ChatGPT

Math Learning App Work Breakdown Plan (iPhone & iPad)

1. Architecture Plan

1.1 Module Map and Dependency Rules

Module	Description	Dependencies
App (Shell)	Main SwiftUI app target that wires together dependency injection, navigation, and environment. Contains `AppDelegate` and `SceneDelegate`.	Depends on `Core`, `DesignSystem`, `ContentModel`, and selected Feature modules.
Core	Shared services: authentication, profiles, progress tracking, persistence (CoreData/SQLite), networking abstraction, analytics, feature flags, remote configuration, error handling, logging, localization.	No dependencies.
DesignSystem	Glassmorphism-based UI components, tokens (colors, typography, spacing, elevation, motion), icons, and illustrations. Exposes reusable SwiftUI views and modifiers. Ensures high contrast and accessible blur levels [1].	Depends on `Core` for localization and theming.
ContentModel	Domain models: `Standard`, `Skill`, `Lesson`, `Item`, `Hint`, `Rubric`. Implements mastery logic, difficulty progression, spacing rules, and scoring. Includes repository protocols and local JSON fixtures.	Depends on `Core`.
Feature Modules	Each grade (`Kindergarten`, `Grade1`… `Grade6`) is a separate SPM package. Contains feature-specific view models, coordinators, screens, and content resources. Uses the shared design system.	Depends on `Core`, `DesignSystem`, `ContentModel`. No direct dependencies between features.
TestingModules	Contains test utilities, mocks, fixture generators, and snapshot test helpers.	Depends on all modules under test.
Tooling	CI/CD scripts, code generators (e.g., SwiftGen), and lint rules.	N/A

Dependency rules: Feature modules must not depend on one another to avoid coupling. Shared modules (`Core`, `DesignSystem`, `ContentModel`) must have no dependencies on downstream features. Navigation is orchestrated via coordinators in each module (`MVVM-C`), decoupling view hierarchy from business logic. Repositories in `ContentModel` abstract data access; networking stubs allow offline development and testing.

1.2 Data Flow and Error Boundaries

1. **User initiates a lesson:** A feature coordinator creates a view model with dependencies injected from `Core` (e.g., content repository, progress service).
2. **View state updates:** The view model exposes `@Published` state. SwiftUI views bind to this state. Business logic lives in the view model; networking or database calls return through repositories and update state.
3. **Content retrieval:** The repository fetches lesson data from local cache or remote API via `Core` networking layer. Failing requests trigger retries and fallback to offline content.
4. **Analytics:** On each screen event (view appear, item answered), the view model logs events through the analytics service (declarative instrumentation). Feature flags and remote config determine experiments and gating.
5. **Error handling:** Errors are mapped to domain-specific error types and surfaced via view state. Centralized error presenters in the design system display friendly messages and retry actions.

Performance budgets: App launch < 2 seconds, cold start memory < 150 MB on iPad; scrolling in item lists > 60 fps; lesson screen should load within 200 ms. Glassmorphic components must not exceed 2–3 layered blurs per screen due to performance and accessibility concerns [2].

1.3 Clean Feature Modules (MVVM-C)

Each feature module follows MVVM-C:

```
// Example feature module skeleton (Grade1)
import SwiftUI
import Core
import DesignSystem
import ContentModel

public struct Grade1Coordinator: View {
    @StateObject private var viewModel = Grade1RootViewModel()

    public var body: some View {
        NavigationStack {
            Grade1HomeView(viewModel: viewModel)
                .environmentObject(viewModel)
        }
    }
}

final class Grade1RootViewModel: ObservableObject {
```

```
    @Published var currentLessonID: UUID?
    private let repository: ContentRepository
    private let progressService: ProgressService
    // Dependency injection via initializer
    init(repository: ContentRepository = LocalContentRepository(),
         progressService: ProgressService = .shared) {
        self.repository = repository
        self.progressService = progressService
    }
}

struct Grade1HomeView: View {
    @ObservedObject var viewModel: Grade1RootViewModel
    var body: some View {
        List {
            ForEach(viewModel.availableLessons) { lesson in
                NavigationLink(value: lesson.id) {
                    LessonRow(lesson: lesson)
                }
            }
        }
        .navigationDestination(for: UUID.self) { id in
            LessonView(viewModel: LessonViewModel(lessonID: id,
                                                  repository:
viewModel.repository,
                                                  progressService:
viewModel.progressService))
        }
    }
}
```

The coordinator is responsible for navigation and dependency injection. View models encapsulate business logic and call repository services. Views bind to state, ensuring separation of concerns.

