concrete features and suggest improvement.

Implementation Impact:
- Specificity validation checks for content anchors
- Vague comment detection triggers teacher warnings
- Growth areas must include: work reference + reason + direction

Principle 5: Timely, concise, and usable now

Feedback designed to be given during or soon after learning, kept short, focused on 1-2 high-leverage priorities.

Implementation Impact:
- Feedback generated in same session as submission
- Mobile-optimized (CTA ≤30 characters)
- Focused on immediate revision, not future tasks

Principle 6: Build feedback literacy and student agency

Prompt students to interpret feedback, self-assess, choose actions, and reflect on next steps.

Implementation Impact:
- Mandatory next step selection (student must choose)
- Reflection prompts included
- Model examples available (not rewrites)

Principle 7: Emotionally safe and motivational

Language is respectful, normalizes mistakes, balances strengths with improvements.

Implementation Impact:
- Forbidden language includes: harsh criticism, ability judgments, peer comparisons
- Strengths always included
- Growth-oriented framing

Principle 8: Tightly aligned to clear intentions and criteria

Every message explicitly connects to learning intentions and success criteria.

Implementation Impact:
- Teacher-defined criteria displayed exactly as written

- Universal Learning Expectations when no criteria provided
- Never infer or fabricate content-specific goals

Principle 9: Support dialogue and multiple feedback roles

Outputs can be used for teacher→student, peer, and self-feedback. Help teachers adjust instruction.

Implementation Impact:
- Teacher review interface
- Class-level analytics for instructional decisions
- (Peer/self modes: post-MVP)

Principle 10: High impact, low workload

Target high-leverage feedback aspects, deliver short, high-quality guidance.

Implementation Impact:
- Default to 2-4 targeted points
- One-click criteria selection for teachers
- Automated first-draft feedback generation

Principle 11: LARA does not remember students. Teachers may.

Session-scoped IDs only. No PII storage. Teacher maintains external mapping.

Implementation Impact:
- Ephemeral session storage
- Automatic data deletion after session end
- Teacher can export session summaries with session IDs

3.2 Non-Negotiable Product Requirements

From "MVP1 Core Feedback Requirements (Non-Negotiable)" document:

A. Start with the Goal (Where am I going?) - REQUIRED

Rule: No feedback may be displayed unless a goal box is rendered first.

Goal Box Requirements:

- Fixed position at top of screen (non-collapsible)
- Always visible during feedback review
- Must display either:
- Teacher's exact criteria (if provided), OR
- Universal Learning Expectations (if no teacher criteria)

Universal Learning Expectations (Default):

Main text: "For this task, success means answering the question directly, explaining your ideas clearly with evidence, and signposting your thinking."

Expandable detail: "This means addressing what the question asks, using relevant examples, organising your ideas logically, and guiding the reader with clear topic sentences and linking words."

Critical Rules:

- Teacher-defined criteria hold full authority (never override or expand)
- LARA must NOT guess, infer, or fabricate task-specific goals
- Universal expectations describe HOW to think/communicate, not WHAT content to include

Forbidden Goal Examples:

- ❌ "Compare two political systems"
- ❌ "Describe causes of World War I"
- ❌ "Evaluate the economic impacts"
- ❌ "Assess the role of government in..."

These are content-specific and curriculum-specific. Universal expectations must remain skill-based.

B. Describe Current Performance (How am I going?) - Structured, Never Blob Text

Rule: Feedback must appear in two structured cards, never as unstructured text.

Card 1: "What's working well"

- Specific and descriptive
- Tied to success criteria or universal expectations
- Focused on task and process, not ability
- Examples:
- ✓ "Your explanation of longshore drift is clear and uses the diagram effectively."
- ✓ "You've organised your argument logically, moving from cause to effect."

Card 2: "What needs attention"

- Must identify specific part of student's work

- Must explain why it needs attention
- Must guide toward improvement without rewriting
- Examples:
- ✓ "Your second paragraph mentions 'impact', but you don't include any examples. Add one to show the effect clearly."
- ✓ "You describe the cause accurately, but you don't explain why it matters. Add one sentence linking cause and consequence."

Forbidden Vague Phrases (unless made specific):
- ❌ "Good job"
- ❌ "Add more detail"
- ❌ "Be clearer"
- ❌ "Explain more"
- ❌ "Work on your structure"

Allowed Specific Versions:
- ✓ "Add more detail about the environmental impact, using one concrete example."
- ✓ "Be clearer by defining the term 'migration' before you use it."
- ✓ "Improve your structure by grouping related ideas into a single paragraph."

Mandatory Specificity Check:
Every growth-area comment must contain at least one content anchor from student's work:
- Paragraph reference (e.g., "paragraph 2", "your opening sentence")
- Concept or term student used (e.g., "migration", "coastal erosion")
- Quote or phrase from their work (e.g., "when you wrote 'impact'")
- Noun phrase describing their work (e.g., "your explanation of the cause")

C. Offer 1-3 Concrete Next Steps (Where to next?) - Mandatory Choice

Rule: Student cannot exit feedback screen until one next step is selected.

Next Step Requirements:
Each next step must include ALL THREE elements:
1. **Action verb** (add, revise, define, explain, restructure, link, clarify, check)
2. **Specific target** (paragraph, sentence, concept, cognitive verb, signposting, evidence)
3. **Success indicator** (brief statement of what will be better)

Examples (Good):
- ✓ "Add one example of environmental impact to paragraph 2 so the reader can see the effect clearly."
- ✓ "Define the term 'migration' before using it so your explanation is clearer."
- ✓ "Add a topic sentence to paragraph 3 to show what idea the paragraph is about."
- ✓ "Explain why this cause matters by adding one sentence linking it to an impact."

Examples (Forbidden):
- ❌ "Improve your work."
- ❌ "Add more detail."
- ❌ "Try again."
- ❌ "Be clearer."

Traceability Requirement:
Each next step must map directly to a growth area. If growth area identifies missing evidence, next step must address adding evidence.

Rationale Requirement:
Each next step must include a "Why this helps" micro-explanation (5-12 words):
- "This will make it clearer how your response answers the question."
- "This helps the reader understand your main idea."
- "This makes your reasoning clearer and easier to follow."

ATQ and Signposting Triggers (Mandatory):
- If student did NOT answer the question (ATQ issue) → at least one next step MUST support ATQ alignment
- If signposting is missing → at least one next step MUST support signposting

Cognitive Load Rule:
Next steps written at Year 8 reading level (unless teacher overrides). No jargon, no dense sentences.

Model Examples Available:
After selecting next step, student can click "Show me an example (not from my work)". System provides:
- Model topic sentence
- Model explanation sentence
- Model linking phrase
- Model example of evidence use

Critical: Model examples must be generic, NOT rewrites of student's specific work.

Revision Success Check:
When student returns after editing, system must:
- Detect whether selected next step was completed
- Show indicator:
- ✓ "You completed your next step!"
- or
- ⚠ "It looks like this step isn't visible in your revision yet. Want help?"
- Ask: "Would you like feedback on your revised answer?"

D. Balance Task/Process/Self-Regulation (Feedback Type Mix)

Rule: Every feedback screen must include a balanced mix of all three types.

Type Definitions:

Task Feedback: About quality, correctness, or completeness of student's work.
- Examples:
- "Your explanation of coastal erosion in paragraph 2 is accurate and clearly linked to the diagram."
- "You correctly identified two causes of population change."
- Characteristics: Specific to content, grounded in criteria, descriptive only (not evaluative)

Process Feedback: About strategies, structure, and methods student used.
- Examples:
- "You grouped similar ideas together; this made your argument easier to follow."
- "Using an example to clarify your point strengthened your explanation."
- Characteristics: Highlights transferable strategies, supports visible thinking

Self-Regulation Feedback: About how student plans, checks, monitors, and evaluates their work.
- Examples:
- "Before submitting next time, read the question again and check whether each paragraph answers it (ATQ)."
- "Re-read your work out loud to notice missing links between ideas."
- Characteristics: Encourages independence, focuses on behaviors not personality

Balance Rules:
- At least ONE Task comment required
- At least ONE Process comment required
- At least ONE Self-Regulation comment required
- Weighting may shift based on need (e.g., more self-reg if student rushes)

Forbidden Self-Level Feedback:
- ❌ "You're smart"
- ❌ "You're talented"
- ❌ "You're a brilliant writer"
- ❌ "You're better than most of the class"
- ❌ "This is wrong"
- ❌ "This is not good enough"
- ❌ "This is excellent work"

ATQ and Signposting Integration:
- ATQ = Task feedback
- Signposting = Process feedback

E. Timely and Immediately Actionable

Timing Requirements:

During Drafting:
- LARA can be invoked while students are working
- Feedback appears within same working session
- Student remains in active editing context

Immediately After Submission:
- Once work submitted for feedback, LARA returns feedback in same session
- Not as delayed notification
- Student cannot leave session without engaging with feedback

Actionability Requirements:

CTA (Call-to-Action) Button:
Every feedback response must include at least one required action student can take NOW.

