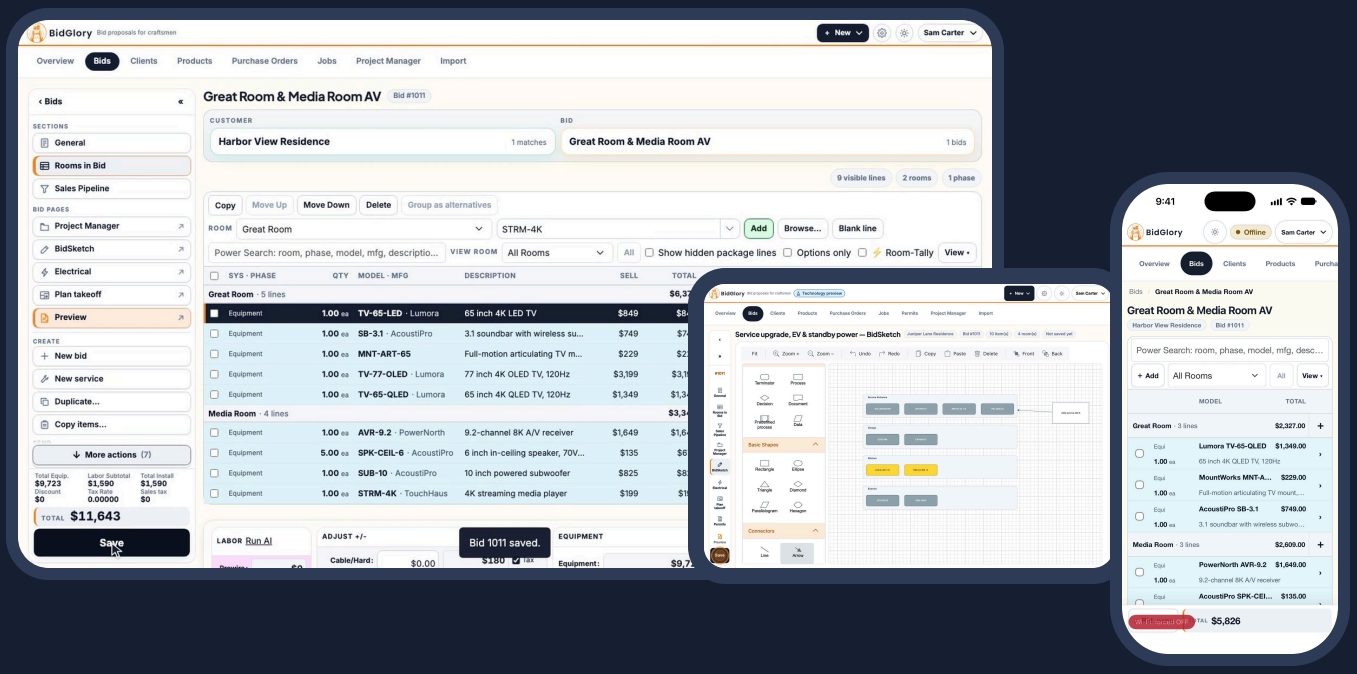
BidGlory
PROPOSAL & PROJECT MANAGEMENT SOFTWARE

A PROJECTTHUNDER CASE STUDY · CROSS-PLATFORM PRODUCT ENGINEERING

One estimating platform, on every screen a contractor owns.

How ProjectThunder, working with AI coding agents, rebuilt a long-serving Windows estimating tool as a web app and a .NET MAUI app that share one codebase and one source of truth. It runs on the desk, in the truck and at the job site, signal or not.



PRODUCT	BUILT BY	PLATFORMS	EDITION
BidGlory	ProjectThunder.com, Inc.	Web (live) · iOS & Android (in testing)	November 2026

AT A GLANCE

Six months, one codebase, four ways to reach the same bid.

BidGlory is estimating and proposal software for low-voltage, AV, security and electrical contractors. It is the successor to BidMagic, a Windows desktop application that contractors have used to price jobs for more than fifteen years. The brief was to bring that product to every device a contractor carries without losing the pricing behaviour their numbers depend on.

1,177
commits to the main line between 20 April and 30 October 2026

69
routable screens, built from 143 Razor components, most of them shared by web and device

315
server operations, each with a written authorization, tenant and idempotency rule

76
versioned database migrations, applied by the app's own upgrader

4
decimal places on unit prices, matching BidMagic exactly so totals reconcile to the cent

0
acknowledged edits held only in memory: a saved change on a device is on disk before the screen says "saved"

What we delivered

- A Blazor Server web app, live on bidglory.net as a technology preview
- A .NET MAUI Blazor Hybrid app for iPad and iPhone (TestFlight) and Android phones and tablets (Google Play closed testing)
- Offline editing on device with a durable outbox, automatic sync and per-user conflict resolution
- Print-ready proposals generated from Word templates on Linux servers
- A customer portal where homeowners and facility managers approve and sign from any device
- Accounting links: ERPNext tested end to end; QuickBooks Online and Desktop built

Contents

01 The product and the problem 3
02 One codebase, many heads 4
03 Where it runs today 5
04 At the desk: the web app 6
05 On the tablet: an electrical bid 8
06 In the pocket: iPhone and Android 10
07 The proposal 11
08 Signal or not: offline and sync 12
09 Design system 14
10 Proof of engineering 15
11 Humans and AI: how it was built 16
12 QA when the developer is an AI 17
13 Results and what's next 18

A trusted desktop tool, stuck on the desk.

BidMagic earned its customers by getting the numbers right. Contractors build a bid room by room, pick products from a catalog, apply labor and markup, and print a proposal the customer signs. All of that happened on one Windows PC in the office.

The work, though, starts at the job site. An estimator walks the house or the building, counts devices, notes the panel, sketches where the equipment goes, then drives back to the office to type it all in. Every handoff is a chance to drop a line item, and every day in the truck is a day the competitor's bid lands first.

We wanted one product our customers could open anywhere: the office browser, the iPad on the walkthrough, the phone in the truck, and the Android tablets many field crews already carry. It had to keep working in a basement with no signal. Most of all, every total had to come out exactly as it would in BidMagic, because contractors compare the two side by side before they trust a new tool with their margins.

