CASE STUDY · PROJECTTHUNDER

Fun With Learning

Rebuilding a children's math app as one .NET MAUI codebase for iPhone, iPad, Android phones and tablets, Windows and Apple-silicon Macs, with a web admin that keeps the content fresh without a new release.

.NET 10 MAUI | iOS & iPadOS | Android | Windows | macOS (Apple silicon) | ASP.NET admin & API







Children's education · Ages 5 to 12

projectthunder.com

AT A GLANCE

A kids' math app that had stopped moving

Fun With Learning teaches elementary math through visual models, games and short live-action videos hosted by real people. The original app was built on Xamarin.Forms. When Microsoft ended Xamarin support, the published app could no longer be updated, and an early port to .NET MAUI compiled but left key screens blank.

1
shared codebase for every platform

5
platforms: iPhone, iPad, Android, Windows, Mac

28
mini games in five math strands

20+
visual math models with guided walkthroughs

5
math worlds, each with three difficulty tiers

30
TestFlight builds from first MAUI build to today

Product	Fun With Learning, ProjectThunder's own children's education app (ages 5 to 12), published on the App Store by ProjectThunder
Our work	Rescue and rebuild of the mobile app, new curriculum features, multi-platform release, hardening and store compliance
Platforms	iOS 15+ and iPadOS, including iPad multitasking and foldable iPhones · Android 8.0+ phones and tablets · Windows 10 and 11 desktop · Apple-silicon Macs (the iPad build, tested through TestFlight for Mac)
Stack	.NET 10, .NET MAUI 10, C#, XAML, MVVM (CommunityToolkit.Mvvm), Autofac, SkiaSharp, Lottie, CommunityToolkit MediaElement, Sentry · ASP.NET API and admin on SQL Server · Azure DevOps Server for backlog, repos and CI
Content	Live-action Video Quests, Story Time read-alouds, celebration videos and a home-screen story feed, all managed from a web admin
Status	Live on the App Store; Android, Windows and Mac builds in distribution and testing

> **"Math without fear": no timers, no leaderboards, no ads, no chat, and a wrong answer only ever gets a kind retry.**

THE CHALLENGE

"It compiles" is not the same as "it works"

Fun With Learning is our own product, and in 2026 it had three problems at once: a store app that could not be updated, a half-finished port, and the security and privacy work the app stores now expect.

A frozen store app

The published app ran on Xamarin.Forms 5. With Xamarin out of support, current Xcode and Android toolchains could no longer produce store builds from it, so even small fixes could not reach families.

A port that only looked finished

An earlier migration to .NET MAUI had reached a clean compile by commenting out the hardest screens. Those were the teaching walkthroughs: area-model multiplication and division, step-by-step long division, and number-bridge subtraction. In the app they rendered as blank pages. Release builds failed outright on stale Xamarin references in the XAML.

Android did not start

A core service was registered for iOS only, so on Android every screen failed to construct. The offline video cache also failed quietly on Android, which left the video library empty.

Security and privacy, to today's standard

Expectations for children's apps have risen sharply. The rebuild was the moment to bring storage, transport security, configuration, the grown-up gate and crash reporting up to what the stores, and parents, expect now.

Store expectations had moved

Apple now requires in-app account deletion, reviews embedded mini-games under Guideline 4.7, and asks for a privacy manifest. Google's Families program adds its own rules for kids' apps. The rebuild had to satisfy all of these, not only run.

The brief

Finish the migration, make every screen real, ship on every device kids use, and make the app safer and easier to update.

OUR APPROACH

Audit first, then themed sprints

We started with a code-level audit of the real app rather than its documentation. That audit rewrote the plan: the sprints had been scoped against an abandoned branch, so we rebuilt the backlog against the code that actually shipped.