CTA Button Requirements:
- Action-specific (includes action verb + specific target)
- Directly linked to a Next Step
- Interface-aware (opens appropriate editing mode)
- Mobile-optimized (≤30 characters)

CTA Text Generation Rules:

Character Limit: Maximum 30 characters (mobile-first constraint)

Content Rules:
CTA text must include only:
- Action verb
- Essential target (no reasoning, no success indicators)

Forbidden in CTA:
- No "so that…" / "so the reader…" clauses
- No full explanations
- No extra descriptors
- No references to "work" (keep learning-focused)

Brevity Rules:
- Replace "paragraph" with "¶" symbol
- Remove filler words (your, the, this, that)
- Use shortest accurate noun

Transformation Examples:

Full Next Step	CTA Text	Length
"Add one example to paragraph 2 so your idea is clearer"	Add example to ¶2	19
"Revise your opening sentence so it introduces your idea"	Revise opening	14
"Define the term 'migration' before you use it in your explanation"	Define 'migration'	18
"Explain why this evidence supports your point"	Explain evidence	16
"Link this idea to the question to stay on track"	Link idea to question	22

Quality Checks for CTA:
A generated CTA is valid only if:
- ☐ Begins with clear action verb
- ☐ Contains no more than two essential nouns
- ☐ Unambiguous without success indicators
- ☐ ≤30 characters
- ☐ Any paragraph reference replaced by ¶
- ☐ Reflects same next step, not weaker version

Fallback Rule:
If CTA generation fails validation twice:
- Log error with full trace (not visible to students)
- Use generic fallback: "Make this change" (16 chars)
- Flag for human review

CTA Behavior and Flow:

CTA Generation:
- Each CTA corresponds to one specific Next Step
- CTA button text includes: Action verb + Specific target

CTA Activation:
- Primary CTA disabled until student selects one Next Step
- Once selected, CTA becomes active and displays action-specific text

CTA Action:
When clicked, CTA must:
- Open relevant editing interface (full response editor, targeted paragraph editor, or planning view)
- Pre-position cursor or highlight specific area to edit
- Keep feedback visible or easily accessible during editing

Completion Detection:
After student acts on CTA, LARA must:
- Detect whether selected action was completed
- Show appropriate response

Exit Requirements:
Students cannot dismiss feedback screen or mark as "done" without:
1. Selecting one Next Step, AND
2. Either: Triggering associated action flow (clicking CTA), OR Explicitly choosing "Skip for now" (with confirmation prompt)

This ensures feedback is not just read, but used, closing loop between "Here's what to do" and "Do it now."

3.3 Teacher Workflow Requirements

From "MVP1 Core Feedback Requirements (Non-Negotiable)" - Section 2:

A. Task Creation Flow (Where am I going?)

Step 1: Select Focus Area(s)
UI must display checkboxes (teacher can select multiple):
- ☐ Understanding content (task)
- ☐ Writing / structure (process)
- ☐ Planning / checking (self-regulation)

Step 2: Choose or Write Success Criteria
Based on teacher's selection(s), show relevant one-click criteria options:

If "Understanding content (task)" selected:
- ☐ Answer the question directly
- ☐ Use relevant examples or evidence
- ☐ Explain your reasoning clearly
- [Add custom criterion]

If "Writing / structure (process)" selected:
- ☐ Organise ideas logically
- ☐ Use clear topic sentences
- ☐ Link ideas with signposting
- [Add custom criterion]

If "Planning / checking (self-regulation)" selected:

- ☐ Check your work before submitting
- ☐ Make sure each paragraph answers the question
- ☐ Revise for clarity and completeness
- [Add custom criterion]

Teacher Can:

- Click to select pre-written criteria (fastest)
- Edit any pre-written criterion inline (flexible)
- Add custom criteria via text field (powerful)
- Select up to 3 criteria total (cognitive load limit)

Why This Works:

- Faster: One-click selection vs typing from scratch
- Guided: Teachers see examples of strong, plain-language criteria
- Flexible: Can edit or add custom criteria
- Pedagogically sound: Pre-written options align with research
- Reduces friction: Don't have to invent criteria on the spot

LARA Validation:

If teacher tries to proceed without selecting at least one criterion:

- Block save
- Show nudge: "Students need a clear goal before they receive feedback. Please select or add at least one success criterion."

Developer Rule:

A task cannot be saved without at least one success statement.

B. Teacher Review & Curation of Feedback

UI Grouping:

LARA displays student feedback to teacher grouped as:

- Where they're going well
- Where they're stuck
- Where to next

Teachers Can:

- Approve
- Edit
- Remove
- Add their own voice

LARA Nudges (Click-Friendly):

If teacher edits feedback to become vague, show popup:

Popup Heading: "This comment needs to be more specific"

Brief Explanation: "Students need clear guidance. Which of these would work better?"

Show 2-3 Context-Specific Suggestions (One-Click Options):
LARA must generate suggestions based on student's actual work.

Example 1: If teacher wrote "Add more detail"
- ☐ "Add more detail about the environmental impact with one example"
- ☐ "Add more detail to paragraph 2 by including evidence"
- ☐ "Add more detail explaining your reasoning for this claim"
- [Edit manually]

Example 2: If teacher wrote "Be clearer"
- ☐ "Be clearer by defining 'migration' before using it"
- ☐ "Be clearer by adding a topic sentence to paragraph 3"
- ☐ "Be clearer by explaining why this evidence matters"
- [Edit manually]

Example 3: If teacher wrote "Improve your structure"
- ☐ "Improve your structure by grouping related ideas into one paragraph"
- ☐ "Improve your structure by using topic sentences to start each paragraph"
- ☐ "Improve your structure by adding linking words between ideas"
- [Edit manually]

Two Buttons at Bottom:
- "Use suggestion" (primary action, green) — applies selected option
- "Keep as is" (secondary, grey) — allows teacher override

Why This Works:
- One-click fix: Teacher selects suggestion instead of rewriting
- Educates teachers: Shows what specificity looks like in practice
- Contextual: Suggestions reference actual student work (paragraphs, concepts, etc.)
- Respects autonomy: "Keep as is" option prevents forcing changes
- Faster workflow: No typing required for most cases

Developer Rule:
Teacher review interface must maintain three feedback questions structurally — not optional.

C. Teacher Analytics Aligned to Principles

Dashboards Must Reflect Feedback Structure:

1. Class Insights
- Most common "Where to next" steps

- Most common misconceptions (from growth areas)
- Most frequent success criteria achieved

2. Student Insights
- Draft progression
- Pattern of strengths
- Pattern of stuck points
- Which next steps they choose most often

3. Process/Self-Reg Data
- Students using effective strategies
- Students not checking work before submission

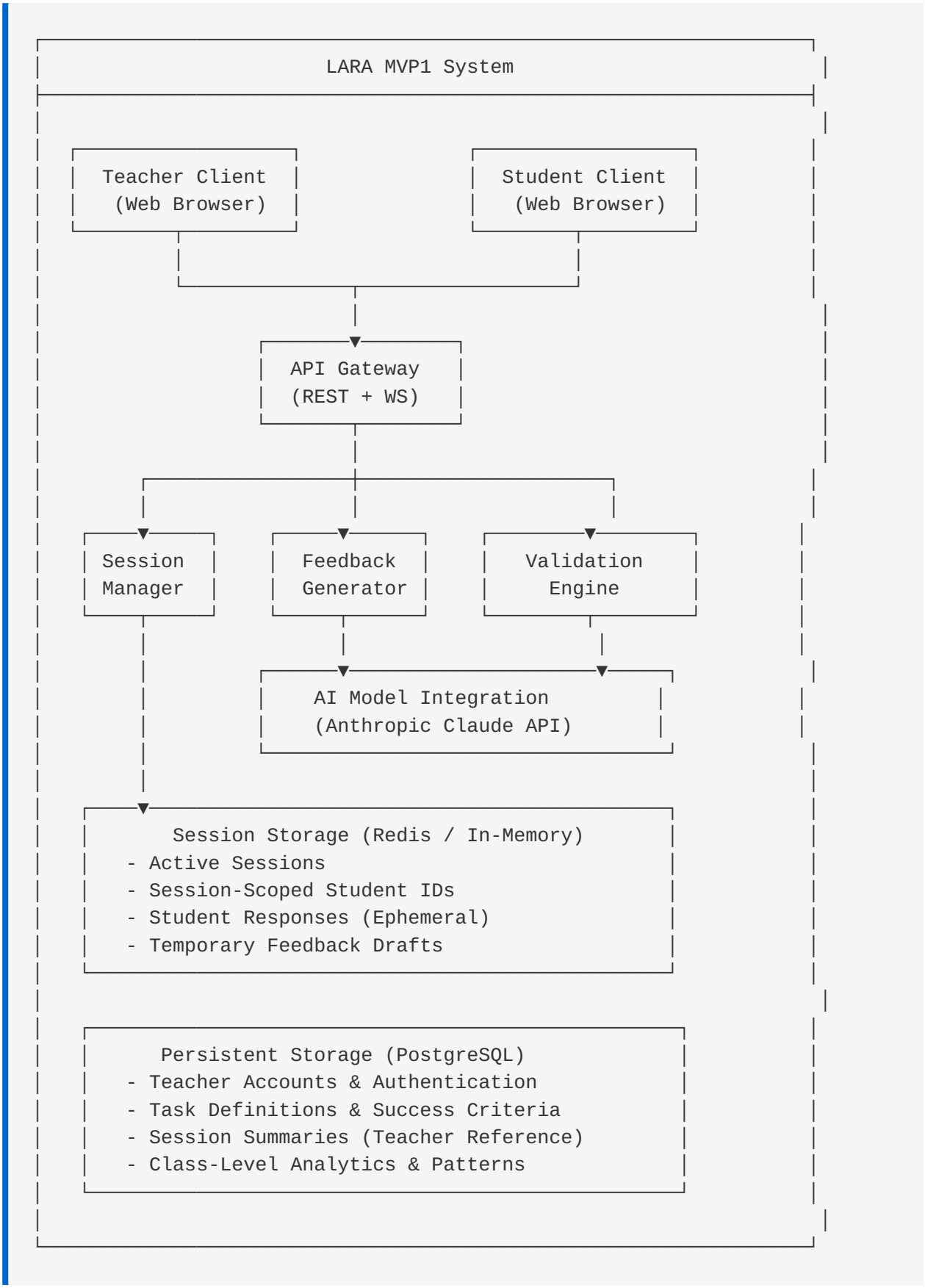
Developer Rule:
Analytics must aggregate based on feedback schema: strengths, growth areas, next steps, criteria.