The constraints we were handed

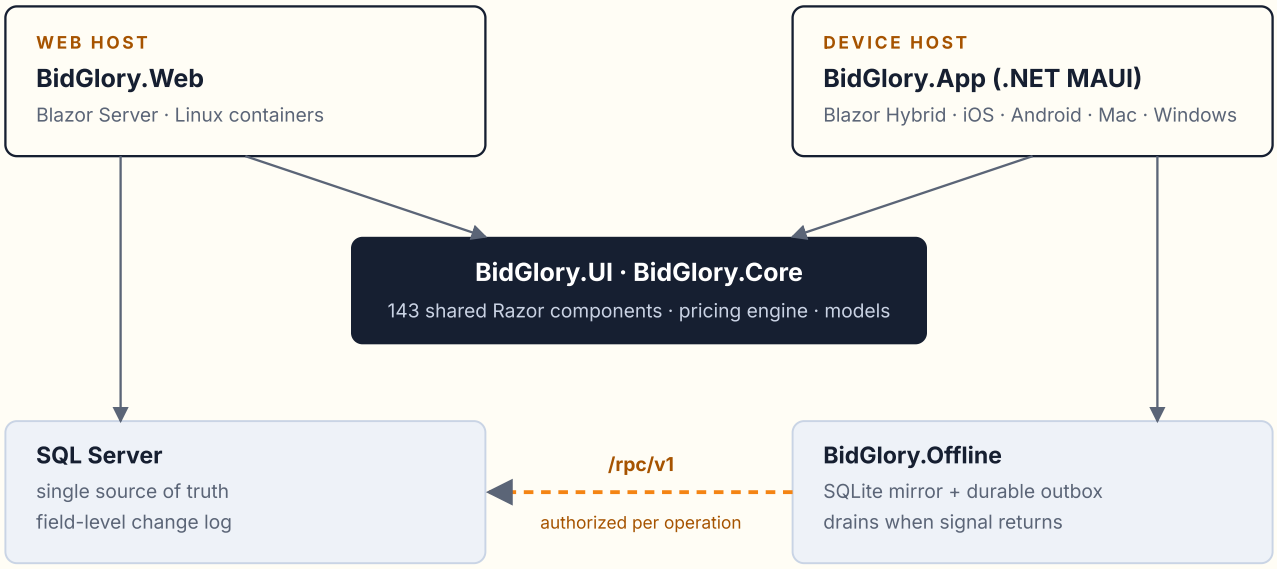
- **Pricing parity.** BidMagic's rules are the specification: four-decimal unit prices, discount applied outside the design base, labor phases under their legacy names.
- **Existing templates.** Thirty-six Word proposal templates customers already send must keep printing.
- **Field reality.** Tablets and phones lose signal; edits made there cannot be lost or silently overwritten.
- **Small team.** One product team cannot maintain separate web, iOS, Android and desktop codebases.
- **Customer trust.** Pricing, cost and customer lists are the contractor's crown jewels; access rules must hold on every device.

"Bid it at the desk. Close it from anywhere." The brief in one line, and the test every screen in this study was held to.

02 · ARCHITECTURE

One codebase, many heads.

The decision that made the rest possible: every screen is written once, as a Razor component, and rendered by two different hosts. In the browser it runs on the server through Blazor. On a device it runs natively inside a .NET MAUI app through Blazor Hybrid.



Why Blazor Hybrid, not a web view

The first iPad build simply wrapped the website in a web view. It was fast to ship, and it was useless in a basement. The production app instead renders the same components natively on the device and talks to the server through a narrow, versioned RPC surface. The device is a real client, so it can work offline, remember a user's column choices, and enforce its own rules.

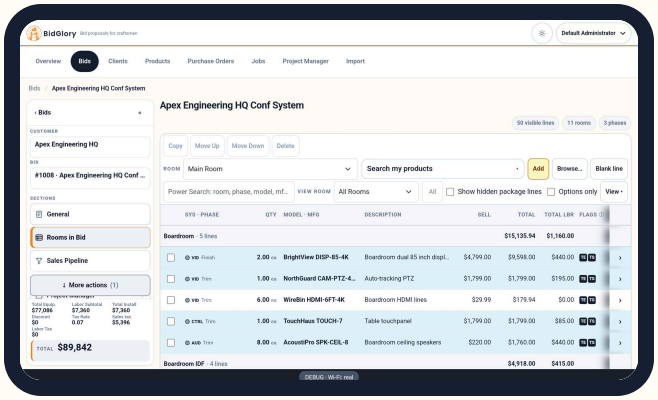
What stays shared, what does not

Pricing, validation, the bid grid, product search and BidSketch are written once. Each host adds only what is genuinely different: authentication (cookies on the web, a device token held in the platform's secure store), data access (direct repository on the server, store-first client on the device), and platform touches like the Android back gesture closing a sheet before it leaves the page.

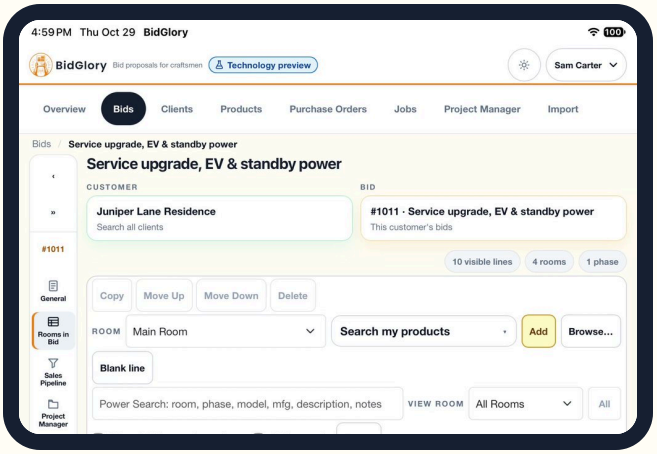
Every platform, stated as it actually stands.

A case study is only useful if its claims are checkable. Here is each platform, its status on 1 November 2026, and how it was verified.

PLATFORM	STATUS	HOW IT RUNS AND HOW WE VERIFIED IT
Mac and Windows desktops Safari, Chrome, Edge, Firefox	LIVE · PREVIEW	The full web app on bidglory.net. Every release ships through the continuous-integration pipeline, from the same main line the apps are built from.
iPad iPadOS 15 and later	TESTFLIGHT	Native .NET MAUI app, release-built and ahead-of-time compiled. Verified on iPad hardware and on the iPadOS 27 simulator, and in TestFlight with internal testers.
iPhone iOS 15 and later	TESTFLIGHT	Same app and same binary as the iPad, laid out for the narrow screen with a bid menu and a persistent total bar.
Android phones and tablets Android 7.0 (API 24) and later	PLAY CLOSED TEST	Same app, built as a signed Android App Bundle targeting API 36. Verified on a Samsung Galaxy Tab S11 Ultra and on phone emulators, including edge-to-edge display, cut-outs, landscape and dark mode.
Native Mac and Windows apps Mac Catalyst · WinUI	NEXT	The MAUI project already targets both. They are not yet released, because the browser covers the desk today. When they ship, they bring offline editing to laptops in the field.



Android tablet. A bid on a Galaxy Tab S11 Ultra in landscape, with sections down the side and the room grid filling the screen.

