

GOOGLE SUMMER OF CODE 2026

Create a NuttX Distribution Layer with Dynamic ELF Application and Library Support

Final Work Product Report

- **Contributor:** Aviral Garg
- **Email:** gargaviral99@gmail.com
- **GitHub:** github.com/aviralgarg05
- **Organization:** Apache Software Foundation
- **Mentor:** Alan Carvalho de Assis, Halysson Carvalho
- **Project:** Create a NuttX Distribution with Dynamic Binary (ELF) Loading (~350 hours)

1. Overview

This project began with a fairly simple question: could I add an application to a NuttX device after flashing the firmware? NuttX already knew how to load an ELF program. What it didn't have was the rest of the experience around that loader: a package description, checks before installation, persistent state, safe updates, and a device-side way to launch what had been installed.

Trying to build all of that at once would have made the project impossible to review. Alan Carvalho de Assis and Halysson Carvalho encouraged me to take the smallest useful step first, so I began with one `hello` package on one board. From there, the work grew in layers. The package became reproducible, the first installer grew into the `nxpkg` lifecycle, and NXStore came only after the lower layers were stable enough to use.

The clearest upstream result is the merged `nxpkg` core. It handles a configurable package root, local and network catalogs, SHA-256 checks, versioned storage, installation, updates, rollback, removal, listing, package locks, and recovery-aware state changes. The ELF and shared-library test packages are generated from the active NuttX build as well, so they don't quietly go stale in the repository.

NXStore is the part a person can actually touch. It is an LVGL application that reads `nxpkg` state through the public API and starts installed programs under a supervisor. I ran that path on an ESP32-S3 touchscreen board with small applications and NXDoom. By the August 21 snapshot, NXStore, Brickmatch support, the `Library.mk` interface, and the shared-library documentation had merged. NXDoom, the calculator, supervised games, metadata and icon export, and the main `nxpkg` and NXStore guide remained under review. **Throughout this report, I only use merged for code that Apache NuttX had accepted.**

The GSoC work is spread across 18 pull requests. Thirteen had merged by August 21 and five were still open. Their GitHub snapshots contained 10,783 additions, 467 deletions, and 107

changed-file entries. Those totals are useful for understanding the review surface. They shouldn't be read as a count of lines written from scratch.

There is still work left: package signing, a maintained public demonstration catalog, FDPIC support, and the optional Dropbear extension. Even with those gaps, **the project now has a merged package-management base and a product path that has been exercised on real hardware. The remaining product-layer work is public and moving through review.**

Final status at a glance

Area	Final state	Evidence
Local package lifecycle	Merged	`nuttx-apps` PR #3474 plus the crash-safety and lifecycle series
Package metadata and validation	Merged	PRs #3519 , #3531 , and #3718
Network catalog and artifact fetching	Merged	PR #3719 ; fresh hardware network re-test was limited by the active board configuration
Install, update, rollback, remove, and recovery	Merged	PRs #3642 and #3720 ; SD-card scenarios exercised on hardware
NXStore LVGL frontend	Merged and hardware-tested	PR #3643
Touchscreen diagnostic	Merged	PR #3659
Loadable touchscreen applications	Partly merged, hardware-tested	Brickmatch PR #3662 merged; PRs #3660 , #3661 , and #3663 remain open
NXDoom module and RGB565 path	Implemented and demonstrated, open	PR #3644 ; simulator and hardware evidence
Declarative shared-library packaging	Merged and hardware-tested	PR #3730 and documentation PR #19888
User and developer documentation	Partly merged	PR #19888 merged; PR #18875 remains open
Signed packages / trust chain	Not implemented	SHA-256 protects integrity but does not authenticate a publisher
Hosted demo distribution / Dropbear	Not completed	Kept outside the final core scope

2. How the project found its shape

Issue [#17351](#) described a broad project: dynamic ELF applications, downloads and updates, a package manager, an LVGL interface, shared libraries, and eventually a demonstration distribution. Dropbear was listed as a useful extra. Each part made sense on its own, but together they touched the loader, build system, filesystems, networking, UI, and board support. There wasn't one obvious place to begin.

The decision that made the project manageable was to prove the smallest complete path first. The plan that came out of my mentor discussions was practical: get one executable package working, keep each upstream change reviewable, and collect evidence on the target before building another layer. That is how I ended up working with a local `hello` package on the Seeed Studio XIAO ESP32S3 Sense rather than starting with a polished app store.

By midterm, that narrow path worked. The package had metadata, board and architecture checks, a SHA-256 digest, staged installation, versioned storage, and a dynamic `hello` program running on the XIAO. It also gave me a much better list of unknowns. Module metadata still needed work. Shared-library packaging, networking, recovery after interrupted operations, and the touchscreen experience were all ahead of me.

I kept that same dependency order after midterm. Repeatable test artifacts came first. Persistent state and failure handling followed. Network and lifecycle commands came next, and NXStore sat on top of all of it. Looking back, that order is what held the project together.