---

4. Architecture Design

4.1 High-Level System Architecture

Component Overview:


LARA MVP1 System
Teacher Client
(Web Browser)
Student Client
(Web Browser)
API Gateway
(REST + WS)
Session
Manager
Feedback
Generator
Validation
Engine
AI Model Integration
(Anthropic Claude API)
Session Storage (Redis / In-Memory)
- Active Sessions
- Session-Scoped Student IDs
- Student Responses (Ephemeral)
- Temporary Feedback Drafts
Persistent Storage (PostgreSQL)
- Teacher Accounts & Authentication
- Task Definitions & Success Criteria
- Session Summaries (Teacher Reference)
- Class-Level Analytics & Patterns

4.2 Key Architectural Decisions

Session-Based, No Student Persistence:
- Student data exists ONLY during active sessions
- Session-scoped IDs generated on join (e.g., "Student 1", "Student 2")
- Automatic data deletion after session ends (with grace period for recovery)

Teacher-Centric Data Model:
- All persistent data tied to teacher accounts
- Teachers own tasks, sessions, and summaries
- Students are anonymous participants with no cross-session identity

Real-Time Feedback:
- Synchronous feedback generation during student session
- WebSocket connections for live updates (student submissions, approval notifications)
- No background jobs or delayed processing

Multi-Stage Validation Pipeline:
- Pre-Flight Validation: Before AI call (checks input validity)
- Post-Generation Validation: After AI call, before rendering (checks output quality)
- Rendering Validation: Final checks before showing to student

Mobile-First UI:
- Responsive design from ground up
- CTA character limits (≤30 chars) enforced
- Fixed goal box optimized for small screens
- Touch-friendly next step selection

4.3 Technology Stack Recommendations

Backend:
- **Language:** Python 3.11+ or Node.js 18+ (for async support)
- **API Framework:** FastAPI (Python) or Express (Node.js)
- **Database:** PostgreSQL 15 (ACID compliance for teacher data)
- **Session Storage:** Redis (high-performance ephemeral storage)
- **AI Integration:** Anthropic Claude API (Claude Sonnet 4)
- **WebSockets:** Built into framework (FastAPI WebSockets or Socket.io)

Frontend:
- **Framework:** React 18 with TypeScript
- **State Management:** Zustand or React Context (lightweight)
- **Routing:** React Router v6
- **UI Library:** Tailwind CSS + Headless UI (accessible components)

- **Form Handling:** React Hook Form + Zod (validation)
- **Real-Time:** WebSocket client library

Infrastructure:
- **Hosting:** AWS (ECS/Lambda) or Vercel (for serverless)
- **CDN:** CloudFront or Cloudflare
- **Monitoring:** Sentry (errors) + Datadog/CloudWatch (metrics)
- **CI/CD:** GitHub Actions or GitLab CI

Development Tools:
- **Version Control:** Git + GitHub
- **API Documentation:** OpenAPI/Swagger (auto-generated)
- **Testing:** pytest (backend), Jest + React Testing Library (frontend)

---

5. Complete User Flows

5.1 Teacher Flow - Detailed Steps

Phase 1: Task Creation

Step 1: Teacher Logs In
- Navigate to `/teacher/dashboard`
- Authentication via email + password
- JWT token stored in browser

Step 2: Create New Task
- Click "Create New Task" button
- Navigate to `/teacher/tasks/create`

Step 3: Enter Task Details
- Form displays:
- Task Title (optional, text input)
- Task Prompt (required, textarea, placeholder: "e.g., Explain the main causes of coastal erosion using examples from the diagram.")

Step 4: Select Focus Areas
- Heading: "What should students focus on in this task?"
- Display checkboxes (multi-select):
- ☐ Understanding content (task)

- ☐ Writing / structure (process)
- ☐ Planning / checking (self-regulation)

Step 5: Choose Success Criteria
- Based on selection(s), dynamically show relevant one-click options
- Pre-written criteria appear as checkboxes
- Each section has "[Add custom criterion]" text input
- Teacher can select multiple, up to 3 total (show counter)

Step 6: Validation
- If no criterion selected → block save
- Show error message: "Students need a clear goal before they receive feedback. Please select or add at least one success criterion."

Step 7: Save Task
- Click "Save Task" button
- Backend creates task record in database
- Redirect to `/teacher/tasks/{task_id}`

Phase 2: Start Session

Step 1: Session Creation Page
- From task detail page, click "Start Session" button
- Navigate to `/teacher/sessions/create?task_id={task_id}`
- Shows task preview (prompt + criteria)

Step 2: Generate Class Code
- Click "Start Session" button
- Backend:
- Generates unique 6-digit class code (e.g., "ABC123")
- Creates session record in database (status: "active")
- Creates session metadata in Redis
- Generates QR code pointing to student join URL

Step 3: Session Dashboard
- Redirect to `/teacher/sessions/{session_id}`
- Display:
- Large, prominent class code display
- QR code for easy mobile join
- Task prompt and success criteria (read-only)

- Real-time student counter (e.g., "5 students joined")
- Student submission list (updates live via WebSocket)

Phase 3: Monitor & Review Feedback

Step 1: Live Monitoring
- Student submission list shows:
- Session-scoped ID (e.g., "Student 1", "Student 2")
- Submission status (e.g., "Response submitted at 10:23 AM")
- Feedback status (e.g., "Feedback pending approval")

Step 2: Automatic Feedback Generation
- When student submits response:
- System triggers automatic feedback generation
- Pre-Flight validation runs
- AI model called with system prompt
- Post-Generation validation runs
- Feedback saved as draft (not visible to student yet)
- Teacher notified: "Feedback ready for review"

Step 3: Review Feedback
- Teacher clicks on student submission
- Feedback review modal opens
- Shows feedback in three-section structure:
- Goal box (read-only, grey background)
- Strengths section (editable)
- Growth areas section (editable)
- Next steps section (editable)

Step 4: Edit Feedback (Optional)
- Teacher can:
- Edit any comment text
- Add new comments
- Remove comments (with warning if removing required types)
- Reorder comments

Step 5: Vague Comment Detection
- If teacher edits comment to become vague:
- System detects vague phrases
- Shows popup: "This comment needs to be more specific"

- Displays 2-3 context-specific suggestions (one-click)
- Teacher selects suggestion or clicks "Keep as is"

Step 6: Approve Feedback
- Click "Approve & Send to Student" button
- Backend:
- Updates feedback status to "approved"
- Notifies student via WebSocket
- Student's feedback screen unlocks

Phase 4: View Class Insights

Step 1: Navigate to Analytics Tab
- Click "Class Insights" tab in session dashboard

Step 2: View Aggregated Data
- **Most Common Next Steps:**
- List showing frequency (e.g., "Add example to paragraph → 12 students")
- Suggests instructional focus areas

  • **Common Misconceptions:**

  • Extracted from growth areas

  • Shows patterns (e.g., "8 students didn't explain cause-effect relationship")

  • **Success Criteria Achievement:**

  • Shows which criteria most/least students met

  • Bar chart or percentage display

  • **Process/Self-Regulation Patterns:**

  • Students using effective strategies

  • Students not self-checking before submission

Step 3: Use Insights for Instruction
- Teacher can decide:
- Whole-class intervention (if pattern is widespread)
- Small-group support (if only some students struggle)
- Individual follow-up (if isolated issues)

Phase 5: End Session

Step 1: End Session
- Click "End Session" button in dashboard
- Confirmation dialog: "Are you sure? Students will no longer be able to access this session."

