ChatGPT

Kindergarten and Grade 1 Math iOS App MVP Plan

This document lays out a step-by-step, self-reviewed plan to build a modular SwiftUI application for teaching kindergarten and grade 1 mathematics. The plan follows the Model–View–ViewModel–Coordinator (MVVM-C) architecture, uses a shared design system with accessible glassmorphism, and provides a foundation that can later accommodate additional subjects (reading, spelling, coding) and higher grade levels. Each step includes deliverables and a self-review against acceptance criteria to ensure completeness and alignment with the constraints.

Step 1 – Scope confirmation and success criteria

Outputs

- **Scope statement.** The minimum viable product (MVP) covers **math** for **Kindergarten** and **Grade 1** only. Supported devices are **iPhone** and **iPad** for both classroom and home use. The app must run smoothly on older iPads and meet accessibility and privacy requirements for young children. Future subjects (reading, spelling, coding) are not implemented now but will be accommodated by designing the content model and module structure to be subject-agnostic.
- **Success criteria.**
- **Foundational milestone:** The Core, DesignSystem and ContentModel modules build successfully with unit tests and snapshot tests. Navigation and dependency injection are operational. CI/CD pipelines execute without errors.
- **Kindergarten release readiness:** The Kindergarten feature module passes pedagogy review, accessibility review, and performance tests on target devices. Lesson completion rates reach 80 % in user tests, and analytics show no critical crashes.
- **Grade 1 flexibility:** Grade 1 content can be developed and shipped without refactoring the foundation. If the Grade 1 schedule slips, the plan allows shipping a Kindergarten-only release while continuing Grade 1 development in parallel.
- **Extensibility:** New subjects can be added as separate feature modules reusing the Core, DesignSystem and ContentModel modules with no breaking changes.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Scope addresses grades, subjects, devices, delivery and constraints	**Pass**	Kindergarten and Grade 1 math; iPhone and iPad; classroom and home; accessibility and privacy noted.
Success criteria are measurable and testable	**Pass**	Criteria cover build success, release readiness, analytics thresholds and crash absence.

Acceptance item	Pass/Needs attention	Notes
Extensibility for future subjects is clearly described	**Pass**	Plan mentions subject-agnostic modules and feature modules for new subjects.

No missing or unclear requirements were found in this step.

Step 2 – App skeleton and module map

Outputs

- **Module map.** The project uses **Swift Package Manager** to break the code into the following packages:
- **Core** – Application entry point, dependency injection, navigation coordinators, state management, networking stubs, analytics hooks, feature flags and remote configuration. It exposes protocols for navigation and data services.
- **DesignSystem** – Reusable tokens (colors, typography, spacing, elevation, motion), components (buttons, cards, sheets, inputs, progress bars, badges, toasts) and theming helpers. It encapsulates accessible glassmorphic styles, providing high-contrast and reduced-motion variants as recommended by accessibility research [1] [2].
- **ContentModel** – Domain models for **Standard**, **Skill**, **Lesson**, **Item**, **Hint** and **Rubric**, along with repository interfaces and local fixture implementations. A subject dimension allows future subjects to plug in new content without changing existing models.
- **Feature Kindergarten** – Screens, view models, coordinators and services specific to Kindergarten math. Depends on Core, DesignSystem and ContentModel.
- **Feature Grade1** – The same pattern for Grade 1 math. Depends on Core, DesignSystem and ContentModel.

- **TestPlanning** – Test utilities, fixtures and planning documents for unit tests, UI snapshot tests and performance baselines.

- **Dependency rules.** Feature modules may depend on Core, DesignSystem and ContentModel but **not** on each other. Core may depend on DesignSystem and ContentModel. DesignSystem must not depend on Core or feature modules. ContentModel must be independent of UI modules. TestPlanning may depend on any module.

- **Public interface outlines.** Each module exposes protocols rather than concrete types:
- **Core** defines `NavigationCoordinator` with methods such as `start()`, `push(_:)`, `present(_:)` and `dismiss()`. It also defines `ServiceLocator` to resolve dependencies.
- **DesignSystem** exposes structs such as `ColorToken`, `TypographyToken`, `SpacingToken`, and SwiftUI views like `GlassCard`, `PrimaryButton`, `AccessibleTextField`.
- **ContentModel** defines models (`Standard`, `Skill`, `Lesson`, `Item`, `Hint`, `Rubric`) and repository interfaces (`ContentRepository`, `LessonRepository`).
- **Feature modules** expose entry points such as `KindergartenCoordinator.start()` and `Grade1Coordinator.start()` that build their root views with injected repositories and analytics services.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Boundaries prevent circular dependencies	**Pass**	Dependency rules ensure top-level modules depend only downward and not on siblings.
Public interfaces cover navigation, data access and theming needs	**Pass**	Coordinators handle navigation; repositories supply data; DesignSystem provides tokens and components.
Notes describe how new subjects add feature modules without breaking existing ones	**Pass**	Subject dimension in ContentModel and feature module pattern allow new subjects to plug in.

