

CodeDrill: Mobile Learning Application for Programming Fundamentals

Professional Project Description

1. Overview

CodeDrill is a mobile learning platform engineered to provide structured, data-driven practice for core computer science concepts. The system addresses a well-defined need across student and professional populations: a lightweight, mobile-optimized environment for drilling fundamental topics with immediate feedback and persistent performance analytics. Traditional learning tools often lack fine-grained topic segmentation, adaptive difficulty, or mobile accessibility; CodeDrill fills this gap through a modular, scalable architecture built on modern cross-platform technologies.

The application is developed using **React Native**, **JavaScript (ES6)**, and the **Expo** toolchain, enabling rapid development, consistent UI rendering across devices, and streamlined iOS deployment. A file-based routing strategy (Expo Router) organizes screens into isolated modules for maintainability. State and history management are implemented through a centralized **Context API provider**, enabling global access to user performance data without prop-drilling and supporting persistent analytics across sessions.

At runtime, CodeDrill loads topic-specific question banks and dynamically filters questions to prevent duplicates, prioritize unanswered items, and deliver continuous, uninterrupted practice. The system includes three difficulty tiers per topic and generates per-question feedback including correctness, selected answer, and reference explanations. The UI is optimized for rapid input, minimal navigation overhead, and accessibility-compliant contrast ratios.

Key results include the successful implementation of a responsive question engine, stable navigation stack, persistent progress tracking, and deterministic question selection logic. Preliminary validation confirms that the app maintains low interface latency, accurate state synchronization through context, and seamless topic switching. The modular architecture supports future integration of cloud-backed storage, spaced-repetition scheduling algorithms, or expansion into additional CS domains.

CodeDrill ultimately provides a technical, extensible framework for interactive programming practice, enabling learners to strengthen competencies in areas such as data structures, recursion, and algorithmic reasoning. Its engineering-first design ensures maintainability, room for expansion, and alignment with professional software development standards.

2. Problem and Requirements

Learners in computer science—ranging from beginners to working professionals—often lack access to a unified, mobile, low-friction platform for structured practice. Existing resources are fragmented across textbooks, desktop-only websites, and inconsistent online question banks. Early learners struggle to reinforce foundational concepts, intermediate learners require targeted interview preparation, and professionals need periodic refreshers to prevent skills decay.

CodeDrill addresses this gap by providing a mobile-optimized drilling system that delivers curated question banks, adaptive difficulty selection, and persistent progress analytics.

Functional Requirements

CodeDrill must support structured drilling across nine core computer science domains: Basics, Strings, Lists, Dictionaries, Stacks, Queues, Linked Lists, Recursion, and Binary Trees. For each domain, the system shall:

- Present curated multiple-choice question banks.
- Support easy, medium, and hard difficulty tiers.
- Allow users to answer questions and receive immediate correctness feedback.
- Persist user history and aggregate topic-level performance.
- Prevent repeat questions until all items in a set are attempted.
- Provide accessible navigation between topics, history, and analytics.
- Maintain session integrity using centralized state management via Context API.

Non-Functional Requirements

- **Performance:** Navigation and question rendering should load within <150 ms.
- **Usability:** Question answering must require ≤ 2 taps.
- **Accessibility:** Maintain readable font sizes and contrast.
- **Reliability:** History data persists across app restarts.
- **Portability:** Runs on iOS using Expo.
- **Maintainability:** Modular file-based routing with reusable UI components.
- **Scalability:** Easily extendable to new topics or question banks without redesign.

Constraints

- Mobile-only application developed with React Native, JavaScript, and Expo.
- Limited development time (single-semester Capstone).

Success Criteria

Technical

- Question engine functions deterministically across all nine topics.
- No navigation or state synchronization errors across screens.
- Persistent history correctly reflects accuracy and completion rates

User-Centered

- Users can complete drills with minimal navigation overhead.
- Performance summaries clearly expose strengths and weaknesses.