Step 2: Backend Processing
- Update session status to "ended"
- Generate session summary
- Save summary to persistent storage (session_summaries table)
- Set TTL on Redis keys (1 hour grace period for recovery)
- After grace period, delete all ephemeral session data

Step 3: Session Summary View
- Redirect to `/teacher/sessions/{session_id}/summary`
- Display:
- Session metadata (date, duration, student count)
- Downloadable CSV with session-scoped IDs and responses
- Class insights summary
- Option to create similar task

5.2 Student Flow - Detailed Steps

Phase 1: Join Session

Step 1: Landing Page
- Navigate to `/student/join` (or scan QR code)
- Display:
- Large heading: "Join Your Class"
- Input field: "Enter your 6-digit class code"
- Button: "Join"

Step 2: Enter Class Code
- Student types 6-digit code (e.g., "ABC123")
- Click "Join" button

Step 3: Validation
- Backend checks:
- Does session exist?
- Is session active?

- Is session not yet ended?
- If invalid → show error: "This class code isn't active. Check with your teacher."

Step 4: Generate Session Student ID
- Backend:
- Creates unique session-scoped student ID (UUID)
- Assigns display name (e.g., "Student 7")
- Stores in Redis with session association
- Returns session data and student ID

Step 5: Store Student ID in Browser
- Frontend stores session_student_id in sessionStorage (not localStorage)
- Used for all subsequent API calls via X-Student-ID header

Step 6: Redirect to Task View
- Navigate to `/student/session/{session_id}`

Phase 2: Submit Response

Step 1: Task View
- Display layout:
- **Goal Box (Fixed at Top):**
- Non-collapsible
- Shows success criteria or Universal Expectations
- Expandable "What does this mean?" section

- **Task Prompt:**
  - Read-only, clear typography
- **Response Input:**
  - Textarea with placeholder
  - Optional word/character counter
  - "Submit for Feedback" button (primary action)

Step 2: Write Response
- Student types their answer
- Auto-save to sessionStorage (local backup, not server)

Step 3: Submit
- Click "Submit for Feedback" button

- Frontend validation:
- Check response not empty
- Check minimum length (e.g., 50 characters)
- If valid → send to backend

Step 4: Backend Processing
- Store response in Redis (session student record)
- Update submission status
- Notify teacher via WebSocket
- Trigger automatic feedback generation (async)

Step 5: Waiting State
- Show message: "Your teacher is reviewing your feedback..."
- Display loading indicator
- WebSocket listener for approval notification

Phase 3: Receive Feedback

Step 1: Notification
- When teacher approves feedback:
- WebSocket message received
- Loading state replaced with "View Feedback" button

Step 2: Feedback Screen
- Navigate to `/student/session/{session_id}/feedback`
- **Pre-Render Validation:**
- System runs final checks before displaying
- Ensures all requirements met

Step 3: Layout

A. Goal Box (Top, Fixed)

```
┌─────────────────────────────────────────┐
│ 🎯 Goal for this task:                  │
│                                         │
│ [Criteria text displayed exactly as written]│
│ or                                      │
│ [Universal Learning Expectations]       │
│                                         │
│ ▼ What does this mean? (expandable)     │
└─────────────────────────────────────────┘
```

B. Card 1: What's Working Well

```
┌──────────────────────────────────────────┐
│ 💚 What's working well                   │
│                                          │
│ • Your explanation of coastal erosion in │
│   paragraph 2 is accurate and clearly    │
│   linked to the diagram.                 │
│                                          │
│ • You've organised your argument logically, │
│   moving from cause to effect.           │
└──────────────────────────────────────────┘
```

C. Card 2: What Needs Attention

```
┌──────────────────────────────────────────┐
│ 🟡 What needs attention                  │
│                                          │
│ • Your second paragraph mentions 'impact', │
│   but you don't include any examples. Add │
│   one to show the effect clearly.        │
│                                          │
│ • You describe the cause accurately, but │
│   you don't explain why it matters. Add  │
│   one sentence linking cause and         │
│   consequence.                           │
└──────────────────────────────────────────┘
```

D. Card 3: Choose One Next Step

```
┌─────────────────────────────────────────┐
│ 🎯 Choose one next step:                │
│                                         │
│ ○ Add one example to paragraph 2        │
│   Why this helps: This makes your       │
│   explanation clearer                   │
│                                         │
│ ○ Explain why this cause matters        │
│   Why this helps: This strengthens your │
│   reasoning                             │
│                                         │
│ ○ Check your signposting                │
│   Why this helps: This helps the reader │
│   follow your ideas                     │
│                                         │
│ ℹ️ Show me an example (not from my work) │
└─────────────────────────────────────────┘

[Continue Button - Currently Disabled]
```

Step 4: Select Next Step
- Student clicks one radio button
- "Continue" button becomes enabled
- Button text updates to show specific action (e.g., "Add example to ¶2")

Step 5: Optional - View Model Example
- Student clicks "Show me an example" link
- Modal displays generic model example:
- If "Add example" selected → shows model example sentence
- If "Add topic sentence" selected → shows model topic sentence
- If "Explain connection" selected → shows model linking sentence
- **Critical:** Examples are generic, NOT based on student's specific content
- Button: "Close" (returns to feedback screen)

Step 6: Trigger Revision
- Student clicks "Continue" button (now enabled)
- Backend logs selected next step
- Frontend determines editing interface based on actionType

Phase 4: Revise Response

Step 1: Editing Interface
- Based on actionType, open appropriate interface:

If actionType = "revise":

- Full response editor opens
- Shows original response (editable)
- Feedback sidebar (collapsible) shows selected next step
- No specific highlighting

If actionType = "improve-section":

- Targeted paragraph editor
- Highlights relevant paragraph based on next step
- Feedback sidebar shows selected next step
- Cursor pre-positioned in relevant section

If actionType = "reupload":

- Full response editor
- Blank slate or original text (teacher preference)

If actionType = "rehearse":

- Planning view
- Shows checklist or scaffolding based on next step
- Encourages reflection before writing

Step 2: Edit Response

- Student makes changes
- Auto-save to sessionStorage (local backup)

Step 3: Save Revision

- Click "Save Revision" button
- Frontend sends revised text to backend
- Backend:
- Stores revision in Redis
- Runs completion detection algorithm
- Returns completion status

Step 4: Completion Detection

- System checks if selected next step was completed
- Simple heuristics:
- If next step was "Add example to ¶2" → check if ¶2 has more content
- If next step was "Define term" → check if definition appears before term usage
- If next step was "Add topic sentence" → check if paragraph now starts differently

Step 5: Completion Feedback

- Show appropriate message:

If completed:

```
┌─────────────────────────────────────────┐
│ ✓ You completed your next step!         │
│                                         │
│ Would you like feedback on your revised │
│ answer?                                 │
│                                         │
│ [Yes, get feedback]  [No, I'm done]     │
└─────────────────────────────────────────┘
```

If incomplete:

```
┌─────────────────────────────────────────┐
│ ⚠ It looks like this step isn't visible │
│   in your revision yet.                 │
│                                         │
│ Want help completing it?                │
│                                         │
│ [Show me the step again]  [I'm done]    │
└─────────────────────────────────────────┘
```

Step 6: Optional - Request New Feedback

- If student clicks "Yes, get feedback":
- Trigger feedback generation for revised response
- Return to waiting state
- Cycle repeats (in full MVP, this is post-MVP1; MVP1 = 1 revision only)

Step 7: Session Complete

- If student clicks "No, I'm done" or "I'm done":
- Show completion message: "Great work today! Your teacher has your responses."
- No persistent data for student
- Session data remains in Redis until teacher ends session

---

6. Component Specifications

6.1 Teacher Components

TaskCreationWizard

Purpose: Multi-step form for creating tasks with success criteria.

State:
- formData: { title, prompt, focusAreas[], criteria[] }
- validationErrors: []
- currentStep: number (1-3)

Methods:
- handleFocusAreaToggle(area): Add/remove from focusAreas
- handleCriteriaSelect(criterion): Add pre-written criterion
- handleCustomCriteriaAdd(text): Add custom criterion
- validateForm(): Check minimum requirements
- saveTask(): Submit to backend

UI Sections:
1. Task Details (title, prompt)
2. Focus Areas (checkboxes)
3. Success Criteria (dynamic based on focus areas)
4. Preview & Save

Validation:
- At least one criterion required
- Show error if saving without criteria
- Maximum 3 criteria (soft limit, can be overridden)

SessionDashboard

Purpose: Real-time monitoring of active session.