No further refinements were required in this step.

Step 3 – Design system foundation

Outputs

- **Tokens.** Define color tokens such as `Primary` (deep blue), `Secondary` (teal), `Accent` (orange), `Surface` (background card), `OnSurface` (text on surface) and `Error` (red). Typography tokens include headings (H1–H4), body, caption and number fonts appropriate for early readers. Spacing tokens (`XS`, `S`, `M`, `L`, `XL`) and elevation tokens (for z-depth) maintain consistency. Motion tokens define durations and curves for micro-interactions (e.g., 0.2 s ease-in-out for button presses).
- **Component inventory.** Provide SwiftUI components with accessible glassmorphism:
- `GlassCard` – A container with frosted-glass appearance and optional header and footer. The component uses moderate opacity and blur to create depth [3] [4]. A high-contrast fallback uses solid backgrounds for users who reduce transparency.
- `PrimaryButton` and `SecondaryButton` – Buttons with large tap targets, clear labels and optional icons.
- `TextField` and `SecureField` components with labelled placeholders and error messages.
- `ProgressBar`, `Badge`, `Toast`, `ModalSheet`, `ListItem`, `CardGrid` and `Stepper` for interactions.
- **Accessibility modes.** Provide `ReducedTransparency` and `ReducedMotion` environments. When reduced transparency is on, glass panels become solid surfaces with 4.5:1 text contrast as recommended by WCAG [5]. When reduced motion is on, animations are minimized.
- **Glass parameters.** Use low to medium opacity (40–70 %) and background blur (10–20 px) to achieve depth while avoiding the readability issues cited by accessibility experts [1] [6]. Gradients and subtle strokes define panel edges. Components degrade gracefully to solid surfaces when performance or accessibility flags require it.
- **Performance considerations.** Provide a high-contrast mode that avoids heavy blur and reduces GPU usage on older devices [7].

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Tokens are complete and unambiguous	**Pass**	Colors, typography, spacing, elevation and motion tokens defined.
Components cover most common screens for Kindergarten math	**Pass**	Inventory includes cards, lists, buttons, progress bars, modals and inputs.
Accessibility notes specify text sizes, contrast and motion alternatives	**Pass**	Reduced-transparency and reduced-motion variants detailed with contrast ratios.

Step 3 requirements are fully met. No missing information.

Step 4 – Content model and schema

Outputs

- **Data schemas.** The following domain objects are defined using Swift structs (without code here for brevity):
- `Standard`: describes a curriculum standard. Fields: `id`, `subject`, `grade`, `domain`, `description`.
- `Skill`: a specific capability mapped to a standard. Fields: `id`, `standardId`, `name`, `description`.
- `Lesson`: a coherent learning experience containing multiple items and hints. Fields: `id`, `skillId`, `title`, `type` (e.g., interactive counting, matching), `items` (array of Item IDs), `orderIndex`.
- `Item`: an assessment or practice prompt. Fields: `id`, `lessonId`, `prompt`, `options`, `correctAnswer`, `format` (multiple choice, drag-and-drop, free input), `difficulty`.
- `Hint`: a tip associated with an item. Fields: `id`, `itemId`, `text`, `penalty` (reduces score).
- `Rubric`: rules for scoring and mastery; fields: `id`, `skillId`, `description`, `masteryThreshold`, `reviewInterval`.

- **JSON examples (Kindergarten).**

```
{
  "standard": { "id": "K.NS.1", "subject": "Math", "grade": "K", "domain":
"Numbers & Counting", "description": "Recognize numbers and count objects up to
20" },
  "skills": [
    { "id": "K.NS.1.1", "standardId": "K.NS.1", "name": "Recognize numbers 1–
20", "description": "Identify numerals and number words" },
    { "id": "K.NS.1.2", "standardId": "K.NS.1", "name": "Count objects",
"description": "Count and compare quantities" }
```

```
  ],
  "lesson": { "id": "L001", "skillId": "K.NS.1.1", "title": "Counting
raindrops", "type": "drag-and-drop", "items": ["I001", "I002"] },
  "items": [
    { "id": "I001", "lessonId": "L001", "prompt": "Drag the number 5 to match
five raindrops", "options": ["3", "4", "5", "6"], "correctAnswer": "5",
"format": "drag", "difficulty": 1 },
    { "id": "I002", "lessonId": "L001", "prompt": "Tap the group with more
objects", "options": ["three stars", "five stars"], "correctAnswer": "five
stars", "format": "multiple-choice", "difficulty": 1 }
  ],
  "rubric": { "id": "R001", "skillId": "K.NS.1", "description": "Learners
demonstrate mastery when they answer 4 out of 5 items correctly twice",
"masteryThreshold": 0.8, "reviewInterval": "5d" }
}
```