The journey at a glance

Period	What I focused on	What changed by the end
Community bonding and early coding	Read the ELF and shared-library examples; define a minimal package and repository model	I had a constrained plan centered on one ELF package rather than a complete distribution image
June 2026	Implement the basic `nxpkg` path and missing module metadata; validate on XIAO ESP32S3 Sense	Local install/list worked and the first package-manager work reached upstream
July 2026	Generate ELF and shared-library fixtures from the active build; add positive and negative tests	Package artifacts became reproducible instead of hand-maintained
Late July to early August	Build NXStore, touchscreen diagnostics, application modules, metadata/icon export, and NXDoom changes	The package work became a device-facing application workflow
August 2026	Rework storage, parsing, network fetch, update/rollback/remove, locks, and recovery	The four-part `nxpkg` hardening series merged on August 17

Period	What I focused on	What changed by the end
Final week	Address review comments, document the stack, validate hardware, and implement declarative library support	The merged core was ready; the product and library layers were public and reviewable

3. Starting small: one package on one board

My first milestone was deliberately plain: install one valid ELF package and cleanly reject an invalid one. PR #3474 added ``system/nxpkg`` and the first lifecycle helper. Its catalog carried the package version, compatible architecture and board, artifact location, and SHA-256 digest. During installation, the payload went into staging, passed its checks, and moved into a versioned directory before the database treated it as installed.

For the XIAO configuration, I packaged the ``hello`` ELF produced by the current build. I installed it with ``nxpkg``, checked that it appeared in the list, and launched it with the NuttX loader. That tiny test exposed the contract more clearly than a large demo would have: the target had to match, the digest had to match, and a failed install couldn't leave behind a convincing but unusable package record.

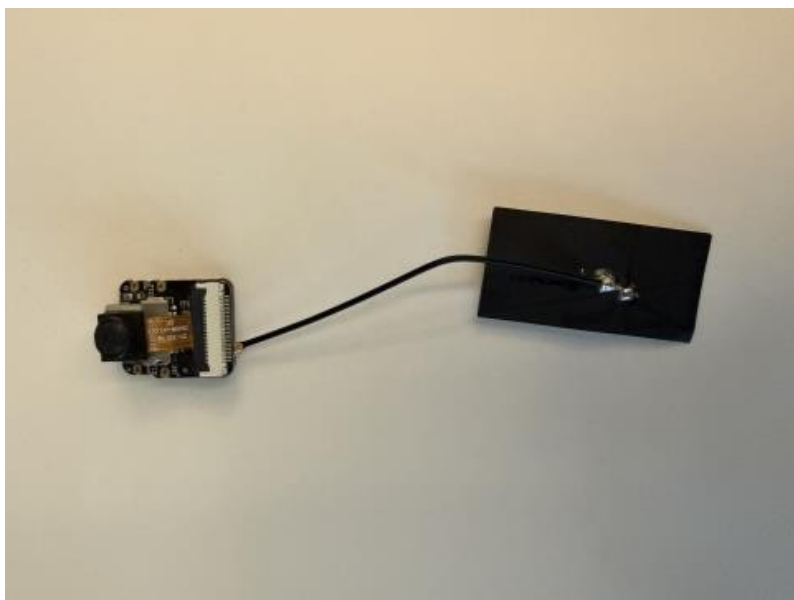


Figure 1. The XIAO ESP32S3 Sense used for the first working ELF package path.

Versioned storage was present from the first implementation because updates and rollback were already on the roadmap. Verification also happened before commit. **Those two early choices saved me from replacing the storage model once the lifecycle became more complicated.**

4. Turning a one-off demo into a repeatable test

A binary that works once on my machine isn't a useful upstream test. An ELF copied into a fixture can drift away from the ABI, toolchain, or configuration that produced it, and the

breakage may stay hidden for months. I moved the examples toward artifacts generated by the build that is actually being tested.

[PR #3519](#) supplied missing module metadata for executable tools. [PR #3531](#) used the ELF from the active configuration to generate a valid index, an intentionally invalid index, and the NSH scripts that exercise both. On the board, the valid package installed. A missing artifact returned `ENOENT`, the package list remained empty, and the memory delta returned to zero when the script finished.

The shared-library fixture followed the same idea. [PR #3526](#) generated `shared-index.json` and `pkgsotest.nsh` from the current `modprint` and `sotest` outputs. Both Make and CMake produce the fixture, which means the test is not tied to one host build path.

This was the point where the work stopped being my local demo and became something another contributor could rebuild. It also changed what I recorded as evidence. A successful command wasn't enough by itself; the failure path and the cleanup afterward mattered too.