State:
- session: { id, class_code, task, status }
- students: [{ id, display_name, submission_status, feedback_status }]
- feedbackQueue: [studentIds with pending approvals]

WebSocket Listeners:
- "student_joined" → Add to students list
- "student_submitted" → Update submission_status
- "feedback_generated" → Update feedback_status
- "student_revised" → Update revision_status

Methods:
- endSession(): Confirm and end session
- viewFeedback(studentId): Open FeedbackReviewModal
- downloadSessionData(): Export CSV

UI Sections:
- Class Code Display (large, prominent)
- QR Code
- Task Preview (collapsible)
- Live Student List
- Class Insights Tab

FeedbackReviewModal

Purpose: Review and edit AI-generated feedback before approval.

Props:
- studentId
- feedback (initial feedback object)
- studentResponse (for context)

State:
- editedFeedback: (modified feedback)
- showVagueWarning: boolean
- suggestedSpecificComments: []

Methods:
- handleCommentEdit(index, newText)
- validateSpecificity(): Check for vague phrases
- generateSuggestions(vagueComment, studentResponse): Context-specific alternatives
- approveFeedback(): Submit approved feedback
- rejectFeedback(): Trigger regeneration

UI Sections:
- Goal Box (read-only, grey)
- Strengths (editable list)
- Growth Areas (editable list)
- Next Steps (editable list)
- Approve/Reject buttons

Vague Comment Popup:
- Triggered when specificity check fails
- Shows heading, explanation, 2-3 suggestions
- One-click selection or manual edit

ClassInsightsPanel

Purpose: Display aggregated class-level patterns.

Props:
- sessionId

Data Sources:
- Aggregated from all student feedback in session
- Computed on backend, cached

Display Sections:

1. **Most Common Next Steps:**
   - Bar chart or list
   - Shows frequency (e.g., "Add example → 12 students")
2. **Common Misconceptions:**
   - Extracted from growth areas
   - Shows patterns (e.g., "8 students: didn't explain cause-effect")
3. **Success Criteria Achievement:**
   - Percentage of students meeting each criterion
   - Visual indicator (green/yellow/red)
4. **Process/Self-Regulation Data:**
   - Students using effective strategies
   - Students not self-checking

Export Options:
- Download as PDF or CSV

6.2 Student Components

JoinSession

Purpose: Entry point for students to join via class code.

State:
- classCode: string
- error: string | null
- loading: boolean

Methods:
- validateCode(code): Check format (6 digits/chars)
- joinSession(): Call API, store session_student_id
- handleError(message): Display error message

UI:
- Large heading: "Join Your Class"
- Input: 6-digit class code
- Button: "Join"
- Error display area

Validation:
- Client-side: 6 characters
- Server-side: Session exists and active

TaskResponseForm

Purpose: Display task and collect student response.

Props:
- task: { prompt, criteria }
- sessionId
- studentId

State:
- responseText: string
- wordCount: number
- isSaving: boolean

Methods:
- handleTextChange(text): Update state, auto-save to sessionStorage
- submitResponse(): Validate and send to backend
- saveLocalBackup(): Store in sessionStorage

UI Layout:
- Goal Box (fixed at top)
- Task Prompt (read-only)
- Response Textarea (main input area)
- Word/Character Counter (optional)
- Submit Button (primary action)

Auto-Save:
- Save to sessionStorage every 30 seconds
- Recover on page reload

FeedbackScreen

Purpose: Display structured feedback and require next step selection.

Props:
- feedback: { goal, strengths, growthAreas, nextSteps }
- canSelect: boolean (true after feedback approved)

State:
- selectedNextStepId: string | null
- showModelExample: boolean
- modelExampleContent: string | null

Methods:
- handleNextStepSelect(id): Update selection
- enableContinueButton(): Activate once selection made
- fetchModelExample(nextStepType): Retrieve generic example
- proceedToRevision(): Navigate to editing interface

UI Layout:
- GoalBox (fixed, always visible)
- StrengthsCard (list of strengths with type badges)
- GrowthAreasCard (list of growth areas with type badges)
- NextStepsCard (radio buttons, one required)
- ModelExampleLink ("Show me an example")
- ContinueButton (disabled until selection)

Disabled State:
- Continue button greyed out until radio selection made
- Hover shows tooltip: "Choose a next step to continue"

RevisionEditor

Purpose: Targeted editing interface based on selected next step.

Props:
- originalResponse: string
- nextStep: { action, target, why }
- feedbackContext: { strengths, growthAreas }
- actionType: string

State:
- revisedText: string
- highlightedSection: { start, end } | null

Methods:
- highlightRelevantSection(): Based on nextStep target (e.g., ¶2)
- saveRevision(): Submit to backend
- checkCompletion(): Validate if next step completed
- autoSave(): Periodic save to sessionStorage

UI Modes (Based on actionType):

"revise" mode:
- Full response editor
- No highlighting
- Feedback sidebar (collapsible)

"improve-section" mode:
- Full response visible
- Highlight relevant paragraph/section
- Cursor pre-positioned
- Feedback sidebar shows selected next step

"reupload" mode:
- Blank editor or original text
- Encourages fresh approach

"rehearse" mode:
- Planning scaffold
- Checklist based on next step
- "Start writing" button → switches to editor

6.3 Shared Components

GoalBox

Purpose: Display success criteria, always visible during feedback.

Props:
- goalText: string
- source: "teacher" | "universal_expectations"
- expandable: boolean
- expandedDetail: string | null

State:
- isExpanded: boolean

UI:
- Fixed position (sticky top or persistent container)
- Non-collapsible
- Optional expandable section ("What does this mean?")
- Background color to distinguish from feedback

Behavior:
- Always renders before feedback
- Cannot be hidden or minimized
- Expandable detail shows on click (if provided)

FeedbackTypeTag

Purpose: Visual indicator of feedback type.

Props:
- type: "task" | "process" | "self_reg"

UI:
- Small badge next to feedback comment
- Color-coded:
- Task: Blue
- Process: Green
- Self-Regulation: Purple
- Hover tooltip explains type

CTAButton

Purpose: Primary action button with ≤30 character constraint.

Props:
- text: string (must be ≤30 chars)
- action: function
- disabled: boolean
- variant: "primary" | "secondary"

Validation:
- Throws error if text >30 characters
- Auto-truncates with "..." if needed (but logs warning)

UI:
- Large, touch-friendly button
- Mobile-optimized spacing
- Loading state when action processing

---

7. Validation & Business Rules

7.1 Pre-Flight Validation (Before AI Call)

Purpose: Prevent invalid feedback requests before calling expensive AI model.

Checklist:

1. **Task Setup Checks:**
   - ✅ Task has at least one teacher-defined success criterion OR Universal Expectations will be used
   - ❗ If neither true → block feedback generation, prompt teacher to add criteria
2. **Student Input Checks:**
   - ✅ Student submission is non-empty
   - ✅ Student work contains parseable text (at least one sentence or paragraph)
   - ❗ If not parseable → show message: "It looks like your response is too short to give feedback yet. Add a little more and try again."

3. **Session Checks:**
   - ✅ Class code session is active
   - ✅ Submission belongs to current task & student
   - ❗ If invalid → show message: "We're reconnecting you to your work…"
4. **System Readiness Checks:**
   - ✅ AI model service reachable
   - ✅ Storage/logging endpoints responsive
   - ❗ If not → show fallback: "Still preparing your feedback…"

Benefits:
- Prevents unnecessary model calls
- Prevents malformed feedback requests
- Ensures smooth user experience

7.2 Post-Generation Validation (After AI Call, Before Rendering)

Purpose: Ensure AI-generated feedback meets all pedagogical and technical requirements.