2. Design System

2.1 Components and Tokens

The design system adopts a **glassmorphism** aesthetic with careful attention to accessibility. According to the Nielsen Norman Group, glassmorphic elements use translucency and blur to create depth, but overuse can cause accessibility challenges [1]. Therefore:

- **Color tokens:** Define a palette with high-contrast pairs. Primary surfaces use semi-transparent backgrounds with 30–50 % opacity and a 20 pt blur. Text and icons meet WCAG 2.2 contrast ratios (4.5:1 for body text) [3].
- **Typography:** San-serif typefaces with variable sizing (Large titles 34 pt, body 17 pt, captions 13 pt). Use SwiftUI's dynamic type support for scaling.

- **Spacing and elevation:** 4 pt baseline grid. Shadow and blur tokens to simulate depth: e.g., `GlassSurface.shadow = Color.black.opacity(0.1).blur(radius: 10)`.
- **Components:** Buttons (primary, secondary), cards, modals, input fields, progress bars, badges, toasts. Each component has an accessible fallback (e.g., no blur in high-contrast mode) and supports localization. Motion specifications define subtle transitions and reduce motion for users with vestibular sensitivities [4].

2.2 Accessibility Considerations

- Limit the number of translucent layers per screen to avoid focus disruption and cognitive load [5].
- Provide clear outlines or strokes on glass panels to differentiate them [6].
- Enable “reduce transparency” and “reduce motion” settings that replace blurred backgrounds with solid colors and disable parallax.
- Ensure text contrast meets 4.5:1 and 3:1 ratios [7] and check with accessibility inspectors.

3. Content Specification

3.1 Domain Model

Entity	Description
Standard	Represents a curricular standard (e.g., “Add within 100”). Contains identifier, grade, domain, and learning objective.
Skill	Subdivision of a standard covering a discrete capability. Skills map to multiple lessons.
Lesson	A sequence of instructional screens and practice items targeting one skill. Includes metadata (grade, skill, estimated duration), lesson type (video, interactive tutorial, practice).
Item	Individual question or exercise with stimulus (stem), possible answers, correct answer, distractors, hint references and rubric. Supports multiple item types: multiple choice, drag-and-drop, numeric input, drawing.
Hint	Step-by-step guidance tied to an item. Provides scaffolding and encourages strategic problem solving.
Rubric	Defines scoring rules, feedback and mastery threshold for an item or lesson.

Content is stored as JSON or delivered via APIs. Each lesson includes `metadata.json` and `items.json`. Local fixture seeding enables offline development.

3.2 Standards Matrix by Grade

Grade	Core Topics / Skills*	Lesson Count Target	Item Count Target
Kindergarten	Number recognition, counting to 20, odd/even, ordinal numbers, simple addition/subtraction, comparisons (more/less).	40 lessons	~400 items
Grade 1	Number charts & patterns, comparing numbers, base-10 blocks, place value, addition & subtraction (up to 20), simple fractions, measurement (length, weight), money & time, basic geometry, data/graphs.	45 lessons	~450 items
Grade 2	Skip counting, place value & rounding, addition and subtraction with regrouping, introductory multiplication, fractions, measurement, money, time, geometry, graphing.	50 lessons	~500 items
Grade 3	Place value & rounding to 10 000, multi-digit addition/subtraction, multiplication/division facts, order of operations, roman numerals, fractions & decimals, area/perimeter, time & calendar, geometry, data/graphs.	55 lessons	~550 items
Grade 4	Place value & rounding to millions, long multiplication and division, fraction operations, decimal operations, factors & prime factors, measurement conversions, geometry (angles, area, symmetry), roman numerals, data/graphs.	60 lessons	~600 items
Grade 5	Multi-digit operations including decimals & fractions, order of operations, integers, factoring & exponents, measurement & volume, geometry (3-D shapes), data analysis, introductory algebra.	65 lessons	~650 items
Grade 6	Scientific notation, complex fraction operations, decimal/percent conversions, integers and rational numbers, exponents, proportions, geometry (surface area, volume), measurement, data & algebra.	70 lessons	~700 items

*These topics correspond to the K5 Learning curriculum analysis previously conducted.

3.3 Mastery and Scoring Rules

- **Mastery Threshold:** A skill is mastered when learners achieve ≥ 80 % accuracy across three consecutive lessons or practice sessions.
- **Spaced Repetition:** If performance drops below 80 %, the app schedules review sessions after 1 day, 1 week, and 1 month.
- **Adaptive Difficulty:** Item difficulty increases when a learner answers three items correctly; decreases after two incorrect answers. Item selection uses an item response theory (IRT) model.