Project

- Complete documentation, installation guide, testing artifacts, and poster delivered.
- Application builds and runs successfully on an iOS device via Expo.

Prioritized Backlog

1. Implement question banks for all nine CS topics.
2. Build question engine with immediate correctness feedback.
3. Implement HistoryContext for persistent analytics.
4. Create Topic Selection dashboard.
5. Add Difficulty Selection screen and logic.
6. Implement non-repeating question behavior.
7. Build Progress Report with per-topic stats.
8. Develop history viewer to review past attempts.
9. Implement local storage syncing for user progress and history data.
10. Add session summary screen displaying accuracy, streaks, and difficulty breakdown.

Full Backlog: <https://github.com/HuedCode88/Mobile-Coding-practice>

3. Technical Architecture

The application uses the **Expo Router architecture**, with a centralized navigation layout and persistent history storage via a custom React context. The application follows a clean component-based structure:

Key Technologies & Tools

- **React Native** — Core UI framework
- **Expo** — Development environment and build tools
- **Expo Router** — File-based routing and navigation
- **JavaScript (ES6+)** — Application logic
- **Context API** — User progress and history management
- **Local asset management** — Application logo and static resources

4. Core Source Files and Responsibilities

4.1 Home.jsx (Homescreen)

The Home screen serves as the application entry point. It displays the CodeDrill logo and two primary navigation actions:

- **Begin Learning** → navigates to the Topics page
- **Progress Report** → navigates to the user's performance analytics

Styling emphasizes readability, large interactive buttons, and a dark-themed interface consistent across the application.

4.2 _layout.jsx (Root Navigation Layout)

This file configures the global application structure using **Expo Router's Stack navigation**. It:

- Provides a consistent header across screens
- Registers routes for the Home page, Topics list, Topic drill screen, About, Contact, and Progress Report
- Automatically formats topic names from slug form (e.g., linked-lists → Linked Lists)

It also wraps the application in a **HistoryProvider**, enabling persistent storage of user performance.

4.3 Topics/index.jsx (Topic Selection Screen)

This screen presents the list of available drill topics, each linking to its respective topic screen. Features include:

- Scrollable layout
- Button-based topic selection
- Automatic slug generation for URL routing
- Consistent color palette and accessible text contrast

This screen marks the entry point into the learning flow.

4.4 Topics/[topic].jsx (Dynamic Topic Drilling Screen)

This is one of the most functionally dense components. It:

- Parses URL parameters to determine selected topic and difficulty
- Retrieves relevant questions from a **central question bank**
- Ensures users only receive *unanswered* or *incorrect* questions, preventing repetition
- Tracks user selections, correctness, and question history via context
- Provides real-time feedback:
 - **Correct** answers highlighted in green
 - **Incorrect** answers highlighted in red with correct answer details
- Offers difficulty selection and a “Next Question” progression mechanism
- Includes direct navigation to:
 - Topic list
 - Full question history

This file controls the complete user learning interaction cycle.

4.5 ProgressReport/index.jsx (Progress Analytics Dashboard)

This screen serves as the application’s progress-tracking module. It:

- Aggregates user history to compute:
 - Total questions answered per topic
 - Correct answers per difficulty (Easy, Medium, Hard)
- Uses collapsible sections to display:
 - Correctly answered questions
 - Incorrect attempts

- User-selected responses vs. correct answers

This functionality assists learners in identifying areas requiring additional practice.

4.6 HistoryContext.jsx (State Management Layer)

Although not included in the snippet above, its referenced usage indicates it is your persistent storage mechanism. It enables:

- Global availability of question history
 - Dynamic UI updates based on user progress
 - Data persistence during navigation
-

5. Implementation Notes

Repository Structure:

- /assets: Images and media resources (e.g., CodeDrill logo).
- /components: Reusable UI components (buttons, cards, modals).
- /contexts: Global state management (HistoryContext.js).
- /questions: Curated question banks, categorized by topic and difficulty.
- /screens: All application screens (Home, Topics, Topic, ProgressReport).
- App.js / _Layout.jsx: Root navigation and provider wrapping.
- package.json: Dependencies including React Native, Expo, and navigation libraries.