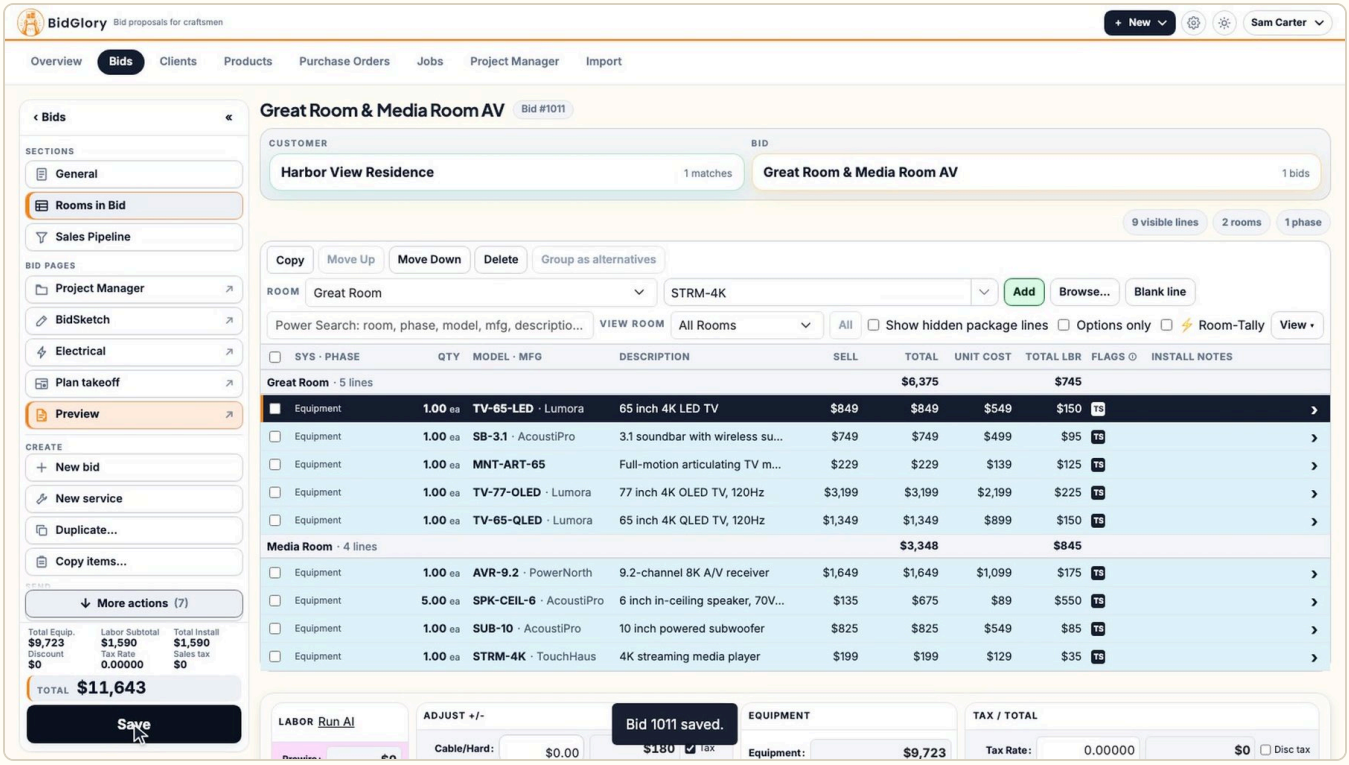


iPad. The same electrical bid from the web walkthrough, opened in the native app and grouped by room.

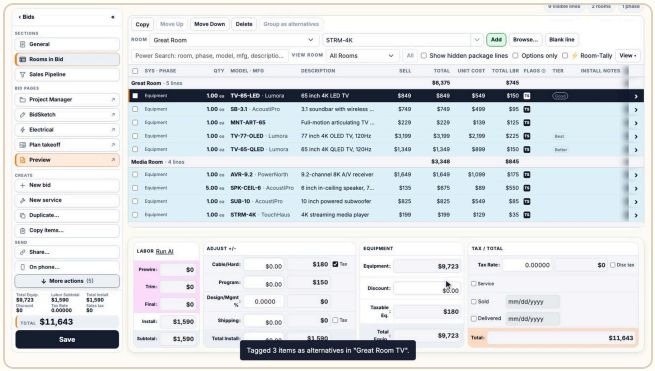
04 · AT THE DESK

A bid built like an estimator thinks: room by room.

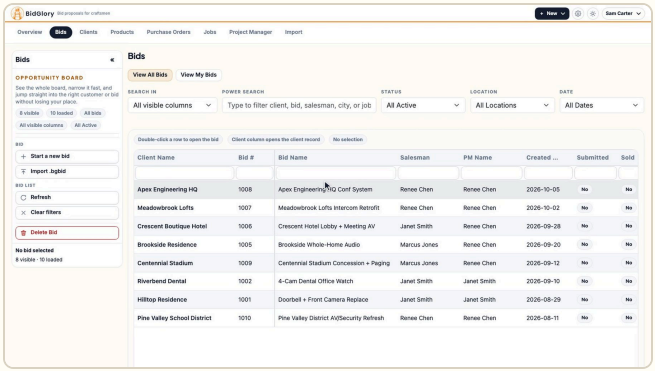
The web app is where most bids start. The grid is dense on purpose, because estimators read it like a spreadsheet. Everything around it is quiet so the numbers stay legible.



Bid workspace. Sections sit in the sidebar; rooms group the lines; sell, cost, labor and install notes sit in one row. The totals panel below updates on every keystroke, with no save-and-wait.

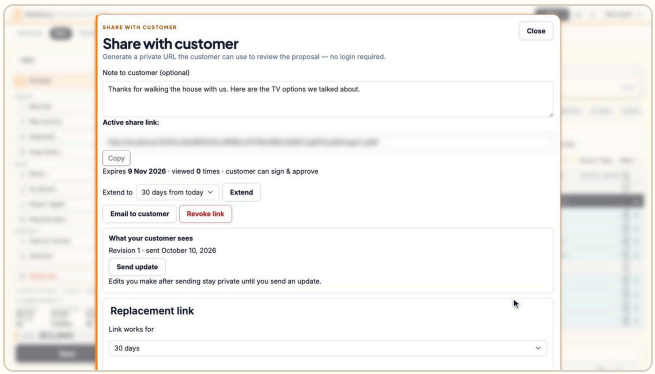


Good / Better / Best. Tag lines as alternatives and the proposal prints them as choices for the customer. Only the picked tier counts toward the total.

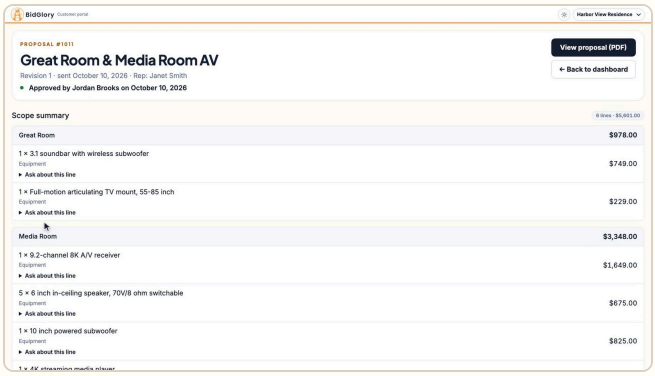


Bid list. Every bid with its stage, salesperson and total, searchable by number, client, city or name.