- **Hints and Remediation:** Hints reduce score weight by 20 %, but allow progress. After repeated incorrect answers, the app triggers remediation lessons or directs to previous grade content.

4. Work Breakdown and Timeline

4.1 Milestones

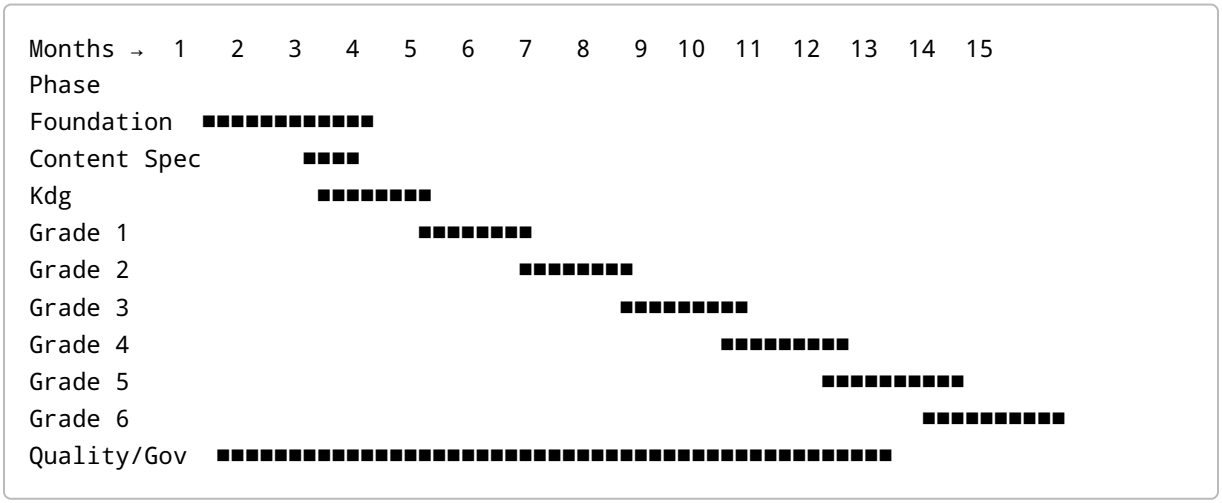
Phase	Duration	Deliverables	Key Risks
Foundation Setup	12 weeks	Project scaffolding with SPM packages, navigation and DI framework, Core services, design system, content models, CI/CD pipeline, baseline unit and snapshot tests.	Tooling complexity, initial performance tuning, alignment on design system.
Content Model & Assessment	4 weeks (overlap)	Define domain models, API/ contracts, scoring rules, seeding of fixtures.	Scope creep in content specification, dataset volume.
Grade 0 (Kindergarten)	8 weeks	Implement Kindergarten feature module; integrate number recognition, counting, simple operations; design interactive activities; test on devices; finalize 40 lessons.	Matching print worksheets with digital interactions; ensuring engagement for young learners.
Grade 1	8 weeks	Build Grade 1 module using shared components; implement base-10 blocks, time and money interactions; produce 45 lessons; pass pedagogy and accessibility reviews.	Complexity of new interaction types (coins, clocks); memory constraints on iPad mini.
Grade 2	8 weeks	Add skip counting games, regrouping operations, intro multiplication features; create 50 lessons.	Balancing difficulty progression; implementing dynamic arrays and multiplication visuals efficiently.
Grade 3	9 weeks	Include division, decimals, roman numerals, geometry tools, calendar; deliver 55 lessons.	Implementing calendar interactions; ensuring decimal/fraction models align.
Grade 4	9 weeks	Long division tutor, fraction/decimal conversions, factorization, advanced geometry; 60 lessons.	Performance of long-division animations; deep factor tree visualisations.

Phase	Duration	Deliverables	Key Risks
Grade 5	10 weeks	Multi-digit operations with decimals/ fractions, exponents, geometry volumes, introductory algebra; 65 lessons.	Ensuring algebraic notation is intuitive; memory footprint of 3-D geometry.
Grade 6	10 weeks	Scientific notation, integer arithmetic, proportions, surface area, advanced data & algebra; 70 lessons.	Complexity of scientific notation and rational number operations; advanced graph interactions.
Quality & Governance	Ongoing	Telemetry dashboards, authoring guidelines, weekly triage, performance tuning.	Data privacy compliance, content review bottlenecks.