5. Growing `nxbpkg` into a full package lifecycle

The midterm installer proved the main idea, but persistent package state raises harder questions. What happens if power disappears during an update? Can a malformed catalog escape the package directory? Who owns a lock after a reset? I split the answers into a chain of pull requests so storage, untrusted input, networking, and lifecycle behavior could each be reviewed on their own.

5.1 Configurable and crash-aware storage

[PR #3642](#) moved all package-manager files below `CONFIG_SYSTEM_NXPKG_ROOT`, letting each board choose a writable filesystem such as its SD card. State updates use temporary files and rename-based replacement when the filesystem supports it.

Locks also gained an owner and a boot identity. During hardware tests, a running task kept its lock, while a lock from an exited task or an earlier boot could be reclaimed. **This gives `nxbpkg` a clear update order and a practical recovery path using the filesystem operations NuttX provides. It does not promise transactions on every filesystem.**

5.2 Treating catalogs and databases as input boundaries

[PR #3718](#) treats catalogs and the local database as untrusted input. Names and versions containing path separators are rejected, field sizes are bounded, current and previous version lists are checked, version parsing avoids overflow, and paths are validated before they are joined to the package root.

That strictness came from a simple observation: JSON is only a format. It doesn't make the contents safe. A truncated catalog, an edited SD card, or a bad remote entry should stop at validation. It shouldn't turn into a write outside the package root or a half-valid selection in the database.

5.3 Adding network synchronization without overstating trust

[PR #3719](#) added catalog synchronization and artifact downloads over HTTP or HTTPS. `nxbpkg sync` retrieves the catalog; `install` and `update` fetch the referenced files. Longer transfers renew

the package lock, and no downloaded payload is committed until its SHA-256 matches the catalog.

A matching SHA-256 tells me that the payload is the one named by the catalog. It doesn't tell me who published that catalog. Plain HTTP lacks server authentication and confidentiality too. Signed catalogs or packages remain future work, and I made that limitation explicit in the documentation.

The network patch built, passed review, and merged. When I later tried to repeat the test on the current hardware image, both Wi-Fi and loopback were disabled in that configuration. I label that run **environment-limited**. It wasn't a failing source test, and it wasn't a new hardware pass either.

5.4 Completing update, rollback, remove, and recovery

PR #3720 completed the command-line lifecycle: ``sync``, ``install``, ``update``, ``rollback``, ``remove``, and ``list``. The database acts as the commit record, so pointer updates and file cleanup are ordered around its state. Conflicting operations use per-package locks. A live owner gets ``-EBUSY``; callers do not wait forever.

On the touchscreen board's SD card, I ran install, update, rollback, and remove. I also forced a live-lock conflict, reclaimed a stale lock, and checked cleanup after a bad digest. **By then, ``nxpkg`` had become a recoverable package lifecycle rather than a one-command installer. The whole core series had merged.**

6. Putting a real interface on top: NXStore

Once package state was dependable enough for another component to consume, I built NXStore in PR #3643. It reads catalog and installed-package state through the public ``<system/nxpkg.h>`` API. Network access, hashing, filesystem work, and process launch all happen outside the LVGL event thread, which keeps the screen responsive during slower operations.

