

Shipping Guide

Hey all! As of 05/05/2026, more than half of projects on #horizons are being sent back for changes needed. This guide documents the requirements for each project type, so that rejections shouldn't come as a surprise! Use the **On this page** menu to jump to your ship type.



Web Playable [🔗](#)

A project that runs in a browser: websites, web apps, browser games, demos.

WHERE TO HOST

Vercel

Generous free tier, instant git deploys, edge functions cover most backend needs. Double-check you're sending the public *.vercel.app URL, not your internal vercel.com/<you>/<project> dashboard link.

Hack Club Nest

Free Linux shell from Hack Club, runs anything including long-lived backends.

GitHub Pages

Super fast to set up, native to GitHub, free, but won't run a backend.

itch.io

Drag-and-drop a zip; perfect for browser games and jam-style projects.

Netlify

Git-based deploys with a generous static-site free tier, similar feel to Vercel.

– **Streamlit Community Cloud**

Zero-config for Streamlit/Python apps, but takes minutes to start up which is a pain to review.

~~localhost only~~

Your site must stay indefinitely reachable; localhost is only up while your machine is. Not acceptable.

~~Tunnels (bore, ngrok, Cloudflare Tunnel)~~

Same problem as localhost: the tunnel dies when your laptop sleeps. Your site must stay indefinitely reachable. Not acceptable.

STEPS

- 1 Push your source code to a public GitHub (or GitLab) repo.
- 2 Deploy somewhere from the recommended list above.
- 3 Open the deployed link in a fresh / incognito window to test it works without sign in.
- 4 Paste the deployed URL into Demo URL, and the repo into Code URL.
- 5 Add 1 or 2 screenshots and a README that says what it does, how to use it and how it works. Ideally include that it is a project for #horizons.

COMMON REJECTION REASONS

- ×28 README is missing the basics: real description, screenshots, tech stack, motivation, and how-it-works section
- ×11 Project was already shipped to another Hack Club program (double-dipping is not allowed without significant new work)
- ×10 README appears to have been written by AI
- ×8 Project relies on AI beyond the 30% limit
- ×7 All code committed in one go: reviewers cannot see incremental progress
- ×7 A visible feature is broken in the deployed demo (dead button, 404 asset, navbar, etc.)
- ×6 Project pre-dates the Horizons start window and was not updated meaningfully
- ×5 Wrong or unrelated Hacktime project linked, or the same Hacktime project re-used across submissions
- ×5 Under the 3-hour minimum



Windows Playable [🔗](#)

A native Windows app or game (.exe or installer).

WHERE TO HOST THE BUILD

- [? GitHub Release](#)

Trusted, version-tagged, and free; the default reviewers expect.

itch.io

Listing page with screenshots, builds, and download counts in one place.

~~**Google Drive (link share)**~~

Your build needs to live somewhere indefinite and trusted. Drive links can be revoked, throttled, or quietly disappear, so they don't meet that bar. Stick to standard development practice and ship from a GitHub Release.

~~**random file host**~~

Same problem: the file source must be indefinite and trustworthy. Anonymous file lockers fail both tests. Use the standard developer workflow and host the build on a GitHub Release.

STEPS

- 1** Build a release binary (dotnet publish, cargo build --release, your engine's Windows export, etc).
- 2** Zip the build folder so reviewers get the .exe plus any required DLLs and assets in one shot.
- 3** Upload the zip to a GitHub Release.
- 4** Put the download link in Demo URL and your source in Code URL.
- 5** In your README, list system requirements and any first-run steps (unblocking the .exe, allowing through SmartScreen).

COMMON REJECTION REASONS

×4 README appears to have been written by AI

×2 Project was already shipped to another Hack Club program

×1 README is missing screenshots, motivation, and details

- Test on a clean virtual machine or a friend's laptop that doesn't have your dev tools installed, to catch missing dependencies.



Mac Playable

A native macOS app (.app bundle or .dmg).

BUILD FORMAT

Universal (arm64 + x86_64)

Runs on every Mac sold in the last decade; friendliest for reviewers.

Apple Silicon only

Fine for M-series Macs (2020+); label it clearly so Intel users know.

Intel only

Works on older Macs and runs under Rosetta on M-series; label the arch.