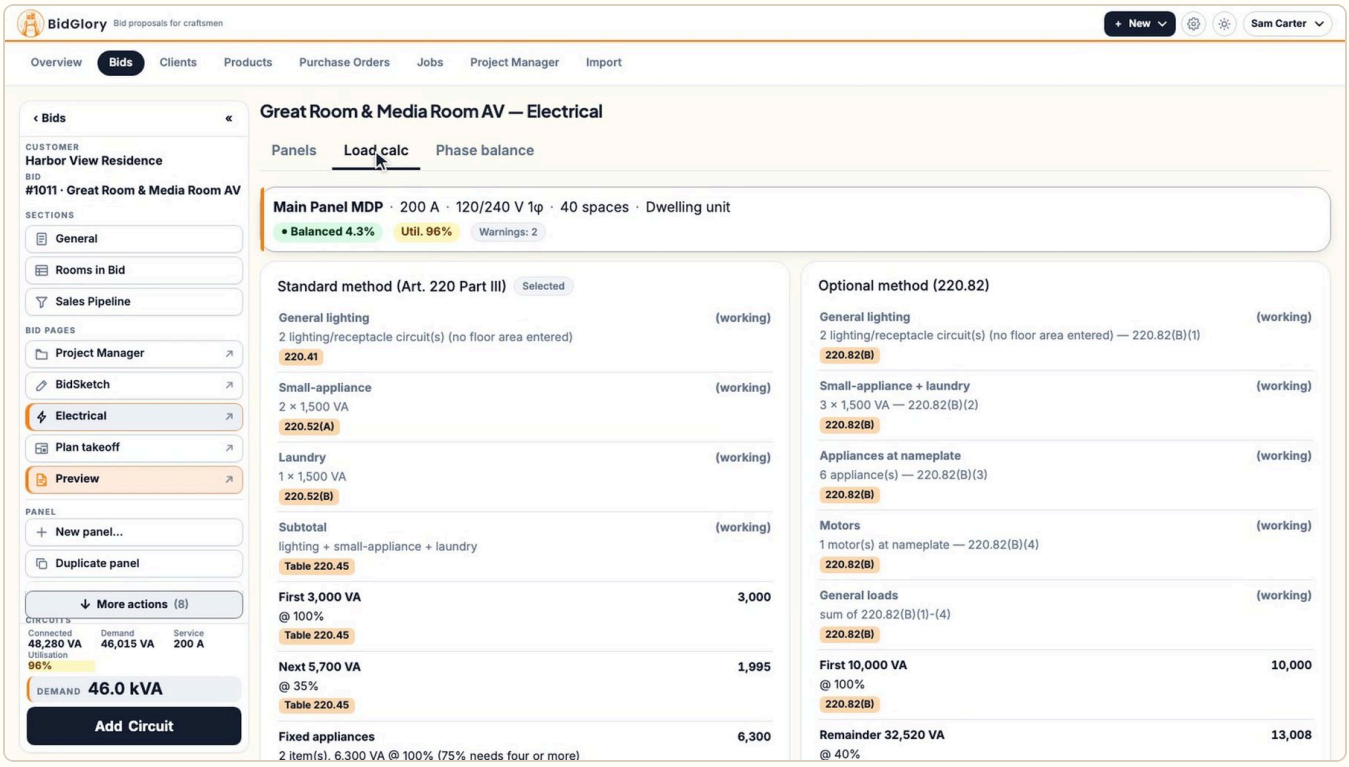
From bid to signature without leaving the browser.



Share with customer. A private link that expires, with an optional note. The estimator chooses whether the customer can sign or only view, so an architect or family member can see the proposal without approving it.



Customer portal. The customer reads the scope room by room, asks about any line, and approves. The approval is recorded with signer and time on the bid.



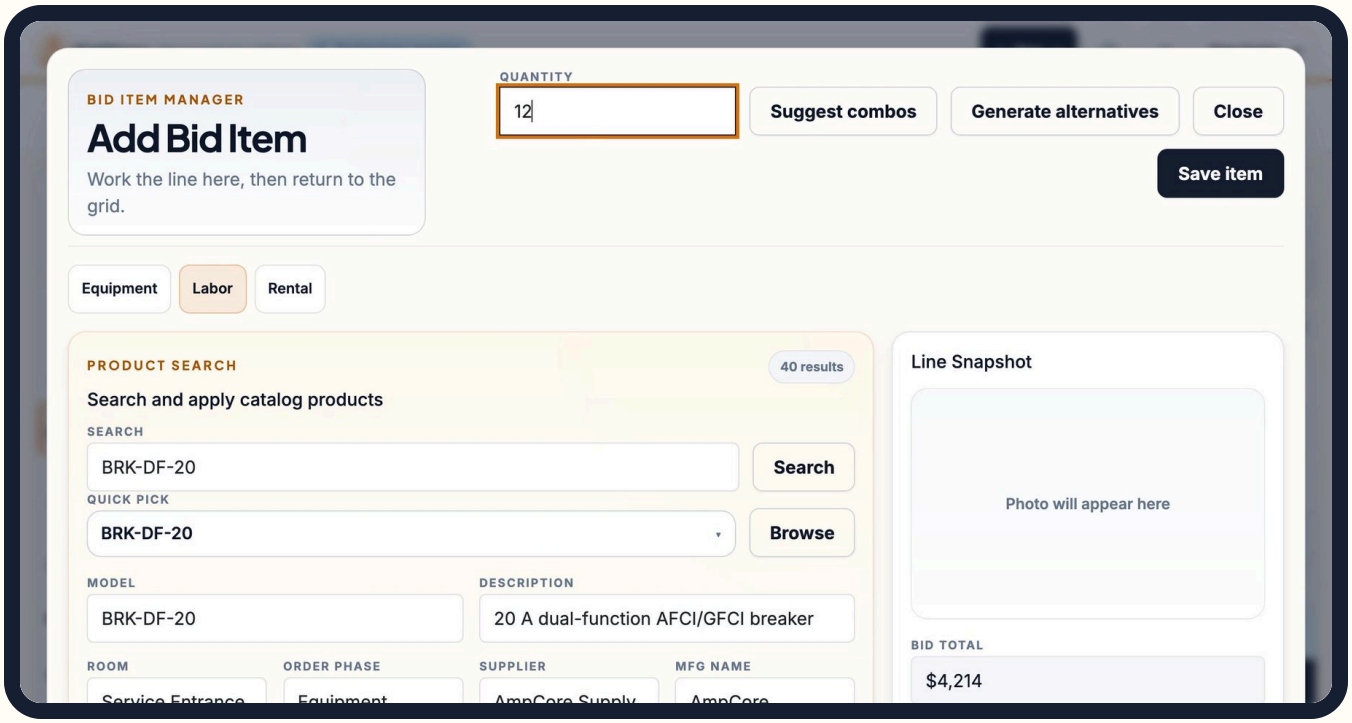
Electrical load calculation. The NEC Article 220 standard and optional methods, computed side by side from the panel's circuits, with the code reference beside every figure. A calculation aid for the estimator; final figures are verified with the authority having jurisdiction (AHJ).

> All bids, clients and products in this study are synthetic demonstration data. No customer's information appears in any screenshot.

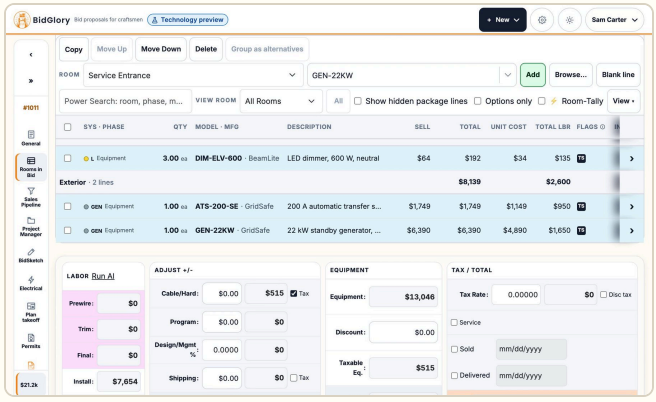
05 · ON THE TABLET

An electrical bid, start to finish, at tablet size.