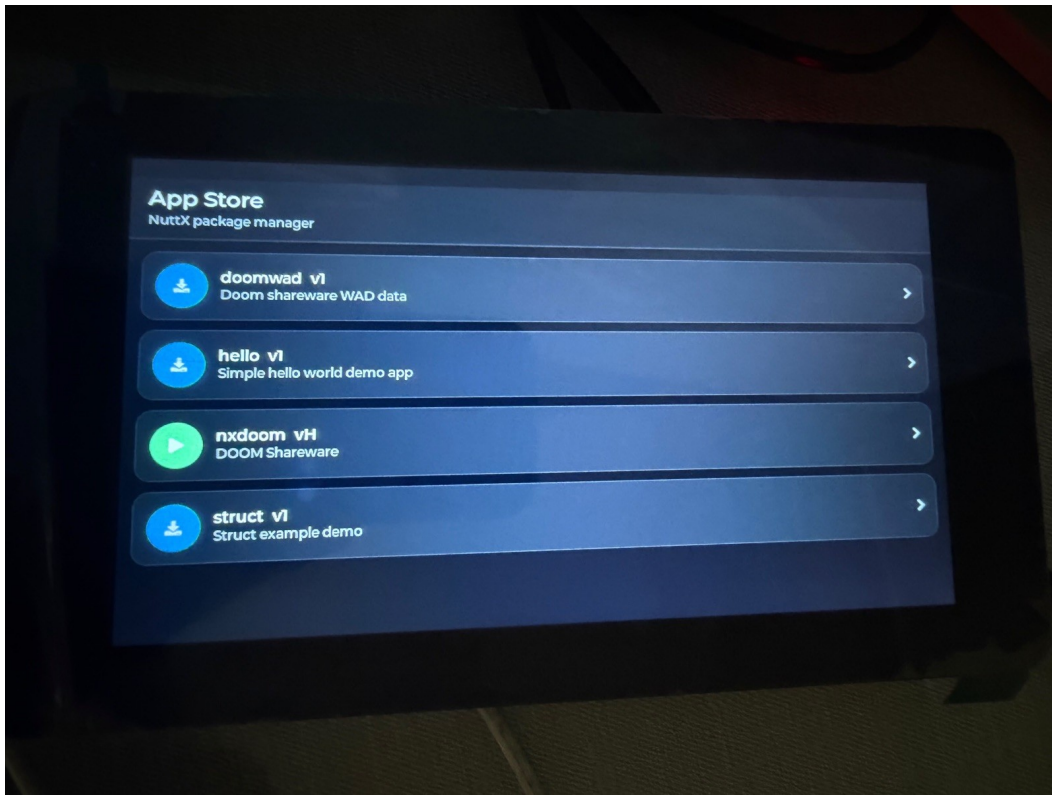


Figure 2. NXStore reading the package catalog on the 800 x 480 ESP32-S3 touchscreen target.

The interesting part starts after installation. NXStore hands the executable to a supervisor, which starts it as a separate child. When the user closes the application, the store can send `SIGTERM`, wait for the child to exit, and show the catalog again.

`cgol`` was my first end-to-end check of that flow. It launched with a 32 KiB stack and appeared under its own PID. The close action reached the application, the process exited, and the PID disappeared. At that point the store was no longer a screen full of package cards; it had a working install, launch, and return loop.

6.1 Applications that exercised different boundaries

[PR #3659](#) added an LVGL diagnostic for `/dev/fb0`` and `/dev/input0``; it merged on August 17. I used it whenever I needed to tell a board input or display problem apart from an NXStore or supervisor problem.

The other applications each stressed a different edge. [PR #3660](#) added a touchscreen calculator, including division-by-zero handling and cooperative shutdown. [PR #3661](#) let `cgol``, `match4``, and `snake`` run either as built-ins or modules, with process behavior suitable for supervision. [PR #3662](#) brought touchscreen and module support to Brickmatch; I played through both win and loss paths. [PR #3663](#) supplied descriptions, categories, and optional RGB565 icons.

Together, these applications exercised small modules, GUI state, touch input, metadata, resource files, repeated launches, and clean shutdown. I chose them to uncover integration bugs. Filling the catalog with attractive demos was secondary.

7. NXDoom showed where "it loads" stops being enough

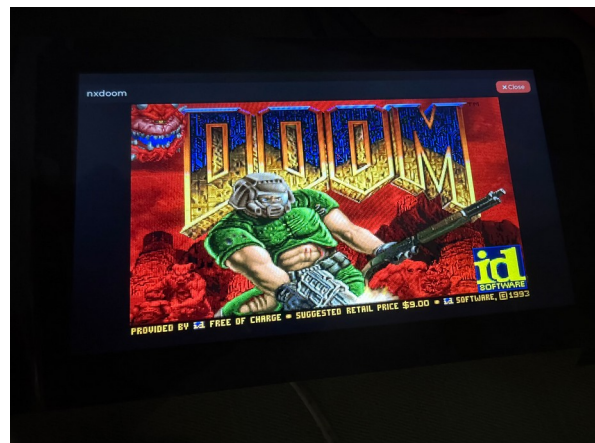
NXDoom became the heaviest system test in [PR #3644](#). One run touched a large loadable module, WAD files, framebuffer format, screen updates, process lifetime, and memory pressure.

The first result looked promising: a module of about 954 KiB loaded, resolved its dependencies, read the Doom shareware data, and entered the renderer and game loop. Then I looked at the screen. The pixels were corrupted. **That was a useful failure because it forced me to record loading, initialization, rendering, input, and shutdown as separate results.**

I moved the same code to the NuttX simulator and watched the VNC output to isolate the display path. The resulting changes query plane information, check framebuffer depth and format, account for the framebuffer origin, and issue `FBIO\_UPDATE` when needed. A later board run reached the NXDoom title screen inside the supervised application view.



Simulator / VNC



ESP32-S3 hardware

Figure 4. NXDoom validation progressed from simulator rendering to the supervised hardware application.

NXDoom also exposed Xtensa linker details that the smaller modules never reached. Literal placement had to agree with the text layout, and unnecessary exception-frame data introduced relocations the loader could not handle. A review thread about static and heap-backed renderer buffers was still unresolved on the report date. The visual evidence is real, but [PR #3644](#) remains in the open column.

8. Shared libraries, for real

The advanced part of the proposal called for an Android-make-style library declaration. [PR #3730](#) adds a root `Library.mk` interface with `LOCAL\_MODULE`, `LOCAL\_SRC\_FILES`, and language-specific flags. It reuses the compiler-runtime choices already made by `Application.mk` and accepts multiple C or C++ source files.