Coding Conventions & Patterns:

- Functional components with React Hooks (useState, useEffect).
- Context API for persistent analytics (HistoryContext).
- Modular separation: UI, data, and logic are isolated per component.
- Consistent styling using StyleSheet.create(); Flexbox layout ensures responsiveness.
- Tradeoffs: Limited local storage persistence (AsyncStorage) over full database integration for MVP delivery.

Reference: Full README includes setup instructions and module layout details.

6. Testing & Evaluation

Testing Strategy:

- **Unit Testing:** Question engine logic, scoring calculations, and feedback display.
- **Integration Testing:** Navigation between screens, state updates in HistoryContext, and difficulty selection logic.
- **End-to-End Testing:** Simulated user flows (topic selection → question answering → progress report).

Coverage & Metrics:

- Track number of questions answered per topic and difficulty.
- Percent correct per session, streaks, and total progress.
- Identify repeatable errors or incorrect scoring.

Defect Summary:

- Early prototype revealed occasional duplicate question selection; resolved by non-repeating question logic.
- Minor layout inconsistencies on smaller iOS screens; addressed with responsive styling.

7. Security, Privacy, Accessibility & Compliance (O5)

Security & Privacy:

- No personally identifiable information (PII) is collected.
- User progress stored locally (AsyncStorage) to minimize exposure.

Accessibility:

- Color contrast sufficient for readability.
- Touchable areas comply with minimum size recommendations.

Ethical & Societal Considerations:

- Inclusive design ensures equal access to learning materials.
- App encourages structured learning without promoting harmful shortcuts.

8. Deployment & Operations

Platform: The application is deployed using **Expo**, a framework and platform for React Native that simplifies building, testing, and distributing mobile apps. The primary target is **iOS**, with potential for cross-platform support.

Running the App:

- During development, the app runs locally via **expo start**, which launches the **Metro bundler** and provides a QR code.
- Users can scan the QR code with the **Expo Go app** on their iOS device to run the application instantly without manual installation or App Store submission.
- Expo handles bundling, asset management, and live reloading during development.

Continuous Integration & Deployment (CI/CD):

- The project integrates linting and automated builds using Expo's build service (**eas build**).
- Over-the-Air (OTA) updates are enabled via Expo's **EAS Update**, allowing new versions of JavaScript code and assets to be delivered directly to users without a full App Store release.

Installation:

- Users install and launch the app by scanning a **QR code** or using a shared **Expo link**.
- No backend server is required for operation; all question banks, progress tracking, and analytics are managed locally via AsyncStorage.

Documentation:

- Detailed **Installation Guide** (Appendix A) and **Admin/Operations Runbook** (Appendix C) provide step-by-step instructions for setup, development, and maintenance.

9. Limitations & Future Work

Current Limitations:

- No cloud sync; progress is device-specific.
- Limited analytics visualization (basic counters; no graphs).
- MVP interface optimized for iOS; Android adjustments pending.

Technical Debt & Shortcuts:

- Hardcoded question banks; future versions may integrate dynamic API-based updates.
- Simple UI design; future iteration could include animations and gamification.

Future Roadmap:

1. Implement cloud storage and cross-device syncing.
2. Expand question banks with additional algorithms and data structures.
3. Add more detailed analytics (charts, difficulty trends, recommendations).
4. Implement Android support and accessibility enhancements.

10. References

Install Expo Router following the steps in the following website: <https://docs.expo.dev/router/installation/>
This will allow you to test and traverse the application on your IOS device.

Poster:  CodeDrill Poster 36x48.pdf

Slides:  CodeDrill Presentation Final.pdf

Videos: Demo: [Demo](#)