FROM IDEA TO APP

Build practical beginner apps with ChatGPT and Floot

Independent workbook | 20 focused lessons | 12 guided practice projects

Revised edition • November 2026

About this guide and how to use it

This is an independent educational workbook, not official documentation from OpenAI or Floot. It teaches planning, prompting, prototyping and checking app ideas. The twelve project briefs are practice exercises, not independently built or verified applications. No screenshots or finished sample apps are promised.

Copyright © 2026. All rights reserved. This copy is licensed to the original purchaser for personal use. Do not redistribute or resell the PDF. This edition does not include individualized technical support or a promise of future updates. For platform assistance, use official documentation and support channels.

Prices, features, limits and interfaces can change. Check official docs before relying on a specific capability. Use fictional records and never put passwords, payment credentials or sensitive client details into AI prompts.

Affiliate disclosure: Some links may earn the publisher a commission at no extra cost to you. Floot referral: https://floot.com/r/N3VJF3. ChatGPT: https://chatgpt.com.

Contents

About this guide · · · 2
Contents · · · 2
Lesson 01: Start with the outcome, not the technology · · · 4
Lesson 02: What ChatGPT and Floot each do · · · 4
Lesson 03: Set up your accounts safely · · · 5
Lesson 04: Write your first app brief · · · 5
Lesson 05: Choose the smallest useful version · · · 6
Lesson 06: Design screens before building · · · 6
Lesson 07: Prepare useful sample data · · · 7
Lesson 08: Build the first screen in Floot · · · 8
Lesson 09: Make forms and data work · · · 8
Lesson 10: Understand logins and privacy · · · 9
Lesson 11: Use AI features only where helpful · · · 9
Lesson 12: Fix bugs systematically · · · 10
Lesson 13: Test on a real phone · · · 10
Lesson 14: Improve the interface without breaking it · · · 11
Lesson 15: Prepare a launch checklist · · · 12
Lesson 16: Make a lead-magnet website · · · 12
Lesson 17: Sell a paid digital edition · · · 13
Lesson 18: Promote with short videos · · · 13
Lesson 19: Maintain and improve your app · · · 14
Lesson 20: Your 30-day app-building plan · · · 15
Project 01: Service request tracker · · · 16

Project 02: Opt-in lead organizer · · · 16
Project 03: Appointment inquiry board · · · 17
Project 04: Home maintenance planner · · · 18
Project 05: Content idea library · · · 19
Project 06: Digital product catalog · · · 20
Project 07: Project expense log · · · 21
Project 08: Event RSVP manager · · · 22
Project 09: Inventory count board · · · 23
Project 10: Client feedback collector · · · 24
Project 11: Study session planner · · · 25
Project 12: FAQ publishing board · · · 26
Reusable templates · · · 28
Platform notes and references · · · 28
Launch and delivery checklist · · · 29

LESSON 01 Start with the outcome, not the technology

What you need to know

A successful first app solves one clearly described problem for one group of people. Before choosing features, write a sentence that starts: “This app helps [person] do [task] without [frustration].” Keep it visible as you build.

Example

A local service owner needs a simple way to record customer requests. A first version might collect a name, service type, preferred date, and status. It does not need payments, artificial intelligence, or a mobile app store listing.

Your action

Write three problems you know from your own work or hobbies. For each, name the user, the recurring task, and how they solve it today. Circle the problem you can demonstrate with five realistic examples.

Prompt to adapt

> Act as a patient product coach. Ask me one question at a time to turn this idea into a one-sentence problem statement, a target user, and a measurable outcome. Do not suggest features until the problem is clear.

Evidence-based checkpoint

State one user, one task and one observable improvement. Reject features that do not support that outcome.

LESSON 02 What ChatGPT and Floot each do

What you need to know

ChatGPT is useful for brainstorming, planning, writing specifications, reviewing copy, creating test cases, and interpreting errors. Floot is an app-building environment where you can turn a description into a working interface and application logic. Neither tool replaces testing or careful review.

Example

Think of ChatGPT as your planning partner and Floot as your workshop. Move between them deliberately: plan a small slice, build it, test it, describe what happened, then revise.

Your action

Create two columns titled “Planning with ChatGPT” and “Building in Floot.” Put each task in the correct column. Include privacy, publishing, and quality checks as shared responsibilities.

Prompt to adapt

> Explain my app idea in plain English. Produce a Floot-ready build brief with user roles, screens, data fields, actions, validation, and acceptance tests. Flag assumptions instead of inventing requirements.

Evidence-based checkpoint

List which work is planning and which requires a real app build. Mark every untested feature as untested.

LESSON 03 Set up your accounts safely

What you need to know

Use your own accounts and a password manager. Enable multi-factor authentication where available. Read current plan and usage terms before committing to a workflow. Keep private API keys out of screenshots, prompts, public repositories, and front-end code.

Example

Create a dedicated project folder for copy, screenshots, sample data, and release notes. Use fake names and addresses while testing. Store real user information only in a system designed to protect it.

Your action

Make a setup checklist: sign in, review billing, find your project dashboard, confirm account recovery, create a test project, and identify where publication settings live.