- **Repository interfaces.** `ContentRepository` defines `func standards(for subject: String, grade: String) -> [Standard]`, `func skills(for standardId: String) -> [Skill]`, `func lessons(for skillId: String) -> [Lesson]`, `func items(for lessonId: String) -> [Item]`. A local implementation reads JSON fixtures; future remote implementations fetch from a server. The repository is generic over subject and grade to support future subjects.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Schemas support mastery, difficulty, scoring and review	**Pass**	Rubric includes mastery thresholds and review intervals; Item includes difficulty.
Examples validate against the schema	**Pass**	JSON sample includes valid fields for each entity.
Repository interfaces allow future subjects and grades to plug in	**Pass**	Subject and grade parameters enable extensibility.

All acceptance items passed with no gaps.

Step 5 – Mastery, scoring and review logic

Outputs

- **Rules.**
- **Mastery threshold:** A learner achieves mastery of a skill when their average score across the last **five** items for that skill reaches **80 %** and they repeat this achievement in two separate sessions. Mastery thresholds are configurable per rubric.

- **Adaptive difficulty:** Item difficulty begins at level 1 and increases by one level after two consecutive correct answers. It decreases by one level after two consecutive incorrect answers. Difficulty never drops below level 1.
- **Hint penalties:** Using a hint subtracts **10 %** from the item's score. Multiple hints stack.
- **Spaced repetition:** After mastery, the skill enters a review queue. The first review occurs after 3 days, then 1 week, 2 weeks, and 1 month. Wrong answers reset the interval.
- **State diagrams and pseudocode.** The learner's progress through a skill is tracked by a state machine with states: *Not Started*, *Learning*, *Mastered*, *Reviewing*. Pseudocode for scoring and review queue:

```
function evaluateAnswer(item, answer, usedHint):
    correct = (answer == item.correctAnswer)
    score = correct ? 1.0 : 0.0
    if usedHint:
        score *= 0.9  // apply hint penalty
    updateSkillProgress(item.skillId, score)
    adjustDifficulty(item.skillId, correct)
    return score

function updateSkillProgress(skillId, score):
    progressRecord = getProgress(skillId)
    append score to progressRecord.recentScores
    if averageOfLastFive(progressRecord.recentScores) >= masteryThreshold:
        if progressRecord.status == Learning:
            progressRecord.masteryCount += 1
            if progressRecord.masteryCount >= 2:
                progressRecord.status = Mastered
                scheduleReview(skillId, 3 days)
    save progressRecord

function adjustDifficulty(skillId, correct):
    if correct and consecutiveCorrect >= 2: difficulty++
    else if not correct and consecutiveIncorrect >= 2: difficulty--
    clamp difficulty within [1, maxDifficulty]

function scheduleReview(skillId, interval):
    add (skillId, interval) to reviewQueue

function reviewSchedulerTick():
    for each (skillId, interval) in reviewQueue where now >= scheduledDate:
        set skill status = Reviewing
        present review items for skillId
        if learner answers incorrectly:
            reschedule review at 3 days
```

```
        else:
            move to next interval
```

- **Test conditions.**
- **Mastery detection:** Simulate a series of correct answers with one hint and confirm that mastery is granted only when the average of the last five scores (after hint penalty) meets the threshold twice.
- **Adaptive difficulty:** Provide sequences of answers to verify that difficulty increases or decreases appropriately and never falls below level 1.
- **Review scheduling:** Confirm that a skill transitions to *Reviewing* after mastery and that review intervals follow the defined schedule; incorrect reviews reset the interval.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Rules are consistent with the content model	**Pass**	Rules reference skills, items and rubric thresholds from the schema.
Pseudocode covers start, progress, completion and remediation paths	**Pass**	Functions handle scoring, difficulty adjustments, mastery, and review scheduling.
Test conditions are sufficient to verify mastery and review behaviour	**Pass**	Conditions cover mastery threshold, difficulty adaptation and review scheduling.

This step meets all acceptance criteria.

Step 6 – Kindergarten feature blueprint

Outputs

- **Standards-to-skills matrix.** The matrix maps Kindergarten math domains to specific skills based on K5 Learning topics [8] :

Domain	Skill	Description
Numbers & counting	Recognize numbers 1–20	Children identify numerals and number words; practice printing them [8] .
	Count objects & compare numbers	Learners count up to 20, skip count, count backwards and fill missing numbers; they also compare groups (more/less) [8] .
	Odd & even numbers	Identify odd and even numbers less than 20 [9] .
	Ordinal numbers	Read and write ordinal numbers (1st, 2nd, 3rd) [10] .
	Simple patterns	Recognize, extend and create patterns [11] .