This also follows Alan's suggestion that real library support could reduce the memory spent on repeated statically linked copies. The declaration stays short:

```
include $(APPDIR)/Make.defs

LOCAL_MODULE := modprint
LOCAL_SRC_FILES := modprint.c
include $(APPDIR)/Library.mk
```

A clean `esp32s3-devkit:sotest` Make build produced the modules I copied to the board. The loader resolved functions, shared data, and the library dependency. Both modules ran and unloaded. I then repeated the exercise with mixed C/C++ sources and checked the compiler-runtime dependency as well.

That is working shared-library packaging, and both the `Library.mk` interface and its companion documentation are now upstream. **It does not prove FDPIC.** FDPIC needs architecture and toolchain evidence that this project hasn't produced. [PR #19888](#) documents the declaration and its current scope.

9. The architecture I ended up with

There wasn't a finished architecture diagram waiting at the start of the project. The boundaries became clearer while the local package path, persistent state, supervisor, and UI began working together. In the design I ended up with, `nxpkg` owns package state, both the CLI and NXStore call the same service, and process execution belongs to the loader and supervisor.

Runtime path used in this project

```
static repository
  |
  | catalog + ELF/library artifact + optional icon
  v
nxpkg: sync -> verify -> install/update/rollback/remove
  |
  | validated database + versioned package files
  v
configured package root
  |
  | selected executable or library path
  v
NuttX ELF loader -> supervisor -> running application
                        ^
                        |
                    CLI / NXStore
```

Names, versions, paths, compatibility fields, and hashes are checked before anything reaches persistent state. All writes stay under `CONFIG\_SYSTEM\_NXPKG\_ROOT`. The database records the selected version; current and previous version directories make rollback possible. Package locks protect lifecycle operations, while the supervisor keeps process lifetime out of the UI callback.

The same boundaries guided debugging. I could inspect repository input, stored package state, the chosen artifact, loader behavior, and the running process one at a time instead of treating the whole path as one opaque application launch.

10. Upstream contribution inventory

This is the public contribution snapshot I used on August 21. **A merged entry had been accepted into the Apache repository. An open entry was implemented in public, but its review could still change it.**

Repository / PR	Status	Contribution	Snapshot size
`nuttx-apps` #3474	Merged	Basic package lifecycle helper	+2,920 / 15 files
`nuttx-apps` #3519	Merged	Missing module metadata for executable tools	+32 -18 / 11 files
`nuttx-apps` #3526	Merged	Packaged shared-library fixtures	+729 -31 / 8 files
`nuttx-apps` #3531	Merged	Extended ELF and `nxpkg` validation fixtures	+687 -3 / 6 files
`nuttx-apps` #3642	Merged	Configurable and crash-aware package storage	+522 -57 / 5 files
`nuttx-apps` #3718	Merged	Catalog, database, path, and version validation	+355 -56 / 2 files
`nuttx-apps` #3719	Merged	Network catalog and artifact fetching	+734 -1 / 4 files
`nuttx-apps` #3720	Merged	CLI lifecycle, rollback, locks, and recovery	+628 -111 / 2 files
`nuttx-apps` #3659	Merged	LVGL touchscreen diagnostic	+266 / 5 files
`nuttx-apps` #3643	Merged	NXStore LVGL touchscreen frontend	+1,941 -87 / 7 files
`nuttx-apps` #3644	Open	NXDoom RGB565, module, and crash fixes	+267 -23 / 16 files
`nuttx-apps` #3660	Open	LVGL touchscreen calculator	+625 / 5 files
`nuttx-apps` #3661	Open	Supervised loadable game processes	+95 -28 / 4 files

Repository / PR	Status	Contribution	Snapshot size
`nuttx-apps` #3662	Merged	Brickmatch touchscreen and module support	+324 -17 / 4 files
`nuttx-apps` #3663	Open	Package metadata and RGB565 icon export	+123 -8 / 1 file
`nuttx-apps` #3730	Merged	Declarative shared-library support	+257 -27 / 8 files
`nuttx` #18875	Open draft	`nxpkg` and NXStore user/developer documentation	+253 / 3 files
`nuttx` #19888	Merged	Shared-library declaration documentation	+25 / 1 file

Contribution statistics

Measure	Merged	Open	Total
Pull requests	13	5	18
GitHub additions	9,420	1,363	10,783
GitHub deletions	408	59	467
Changed-file entries	78	29	107

The diff totals give a sense of how much code reviewers had to consider. They aren't a clean-room measure of my authorship. The useful record is the linked work itself, its review state, and the behavior another contributor can reproduce.