1. **Audit and re-plan.** We catalogued every screen, service and dependency, logged the security findings as backlog items, and mapped the app's teaching methods against the full US whole-number curriculum. That gave a roughly 2,000-hour roadmap of about 200-hour sprints in Azure DevOps.
2. **Get it running everywhere.** We fixed the Android startup failure, cleared the stale XAML so Release builds pass on both mobile platforms, set the correct app identity and signing, and shipped the first MAUI build to TestFlight.
3. **Lights On.** We restored the four blank teaching screens and rebuilt their animation in native MAUI, replacing the Lottie behaviour that hadn't survived the port.
4. **Lockdown.** A dedicated security and privacy sprint: secure storage, strict transport security, build-time configuration, a stronger grown-up gate and crash reporting.
5. **Curriculum sprints.** Each sprint added a strand of visual math, named for the families of models it introduced:

   Gold Hops · number lines
   Ten-Frame Meadow · ten frames, doubles, subitizing
   Bonds and Buddies · number bonds, fact families
   Block Party · base-ten, partial sums, expanded form
   Bar None · bar models, comparison
   Array Avenue · arrays, equal groups, halving
   Counting Cove · rekenrek, hundred chart, compensation
   Classic Corner · carrying, borrowing, multi-digit multiplication

6. **Tester rounds.** A small QA group of parents, teachers and our own team tested every TestFlight build. Each round of notes became backlog items, was reproduced on simulators and devices, fixed, and shipped in the next build.

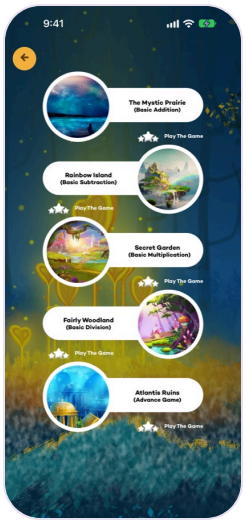
How we verify

A change counts as done only after it has been seen working on a simulator or device, with screenshots of the destination screen, not merely after it builds. Repeatable UI-automation scripts (Appium and XCUITest) drive the real app through math worlds and mini games. Larger changes also get a separate adversarial code review before they ship.

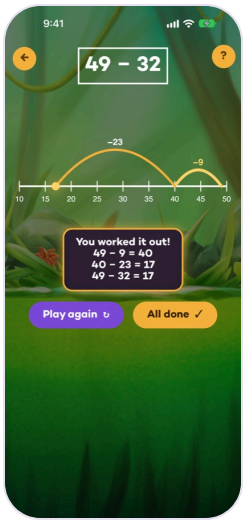
THE SOLUTION · LEARNING

Five worlds, and a model behind every answer

Practice lives in five painted worlds: Mystic Prairie (addition), Rainbow Island (subtraction), Secret Garden (multiplication), Fairy Woodland (division) and Atlantis Ruins (mixed). Each has three tiers, Rookie, Hotshot and Rock Star, and a session is 15 questions.



The five worlds



A hint solved as hops on a number line

Hints that teach the method

Tapping Hint doesn't reveal the answer. It opens a guided walkthrough of the strategy behind it:

- **Area model** for multiplication, with partial products lighting up
- **Partial quotients** for division, built proportionally from the real numbers
- **Long division** step by step, with a multiply-back check
- **Number-line hops** for making ten, counting back and skip counting
- **Ten frames and dice** for early addition

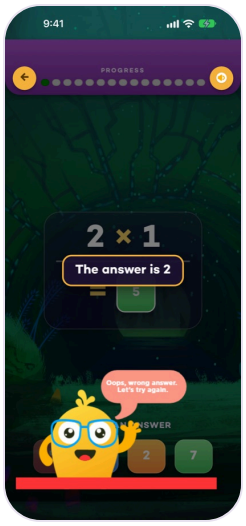
A **division spotlight** introduces each new problem in plain English ("Share 594 into 33 equal groups") while the music dips, then fades into play.

Gentle by design

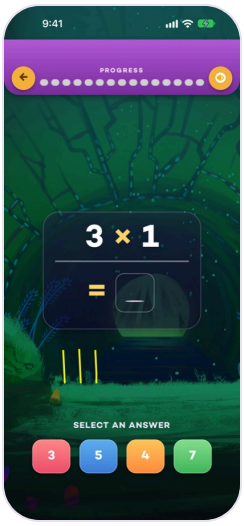
A wrong answer shows the right one, kindly, and the mascot Zimbo says "Oops, let's try again." There are no countdowns, no red buzzers and no comparing children with each other. Progress bars use a neutral slate for misses, not red.