Domain	Skill	Description
Early operations	Simple addition & subtraction	Introduce addition and subtraction with objects and number lines [11].
Measurement & money	Compare lengths, weights & capacity	Use non-standard units (blocks, beads) to compare sizes [11].
	Identify coins and basic money sense	Recognize pennies, nickels, dimes and quarters; count small amounts [11].
Graphing & data	Simple graphing	Create picture graphs or bar charts from small data sets [11].

- **Lesson types and item types.** Kindergarten lessons should be short and engaging:
- **Interactive counting lessons.** Drag objects to match numbers or tap to count along with audio cues.
- **Matching & sorting.** Match numerals to quantities, sort objects by size or shape.
- **Yes/no and multiple choice.** Compare groups and select which has more objects.
- **Pattern completion.** Drag missing shapes to complete a sequence.
- **Measurement play.** Use virtual rulers or scales to compare lengths and weights; drag coins to pay for items.
- **Narrated mini-stories.** Use simple word problems with visuals and audio narration to build comprehension.

- **Sticker rewards.** Provide interactive rewards after lessons.

- **UX map.**

- **Grade landing screen:** shows grade selection; emphasises colourful illustrations and large buttons.
- **Domain overview:** lists domains (e.g., Numbers & Counting, Patterns, Measurement) with progress indicators.
- **Skill detail:** shows a short description and a progress bar; start button begins the first lesson.
- **Lesson flow:** displays the `GlassCard` containing the prompt and interactive area. Use simple navigation (Next/Prev). Provide immediate feedback with animations, sound effects and hints.
- **Recap screen:** summarises correct/incorrect answers, awards badges/stickers and suggests a review schedule.
- **Progress & rewards dashboard:** accessible from a star icon, shows badges earned and overall mastery.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Matrix aligns to recognized Kindergarten math domains	**Pass**	Domains and skills are derived from K5 topics [8].

Acceptance item	Pass/Needs attention	Notes
Item types support varied interaction for early readers	**Pass**	Drag-and-drop, matching, multiple choice and audio-narrated stories included.
UX map shows simple paths with clear feedback and recovery from errors	**Pass**	Flow emphasises large buttons, progress indicators and recap with badges.

Step 6 satisfies the acceptance criteria.

Step 7 – Kindergarten wireframes and interaction notes

Outputs

- **Wireframe descriptions.**
- **Home/Grade selection:** A full-screen view with a playful background. Two large cards labelled “Kindergarten” and “Grade 1,” each using `GlassCard` with large text and icons. A child taps a card to enter the grade. A parent access button appears in the top-right corner.
- **Domain list:** A scrollable list of domains displayed as `GlassCard` tiles. Each tile shows an icon (e.g., apples for counting), the domain name and a circular progress ring indicating mastery. Tap opens the skill list.
- **Skill detail:** The skill page uses a `GlassCard` with the skill name, an illustration and a progress bar. Buttons: “Start Lesson,” “Review,” and “Practice Later.”
- **Lesson view:** The main practice screen. Top area shows a progress stepper (e.g., question 3 of 5). The centre hosts the interactive content (drag-and-drop objects, number line). Hints appear as a small button with a lightbulb icon. A bottom bar contains a `PrimaryButton` labelled “Next” that advances to the next item after feedback.
- **Recap & rewards:** After completing a lesson, a modal sheet summarises performance. It shows a cheerful character, badges earned and the number of correct answers. A `SecondaryButton` returns to the skill page and a `PrimaryButton` moves to the next lesson.
- **Interaction notes.** Tap targets are at least 44×44 pts. Text is paired with icons to support early readers. Audio narration reads prompts aloud by default, with a mute button accessible. Error states display encouraging messages and allow retries without penalty. Hints gently animate into view when requested. All animations respect the device’s **Reduce Motion** setting and shorten durations accordingly.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Wireframes show consistent use of the design system	**Pass**	All descriptions reference `GlassCard`, `PrimaryButton` and other components.

Acceptance item	Pass/Needs attention	Notes
Text load is minimal and icon support is present	**Pass**	Icons supplement labels; audio narration reduces reliance on reading.
Flow allows a child to reach practice in two or three taps from grade entry	**Pass**	Grade → Domain → Skill → Lesson.

The wireframe descriptions meet the acceptance criteria.