4.2 Grade Release Order

We recommend shipping **Kindergarten → Grade 1 → Grade 2 → Grade 3 → Grade 4 → Grade 5 → Grade 6**, aligning with the natural progression of skills. Early grades require simpler interactions and allow the team to refine the design system and underlying mechanics before tackling more complex content.

4.3 Timeline Overview



(Darker bars indicate overlapping efforts.)

5. Definition of Done (DoD)

- **Coding Standards:** Follow Swift API design guidelines, MVVM-C architecture, and modular SPM structures. All code must have documentation comments and adhere to naming conventions.

- **Test Coverage:** ≥ 85 % unit test coverage for view models and services; snapshot tests for UI components using `ViewInspector` and `SnapshotTesting` frameworks. Performance tests for heavy operations.
- **Accessibility Checks:** All screens must pass VoiceOver navigation, contrast tests, and dynamic type scaling. Provide accessibility identifiers for UI tests.
- **Analytics Events:** Each feature emits events defined in the analytics schema: screen view, item attempt, hint usage, session duration. Events propagate to dashboards used for A/B testing and engagement analysis.
- **App Store Checklist:** Localisation coverage, privacy manifest, data collection disclosure, parental gate, offline mode verification, app icon and screenshots.
- **Review Gates:** Each grade must pass pedagogy review (content accuracy), accessibility review (contrast, voiceover, motion reduction), and performance thresholds (launch time, memory). QA sign-off and stakeholder approval required before release.

6. Risk and Mitigation Log

Risk	Impact	Mitigation
Content scale growth	Growing content library may bloat app size and hinder performance.	Use on-demand resource packs; compress assets; stream videos; implement lazy loading.
Device memory limits	Older iPads/iPhones might struggle with heavy blur effects and 3-D interactions [4].	Limit blur layers; provide low-performance mode that uses solid backgrounds; test on lower-end devices.
Offline needs	Learners may have limited connectivity, affecting content downloads.	Cache lessons locally; implement background download tasks; provide offline progress sync.
Accessibility pitfalls	Glassmorphism may reduce contrast or cause motion discomfort [8].	Provide high-contrast mode; let users disable blur and motion; ensure text always has solid background.
Data privacy	Handling children's data requires compliance with COPPA/FERPA.	Implement parental consent flows; anonymize analytics; encrypt sensitive data; review privacy policies.
Third-party dependency updates	External libraries might break builds or introduce vulnerabilities.	Pin versions; monitor security advisories; integrate Dependabot and run regular vulnerability scans.
Feature creep	Expanding scope may delay releases.	Maintain strict scope boundaries; use feature flags to defer incomplete features; enforce acceptance criteria.

7. Grade Order and Scope Details

Kindergarten

- **Features:** Number tracing, counting games, odd/even sorting, ordinal activities, simple addition/subtraction with pictures, reward animations.
- **Lesson & Item Counts:** 40 lessons; ~400 items.
- **Mastery & Remediation:** Mastery threshold at 80 %. Provide frequent hints; revisit skills after 3 days if accuracy < 70 %.
- **Dataset Needs:** Number illustrations, audio for counting, simple animations.
- **Nonfunctional Requirements:** Memory usage < 100 MB during games; load times < 0.5 s; voiceover friendly; interactive areas sized for small fingers.

Grade 1

- **Features:** Interactive number charts, pattern games, base-10 blocks, addition/subtraction practice, simple fractions, measurement lab, money and time activities, shape builder, bar graphs.
- **Lesson & Item Counts:** 45 lessons; ~450 items.
- **Mastery & Remediation:** Automatic skill review after two incorrect answers; incorporate hints gradually.
- **Dataset Needs:** Base-10 block assets, clock and coin images, measurement units audio.
- **Nonfunctional Requirements:** Memory < 120 MB; support for split-screen multitasking; offline caching of lessons.

Grade 2

- **Features:** Skip-counting games, place value and rounding, addition/subtraction with regrouping, introductory multiplication arrays, fraction visuals, measurement lab, money/time tasks, geometry puzzles, graphing modules.
- **Lesson & Item Counts:** 50 lessons; ~500 items.
- **Mastery & Remediation:** Increase difficulty after three correct answers; schedule spaced practice every week.
- **Dataset Needs:** Multiplication arrays art, fraction bars, measurement conversions.
- **Nonfunctional Requirements:** Memory < 130 MB; ensure multiplication visuals remain responsive.