Validation Rules:

1. Goal Validation

- ✅ goal.text exists and is non-empty
- ✅ If teacher criteria exist → goal.source = "teacher"
- ✅ If no criteria → goal.source = "universal_expectations"
- ❗ Goal must NOT contain invented curriculum/content criteria
- **Check Method:** Scan for content-specific command verbs (compare, describe, evaluate, analyze) paired with domain content

2. Strengths Validation

- ✅ At least one strength present
- ✅ Each strength contains type tag (task/process/self_reg)
- ❗ Must reference specific evidence:
- workReference field populated, OR

- criteriaReference field populated, OR
- Explicit skill reference in comment
- ❗ Must pass specificity check (contains content anchors from student work)

Specificity Check Algorithm:
Scan comment text for:
- Paragraph references (paragraph 2, ¶2, your opening, etc.)
- Quoted terms from student work
- Concept names student used
- Specific nouns/verbs from response
- At least 2 content words overlapping with student response (>4 letters)

3. Growth Area Validation

- ❗ At least one growth area required
- ❗ Each must include:
- workReference (specific part of student work)
- reason (why it needs attention)
- direction (guidance for improvement, not rewrite)
- ❗ Must pass specificity check
- ❗ Must contain valid feedback type (task/process/self_reg)

4. Feedback Type Balance

Across strengths + growth areas:
- ❗ At least ONE Task comment
- ❗ At least ONE Process comment
- ❗ At least ONE Self-Regulation comment
- If missing any type → regenerate feedback

5. Next Steps Validation

Every next step must have:
- ❗ Action verb
- ❗ Specific target (paragraph, concept, or growthAreaId)
- ❗ Success indicator
- ❗ Short rationale ("why this helps", 5-12 words)
- ❗ CTA (≤30 characters)
- ❗ actionType (determines UI mode)

Additional Rules:

- ❗ Must map to at least one growth area (traceability)
- ❗ Must not duplicate another next step
- ❗ Must not rewrite student work
- ❗ Must pass specificity check (contains content anchor)

6. ATQ / Signposting Logic

- If any growth area contains "ATQ" or "answer the question" → must include at least one ATQ-focused next step
- If any growth area contains "signposting" → must include at least one signposting-focused next step

7. CTA Validation

- ❗ At least one primary CTA must be present
- ❗ CTA must be action-specific (verb + target)
- ❗ Student cannot exit until next step selected

Generic CTAs (Forbidden):

- ❌ "Try again"
- ❌ "Improve this"
- ❌ "Continue" (before next step chosen)

8. Forbidden Language Check

Scan all comments for:

- ❌ Vague phrases: "add more detail", "be clearer", "good job", "improve this"
- ❌ Ability praise: "smart", "talented", "brilliant writer", "clever", "gifted"
- ❌ Peer comparison: "better than most", "top of the class", "compared to others"
- ❌ Judgmental language: "this is wrong", "not good enough", "excellent work", "poor quality"

Special Case - Vague Phrase Specificity:

If vague phrase detected, check if made specific:

- "add more detail" → acceptable if followed by "about [specific thing]" or "to paragraph X"
- "be clearer" → acceptable if followed by "by defining [term]" or "by [specific action]"
- If not made specific → reject

9. Renderer Compatibility Check

- ❗ All objects match JSON schema

- ❗ No missing fields required by UI
- ❗ No empty arrays where prohibited (strengths, growthAreas, nextSteps)

7.3 Regeneration Logic

Trigger: If any validation rule fails after AI generation.

Process:
1. Reject invalid feedback (do not render to student)
2. Log validation errors with full trace
3. Attempt regeneration with explicit validation flags in system prompt
4. Maximum 2 regeneration attempts

After Max Retries:
- Log critical error
- Show safe message to student: "Preparing your feedback… thanks for waiting."
- Notify teacher: "Feedback generation failed for Student X. Please review manually."

Fallback Feedback (Emergency Only):
- Never shown to student automatically
- Teacher must manually write feedback
- System logs failure for analysis

7.4 Specificity Check Implementation

Purpose: Detect generic, vague feedback that lacks connection to student work.

Method:

Check 1: Content Anchor Detection
Scan comment text for patterns:
- Paragraph references: "paragraph 2", "¶2", "your opening", "your first sentence"
- Quoted student text: words inside quotation marks
- Concept references: terms that appear in both comment and student response
- Structural references: "your explanation of", "when you described"

Check 2: Word Overlap
- Extract content words (>4 letters) from student response
- Extract content words from comment

- Calculate overlap
- Require at least 2 overlapping content words for specificity

Check 3: Generic Phrase Detection
- Maintain blacklist of generic phrases
- Flag comment if majority of text is generic
- Examples: "your work", "this response", "the content", "your writing"

Pass Threshold:
Comment passes specificity check if:
- (At least 1 content anchor detected) OR
- (At least 2 overlapping content words) OR
- (Structural reference present + 1 overlapping word)

Fail Action:
- If comment fails → mark for regeneration
- If multiple comments fail → regenerate entire feedback object

---

7.5 Teacher Edit Validation

Purpose: Catch when teachers make feedback vague during editing.

Trigger: On save/approve in FeedbackReviewModal.

Process:
1. Run specificity check on all edited comments
2. Run forbidden language check
3. If any comment fails:
- Block save
- Show "This comment needs to be more specific" popup
- Generate 2-3 context-specific suggestions
- Allow "Keep as is" override (log override for analysis)

Context-Specific Suggestion Generation:

Input:
- Vague comment (e.g., "Add more detail")
- Student response text
- Original growth area (if editing existing comment)

Algorithm:
1. Identify what type of detail is missing (from original growth area)

2. Find specific paragraph or section referenced
3. Generate 3 alternatives that:
- Specify WHERE (which paragraph)
- Specify WHAT (which type of detail)
- Specify HOW or WHY (purpose/benefit)

Output:
- Array of 3 one-click alternative comments
- Each is specific, actionable, and grounded in student work

---

8. API Design

8.1 Authentication Endpoints

POST /api/auth/register
- Body: { email, password, name }
- Response: { user_id, token }
- Creates teacher account

POST /api/auth/login
- Body: { email, password }
- Response: { user_id, token, name }
- Returns JWT token

POST /api/auth/logout
- Headers: Authorization: Bearer {token}
- Response: { success: true }
- Invalidates token

GET /api/auth/me
- Headers: Authorization: Bearer {token}
- Response: { user }
- Returns current user info

8.2 Task Management Endpoints

POST /api/tasks
- Headers: Authorization: Bearer {token}
- Body: { title, prompt, success_criteria[], use_universal_expectations }
- Response: { task_id, task }
- Validation: At least one criterion required (unless use_universal_expectations=true)

GET /api/tasks
- Headers: Authorization: Bearer {token}
- Query: ?page=1&limit=20
- Response: { tasks: [], total, page }
- Returns teacher's tasks

GET /api/tasks/:task_id
- Headers: Authorization: Bearer {token}
- Response: { task }
- Returns single task details

PUT /api/tasks/:task_id
- Headers: Authorization: Bearer {token}
- Body: { title, prompt, success_criteria[] }
- Response: { task }
- Updates task (validation same as POST)

DELETE /api/tasks/:task_id
- Headers: Authorization: Bearer {token}
- Response: { success: true }
- Soft delete (mark as deleted, don't remove from DB)

8.3 Session Management Endpoints

POST /api/sessions
- Headers: Authorization: Bearer {token}
- Body: { task_id }
- Response: { session_id, class_code, qr_code_url }
- Creates active session, generates class code

GET /api/sessions/:session_id
- Headers: Authorization: Bearer {token}

- Response: { session, task, students: [] }
- Returns session details with live student list

PATCH /api/sessions/:session_id/end
- Headers: Authorization: Bearer {token}
- Response: { success: true, summary_id }
- Ends session, generates summary, schedules cleanup

GET /api/sessions/:session_id/analytics
- Headers: Authorization: Bearer {token}
- Response: { class_insights, student_insights }
- Returns aggregated patterns and metrics

WebSocket /ws/sessions/:session_id
- Headers: Authorization: Bearer {token}
- Events:
- student_joined: { student_id, display_name }
- student_submitted: { student_id, timestamp }
- feedback_generated: { student_id }
- student_revised: { student_id, timestamp }

8.4 Feedback Review Endpoints (Teacher)

GET /api/sessions/:session_id/students/:student_id/feedback
- Headers: Authorization: Bearer {token}
- Response: { feedback, student_response }
- Returns draft feedback for review

PUT /api/sessions/:session_id/students/:student_id/feedback
- Headers: Authorization: Bearer {token}
- Body: { feedback }
- Response: { feedback }
- Saves edited feedback (runs validation)

POST /api/sessions/:session_id/students/:student_id/feedback/approve
- Headers: Authorization: Bearer {token}
- Response: { success: true }
- Approves feedback, makes visible to student, sends WebSocket notification

POST /api/sessions/:session_id/students/:student_id/feedback/regenerate
- Headers: Authorization: Bearer {token}

- Response: { feedback }
- Triggers new AI generation

8.5 Student Session Endpoints

POST /api/student/join
- Body: { class_code }
- Response: { session_id, session_student_id, task }
- Validates class code, creates ephemeral student record
- Returns session_student_id for use in subsequent calls

GET /api/student/session/:session_id
- Headers: X-Student-ID: {session_student_id}
- Response: { task, status }
- Returns task details for student

POST /api/student/session/:session_id/submit
- Headers: X-Student-ID: {session_student_id}
- Body: { response_text }
- Response: { success: true, feedback_pending: true }
- Stores response, triggers feedback generation

GET /api/student/session/:session_id/feedback
- Headers: X-Student-ID: {session_student_id}
- Response: { feedback, status: "approved" | "pending" }
- Returns feedback if approved, otherwise returns pending status

POST /api/student/session/:session_id/select-next-step
- Headers: X-Student-ID: {session_student_id}
- Body: { next_step_id }
- Response: { success: true }
- Logs student's next step selection