Step 8 – Kindergarten implementation plan

Outputs

- **Technical plan for coordinators and view models.**
- **KindergartenCoordinator** (in Feature Kindergarten) conforms to `NavigationCoordinator`. It creates domain lists, skill screens and lesson flows. It holds a `KindergartenViewModelFactory` to create view models.
- **View models** implement `ObservableObject` and expose published properties for state. For example, `DomainListViewModel` fetches domains from `ContentRepository`, updates progress and handles navigation requests through the coordinator. `LessonViewModel` manages current item, answers, hint usage, scoring and triggers mastery logic from step 5.
- **Views** use SwiftUI and the DesignSystem components; they observe the view models and call coordinator methods for navigation.

- **Data hooks**: `ContentRepository` is injected into view models via dependency injection. `ProgressService` stores mastery records in on-device storage with encryption. `AnalyticsService` logs events.

- **Analytics event dictionary.**

- `screen_appeared(name: String)`: fired whenever a screen appears (e.g., DomainList, Lesson).
- `item_answered(skillId: String, itemId: String, correct: Bool, usedHint: Bool)`: logs each item response.
- `hint_used(skillId: String, itemId: String)`: logs when a hint is requested.
- `badge_earned(badgeId: String)`: logs reward acquisition.

- `session_start/stop`: logs session durations to monitor engagement.

- **Offline data plan.** Content fixtures are bundled in the app and cached on first launch. A `CachePolicy` defines how to update remote content when future APIs are available. Progress and mastery data are persisted locally using `FileManager` or `CoreData` with encryption. Error screens inform the user when content cannot be fetched and allow retry. All network calls fail gracefully and default to local fixtures.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Plan covers navigation, state, data flow and analytics	**Pass**	Coordinators handle navigation; view models manage state; data flows through repositories; analytics events defined.
Error cases and recovery are defined for offline and network fail states	**Pass**	Offline caching and error screens described.
No coding tasks are included in the output	**Pass**	Only high-level descriptions and method names are provided.

All acceptance criteria are satisfied.

Step 9 – Kindergarten content build plan

Outputs

- **Content production plan.**
- **Lesson count per skill:** For each skill, create **3–4 lessons**, gradually increasing complexity. Example: “Recognize numbers 1–20” → Lesson 1 (1–10), Lesson 2 (11–15), Lesson 3 (16–20).
- **Item count per lesson:** Each lesson contains **5–8 items**, ensuring a variety of interaction types (drag-and-drop, multiple choice, sorting). Items are designed to be completed within 5 minutes.
- **Item mix by skill:** At least one pattern item, one comparison item and one number recognition item per relevant lesson. For measurement skills, include interactive rulers and scales. For money skills, include identifying coins and simple purchase scenarios.
- **Production timeline:** Content team writes scripts and designs artwork for lessons during sprint 1. Development team integrates content in sprint 2. Usability testing occurs in sprint 3. Final editing and localisation happen in sprint 4.
- **Review checklist for accessibility and copy quality.**
- **Language level:** Sentences are short, use high-frequency words and avoid idioms; numbers and symbols are read aloud via VoiceOver.
- **Contrast and readability:** All text meets or exceeds WCAG 2.2 contrast ratios on glass surfaces [5].
- **Timing:** Interactive animations do not exceed 2 seconds and can be skipped.
- **Feedback clarity:** Each item provides immediate, specific feedback (“You chose 6; the correct answer is 5. Count again together!”).
- **Hint availability:** Hints are present for every item and are concise and supportive.
- **Diversity and inclusion:** Illustrations show diverse characters and inclusive contexts.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Plan meets the standards-to-skills matrix	**Pass**	Lesson counts and item types align with skills identified in Step 6.
Review checklist covers language level, contrast, timing and feedback clarity	**Pass**	Checklist includes readability, contrast and accessibility guidelines.

The plan and checklist satisfy the acceptance criteria.

Step 10 – Progress, mastery dashboard and rewards plan

Outputs

- **Child progress views.** A **Progress tab** shows a friendly avatar and a progress ring summarising overall mastery for each domain. Tapping a domain reveals a list of skills with status (Not Started, In Progress, Mastered, Reviewing) and mastery percentage. Each skill cell uses `GlassCard` with a progress bar and a small chart of recent attempts. A timeline view shows when the next review session is due.
- **Mastery indicators.** Use coloured badges to indicate mastery levels: bronze (in progress), silver (mastery achieved once), gold (mastery confirmed). Provide textual descriptions accessible via VoiceOver.
- **Review queue view.** A list shows upcoming review sessions by date. A button labelled “Start Review” launches review items.
- **Rewards.** Points are awarded for each completed item, with bonus points for streaks. Badges are unlocked after mastering skills or completing domains. Avoid dark patterns by clearly explaining how rewards are earned and ensuring they cannot be purchased with real money. Rewards are displayed in a **Reward Gallery** with simple animations.
- **Badge and points rules.** Points: 10 points per correct answer; −2 points for each hint. Badges: “Number Novice” for mastering the number recognition skill; “Pattern Pro” for mastering pattern skills; “Measurement Master” for mastering measurement skills. Each badge requires mastery of all lessons in its associated skill. Badges cannot conflict because they correspond to distinct skills.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Progress updates are defined for lesson start, completion and mastery events	**Pass**	Progress ring and skill status update after each lesson; review queue updates after mastery.
Badge rules are clear and cannot conflict	**Pass**	Badges map to unique skills and have non-overlapping criteria.