11. What I actually ran

11.1 Targets and their evidence boundaries

Target	How I used it	What it proves
Seeed Studio XIAO ESP32S3 Sense	Early ELF, install/list, and fixture validation	The midterm and local executable package path
ESP32-S3 800 x 480 touchscreen board	SD-card lifecycle, LVGL input, NXStore, supervised apps, and NXDoom	The primary final device workflow
NuttX simulator with VNC	Isolate NXDoom framebuffer and update behavior	Simulator rendering behavior only; ESP32-S3 timing and memory remain outside this evidence
`esp32s3-devkit:sotest` configuration	Shared-library build and loader exercise	The tested Xtensa shared-library path; FDPIC and other architectures remain outside this evidence

The touchscreen work was built on Alan's upstream board support. I used a few local display and control-path experiments while debugging, but they were validation aids. They are not listed as my GSoC pull requests.

11.2 Validation matrix

Scenario	Expected result	What I observed	Status
Install valid ELF fixture	Hash and compatibility pass; package appears in list	Installed and listed on hardware	Pass
Install missing artifact	Return <code>-ENOENT</code> ; no installed record	Failed with <code>-2</code> ; package list stayed empty	Pass
Install bad SHA-256	Reject artifact and remove staging data	Digest failure and cleanup observed	Pass
Live package lock	Conflicting request returns busy	<code>-EBUSY</code> observed	Pass
Stale package lock	Reclaim after owner exit or old boot	Stale ownership reclaimed	Pass
Update then rollback	New version current; prior version recoverable	Current and previous selections exchanged	Pass
Remove	Database and files converge to absent package	Package removed from SD-card state	Pass
Launch and close <code>`cgol`</code>	Child starts, handles termination, and exits	PID launched and disappeared after close	Pass
Calculator repeated use	Arithmetic works across repeated lifecycle	Arithmetic and repeated close tested	Pass
Brickmatch touchscreen	Touch maps to paddle and reaches end states	Win and loss paths exercised	Pass
NXDoom module load	Large module resolves and reaches game loop	Loader and game initialization reached	Pass
NXDoom visual output	Correct RGB565 and update path	Simulator gameplay and hardware title screen captured	Pass with open review item
Shared-library modules	Resolve function/data dependencies and unload	Both modules loaded, ran, and unloaded	Pass
Fresh network re-test	Reach repository over configured interface	Wi-Fi/loopback path absent in active image	Environment-limited

I keep an unavailable test environment separate from a failed test. The same applies to simulator rendering and execution on the board; each proves something different.

12. Decisions that made more sense after using the system

12.1 A static repository before a custom server

A JSON catalog and ordinary files were enough for the first repository. Anyone can inspect them, and simple static hosting can serve them. That let me spend the GSoC period on the embedded client. Signing, repository policy, and discovery can be designed separately.

12.2 Versioned storage before in-place replacement

Separate version directories use more storage, but the state is easy to reason about. The database can select another version without rewriting a binary that may still be running. Recovery also has two concrete places to look: current and previous.

12.3 SHA-256 before commit

The digest is checked before an artifact becomes installed. That catches corruption and a mismatch between the catalog and the downloaded file. A digest cannot replace a package signature, so the trust limitation appears in both the documentation and this report.

12.4 The database as the commit record

Pointer changes and file deletion follow the package-state transition recorded in the database. Since that database is the source of truth, I validate it as strictly as the remote catalog.

12.5 Per-package locking

One global lock would be easy to implement and frustrating to use, since an operation on one package would block every other package. Per-package ownership keeps the contention local. A boot identity also makes it possible to recognize a lock left behind by an earlier boot.

12.6 Work outside the UI thread and cooperative shutdown

Hashing, filesystem traversal, downloads, and module loading can all take longer than an LVGL callback should. NXStore sends that work outside the event thread. When an application closes, the launcher requests `SIGTERM`, and the examples clean up before control returns to the store.

13. The problems that changed my approach

13.1 A demo is not yet a subsystem

The first installer proved the concept, yet it bundled too many concerns into one review. I split storage, parsing, transport, and lifecycle into dependent patches. Moving parent PRs caused some temporary CI noise, but each change became easier to explain and the core eventually merged.

13.2 Resets can happen between any two writes

On a small board, I can't assume desktop filesystem or process behavior. Power or a reset can interrupt any write. That is why the implementation uses staging, bounded parsing, temporary-file replacement, versioned directories, explicit state ordering, and stale-lock recovery.

13.3 A successful loader return can still produce a broken application

NXDoom loaded and entered its game loop while the display was still wrong. I had to check framebuffer format, origin, update behavior, resources, and shutdown separately. Since then, I have recorded build, load, initialization, rendering, interaction, and exit as separate results.

13.4 Board enablement and my GSoC contribution are not the same thing

The touchscreen target depended on board-specific display and I/O support. I used that work while validating the project and credited the upstream base. The contribution table remains limited to my own GSoC pull requests.