POST /api/student/session/:session_id/submit-revision
- Headers: X-Student-ID: {session_student_id}
- Body: { revision_text }
- Response: { completion_status, suggestion }
- Stores revision, runs completion detection

8.6 Internal Feedback Generation Service

POST /internal/feedback/generate
- (Internal only, not exposed to clients)
- Body: { task_id, student_response, success_criteria }
- Response: { feedback } or { error, retry: true }

Process:
1. Run pre-flight validation
2. Build system prompt from LARA principles + task criteria
3. Call AI model (Anthropic Claude API)
4. Parse JSON response
5. Run post-generation validation
6. If validation fails:
- Log errors
- Attempt regeneration (max 2 times)
- Return error if all attempts fail
7. Return validated feedback

---

9. UI/UX Requirements

9.1 Design Principles

Mobile-First:
- All screens designed for mobile devices first
- Desktop layout is enhanced version, not separate design
- Touch targets minimum 44x44 pixels
- CTA buttons maximum 30 characters for mobile readability

Accessibility:
- WCAG 2.1 AA compliance minimum
- Semantic HTML (proper heading hierarchy)
- ARIA labels for interactive elements
- Keyboard navigation support
- Screen reader friendly

Clarity Over Cleverness:
- Plain language, no jargon

- Clear visual hierarchy
- One primary action per screen
- Avoid overwhelming students with choices

Consistency:
- Three-question structure visible in every feedback screen
- Color coding for feedback types (task=blue, process=green, self-reg=purple)
- Fixed goal box position
- Predictable navigation

9.2 Teacher Interface Requirements

Task Creation

Layout:
- Clean, step-by-step wizard
- Progress indicator (Step 1 of 3)
- Clear labels and instructions
- Inline validation with helpful error messages

Criteria Selection:
- Checkboxes for pre-written options
- Visual grouping by type (task/process/self-reg)
- Clear affordance for adding custom criteria
- Counter showing how many selected (e.g., "2 of 3 criteria")

Validation Messages:
- Error: Red text with icon, above submit button
- Success: Green confirmation after save
- Warning: Yellow for optional improvements

Session Dashboard

Class Code Display:
- Large, prominent font (48-72px)
- High contrast background
- QR code next to it
- Copy-to-clipboard button

Student List:
- Real-time updates (WebSocket)
- Color-coded status indicators:
- Grey: Joined, not submitted
- Blue: Submitted, feedback pending
- Green: Feedback approved
- Yellow: Revised
- Click on student row to view feedback
- Filter options (e.g., "Show only pending approval")

Class Insights:
- Tab navigation (Students | Insights)
- Visual charts (bar charts for common patterns)
- Sortable tables
- Export button

Feedback Review Modal

Layout:
- Full-screen modal or side panel (desktop)
- Scrollable, three-section structure
- Fixed header with student ID and task name
- Fixed footer with Approve/Reject buttons

Editing:
- Inline editing (click to edit)
- Auto-save draft while editing
- Undo/redo support
- Visual diff if comparing to original AI version

Vague Comment Popup:
- Modal overlay
- Clear heading and explanation
- Radio buttons for suggestions
- Prominent primary action ("Use suggestion")
- Secondary escape ("Keep as is")

9.3 Student Interface Requirements

Join Screen

Layout:
- Centered content
- Large, friendly heading
- Single input field (6-digit code)
- Large, touch-friendly Join button
- No distractions, single focus

Error Handling:
- Clear error messages ("This class code isn't active")
- Suggest next steps ("Check with your teacher")
- Avoid technical jargon

Task Response Form

Layout:
- Fixed goal box at top (sticky on scroll)
- Task prompt clearly separated
- Large textarea (minimum 300px height)
- Submit button at bottom, always visible

Goal Box:
- Non-collapsible
- Light background color (e.g., blue-50)
- Border to distinguish from content
- Expandable "What does this mean?" in collapsed state
- Icon to indicate expandability

Auto-Save:
- Save to sessionStorage every 30 seconds
- Show "Auto-saved" indicator briefly
- No server calls (local only)

Feedback Screen

Layout:
- Vertical card stack (mobile)
- Goal box pinned at top
- Each card has clear visual separation

- Next steps as radio buttons (large touch targets)
- Continue button anchored at bottom

Card Design:
- Rounded corners, subtle shadow
- Icon at top left (💚, 🟡, 🎯)
- Padding for readability
- Type badges inline with comments

Next Step Selection:
- Radio buttons with large labels
- "Why this helps" text in smaller, secondary font
- Visual feedback on selection (border highlight)
- Disabled button becomes enabled on selection

Model Example Link:
- Small, unobtrusive link ("ℹ️ Show me an example")
- Opens modal with generic example
- NOT inline to avoid confusion with student's work

Revision Editor

Layout:
- Split view or tabbed view:
- Edit tab: Full editor
- Feedback tab: Collapsible feedback reference
- Highlight target section (if improve-section mode)
- Save button always visible

Highlighting:
- Yellow background for target paragraph
- Smooth scroll to highlighted section on load
- Cursor pre-positioned in highlighted area

Completion Feedback:
- Modal or banner after save
- Clear success message or improvement suggestion
- Option to continue or view feedback

9.4 Responsive Breakpoints

Mobile (320px - 767px):
- Single column layout
- Stacked cards
- Full-width buttons
- Goal box sticky at top
- Collapsible feedback sidebar in revision editor

Tablet (768px - 1023px):
- Two-column layout for teacher dashboard (student list | feedback preview)
- Larger touch targets
- Side-by-side cards for feedback (where space allows)

Desktop (1024px+):
- Multi-column layouts
- Side panels for modals
- Larger typography
- More information density (but not cluttered)

9.5 Loading & Error States

Loading Indicators:
- Spinner for async operations (feedback generation, save)
- Skeleton screens for data loading (student list)
- Progress bars for multi-step processes (task creation)

Error Messages:
- Inline for form validation
- Toast notifications for async errors
- Full-screen error page for critical failures
- Always suggest next action

Empty States:
- Friendly illustrations
- Clear explanation ("No sessions yet")
- Prominent CTA ("Create your first task")

---

10. Session Management

10.1 Session Lifecycle

Creation:
- Teacher clicks "Start Session" for a task
- Backend generates unique 6-digit class code
- Creates session record in PostgreSQL (persistent metadata)
- Creates session state in Redis (ephemeral data)
- Returns session_id and class_code to teacher

Active:
- Students join using class code
- Each student gets unique session-scoped ID (UUID)
- Student IDs stored in Redis under session key
- Real-time updates via WebSocket
- Student responses and feedback stored in Redis

Ending:
- Teacher clicks "End Session"
- Backend:
- Updates session status to "ended" in PostgreSQL
- Generates session summary
- Saves summary to session_summaries table
- Sets TTL on Redis keys (1 hour grace period)
- After grace period:
- Redis automatically deletes expired keys
- No manual cleanup required

Recovery:
- If session crashes during active state:
- Teacher can reload dashboard
- Redis data persists until TTL
- WebSocket reconnects automatically
- If teacher accidentally ends session:
- Grace period allows recovery (1 hour)
- After grace period, data is gone

10.2 Class Code Generation

Requirements:
- 6 characters (alphanumeric)
- Easy to type and read (exclude ambiguous characters: 0/O, 1/I/l)
- Unique among active sessions
- Case-insensitive for student input

Algorithm:
1. Generate random 6-character string from allowed charset
2. Check if code already in use by active session
3. If collision, regenerate
4. Store mapping: class_code → session_id in Redis
5. Set TTL to match session duration (default 24 hours max)

Allowed Charset:
- Uppercase letters: A-Z (excluding O, I)
- Digits: 2-9 (excluding 0, 1)
- Total: 24 letters + 8 digits = 32 characters
- Probability of collision: Very low (32^6 = 1 billion combinations)

QR Code:
- Generated server-side using library (e.g., qrcode in Python)
- Encodes URL: `https://lara.app/student/join?code={class_code}`
- Returned as data URL (base64) or served as image

10.3 Session-Scoped Student IDs

Purpose:
- No persistent student accounts
- No PII stored by LARA
- Teacher maintains external ID-to-name mapping

Implementation:
- On join, generate UUID for student
- Store in Redis: `session:{session_id}:student:{uuid}`
- Assign display name: "Student 1", "Student 2", etc. (incremental)
- Student ID stored in browser sessionStorage (not localStorage)
- Used in X-Student-ID header for all API calls

Display Name Assignment:
- Incremental counter in session metadata
- First student joins → "Student 1"
- Second student joins → "Student 2"
- If student refreshes page → same ID retrieved from sessionStorage
- If student joins from new browser → new ID assigned

Privacy Benefits:
- No cross-session tracking possible
- No student profile or history
- No identifiable information in LARA's database
- Teacher exports CSV with session IDs, adds names externally