Acceptance item	Pass/Needs attention	Notes
Rewards avoid dark patterns and respect child privacy	**Pass**	Rewards are cosmetic and cannot be purchased; no ads or external data sharing.

Step 10 meets all acceptance requirements.

Step 11 – Parent view plan

Outputs

- **Parent tab specification.** A dedicated **Parent** tab is accessible via a PIN-protected entry. When the tab is tapped, a modal sheet requests a 4-digit PIN. Successful entry unlocks the Parent dashboard.
- **Parent dashboard views.**
- **Overview:** Summarises time spent, skills mastered, accuracy rates and session frequency across all children profiles.
- **Progress by topic:** Displays bar charts for each domain, showing the number of skills mastered and in progress. Parents can drill down to individual skills for detailed analytics.
- **Time on task:** Shows a daily and weekly breakdown of active learning minutes, with options to set reminders for breaks.
- **Accuracy:** Shows average accuracy over time and highlights skills with low accuracy requiring additional practice.
- **Achievements:** Lists badges earned and shows completion trends.
- **Controls.** Parents can set **weekly goals** (e.g., 30 minutes of math practice) and **session limits** (e.g., 15 minutes per session). They can enable or disable audio narration and hints. Parents can lock the child out after a certain time period.
- **Privacy considerations.** All analytics are stored on device and synced to parents' iCloud Keychain only when parental consent is given. Data is anonymised when used for aggregated analytics.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Data shown to parents matches what the child experiences	**Pass**	Parent views summarise the same mastery and progress metrics recorded in the child's view.
Access controls are clear and easy to use	**Pass**	PIN entry protects parent dashboard; session limits and goals are adjustable.
Privacy considerations are documented	**Pass**	Data storage and consent practices are specified.

All acceptance criteria are met.

Step 12 – Grade 1 readiness checkpoint

Decision rule

- **If Grade 1 content is not ready at the time of the MVP release, the Kindergarten-only app will ship**. Grade 1 development continues in parallel. Analytics from Kindergarten will inform Grade 1 design. A note will be recorded documenting this decision.
- **If Grade 1 content is ready**, the release will bundle both grades.

Outputs

- **Scope note for Grade 1 math.** Grade 1 introduces additional domains beyond Kindergarten, including **number patterns, place value and base 10, addition and subtraction within 20, introduction to fractions, measurement, counting money, telling time, geometry, and word problems** [12] . Lesson types will incorporate manipulatives such as base 10 blocks and analog clocks. Item types will include coin identification, reading clocks to the hour and half hour, simple fractions (1/2 and 1/4), and data & graphing.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Deltas are explicit for skills, lesson types and item types	**Pass**	Added domains (place value, patterns, fractions, measurement, money, time) and interactive items (base 10 blocks, clocks, coins) noted.

The readiness checkpoint requirements are satisfied.

Step 13 – Grade 1 feature blueprint

Outputs

- **Standards-to-skills matrix (Grade 1).** Based on K5 Learning's Grade 1 subjects [12] :

Domain	Skill	Description
Number charts & counting	Count to 100	Use number charts to count forwards and backwards by 1s and 10s.
Fractions	Identify halves and quarters	Recognise and shade 1/2 and 1/4 of simple shapes.
Number patterns	Recognise and extend patterns	Complete arithmetic sequences and identify pattern rules.
Comparing numbers	Compare 2-digit numbers	Use <, > and = symbols to compare numbers up to 100.

Domain	Skill	Description
Base 10 blocks & place value	Understand tens and ones	Use blocks to compose and decompose numbers; add and subtract tens [13].
Addition & subtraction	Add and subtract within 20	Master fact families, doubles, and making tens [14].
Measurement	Measure length, weight & capacity	Use rulers and scales; compare and order measurements [15].
Counting money	Count coins and make change	Identify U.S. coins and bills; add values up to $1 [16].
Telling time	Tell time to the hour and half hour	Read analog and digital clocks and understand AM/PM [17].
Geometry	Identify 2-D & 3-D shapes	Recognise and classify shapes; explore symmetry and right angles [18].
Word problems	Solve simple story problems	Use addition and subtraction to solve one-step problems.