13.5 The deadline arrived before every review finished

The package-manager core merged during the final week. By August 21, NXStore, Brickmatch, `Library.mk`, and the shared-library documentation had followed it upstream. Five pull requests were still open: NXDoom, the calculator, supervised games, metadata and icon export, and the main `nxpkg` and NXStore guide. **I would rather show that boundary clearly than let successful screenshots imply that every part had already merged.**

14. Assessment against the original proposal

Proposed deliverable	My assessment	Evidence and boundary
Validate dynamic ELF applications	Completed	Small fixtures and the large NXDoom module exercised; metadata fixes merged
Create an LVGL application	Completed; partly merged	NXStore merged and was hardware-tested; calculator remains open
Support application updates	Completed and merged in core	Update, previous-version tracking, rollback, and cleanup in `nxpkg`
Download applications	Completed and merged in core	Local and HTTP(S) sources; final fresh board network re-test environment-limited
Package manager install/update/remove	Completed and merged in core	Includes sync, list, rollback, locks, and recovery
Android-make-style library declaration	Completed and merged	`Library.mk` in PR #3730 ; not FDPIC proof
Package and run shared libraries	Completed and merged	Generated fixtures and the declaration interface are upstream
Graphical application store	Completed, merged, and hardware-tested	NXStore PR #3643 ; catalog icons are not yet shown before installation

Proposed deliverable	My assessment	Evidence and boundary
Demonstration distribution	Partially completed	Static repository format and device client exist; no maintained reference image/catalog for ESP32-S3 or STM32H7
Dropbear	Not attempted	Optional extension kept outside the core path

My assessment is that the central package-distribution goal and most of the advanced lifecycle work are complete. The package manager, NXStore, and shared-library declaration are now upstream, while several applications, the metadata and icon export, the main guide, and trust-model work remain. I see the result as a strong upstream foundation with an integrated prototype on top. Pre-install application previews and a maintained board distribution are two visible gaps between that prototype and a complete product.

15. Future work

The next work is not one more isolated feature. It is the work needed to make the package path understandable to users, repeatable for board owners, and maintainable after a demo.

15.1 Close the current upstream path

- Work through the remaining review comments on NXDoom, the calculator, supervised games, metadata and icon export, and the `nxpkg` and NXStore guide.
- Rebase each remaining dependent pull request on the merged package and store layers, then rerun its checks.
- Repeat the repository download test on firmware with the required network interface enabled.
- Update the public work-product page if a review state changes before the deadline.
- Send Alan and Halysson the stable work-product URL for a preview, while keeping the submission deadline independent of their reply.

15.2 Show an application before asking the user to install it

[PR #3663](#) added descriptions, categories, and optional RGB565 icon export to the application metadata. That supplies the content, but NXStore does not yet display those icons while the user browses the catalog. The next UI step is to show a recognizable icon and a compact preview before installation, with the package name, version, category, compatibility, and download size close by.

- Define how NXStore retrieves, validates, caches, and decodes icon assets without blocking the LVGL event thread.
- Provide a clear fallback when an icon is absent, corrupt, too large, or uses an unsupported format.
- Test catalog scrolling and memory use with several icons rather than a single happy-path package.

This would not change the package lifecycle, but it would change how understandable the store feels. A person should have some idea of the program before committing storage and bandwidth to it.

15.3 Build a real NuttX distribution around the package path

A useful next milestone is a reproducible reference distribution, not just a screenshot and a collection of pull requests.

- Start with one supported ESP32-S3 touchscreen profile that includes the filesystem layout, network configuration, NXStore, a versioned catalog, a small application set, and documented update and recovery steps.
- Bring the same distribution contract to an STM32H7 target to expose assumptions about architecture, storage, display, input, and toolchain support.
- Add other capable boards only through explicit profiles, keeping board-specific setup outside the package and store logic.
- Publish a repeatable image and catalog release process, with a small hardware test matrix and a clear statement of what is supported.

I would call this a NuttX distribution only when another person can build or flash it, reach the same catalog, install and update an application, and recover from a bad package without reconstructing my development environment.

15.4 Harden the platform underneath it

- Design signed catalog or package metadata, including key rotation and failure behavior.
- Add expiry and replay rules for repositories, then document how TLS credentials are provisioned.
- Exercise more reset and fault-injection cases around database and pointer changes.
- Add storage quotas, dependency and conflict rules, and cleanup for unused versions.
- Run the package path across the ESP32-S3 and STM32H7 reference profiles, then extend the architecture and filesystem matrix.
- Extend the merged `Library.mk` support with an appropriate CMake interface.
- Explore FDPIC on an architecture and toolchain that can provide meaningful evidence.
- Keep the demonstration catalog small and maintained as the package format settles upstream.

16. Reproducing the work

16.1 Build NuttX with package and loader support

Start with a writable filesystem and point `CONFIG\_SYSTEM\_NXPKG\_ROOT` at its mount point. Enable the JSON, hashing, network, ELF or module, and filesystem options used by the chosen test. NXStore also needs LVGL, framebuffer and input devices, the supervisor dependencies, and whichever application modules will appear in the catalog.

```
./tools/configure.sh <board>:<configuration>  
make -j
```

[PR #18875](#) collects the package-manager and NXStore dependencies, repository and manifest formats, command-line behavior, recovery model, and transport limitations.