---

10.4 Data Cleanup

Ephemeral Data (Redis):
- Automatically deleted when TTL expires
- TTL set on session end + 1 hour grace period
- No manual deletion required

Persistent Data (PostgreSQL):
- Session metadata remains (for teacher analytics)
- Session summaries remain (for teacher reference)
- Teacher can delete summaries manually if desired

Student Data Guarantee:
- No student responses stored after session ends + grace period
- Session IDs in summaries are meaningless without teacher's external mapping
- LARA cannot identify students across sessions

10.5 Concurrent Session Limits

Per Teacher:
- Maximum 5 active sessions at once (soft limit, configurable)
- Prevents resource abuse
- Typical use case: 1 session at a time

Per Session:
- Maximum 200 students (soft limit, configurable)
- Large class support (typical class size: 20-30)
- WebSocket connections scale to limit

Class Code Uniqueness:
- Enforced globally across all active sessions
- Collision handling: Regenerate code automatically
- Active code pool limited to current active sessions only

---

11. Feedback Generation System

11.1 AI Model Integration

Provider: Anthropic Claude API

Model: Claude Sonnet 4 (claude-sonnet-4-20250514)

Why Sonnet 4:
- Balance of speed and quality
- Strong instruction following
- JSON output reliability
- Cost-effective for MVP scale

API Configuration:
- Max tokens: 1000 (sufficient for structured feedback)
- Temperature: 0.3 (low for consistency, some variability for naturalness)
- Timeout: 30 seconds
- Retry: 2 attempts on failure

11.2 System Prompt Construction

Components:

1. **Role Definition:**
   - "You are LARA, an educational feedback engine."
   - Establishes context and purpose

2. **Feedback Principles (Abbreviated):**
   - Formative by design
   - Answer three questions
   - Focus on task/process/self-regulation
   - Be specific and improvement-focused
   - Timely and usable now
   - Build feedback literacy
   - Emotionally safe
   - Aligned to criteria
   - High impact, low workload
3. **Success Criteria:**
   - If teacher-defined: Include exact criteria text
   - If universal: Include Universal Learning Expectations text
   - Critical: NEVER infer content-specific goals
4. **Task Prompt:**
   - Student-facing task question/prompt
5. **Output Requirements:**
   - JSON format with specified schema
   - At least one strength, one growth area, 1-3 next steps
   - Balance of task/process/self-reg types
   - Specific references to student work
   - Avoid forbidden language
6. **Forbidden Language:**
   - No vague praise
   - No ability judgments
   - No peer comparisons
   - No rewrites
7. **JSON Schema:**
   - Exact structure expected
   - Field descriptions

Template:

```
You are LARA, an educational feedback engine. Your role is to provide
formative feedback aligned with research-based principles.

FEEDBACK PRINCIPLES:
[abbreviated principles]

SUCCESS CRITERIA FOR THIS TASK:
[teacher criteria OR universal expectations]

TASK PROMPT:
[task.prompt]

FORBIDDEN:
- Vague phrases without specifics
- Ability praise (smart, talented, etc.)
- Peer comparisons
- Content-specific goal inference
- Rewriting student work

REQUIRED:
- At least one strength
- At least one growth area
- 1-3 concrete next steps
- Balance of task, process, self-regulation feedback
- Specific references to student work (paragraphs, concepts, terms)
- CTAs ≤30 characters

RESPOND IN JSON FORMAT:
{schema}
```

11.3 Response Parsing & Validation

Parsing:

1. Extract JSON from AI response (handle markdown code blocks if present)
2. Parse into feedback object
3. Validate against schema

Schema Validation:

- Check all required fields present
- Check field types correct
- Check array lengths (e.g., strengths not empty)

Content Validation:

- Run specificity check on all comments

- Run forbidden language check
- Check feedback type balance
- Validate next step requirements (action verb, target, CTA length, etc.)
- Check ATQ/signposting triggers

Error Handling:
- If parsing fails → log error, trigger regeneration
- If validation fails → log specific failures, trigger regeneration
- Maximum 2 regeneration attempts
- After max retries → log critical error, notify teacher

11.4 CTA Generation Algorithm

Input: Full next step text (e.g., "Add one example to paragraph 2 so your idea is clearer")

Process:

1. **Extract Action Verb:**
   - Scan for known action verbs (add, revise, define, explain, link, clarify, check, organise)
   - Capitalize first letter
   - Default to "Improve" if no verb found
2. **Strip Success Indicators:**
   - Remove everything after "so that", "so the", "to show", "to make"
   - Keep only core action and target
3. **Replace Paragraph References:**
   - "paragraph 2" → "¶2"
   - "paragraph X" → "¶X"
4. **Remove Filler Words:**
   - Strip: your, the, this, that, a, an
   - Keep only essential nouns and verbs
5. **Truncate to ≤30 Characters:**
   - If still too long, prioritize verb + first target noun
   - Use ellipsis if necessary (but avoid if possible)
   - Log warning if truncation required

Examples:

Input	Output	Length
Add one example to paragraph 2 so your idea is clearer	Add example to ¶2	19
Revise your opening sentence so it introduces your idea	Revise opening	14
Define the term 'migration' before you use it	Define 'migration'	18
Explain why this evidence supports your point	Explain evidence	16
Organise your points into a clearer sequence	Organise points	15

Validation:
- CTA must be ≤30 characters (hard requirement)
- Must begin with action verb
- Must reflect same intent as full next step
- If validation fails twice, use fallback: "Make this change" (16 chars)

11.5 Completion Detection

Purpose: Determine if student completed selected next step in revision.

Approach: Rule-based heuristics (not AI-powered in MVP1).

Algorithm:

1. **Identify Next Step Type:**
   - Based on selected next step's actionType and target
2. **Run Appropriate Check:**

If "Add example to ¶X":
- Compare paragraph X length before and after revision
- Check if length increased by >50 characters
- Result: completed if increased, incomplete if not

If "Define term before using it":
- Search for term in revised text
- Check if definition-like text appears before first usage
- Result: completed if definition found early, incomplete if not

If "Add topic sentence to ¶X":
- Compare first sentence of paragraph X before and after
- Check if first sentence changed
- Result: completed if changed, incomplete if not

If "Check signposting":
- Count linking words (however, therefore, furthermore, etc.)
- Compare before and after
- Result: completed if count increased, incomplete if not

If "Explain why" or "Add reasoning":
- Check paragraph length or sentence count
- Result: completed if increased, incomplete if not

1. **Return Status:**
   - completed: true/false
   - suggestion: (if incomplete) brief hint on what's missing

Limitations:
- Simple heuristics, not perfect
- False positives possible (e.g., student adds text but not relevant example)
- Post-MVP: Could use AI to check completion more accurately

UI Impact:
- If completed → show success message
- If incomplete → show gentle nudge, offer help

---

12. Success Criteria

12.1 MVP1 is Complete When:

Teacher Functionality:

- ✅ Teacher can register and log in
- ✅ Teacher can create task with at least one success criterion
- ✅ Teacher cannot save task without criteria (validation enforced)
- ✅ Teacher can start session and generate class code
- ✅ Teacher can view live student list (real-time updates)

- ✅ Teacher can review AI-generated feedback
- ✅ Teacher can edit feedback (vague comment popup works)
- ✅ Teacher can approve feedback for release
- ✅ Teacher can view class-level insights
- ✅ Teacher can end session
- ✅ Teacher can view session summary

Student Functionality:

- ✅ Student can join session with 6-digit class code
- ✅ Student can see task prompt and success criteria (fixed goal box)
- ✅ Student can submit written response
- ✅ Student receives feedback after teacher approval
- ✅ Student sees feedback in three-question format (goal, strengths, growth areas, next steps)
- ✅ Student must choose one next step before proceeding
- ✅ Student can submit one revision
- ✅ Student sees completion status
- ✅ Student completes session (no persistent data)

System Requirements:

- ✅ No feedback appears without goal being shown first
- ✅ All feedback includes task, process, AND self-regulation comments
- ✅ All growth areas are specific (pass content anchor check)
- ✅ All next steps map to growth areas
- ✅ ATQ and signposting next steps included when relevant
- ✅ CTAs are ≤30 characters
- ✅ Student cannot exit without selecting next step
- ✅ No vague, judgmental, or ability-based language reaches students
- ✅ Session data is ephemeral (auto-deleted after session)
- ✅ Teacher maintains authority over all feedback

END OF IMPLEMENTATION PLAN

Next Steps:

1. Review this plan with full development team
2. Confirm technical stack and architecture decisions
3. Set up development environment and project structure
4. Begin implementation following phased approach
5. Regular check-ins against success criteria

Contact:

For questions or clarifications, refer back to source documents:
- LARA Feedback Principles
- MVP1 Core Feedback Requirements (Non-Negotiable)
- Canonical MVP1 User Flow (One-Page Summary)