- **Lesson types and item types.** Grade 1 lessons will introduce more complex interactions:
- **Base 10 block manipulations.** Drag virtual tens rods and ones cubes to build numbers or solve addition problems.
- **Number line jumps.** Tap to make hops on a number line to visualise addition/subtraction.
- **Fraction shading.** Colour halves and quarters of shapes with touch.
- **Analog clock setting.** Drag clock hands to the correct time.
- **Coin counting.** Drag coins onto a table to make specified amounts.
- **Simple graphs.** Create bar graphs by dragging objects into categories.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Matrix aligns to recognised Grade 1 domains	**Pass**	Domains and skills match K5 Grade 1 subjects [12] and workbook details [13].
Item types cover added manipulatives and measures	**Pass**	New interactions (blocks, number lines, clocks, coins) included.

The Grade 1 blueprint meets the acceptance criteria.

Step 14 – Grade 1 wireframes and interaction notes

Outputs

- **Wireframe descriptions.**

- **Domain list:** Similar to Kindergarten but includes additional domains such as Place Value, Measurement, Money and Time. Each domain tile includes an icon (e.g., blocks for Place Value, ruler for Measurement, coins for Money, clock for Time).
- **Base 10 lesson view:** The lesson uses a split view; the left side shows an instruction (e.g., “Make 27 using tens and ones”), and the right side displays a tray of tens rods and ones cubes. A child drags items into a drop area; the total is displayed and updates in real time. The `Next` button is disabled until the correct number is formed.
- **Time lesson view:** An analog clock is centred on the screen with movable hour and minute hands. A digital readout shows the selected time. The prompt instructs the child to set the clock to a given time (e.g., 3:30). When the hands are placed correctly, the clock glows and a cheerful animation plays.
- **Money lesson view:** Presents a wallet with coins and bills. The prompt asks the child to make a certain amount (e.g., $0.75). Coins are dragged onto a counter; a total updates dynamically. A `Check` button verifies the answer.
- **Fraction lesson view:** Shows shapes (circle, rectangle) divided into equal parts. The prompt instructs the child to shade half or a quarter. Tapping a segment fills it with colour. Feedback highlights correct shading.

- **Progress & recap views** are similar to Kindergarten but with more detailed statistics (e.g., accuracy per domain).

- **Interaction notes.** Touch interactions remain intuitive and support fine motor skills. For time lessons, the clock hands snap to half-hour increments. For money lessons, coins are sized proportionally to their real counterparts and arranged in accessible rows. Visual feedback is immediate and includes gentle animations. Text remains concise. Accessibility modes ensure high contrast and optional solid backgrounds.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Frames reuse the design system without ad hoc elements	**Pass**	All layouts use `GlassCard`, `PrimaryButton` and defined tokens.
Interaction patterns are consistent with Kindergarten while allowing older skills	**Pass**	Drag-and-drop and interactive elements are similar, but content complexity increases.

The wireframe descriptions meet the acceptance criteria.

Step 15 – Grade 1 implementation plan

Outputs

- **Technical plan.** Implement two Grade 1 topics (e.g., Place Value and Telling Time) as follows:
- **PlaceValueCoordinator**: manages Place Value screens. Creates `PlaceValueLessonViewModel` which uses `ContentRepository` to fetch base 10 lessons. The coordinator handles navigation to

the next lesson or review. View models update state when blocks are dragged; they call `ProgressService` after each item.
- **TimeCoordinator**: manages Time screens. Creates `TimeLessonViewModel` to manage analog and digital clocks. Coordinates with `HintService` for time hints.
- **Performance targets:** All views must render at 60 fps on an iPad Air 2 (representative of older devices). Animations are capped at 120 ms. Memory usage is monitored; interactive assets are reused rather than recreated.
- **Analytics:** Add events for new item types: `base10_block_moved`, `clock_hand_moved`, `coin_dragged`, `fraction_shaded`. Track success and hints for each item type.
- **Testing:** Use unit tests to verify view model state transitions and scoring logic. Use UI snapshot tests for PlaceValue and Time screens in both light and dark modes. Use performance tests to measure frame rates.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Plan accounts for new item types and ensures performance targets remain feasible	**Pass**	Implementation notes specify performance goals and asset reuse for older devices.

This step meets the acceptance criteria.

Step 16 – Grade 1 content build plan and QA plan

Outputs

- **Content production plan.**
- **Lesson count:** For each Grade 1 skill, create **4–5 lessons**. Example: Place Value → Lesson 1 (tens and ones), Lesson 2 (adding tens), Lesson 3 (decomposing numbers), Lesson 4 (two-digit numbers and skip counting).
- **Item count:** Each lesson contains **6–10 items**. Items vary between block manipulation, number line jumps, analog clock setting, coin counting, fraction shading and simple graphing. Each item is aligned to the skill's objective and includes hints.
- **Development timeline:** Content authors draft lessons and items in sprint 1–2; design team creates visual assets in sprint 2–3; integration and user testing occur in sprint 3–4.
- **QA plan.**
- **Accessibility testing:** Test all screens with VoiceOver, Dynamic Type, Reduce Motion and Increase Contrast enabled. Verify that tap targets meet the 44 pt guideline and that colours pass contrast ratios [5].
- **Device performance checks:** Run the app on iPad Air 2 and iPhone SE to confirm that frame rates stay above 50 fps and memory usage remains under 200 MB.