Grade 3

- **Features:** Multi-digit operations, multiplication/division facts, roman numeral converter, decimal/fraction converter, area & perimeter explorers, calendar/time apps, classification of shapes, data graphs and line plots.
- **Lesson & Item Counts:** 55 lessons; ~550 items.
- **Mastery & Remediation:** Adaptive difficulty using IRT; targeted reviews of misconceptions.
- **Dataset Needs:** Roman numeral assets, decimal grids, shape library.
- **Nonfunctional Requirements:** Memory < 140 MB; maintain 60 fps on geometry tools.

Grade 4

- **Features:** Long multiplication and division tutor, fraction/decimal operations, prime factorization tool, measurement conversions, advanced geometry (angles, symmetry), data graphs, roman numerals, factoring mini-games.
- **Lesson & Item Counts:** 60 lessons; ~600 items.
- **Mastery & Remediation:** Track progression across multi-step operations; provide step-by-step breakdowns for long division.
- **Dataset Needs:** Factor tree diagrams, long division worksheets, fraction/decimal conversion visuals.
- **Nonfunctional Requirements:** Memory < 150 MB; ensure long-division animations do not stutter; provide low-performance fallback.

Grade 5

- **Features:** Multi-digit decimal and fraction operations, order of operations, integers, exponents and roots, factoring, measurement & volume tasks, coordinate geometry, data analysis (mean/median/mode), introductory algebra with variables.
- **Lesson & Item Counts:** 65 lessons; ~650 items.
- **Mastery & Remediation:** Introduce mastery badges for advanced topics; remediation via simpler examples and video explanations.
- **Dataset Needs:** Algebra variable widgets, exponents cards, coordinate grid visuals.
- **Nonfunctional Requirements:** Memory < 160 MB; handle complex graphs and variable interactions smoothly.

Grade 6

- **Features:** Scientific notation, complex fraction/decimal/percent conversions, integer arithmetic including negatives, exponents and powers, proportions and ratios, surface area and volume calculation, geometry of circles and prisms, introductions to linear equations and functions, advanced data handling.
- **Lesson & Item Counts:** 70 lessons; ~700 items.
- **Mastery & Remediation:** Provide interactive step-by-step solutions for complex calculations; schedule review sessions at 1 week and 1 month intervals.
- **Dataset Needs:** Scientific notation resources, integer number line, ratio tables, surface area nets, algebra function plots.
- **Nonfunctional Requirements:** Memory < 170 MB; ensure math operations remain responsive; test across iPad models.

8. Acceptance Criteria

1. **Foundation milestone passes CI:** All modules build without warnings; unit and snapshot tests run green; linting and formatting checks pass.
2. **Design system coverage:** Components and tokens cover ≥ 80 % of UI patterns; high-contrast mode supported; glassmorphic effects used sparingly and meet contrast requirements [1] [3].
3. **Pedagogy review:** Each grade's lessons align with specified standards; content accuracy validated by educators.
4. **Accessibility review:** Screens support VoiceOver, Dynamic Type, Switch Control; text contrast meets WCAG 2.2; motion and blur can be disabled [8].

5. **Performance thresholds:** Launch time < 2 s; memory usage within grade-specific limits; smooth interactions at 60 fps.
6. **Analytics integration:** Events recorded for page views, item attempts, hints, mastery; dashboard displays key metrics (engagement, accuracy, time on task, retry patterns).
7. **Documentation:** Public APIs documented; SPM modules expose minimal surface; code comments explain complex logic.
8. **Governance:** Telemetry dashboards inform release decisions and A/B tests; content authoring guidelines defined; weekly triage meetings scheduled.

Conclusion

This work breakdown plan provides a structured path for delivering a modular, scalable, and accessible math learning app using SwiftUI. By establishing a strong shared foundation, a robust design system with accessible glassmorphism, and clear content specifications per grade, the team can iteratively ship grade-level features while maintaining high standards for pedagogy, accessibility, and performance. The recommended release order and timeline allow early validation of core interactions on younger grades before tackling more complex topics. Continuous quality governance and risk mitigation ensure the app remains maintainable and compliant throughout its lifecycle.

---

1 6 Glassmorphism: Definition and Best Practices - NN/G
https://www.nngroup.com/articles/glassmorphism/

2 3 4 5 7 8 Axess Lab | Glassmorphism Meets Accessibility: Can Glass Be Inclusive?
https://axesslab.com/glassmorphism-meets-accessibility-can-frosted-glass-be-inclusive/