Work it out by hand

Every question has a scratch layer, so children can draw on the problem with a finger or stylus. On iOS we turn off the edge-swipe back gesture on these screens, so a stroke near the left edge doesn't close the page mid-sum.



A miss gets a kind retry

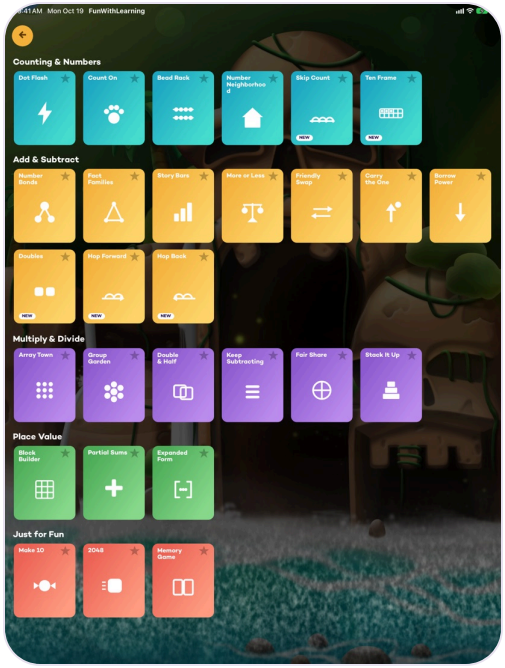


The scratch layer

THE SOLUTION · PLAY

28 mini games, organised by strand

The mini-game hub groups games into five strands, each with its own colour, and keeps a favourites strip and a recently-played row. Every game has a first-visit "how to play" card, a gentle retry hint, round-by-round progress and a Zimbo-coin reward.



On iPad, all 28 games fit on one screen

Strand	Games
Counting & Numbers	Dot Flash, Count On, Bead Rack, Number Neighborhood, Skip Count, Ten Frame
Add & Subtract	Number Bonds, Fact Families, Story Bars, More or Less, Friendly Swap, Carry the One, Borrow Power, Doubles, Hop Forward, Hop Back
Multiply & Divide	Array Town, Group Garden, Double & Half, Keep Subtracting, Fair Share, Stack It Up
Place Value	Block Builder, Partial Sums, Expanded Form
Just for Fun	Make 10, 2048, Memory Game

Custom-drawn manipulatives

The models children touch are custom SkiaSharp controls we built for the app: a number line with animated hops, a hundred chart, base-ten blocks that regroup, counter frames, and dice with standard subitizing patterns. One implementation of each renders the same on iOS, Android and Windows.

Rebuilt native for store review

Two of the original games, 2048 and Memory, were web pages loaded inside the app. Apple's Guideline 4.7 asks for an index of any such non-embedded games, and one page pulled a third-party script from a CDN. We rewrote both as native games: no web views remain in the app, both work offline, and both now award coins.

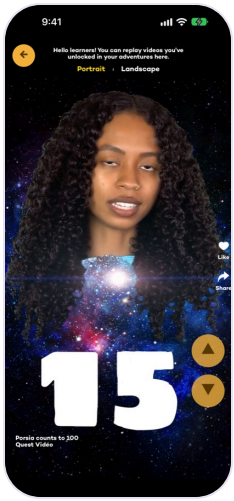
Logic tested before UI

Game rules live in plain C# classes with no UI types, so we test them in a console harness before building any screens. That covers edge cases such as 2048's "2-2-2-2 merges to 4-4, never 8" and a third tap during Memory's mismatch flip-back. Make 10 checks after every refill that a sum of ten is still possible, so the board can never strand a child.

THE SOLUTION · REAL PEOPLE, REAL STORIES

Live-action video, a story to follow, and things to earn

What sets the brand apart is real people on camera instead of animation or synthesized narration. The app is built around that.