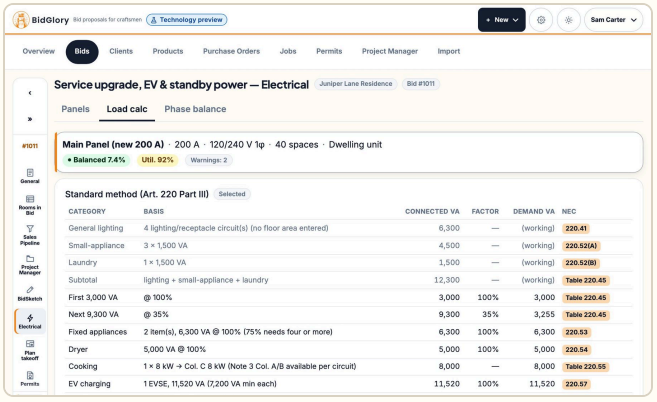
The scenario: a residential service upgrade with an EV charger and a standby generator. The estimator builds it on a tablet, the way they would at the kitchen table with the homeowner.



Adding a line. Search the catalog, set quantity and labor, and see the line's total before saving. The editor is a sheet over the grid, so the bid is never out of view.



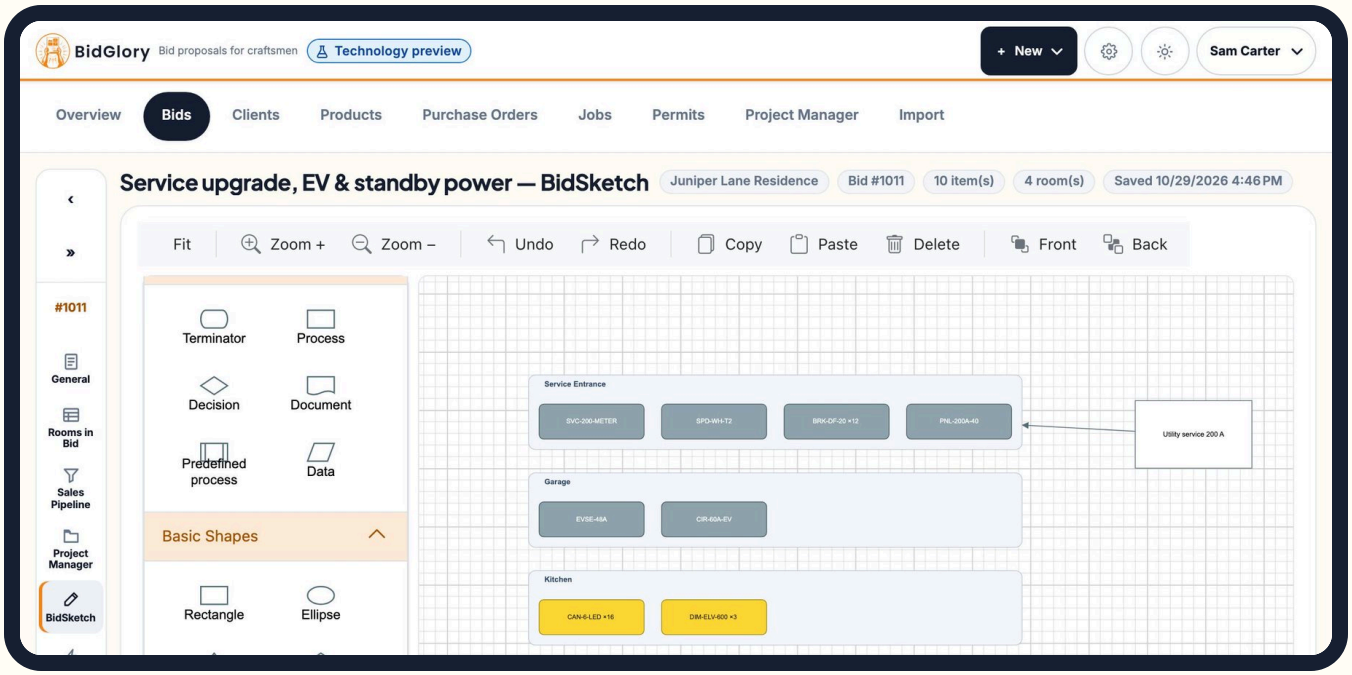
Live totals. Equipment, labor, tax and grand total recalculate as each room fills in.



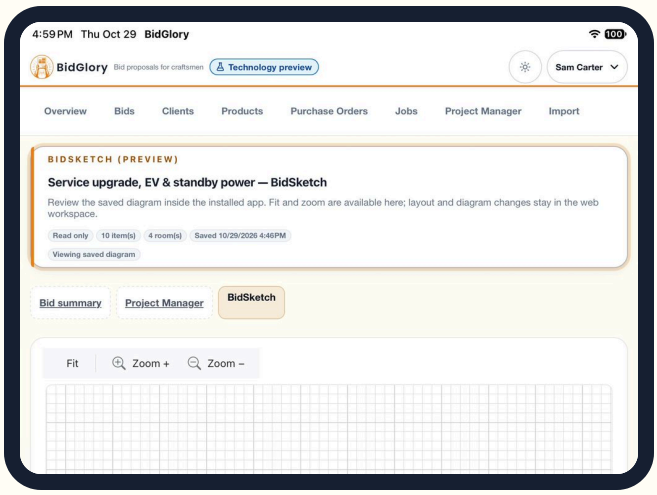
Panel schedule. A 200 A panel's circuits, priced lines and the load calculation (a calculation aid, verified with the AHJ), in one place.

Sketch the system. Print the sketch.

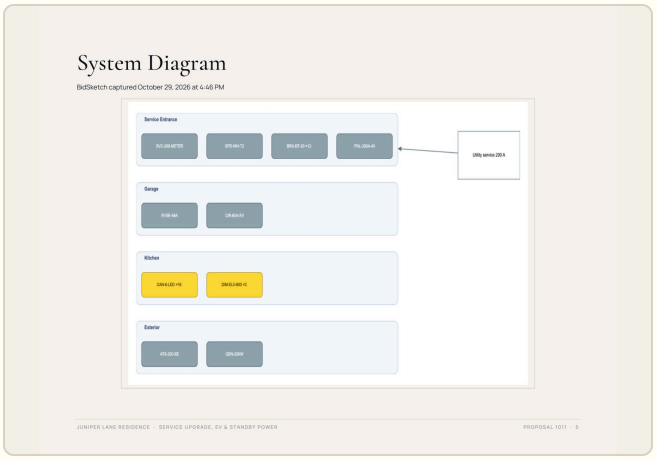
BidSketch is a system diagram built from the bid's own lines. Drag equipment onto the canvas, connect it, save. The same diagram opens on the iPad and prints as a landscape page in the proposal.



BidSketch on the web, at tablet size. Shapes and connectors on the left; the bid's equipment on the canvas, labelled by room. A "Diagram saved" confirmation tells the estimator the sketch is stored with the bid.



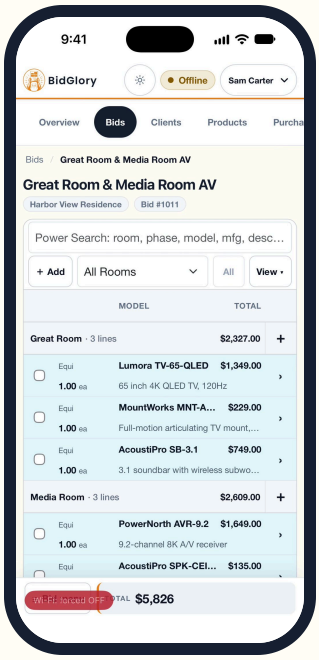
Same sketch, native iPad app. Saved on the web, opened on the device. One record, no export step.