STEPS

- 1 Build a release .app bundle and zip it, or package as a .dmg.
- 2 Upload to a GitHub Release.
- 3 Add the download link as Demo URL and the repo as Code URL.
- 4 Document the unsigned-app workaround: right-click then Open the first time (you almost certainly won't have a paid Apple Dev cert).
- 5 Note in the README whether the build is Intel, Apple Silicon, or universal.



Linux Playable

A native Linux build: AppImage, .deb, .tar.gz, or a build script.

FORMAT CHOICE

AppImage

Single executable that bundles its own deps; runs on most modern distros.

Tarball + launcher

Works anywhere with the right runtime, but reviewers do more setup.

.deb / .rpm

Native package install on Debian/Ubuntu (deb) or Fedora/RHEL (rpm); distro-locked.

~~Build from source only~~

Reviewers won't install your toolchain just to try it.

STEPS

- 1 Produce a runnable artifact. AppImage is easiest because it bundles its own deps.
- 2 Upload to a GitHub Release.
- 3 Add the download link as Demo URL and the repo as Code URL.

- 4 In your README, list the distro you tested on and any system packages reviewers need.
- 5 Make sure the file is executable (chmod +x) before zipping if relevant.

COMMON REJECTION REASONS

- ✖1 README appears to have been written by AI
- ✖1 Demo URL points to the GitHub repo instead of an actual demo
- ✖1 Under the minimum-hours threshold



Cross Platform Compatible [🔗](#)

You ship builds for more than one OS. Reviewers only need at least one of them to actually work, so pick the platforms you can confidently test and don't spread yourself thin.

DISTRIBUTION

📄 Single GitHub Release

All builds in one place so reviewers can compare and grab the right one.

📄 GitHub Actions matrix build

Reproducible cross-compile that proves your CI works; free for public repos.

STEPS

- 1 Produce a build for each OS you want to advertise. As long as one of them runs cleanly for reviewers, you're fine.
- 2 Upload every build to a single GitHub Release.
- 3 Label each download clearly (myproject-win64.zip, myproject-mac-universal.dmg, myproject-linux.AppImage).
- 4 Link the Release page in Demo URL.
- 5 In your README, write a tiny table: platform, arch, tested OS version, and which one you tested most thoroughly.
- 6 If you cross-compiled with GitHub Actions, link the workflow file. Reviewers love that.

COMMON REJECTION REASONS

✖4 Demo link is broken or unreachable (cannot SSH, cannot open in a browser, video link broken)

✖4 README is AI-written or missing the basics (description, tech stack, motivation)

✖2 Wrong or unrelated Hacktime project linked

✖2 Hacktime account is banned (referred to the Fraud Squad)

✖2 Project pre-dates the Horizons start window

✖2 Under the 3-hour minimum

✖1 Repo is empty or only contains a README

- You don't need to ship every OS. Two builds you actually tested beats three you guessed at.

</>

Python

A standalone Python script, CLI tool, or desktop app. Reviewers won't install Python, your dependencies, or set up a virtualenv just to try your project, so you must ship a compiled binary they can double-click.

! Got a web app with a Python backend (Flask, FastAPI, Django, Streamlit)? That belongs under Web Playable, not here. The compile-to-binary path is for standalone scripts and desktop apps only.

HOW TO COMPILE TO A BINARY

auto-py-to-exe (Windows)

Friendly GUI on top of PyInstaller; produces a Windows .exe. Easiest first-time packager.
pypi.org/project/auto-py-to-exe

PyInstaller (Windows / macOS / Linux)

The underlying tool auto-py-to-exe wraps. Run it on Windows for .exe, macOS for .app, Linux for an ELF binary. Does not cross-compile.

py2app (macOS)

macOS-native packager that builds proper .app bundles.

❓ **Nuitka (Windows / macOS / Linux)**

Compiles Python source to C and then to a native binary. Faster runtime and smaller surface area than the freezers.

❓ **cx_Freeze (Windows / macOS / Linux)**

Alternative freezer; cross-platform but again must be run on each target OS.

❓ **Briefcase / BeeWare (cross-platform + mobile)**

Targets desktop and mobile (iOS / Android) from a single Python codebase.

❓ ~~Shipping the raw .py file~~

Reviewers won't install Python or pip your deps. Always compile to a binary.