Video Quests & Story Time

Hosts lead counting quests, science demonstrations and mindfulness breaks, and read picture books aloud. The video player is full-bleed, with Fill and Fit modes and buffering feedback. Celebration clips play after a finished session, and repeats are avoided.

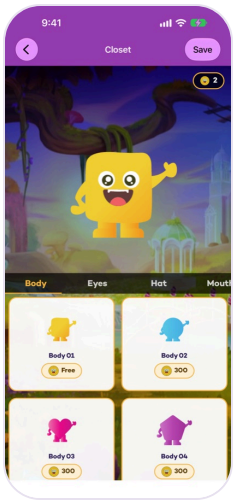


Story mode

The villain has hoarded the world's numbers and drained five regions grey. Children free each region by finishing its math, and the villain is out-counted, never fought. The story is 13 short video beats, each skippable. Beats are tracked per child, and a switch turns story mode off.

Zimbo coins: a closed loop with no real money

Children earn coins in math worlds, in every mini game and from a once-a-day spin. They spend them in the **Closet** to dress their avatar. Coins can't be bought or cashed out: we removed in-app purchases from the app entirely.



The Closet

The Zimbo Market

A pretend price for the coin rises and falls on a small chart, and each day it poses one real word problem built from the chart's own numbers: change, comparison, average or percentage. It teaches graph reading with no money involved. The price is computed on the device from the date and a per-child seed, so it works offline and needs no server.

One device, many children

Each child has a profile with their own coins, stars per world, levels completed, story progress and sound, haptics and story settings. Profiles are created with a name and an avatar; we removed photo capture so no child images are collected.

THE SOLUTION · CONTENT ADMIN

A web admin that keeps the app fresh

Much of the app's content is served rather than compiled in. A password-protected web admin, running on the same ASP.NET server as the app's API, lets the content team change what children see without an app release or store review.

Video library

Add, edit, retire and re-activate videos. Each entry has a title, a streaming link, a type (quest, story, celebration), an orientation and an uploaded thumbnail. The app picks up changes on next launch and caches the catalogue for offline use.

Celebration videos

Mark clips as celebrations and the app plays one after each finished session, rotating through the set and avoiding immediate repeats. Adding a clip in the admin widens the rotation for every child.

Home-screen stories

Publish short story sets to the rail of circles on the home screen. Each set has a title, a thumbnail and a sequence of image or video items with captions, durations and an expiry date. The app shows newest first.

Closet catalogue

Manage the avatar items children can earn, including names, types, layered artwork parts and image uploads. New outfits reach the app without a build.

Notifications

Create and schedule announcement messages through a push-notification service integration.

Admin users and audit

Separate admin accounts with sign-in, plus an audit log of changes and server-side error logging.