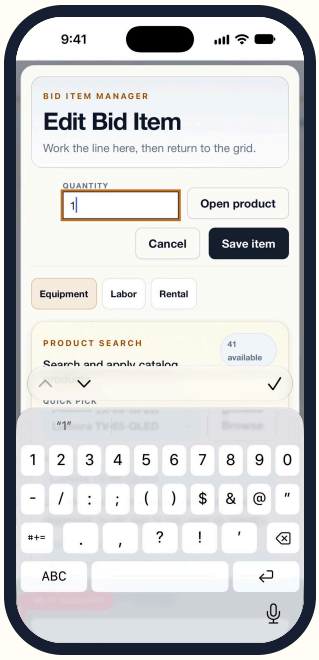
Printed. The proposal's System Diagram page, generated from the saved sketch.

The same bid on a phone, iOS or Android.

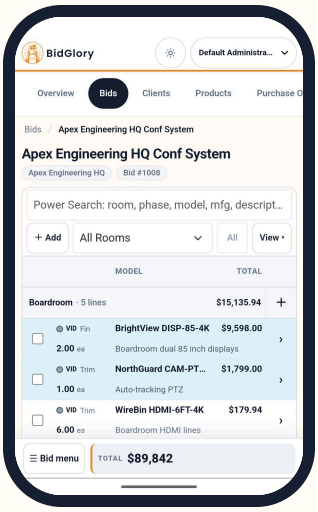
On a narrow screen the sidebar folds into a bid menu and the total pins to the bottom edge. The rows reflow to two lines. Nothing is a separate "mobile site": it is the same component, laid out for the space it has.



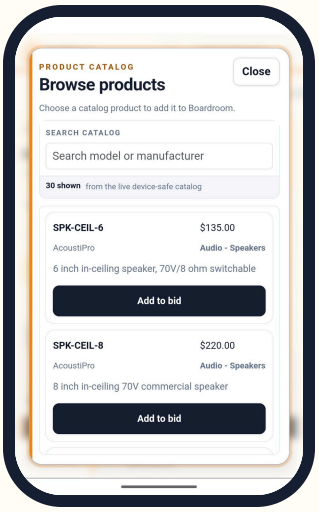
iPhone. Bid open, offline chip in the header.



iPhone. Editing a line with the numeric keypad.



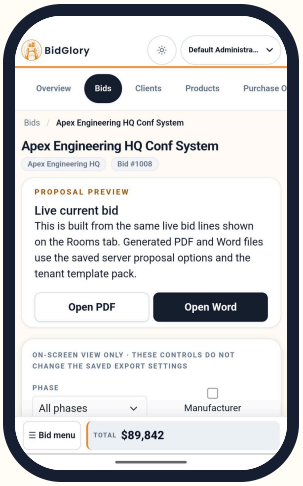
Android. Rooms, lines and a pinned total.



Android. Adding from the catalog.

Platform manners, not lowest common denominator

- Android's back gesture closes an open sheet before it leaves the page.
- Content runs edge to edge on current Android and respects camera cut-outs in both orientations.
- Confirmation dialogs stay above the iPad keyboard instead of hiding behind it.
- Column choices are remembered per device through the platform's own preferences store.



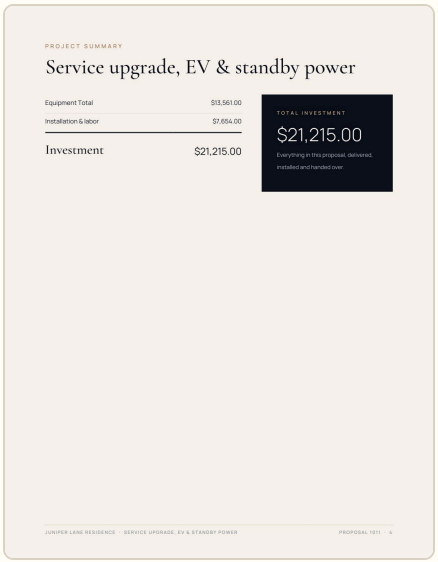
Android. Proposal preview, opening the generated PDF or Word file from the live bid.

The document the customer actually signs.

Proposals are merged from Word templates on the server and rendered to PDF. That happens on Linux containers, with no Microsoft Office anywhere in the stack. The thirty-six templates BidMagic customers already use print unchanged; a new template family is in development.



Cover.



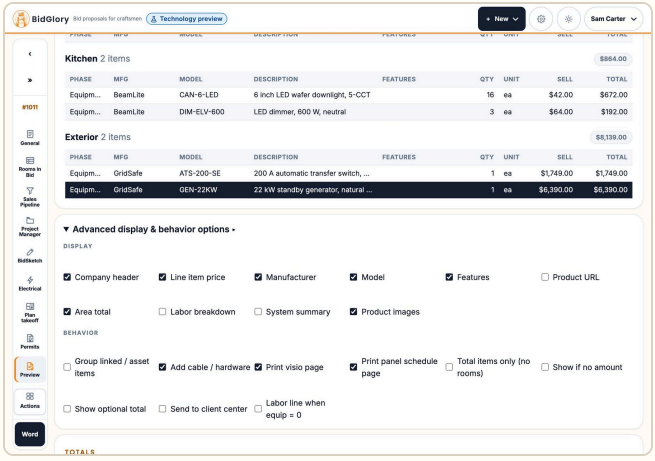
Scope by room.



Panel schedule and load calc.

What goes in, chosen per bid

Line items or room totals; manufacturer and model or plain descriptions; labor shown or folded in; optional Good/Better/Best tiers; the BidSketch diagram; the panel schedule and load calculation; signature and payment terms. Print options live with the bid, so a re-print a month later matches the original.



Print options. Advanced display settings for the proposal, saved with the bid.

08 · SIGNAL OR NOT

Offline is a design, not a fallback.

We wrote the sync model down before writing any sync code. A sync model decided while coding is one nobody can reason about later. The design rests on two invariants that every later change was tested against.

INVARIANT 1

No acknowledged edit exists only in memory.

On the device, a save writes to a local SQLite mirror *and* appends to a durable outbox in one transaction. The screen says "saved" only after that commit. If the app is killed mid-bid, the work is already on disk.

INVARIANT 2

The device can never delete what it did not see.

The outbox carries small, specific operations, never "here is the whole list." A queued change replayed hours later cannot wipe out lines the office added in the meantime. Each operation is re-authorized, re-priced and re-bound by the server when it lands.



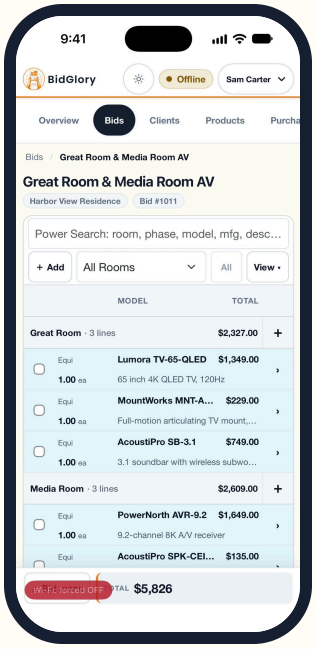
Details that make it trustworthy

- **Idempotency keys** survive retries and app restarts, so a change sent twice lands once.
- **Coalescing**: five offline edits to one line become one queued update, never five round trips.
- **Optimistic concurrency** through row versions: if the office changed the same line, the device is told, not overwritten.
- **Default-deny queue**: an operation can be queued offline only if the authorization manifest marks it safe, and that list is generated, not hand-edited.