STEPS

- 1 Pick a packager for your target OS from the list above.
- 2 Run it on each OS you want to support. PyInstaller, py2app, Nuitka, and cx_Freeze do not cross-compile.
- 3 Upload the binaries to a GitHub Release, one zip per platform.
- 4 Put the Release link in Demo URL and your repo in Code URL.
- 5 In the README, list each binary's OS and arch, plus any first-run steps (unblocking on Windows, right-click-Open on macOS).

COMMON REJECTION REASONS

✖3 README appears to have been written by AI

✖2 Project relies on AI beyond the 30% limit

✖2 All code committed in one go: reviewers cannot see incremental progress

✖2 Project was already shipped to another Hack Club program

✖1 Wrong or unrelated Hackatime project linked

✖1 Script needs to be shipped as a compiled binary (PyInstaller, py2app, etc.) instead of a raw .py file

- Test each binary on a clean virtual machine or a friend's laptop that doesn't have Python or your dev tools installed.



Android Playable [🔗](#)

An Android app: APK or AAB.

HOW REVIEWERS INSTALL IT

📄 **GitHub Release APK**

Trusted distribution path with version tags; what reviewers expect.

📄 ~~Google Drive APK~~

Drive triggers "scan for harmful files" warnings, throttles big downloads, and can revoke the link. Use a GitHub Release for the APK.

– **Play Store upload**

Great if you already have it set up, but the \$25 developer fee plus review wait is not required for Horizons. A GitHub Release APK is enough.

STEPS

- 1 Build a release APK. Signed with a debug key is fine for review.
- 2 Upload the APK to a GitHub Release.
- 3 Put the APK link in Demo URL and your repo in Code URL.
- 4 In your README, list the minSdk and any permissions the app requests.
- 5 Mention "install from unknown sources" must be allowed. This is normal for sideloading.

COMMON REJECTION REASONS

✖2 README is AI-written or missing screenshots, motivation, and details

- Record a short screen capture of the app running and link it from the README.



iOS Playable [🔗](#)

An iOS app. The trickiest to ship because Apple doesn't allow easy sideloading. A demo video linked in your README is always required, no exceptions.

HOW REVIEWERS SEE IT

Demo video linked in README

Required for every iOS submission. Reviewers can't install unsigned builds, so the video is the only way they see your app working.

Reproducible Xcode project

Required alongside the video: reviewers clone, open in Xcode, run on Simulator to verify the code matches what's in the video.

TestFlight invite

Nice extra if you have it set up; lets reviewers test on real hardware. Does not replace the README demo video.

~~Just an .ipa link~~

Useless without a paid Dev cert; reviewers can't install unsigned builds.

STEPS

- 1 Record a screen capture or device video showing the app running, and host it (YouTube, Vimeo, direct link).
- 2 Embed (or link to) that video in your README. This is required for every iOS ship.
- 3 Push the full Xcode project to a public GitHub repo.
- 4 Write README build instructions: clone, open in Xcode, run on Simulator.
- 5 Use Demo URL for the same video link.
- 6 TestFlight is a nice extra but the README video + reproducible Xcode project is the required path.

- iOS Simulator is fine for review. No physical device needed.
- Make sure the project builds without a paid Apple Dev account (no signing-required entitlements).



Hardware

A physical project: PCB, electronics, robot, sensor rig, enclosure. It also can be a hardware design that hasn't been built yet. (We encourage you to build it, you can use the hardware grants in the shop.)

WHAT GOES IN THE REPO

Schematic PDF

Lets reviewers quickly understand your circuit without installing CAD software.

❓ **Board files (KiCad / EasyEDA), or full schematic for non-PCB projects**

Source files prove you designed the hardware and let others remix or manufacture it.

❓ **Bill of Materials**

Shows what parts are required and the approximate cost of reproducing the design.

❓ **Firmware source (if applicable)**

Include the code needed to program and test your hardware.

❓ **Photos of the assembled build (if built)**

Build photos are encouraged, but not required. Design-only hardware projects are fully acceptable.

❓ **Gerbers (if PCB designed)**

Manufacturing files make your PCB fully reproducible.

❓ ~~Only screenshots or renders~~

Your repository must include the original design source (schematics, CAD files, firmware, etc.), not just images.