The app-facing API

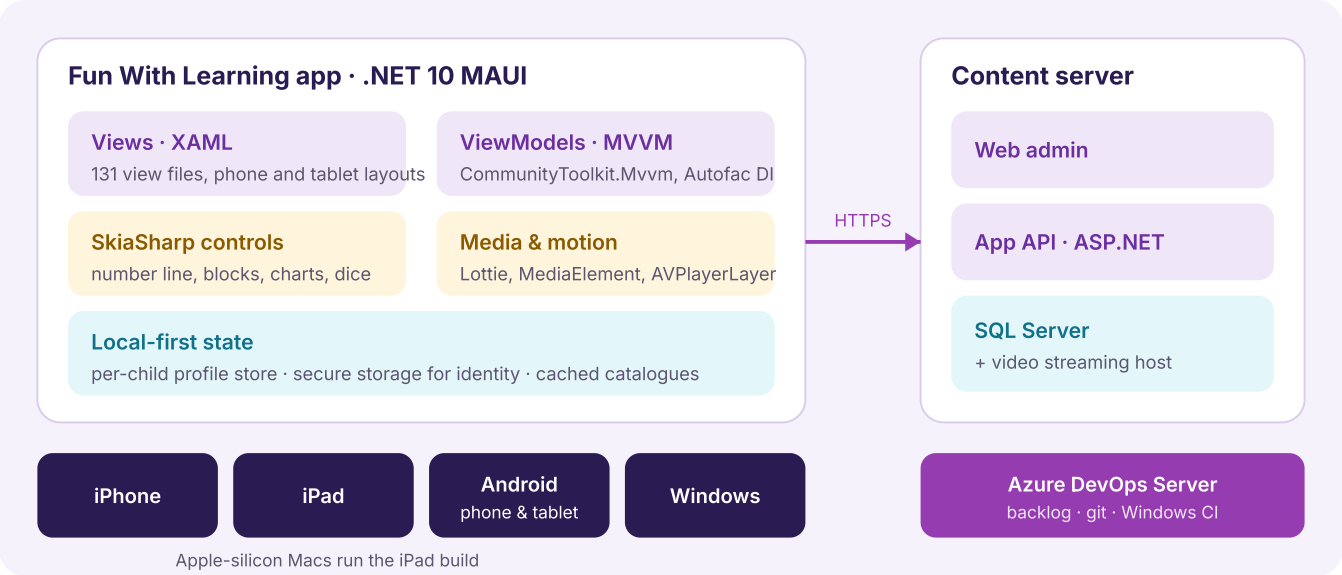
The same server exposes the API the app calls: video and story catalogues, profiles, coin balances, avatar ownership and account deletion. As part of this engagement we:

- Added a real **account-deletion** endpoint, which Apple requires and which the in-app delete flow relies on
- Made coin saves **update** the existing balance instead of adding a new row on every save
- Ported the app-facing API to **.NET 10 / ASP.NET Core** alongside the original server, matching the existing database and JSON shapes so the app can switch over without changes

ARCHITECTURE

One codebase, four operating systems

The app is a single .NET MAUI project. About 34,000 lines of C# and 17,000 lines of XAML compile to native apps for iOS, Android and Windows, and the iPad build runs on Apple-silicon Macs.



Local-first, sync when possible

Each child's coins, stars and settings are written on the device first and synced to the server in the background, one request at a time. A slow or failed server reply can never overwrite a newer local total, and the screen never waits on the network after a correct answer. If the server is unreachable the app keeps working, and requests give up after 15 seconds instead of hanging.

Shared building blocks

Common behaviours are written once and attached anywhere: answer feedback (a spring on correct, a shake on wrong, haptics, sound), press feedback on tiles, the how-to-play overlay, coin toasts and a serialised navigation queue.

Platform-specific where it matters

- **iOS:** a custom video control drawn on a bare AVPlayerLayer for true edge-to-edge playback; iOS 27 scene lifecycle; foldable-iPhone and iPad multitasking support
- **Android:** MediaElement video, target SDK 36, signed builds for the store and for direct download
- **Windows:** an unpackaged, self-contained WinUI app with a fixed portrait window so phone and tablet layouts don't stretch

Observability

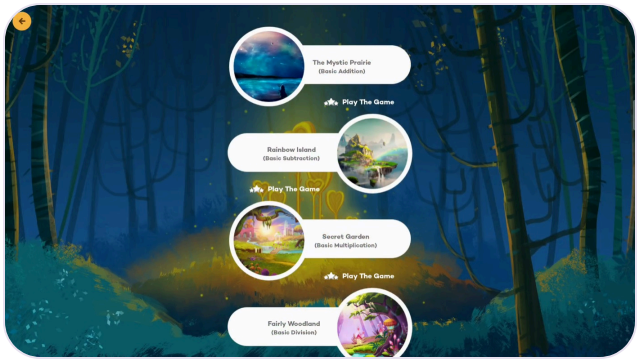
Sentry for crash reporting, configured for children (no user identity, no screenshots), plus managed-exception logging.

DELIVERY

Shipping on every device

Each platform has its own build pipeline, signing and distribution, all produced from the same source tree.