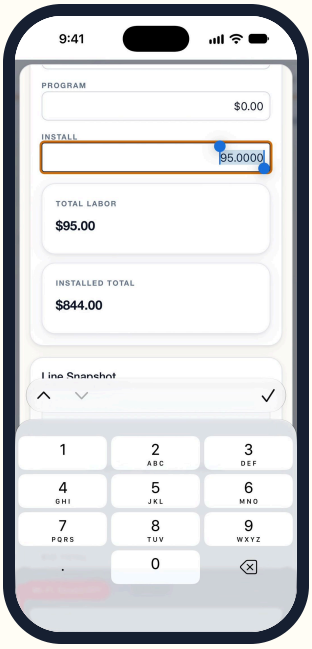
Between devices

There is one source of truth, the server. The office browser on a Mac or Windows PC, the iPad on site and the Android phone in the truck all read and write the same bid. A device that was offline catches up the moment it reconnects. Every field the office changes is written to an append-only change log with before and after values. That log is the foundation for field-level merge.

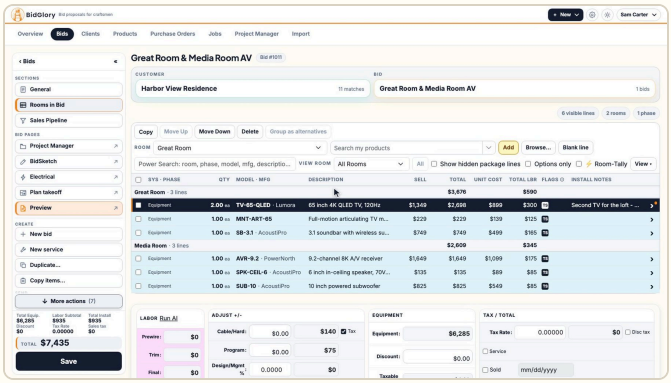
Edit in the basement. The office sees it on reconnect.



1 · No signal. The header chip switches to Offline. The bid stays open and editable.



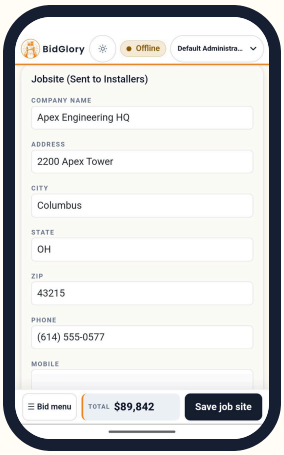
2 · Keep editing. Changes are saved to the device's outbox.



3 · Back online. The outbox drains on its own, the chip clears, and the office's browser shows the device's change with the server's re-priced total. No "sync" button to forget.

How we proved it

To test without driving to a basement, we built a debug-only "airplane mode" for the simulator, compiled out of release builds entirely. Lifting it drains the queue exactly as walking back into coverage does. That is why the device frames above show a test chip at the bottom edge. Running the real app through that switch caught two defects no unit test had: an editor that silently opened read-only on device, and a sign-in session the offline engine could not see. Both were fixed and are now pinned by assertions that we confirmed fail when the fix is removed.

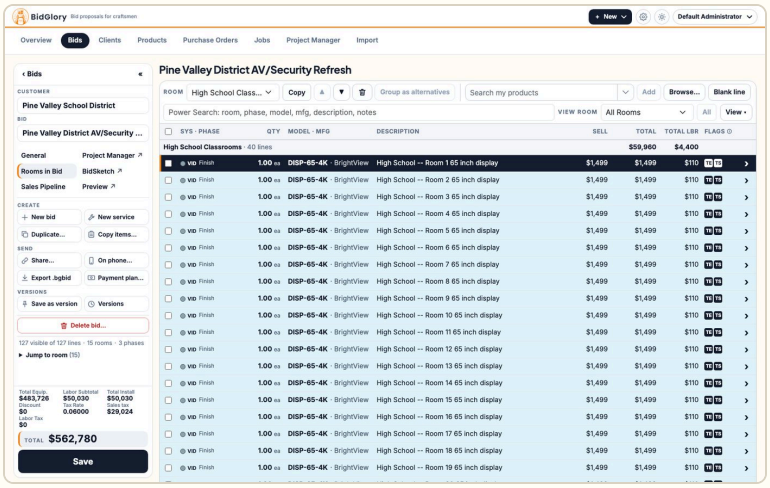


Android, offline. Saving the job site with no connection. It queues and syncs like every other change.

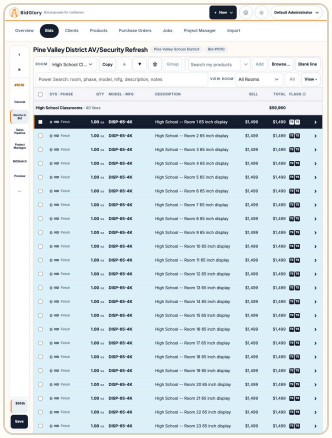
Structure first, paint second.

The interface is built from the brand's own colors: a deep navy that carries every action, a warm paper ground, and orange used only as an accent. Text is never set in orange on light grounds, because bright orange fails accessibility contrast; a deeper burnt orange carries the labels instead.

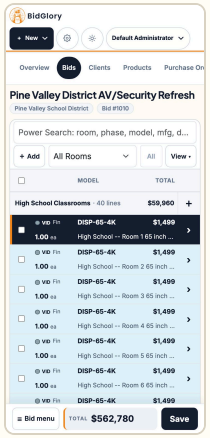
#161F31	#FFFDF5	#F48413	#A65300	#8FA3C7
Navy · actions, ink	Paper · ground	Orange · accent only	Burnt orange · text on light	Slate · tints in dark mode



Desktop, 1440 px. Sidebar and grid side by side.



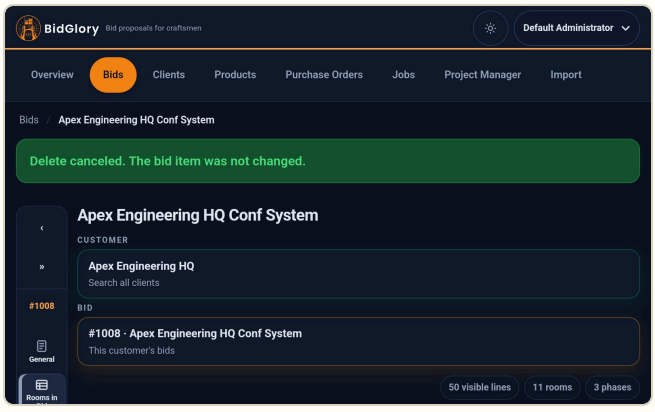
iPad portrait. Sidebar collapses to icons.



Phone. Bid menu, pinned total.