16.2 Generate fixtures and exercise the lifecycle

Enable either the ELF fixture or the `sotest` fixture. The host helper builds the index and NSH scripts from artifacts produced by that configuration, including their SHA-256 values. The resulting repository can be copied to the target filesystem or served through static HTTP(S) hosting.

```
nxpkg sync <source>
nxpkg install <package>
nxpkg list
nxpkg update <package>
nxpkg rollback <package>
nxpkg remove <package>
```

A useful run covers more than the happy path. I would include a missing artifact, a bad hash, a live-lock conflict, stale-lock recovery, and behavior after restart alongside the successful lifecycle.

16.3 Build and use a shared library

Using the merged `Library.mk` interface from [PR #3730](#), declare the sources, build the application module, and run the `sotest` fixture. Check function calls, shared data, dependency resolution, and unloading. This covers the shared-library path implemented in the project. FDPIC needs a separate test.

16.4 Add a package to NXStore

Give the package a stable name and version, compatibility fields, an artifact location, and a SHA-256 digest. Description, category, and icon metadata are optional. After installing it with `nxpkg`, check that NXStore shows the same state. Full pre-install icon rendering remains future work; the store will need to validate and cache the asset, enforce a supported format and size, and fall back cleanly when it cannot display one. An application that returns to the store should validate its devices and resources at startup and handle cooperative termination cleanly.

17. What stayed with me

This project changed what the phrase "install an app" means to me on an embedded system. Behind that small UI action are input validation, storage layout, hashing, write ordering, concurrency, binary compatibility, process lifetime, and the capabilities of the board itself. **The feature only felt convincing once I could test those boundaries separately.**

I've become much more comfortable entering an unfamiliar subsystem and finding the smallest change that can move upstream. The work took me through Kconfig, Make and CMake, C and C++, JSON parsing, filesystems, networking, LVGL, ELF loading, Xtensa behavior, framebuffer APIs, and Sphinx documentation. The most valuable lesson was less technical: code I built, code that merged, and behavior shown in a screenshot are three different kinds of evidence.

Alan Carvalho de Assis and Halysson Carvalho helped turn a broad proposal into a sequence I could actually execute. Their reviews covered the package manager, loader, UI, and hardware path. Reviews from the wider Apache NuttX community changed the implementation too, especially around configurable storage, stricter input handling, and explicit failure states.

During GSoC, I got to follow one feature through design, failed experiments, debugging on a real board, review, documentation, and integration. **I want to keep working through the open pull requests and the remaining trust and portability gaps after the program ends.**

18. Public work-product links

Project and program references

- [Project issue #17351](#): original project scope
- [GSoC work-product guidelines](#)
- [Apache NuttX](#)

Merged `nuttx-apps` contributions

- [PR #3474](#): basic package lifecycle helper
- [PR #3519](#): module metadata for executable tools
- [PR #3526](#): packaged shared-library fixtures
- [PR #3531](#): ELF and `nxpkg` validation fixtures
- [PR #3642](#): configurable and crash-aware storage
- [PR #3718](#): catalog and database validation
- [PR #3719](#): network catalog and artifact fetching
- [PR #3720](#): complete lifecycle and command line
- [PR #3659](#): LVGL touchscreen diagnostic
- [PR #3643](#): NXStore LVGL frontend
- [PR #3662](#): Brickmatch touchscreen and module support
- [PR #3730](#): declarative shared-library support

Open `nuttx-apps` contributions

- [PR #3644](#): NXDoom RGB565 and module support
- [PR #3660](#): LVGL touchscreen calculator
- [PR #3661](#): supervised loadable games
- [PR #3663](#): package metadata and RGB565 icon export

NuttX documentation contributions

- [PR #19888](#): shared-library declaration documentation (merged)
- [PR #18875](#): `nxpkg` and NXStore guide (open)

19. Where the project stands now

The project has traveled a long way from the first local `hello` package on the XIAO board.

Its artifacts are now reproducible, both `nxpkg` and NXStore are upstream, and the merged `Library.mk` interface packages shared dependencies for loadable applications. NXDoom and the shared-library tests pushed the same path far beyond the original small example.

Everything needed to understand the current state is public: the merged package, store, and library layers; the remaining application pull requests; fixtures; board and simulator evidence; documentation; and the list of unfinished work. A contributor can start from the upstream stack, repeat the fixture and board tests, and see where trust, portability, and review work still remain.

GSoC 2026 gave me a deadline and a reason to bring the pieces together. Some pull requests were still open when that deadline arrived, and the project continues beyond the submission. **I am finishing the program with a foundation already upstream, a product path running on hardware, and a much clearer idea of what I want to complete next.**