Platform	How it ships	Notes
iPhone & iPad	App Store, with TestFlight for every build before release	iOS 15 and later. Built with the iOS 27 SDK, including the new scene lifecycle, iPad multitasking and foldable-iPhone layouts. Privacy manifest declared.
Android phones & tablets	Google Play app bundle, plus a signed APK for direct download from funwithlearning.net	Android 8.0 and later, target SDK 36. Tested on phone and tablet emulators and a real Samsung tablet. The direct-download build updates itself; the store build leaves updates to the store.
Windows	Self-contained desktop app built by an Azure DevOps pipeline on a Windows build agent	Windows 10 (19041) and later, x64. Distributed as a downloadable package outside the Microsoft Store.
Mac	The iPad build on Apple-silicon Macs, distributed to testers through TestFlight for Mac	No separate codebase or Mac-specific build to maintain.



Math worlds on an Android tablet, landscape

Release discipline

- One version number in the project file drives every platform
- Release builds always start from a clean build folder. Incremental builds produced subtly broken binaries on both iOS and Android, so we don't ship them.
- Every TestFlight build carries written "What to Test" notes for the QA group
- On Android, builds go to a testing track first, so Google's pre-launch device report runs before anything reaches production

ENGINEERING NOTES

Problems worth writing down

Cross-platform work is mostly smooth, until it isn't. A few of the issues we solved, and what they taught us.

The answer buttons that ignored taps

After a visual redesign, every answer button in the math worlds stopped responding. The new bottom panel was marked input-transparent, and in MAUI that setting passes down to every child inside it. Removing it from the container, and keeping it only on the purely decorative layer, brought the buttons back. We caught it on a device before the build shipped.

Back buttons and impatient fingers

Children tap Back three times in a row. A rapid second tap could arrive mid-transition, iOS would drop it, and Back would stop working. We replaced the guard flags with a small navigation queue: one pop at a time, a short settle between pops, and extra taps folded into the queue instead of discarded.

A crash on the first system dialog

The first MAUI TestFlight build crashed whenever iOS showed any system prompt. A lifecycle override skipped the base call, so MAUI never recorded the window as active and threw on the first deactivation. We reproduced it on a simulator by forcing the app into the background, then fixed it.

Android Release vs. the trimmer

Debug builds were fine, but Release builds froze after a popup closed: the page drew but ignored every touch. The .NET linker had trimmed code a popup library reaches by reflection. Android Release now ships untrimmed. The package is larger, and every screen works.

Every child shared one coin balance

Two settings classes used the same setting name, but one stored it securely and deleted the plain copy. The others then read an empty ID, so every profile on a device shared one balance and one set of progress. A single resolver for the active profile, plus a one-time migration, fixed it. We confirmed on a device that child B's coins never land on child A.

"Brush.Default" is not null

Answer-colour feedback silently stopped after a redesign because code checked whether a background was null. In MAUI the default background is an empty brush object, not null. We switched to MAUI's own emptiness check, and the green and rose feedback came back on every button.

HOW WE BUILT IT

People and AI agents, one team

We built Fun With Learning with AI coding agents (Claude Code) working alongside our developers. People set the direction and approved every release; agents wrote much of the code, ran the simulators and checked each other's work.



Who wrote today's code

Of the roughly 51,000 lines of C# and XAML in the current app, **about 59% were last written by AI agents** and 41% by human developers. Much of the human share is the original app, which our developers carried over from Xamarin and then redesigned. Measured from line-by-line git history (each line is credited to whoever last changed it). Agents added 44,000 lines across 180 commits, and no release shipped without a person signing it off.

72%
of commits since the rebuild began came from agent sessions

26
days from the first MAUI TestFlight build to live on the App Store

68
backlog items and bugs closed, each with a written record

What people did

- Product direction, curriculum choices and the "math without fear" rules
- Visual design, including the gradient answer-button redesign
- Testing on real phones and tablets with parents and teachers, written up as notes
- Every legal agreement, store submission and release decision

What the agents did

- Most of the 20+ visual models, new mini games and story mode
- Turning tester notes into backlog items, then reproducing and fixing them
- Builds, signing and uploads for iOS, Android and Windows
- Release notes and "What to Test" notes for every build