STEPS

- 1 Push your schematics, board files, firmware (if any), and Bill of Materials to GitHub.
- 2 Export your schematic and PCB layout as PDFs or PNGs so reviewers can inspect them without installing CAD software.
- 3 Write a **Steps to Reproduce** section in your README explaining how someone can manufacture, assemble, flash, and test the project.
- 4 Include a complete Bill of Materials with quantities, important part numbers, and approximate costs.
- 5 Commit your work frequently. Many small commits help reviewers understand how your project evolved.
- 6 Track your work using lapse.hackclub.com. Lapse now supports both traditional recordings and Lookout by simply choosing the Desktop mode in lapse, so if you prefer using Lookout, you can now have your recordings synced with your Hacktime project. Include your recording links in your `JOURNAL.md` file. Other recording methods (such as YouTube videos) are not accepted for time tracking.
- 7 Keep a development journal with entries roughly every session of work.
- 8 If you build the project, include photos showing the completed hardware and showing it working. Design-only projects do not require a physical build.
- 9 Research time may be tracked, provided it remains a reasonable portion of the project and does not dominate your logged hours.

- 10 You may design a zine page for your project. Up to **30%** of your project's tracked time may be counted towards the zine page or any other graphic design work.
- 11 Document what your project does, flashing instructions (if applicable), and any important safety notes such as LiPo batteries, mains voltage, or lasers.

- Design-only hardware projects are welcome. Building the hardware is optional, but all design source files must be included.
- A complete Bill of Materials is required so reviewers can understand the project and reproduce it.
- Use **Lapse** for all tracked work. Lapse now supports **Lookout**, allowing Lookout recordings to be synced with your Hacktime project. Include your recording links in `JOURNAL.md`.
- Research and documentation are valid project work when they directly support your build, but they should not make up the majority of your tracked time.
- Zine page design is encouraged, but only up to **30%** of your project's original tracked time may count toward it.



CAD Models

Designs whose deliverable is a CAD file: 3D-printed parts, laser-cut designs, or mechanical assemblies. Review takes longer for this category because reviewers physically reproduce the design (printing, cutting, assembling), so the bar for documentation and reproducibility is higher than for software submissions.

FILE FORMATS

Slicer-ready export (.stl / .3mf, or .dxf for laser cut)

Required. This is the file reviewers drop straight into a slicer or laser-cutter workflow to reproduce your design.

STEP source (.step / .stp)

Universal interchange format that stays fully editable in any CAD tool.

Native CAD source (.f3d, .scad, .ipt, .blend)

Commit alongside the slicer-ready export so the design remains editable and remixable.

~~Source files only, with no slicer-ready export~~

Reviewers cannot open .blend, .f3d, or similar source files in a slicer. Without an .stl, .3mf, or .dxf export the design cannot be reproduced.

STEPS

- 1 Commit a slicer-ready export (.stl or .3mf for 3D printing, .dxf for laser cutting). This is the file reviewers will actually use to reproduce your design.
- 2 Commit the native CAD source alongside it so the design remains editable and remixable.
- 3 Write a detailed README. Document all production settings that apply to your process: layer height, infill, supports, orientation, material, and kerf.
- 4 Provide assembly instructions for any multi-part design. A labelled exploded view, photo sequence, or short build video are all acceptable.
- 5 For mechanisms, prefer 3D-printed components over fasteners. Where fasteners are unavoidable, specify standard sizes (for example M3 or M4) so reviewers can source them.
- 6 Commit progress regularly. Aim for roughly one commit per hour of work so reviewers can see the design evolve rather than appear in a single dump.
- 7 Add 2 to 3 renders to the README and a photograph of the physical result next to the digital model. Use Demo URL for an exploded view or turntable video.

- Because CAD reviews require physical reproduction, ship everything a stranger would need to print or cut your project from scratch with no follow-up questions.
- If your CAD tool offers free public sharing (Onshape public documents, Fusion 360 web viewer), link that as well.



Other / Not Sure

For projects that don't fit any of the categories above. The bar here is simple: write a good README, and if you aren't sure your project is shippable, ask in #horizons-help on Hack Club Slack before submitting.

STEPS

- 1 Write a detailed README that clearly explains what the project is, how it works, and how a stranger would experience it.
- 2 If you aren't sure the project qualifies, post a quick description in #horizons-help on Hack Club Slack and a reviewer will let you know before you ship.