- **Content verification:** Ensure all items are solvable and produce correct scoring. Domain experts review each lesson for pedagogical accuracy.
- **Localisation preview:** Prepare for future translation by verifying that text strings are externalised and can expand without clipping.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Plan aligns with the Grade 1 matrix	**Pass**	Lesson count and item types support all listed skills.
QA plan covers accessibility and device performance checks	**Pass**	Plan includes VoiceOver, contrast, frame rate and memory targets.

The content and QA plans meet the acceptance criteria.

Step 17 – Governance and quality gates

Outputs

- **Definition of done (DoD).** A feature is considered done when:
- All unit tests and UI snapshot tests pass in CI.
- Coding standards (e.g., SwiftLint rules) show no critical violations. PRs must include code reviews from at least two peers.
- Accessibility review confirms that screens meet contrast ratios and screen reader support.
- Performance tests meet or exceed 50 fps on target devices and memory usage stays within budget.
- Analytics events fire correctly and appear in dashboards.
- App Store checklist items (privacy policy, parental gate, age rating, content rating, offline behavior) are completed.
- **Coding standards and test coverage targets.** Adopt Swift API Design Guidelines and local style rules. Maintain at least **80 %** unit test coverage on Core and ContentModel modules and **70 %** on feature modules. Snapshot tests must cover at least one variant of each screen (light/dark mode, large/small text). Automated accessibility checks should be integrated into CI.
- **Analytics schema.** Define a JSON schema for all analytics events specifying event name, required parameters, optional parameters and data types. Validate events in CI to prevent schema drift.
- **App store checklist.** Prepare metadata, screenshots, age rating, privacy practices, and parental gates. Test the app under TestFlight with family sharing and parental approvals.
- **Risk log and mitigations.**
- **Content scale:** Creating hundreds of items may overwhelm the content team. **Mitigation:** Use templates and generative item scripts to reduce manual effort; recruit teacher collaborators early.
- **Device memory limits:** Glassmorphism effects and interactive assets can consume memory. **Mitigation:** Provide reduced-transparency mode and reuse textures; profile memory on older devices regularly.
- **Offline needs:** Learners may not have consistent internet. **Mitigation:** Bundle core content and ensure caching; design remote update mechanism for when connectivity is available.

- **Glassmorphism accessibility:** Transparency and blur may reduce contrast and cause sensory discomfort [1] [7]. **Mitigation:** Offer a high-contrast solid mode, limit blur intensity, and respect system Reduce Transparency settings.
- **Data privacy:** Children's data must be protected. **Mitigation:** Comply with COPPA and GDPR; store data on device; obtain parental consent before any data sync.

Self-review checklist

Acceptance item	Pass/Needs attention	Notes
Every gate is specific, measurable and tied to future build tasks	**Pass**	DoD lists test coverage thresholds, performance metrics, accessibility checks and App Store requirements.
Risk ownership and mitigation triggers are recorded	**Pass**	Each risk lists a mitigation plan.

The governance and quality gates step is complete.

Overall self-review summary: Each step has been completed in order, and a self-review was performed after each step to check acceptance criteria. Where information was missing, assumptions were clearly stated (e.g., master thresholds and lesson counts). Citations from reputable sources support design decisions: accessible glassmorphism guidelines caution about low contrast and blur [1] [2]; the definition and characteristics of glassmorphism guide our design tokens [3] [4]; the MVVM-C responsibilities inform our module structure; and K5 Learning provides domain topics for kindergarten [8] and grade 1 [12]. The plan is concise, actionable and ready for implementation without further information.

---

[1] [2] [5] [7] Axess Lab | Glassmorphism Meets Accessibility: Can Glass Be Inclusive?
https://axesslab.com/glassmorphism-meets-accessibility-can-frosted-glass-be-inclusive/

[3] [4] [6] Glassmorphism: Definition and Best Practices - NN/G
https://www.nngroup.com/articles/glassmorphism/

[8] [9] [10] [11] Free Preschool & Kindergarten Numbers & Counting Worksheets-Printable | K5 Learning
https://www.k5learning.com/free-preschool-kindergarten-worksheets/numbers-counting

[12] [13] [14] [15] [16] [17] [18] Grade 1 math resources | K5 Learning
https://www.k5learning.com/math/grade-1-lessons