How AI does QA here

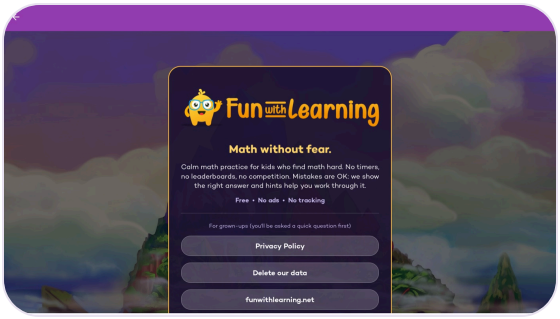
- **Agents drive the real app** on iPhone, iPad and Android simulators, and screenshot the destination screen before calling anything fixed.
- **Scripted play-throughs** read each equation on screen, work out the answer and tap it, through every math world and tier.
- **Agents review agents.** Reviewer agents try to prove each larger change wrong; one pass ran 59 in parallel. Only findings that survive verification get fixed.
- **A second AI model audits.** A different vendor's model audited the code independently and found two real data bugs, both fixed and verified on a device.
- **Rules tested before screens** in console harnesses: 14 of 14 cases for 2048, 19 of 19 for Memory.

CHILD SAFETY & PRIVACY

Built for kids, reviewed like it

A children's app is judged by what it leaves out as much as by what it does. These are the commitments the codebase enforces.

- **No ads, no in-app purchases, no tracking.** Purchases were removed from the app and from sale. No advertising ID is used.
- **No chat and no AI chatbot.** The avatar buddy uses canned lines only.
- **A real grown-up gate.** Sharing and profile deletion sit behind a birth-year gate that only an adult passes (18 or older).
- **In-app account deletion.** Deleting a profile removes it on the server and wipes the data stored on the device, as Apple requires.
- **No child photos.** Profile photos were replaced with a dress-up avatar, and the camera and photo-library permissions were removed.
- **Identity in secure storage.** Names and device identifiers live in the platform keychain or keystore. Android backup of app data is off.
- **Encrypted transport.** Strict HTTPS on both platforms, with one narrowly scoped exception documented for a legacy image host.
- **Privacy-safe diagnostics.** Crash reports carry no user identity or screenshots.



The About screen says it plainly

Store compliance

App Store privacy manifest with required-reason APIs declared · export compliance declared · Guideline 4.7 resolved by making every game native · Google Play Families declarations, data-safety form and content rating completed.

RESULTS

From frozen to shipping

The app is back in active development, released through the App Store and tested continuously, with one codebase serving every device a family is likely to own.

Unfrozen

Moved from end-of-life Xamarin to .NET 10 MAUI and the iOS 27 SDK, so updates can reach families again.

Finished

Every blank teaching screen restored, and the curriculum grew to 20+ visual models and 28 mini games.

Everywhere

iPhone, iPad, Android phones and tablets, Windows desktop and Apple-silicon Macs, all from one source tree.

Safer

Secure storage, strict HTTPS, a real adult gate, account deletion, and no ads or purchases.

Easier to update

Videos, stories, closet items and announcements change from the web admin, with no app release.

Observable

Crash reporting, written test notes on every build, and a verified-on-device definition of done.

What we'd tell the next team

- **Audit the code that ships, not the plan.** Our first plan was written against the wrong branch, and an hour of reading the real code saved weeks.
- **A clean compile is not done.** Verify on the destination screen, on a device, with a screenshot.
- **Treat Release builds as their own platform.** Trimming, ahead-of-time compilation and stale build folders cause bugs that Debug builds never show.
- **Keep kids' data local and minimal.** It is simpler to build, easier to review and kinder to families.



Planning a cross-platform app?

ProjectThunder designs, builds and rescues mobile, desktop and web software in .NET, from first prototype to store release. Learn more at **projectthunder.com**. See the app at **funwithlearning.net**.

Screenshots are from the current app on iPhone, iPad and an Android tablet. Product names are trademarks of their respective owners. Fun With Learning and Zimbo are ProjectThunder products.