Light and dark, following the device

BidGlory follows the operating system's appearance by default and only remembers a choice the user makes. Dark mode is not an inversion: tints are recomputed from slate rather than mixing orange into navy, which turns brown. Focus rings are tuned to clear 3:1 contrast on both grounds.



Dark mode on an Android tablet. The same bid, the same layout, re-tinted for the dark ground.

Checks that can fail, and do.

Quality on a product like this is not a claim; it is a set of gates the build has to pass. These are the ones that matter most, and the reasoning behind each.

1 A written rule for every server operation

All 315 operations the apps can call are listed in an authorization manifest. Each entry states who may call it, which tenant's data it may touch, whether it is idempotent, and whether it may be queued offline. Code generation reads that manifest and refuses anything not marked ready. A new device route cannot appear without someone deciding, in writing, that it is safe.

2 Writes bound to their parent, mechanically

Reviews kept finding one class of mistake: an update that could re-point a child row at someone else's bid. After the third review found the next instance, we stopped fixing instances and built a CI check. Any write reachable from a device must be provably bound to a live parent, or the build fails.

3 Tests we have watched fail

A green test proves nothing until it has gone red. New guards are verified by deleting the condition they protect and confirming the suite catches it. One such run turned up nine failures that a passing suite had been hiding.

4 Release builds, verified before upload

Debug builds neither trim nor precompile, so they can pass while the shipped app crashes. An early TestFlight build did exactly that. Since then the upload script forces a clean release build, and every build is run in release mode on a simulator before it goes to testers.

5 Pricing parity, pinned

BidMagic is the specification. Unit prices carry four decimals and discounts apply outside the design base, exactly as BidMagic does. Golden-file tests render reference proposals and fail on any change to a single figure.

6 Every screen, every theme, scanned

For the brand refresh, an automated sweep opened every route in light, dark and system-dark modes and checked color contrast with axe. It also read computed styles to catch stray off-brand colors that a source-code search would miss.

95% of the code was written by AI. 100% of the decisions were made by people.

BidGlory was built by a small human team directing AI coding agents. The people set the product, made every judgment call and owned every release. The agents wrote, refactored, tested and documented the code. The split below comes straight from the repository's history.

LINES OF CODE COMMITTED, 20 APRIL – 30 OCTOBER 2026



Human developers · 4.9%

about 691,000 lines added across code, tests and scripts

1,018
of 1,177 commits carry an AI co-author signature

5
AI model generations used, upgraded mid-project with no rewrite

103
commits on the busiest single day, 17 October

Counted from git: lines added to source, test, script and SQL files, excluding generated files and recorded baselines. Large refactors count the lines they moved, and those moves were also done by agents.

What the people did

- **Product calls.** "BidMagic is the specification." Four-decimal prices, how copied lines behave, which screens go native first.
- **Sign-in and real devices.** Agents never type credentials. A person signs in on the iPad, the Galaxy Tab and TestFlight.
- **Every release.** Pushes to the main line, store uploads and production deploys need a human yes.
- **Guardrails.** An agent cannot widen what a device may call, or change a recorded build baseline, without explicit approval.
- **Testing in the field.** Human testers' notes become structured findings the agents work through.

Many agents, one codebase

Work runs in parallel lanes: several agent sessions at once, each in its own git worktree with its own throwaway database, so one lane's half-finished change never touches another's. A shared coordination log records who owns what, which review is pending and which decisions are final. It is the agents' stand-up meeting, in writing.

Big moves become cheap

On 22 August an agent split a 46,566-line test file and a 24,000-line data repository into focused parts in one day, with the full suite green after each step. That is the kind of restructuring teams usually postpone indefinitely. Here it was one day's work, which is why the codebase could stay tidy while it grew this fast.

The writer and the reviewer are never the same model.

An AI that checks its own work agrees with itself. So BidGlory's code is written by one AI and audited by another from a different vendor, at agreed phase gates, with a written GO or BLOCK. Everything else is designed on the assumption that a confident answer can still be wrong.

1 Adversarial review at the gate

At the end of each phase the writing agent sends one package: the commit range, how each acceptance criterion is met, device evidence and known gaps. The reviewing agent tries to break it. Blockers come back as a batch, are fixed, and are re-reviewed.

2 Findings become checks

Review finds instances; it cannot find the next one. When the same kind of defect turned up in three consecutive reviews, the agents stopped fixing instances and built a CI check for the whole class. The build now rejects it mechanically.

3 Every test is broken on purpose once

A test an AI wrote and saw pass proves little. Each new guard is verified by removing the code it protects and confirming the suite fails, then putting the code back.

4 Agents drive the real app

Agents run the web app in a browser, the iOS app in a simulator and the Android app in an emulator, and screenshot every screen in light and dark. Each test run gets a fresh throwaway database seeded with synthetic data, never customer data.

5 Demo videos are test runs

The product videos are recorded by agents driving the real app. Filming the electrical bid in this study surfaced 13 defects. They were filed as tickets with repro steps, fixed and re-shot the same week.

6 Everything lands in the backlog

Agents file their own bugs into Azure DevOps with reproduction steps, link the commit that fixes each one and close it when the fix ships. Nothing important lives only in a chat window.

179
bugs filed since April, 158 closed (88%)

566
test files; about 31% of all code written is test code

726
backlog items and tasks planned and tracked in Azure DevOps

Where the product stands, and where it goes.

Delivered

- A web app live on bidglory.net as a technology preview, deployed continuously from the main line.
- One MAUI app in testing on iPad, iPhone and Android, rendering the same components as the web.
- Offline editing of bid lines and job sites, with automatic sync and conflict resolution.
- Electrical tooling: panel schedules, NEC Article 220 load calculation aids (verified with the AHJ) and BidSketch diagrams, all printed in the proposal.
- A customer portal for review, questions and signature from any device.
- ERPNext accounting integration passing its full 45-check suite against a running ERPNext server.

Next, in order

- **Field-level merge** for bids, lines and products, built on the change log already recording every edit.
- **Native Mac and Windows apps** from the existing MAUI project, bringing offline to field laptops.
- **Plan takeoff:** today estimators place markers by room; measured, to-scale takeoff on the plan is next.
- **QuickBooks** verification against real company files before it is offered to customers.
- **Public store release** on the App Store and Google Play after testing closes.

What this project says about how we work

Write the model down before the code. State every platform's status as it is, not as it will be. Turn every recurring review finding into a check that fails the build. Keep the legacy product's behavior as the specification until its users decide otherwise. None of this is exotic. It is what lets a small team ship one product to every screen without the numbers drifting between them.



The finish line. The customer's view of an approved proposal, readable on any device.

BidGlory
PROPOSAL & PROJECT MANAGEMENT SOFTWARE

See it for yourself.

BidGlory is available as a technology preview at **bidglory.com**. To discuss a cross-platform build of your own (web, iOS, Android, Mac and Windows from one .NET codebase), talk to the team that built it.

ProjectThunder.com, Inc.
Product engineering · .NET MAUI · Blazor · cloud and on-premises delivery
projectthunder.com

Screens show synthetic demonstration data. Platform statuses are as of 1 November 2026. BidGlory is in technology preview; features and availability may change. Apple, iPad, iPhone, macOS and TestFlight are trademarks of Apple Inc. Android and Google Play are trademarks of Google LLC. Windows is a trademark of Microsoft Corporation.