Prompt to adapt

> Create a beginner setup checklist for ChatGPT and Floot. Distinguish settings I must verify myself from steps you can explain. Include security and spending precautions.

Evidence-based checkpoint

Confirm you can access your accounts securely and locate the current plan and publishing information.

Important limitation

Never paste API keys into chat or client-side code. Use the platform secret configuration for server-side use.

LESSON 04 Write your first app brief

What you need to know

A good brief defines the user, the job, the simplest workflow, and what “done” means. Avoid “build an amazing app with everything.” Use a short document that can be checked against the result.

Example

Example: “Build a mobile-friendly request tracker for a one-person repair service. The owner can add requests, update status, search by customer, and export records. Do not add payments or social login in version one.”

Your action

Write a brief with five headings: user, problem, core actions, data, and acceptance criteria. Remove anything that is not needed to demonstrate the primary workflow.

Prompt to adapt

> Rewrite this rough idea as a concise build specification. Include three must-have features, three non-goals, and six testable acceptance criteria. Ask about any critical ambiguity.

Evidence-based checkpoint

Check that each must-have feature has a test with an expected result; remove vague promises.

LESSON 05 Choose the smallest useful version

What you need to know

The minimum viable product is not a broken or ugly prototype. It is the smallest complete experience that delivers a real benefit. Limit your first release to one primary user journey and a few screens.

Example

For a lead organizer: list contacts, add contact, edit status, and see next follow-up. Automated prospecting, email campaigns, analytics, and social integrations can wait.

Your action

Draw the user journey from opening the app to completing one task. Count the taps. Remove steps that do not help the user reach the outcome.

Prompt to adapt

> Here are my proposed features. Sort them into must-have, later, and avoid-for-now. Explain each decision in one sentence and propose a two-week MVP scope.

Evidence-based checkpoint

Demonstrate the primary user journey end to end without relying on a future feature.

LESSON 06 Design screens before building

What you need to know

Sketch the screens on paper or in a simple document. A typical small app needs a welcome screen, a main list or dashboard, an add/edit form, and a confirmation or details screen. Design for phone screens first if users will work on mobile.

Example

Use consistent buttons, labels, spacing, and feedback. A primary action should be obvious. Empty states should tell people what to do. Errors should say how to recover, not merely “something went wrong.”

Your action

Sketch four screens and label each action. For every form field, decide whether it is required, optional, or computed.

Prompt to adapt

> Act as a UX designer. Propose a four-screen mobile-first flow for my app, including screen headings, primary buttons, empty states, and accessible error messages.

Evidence-based checkpoint

Check narrow-screen readability, labels, keyboard navigation and helpful empty states.

LESSON 07 Prepare useful sample data

What you need to know

Sample records help reveal design problems before real customers use the app. Create varied cases: short and long names, missing optional fields, overdue tasks, duplicates, and unusual punctuation. Never use actual customer details in demos.

Example

For a request tracker, add records in statuses New, In Progress, Waiting, and Done. Include one record with a long note and one with no optional phone number.

Your action

Create ten fake records and use them to test sorting, search, edit, and delete. Confirm that each field behaves correctly after refreshing the app.

Prompt to adapt

> Generate ten clearly fictional test records for this schema. Include edge cases but no real personal data. Explain what each record is meant to test.

Evidence-based checkpoint

Use ten fictional records to test missing fields, long text, search and duplicates.

LESSON 08 Build the first screen in Floot

What you need to know

Start with the core screen, not every feature at once. Provide the brief and request one working vertical slice. After a change, preview the result and compare it to the acceptance criteria. Save a checkpoint before major revisions.

Example

Ask for a contact list with an Add button and an empty state. Verify it on a narrow phone viewport before adding editing, filtering, and persistence.

Your action

Build only the first screen and check: title visible, navigation understandable, text readable, buttons responsive, no content clipped on mobile.

Prompt to adapt

> Build only the main screen described below. Do not add extra features. Make it responsive, show a helpful empty state, and explain what remains mocked versus functional.

Evidence-based checkpoint

Compare the first screen to the brief and verify buttons do what their labels promise.

LESSON 09 Make forms and data work

What you need to know

A polished form is not enough: data must be validated, saved, retrieved, and edited correctly. Decide where information lives, what happens if saving fails, and who may view it. Test refreshing and reopening the app.

Example

A new request must not disappear when the browser refreshes. A blank required name should show a specific validation message. Duplicate submissions should not create accidental duplicate records.

Your action

Add one record, refresh, edit it, refresh again, then delete it. Repeat with a missing required field and a simulated slow connection.

Prompt to adapt

> Review this form workflow for missing validation, unsafe assumptions, duplicate submissions, and persistence failures. Return concrete test cases and expected outcomes.

Evidence-based checkpoint

Save, refresh, edit and delete a fictional record; confirm permissions on the backend.

Important limitation

Floot preview and published app share the project database. Preview changes can alter live records; use a separate project/database for isolated testing.

LESSON 10 Understand logins and privacy

What you need to know

Authentication establishes who someone is; authorization determines what they can access. Do not assume a hidden button protects data. Sensitive records require server-side access rules and a suitable storage system.

Example

In a lead organizer, each owner should see only their own contacts. Test this with two separate accounts and confirm one account cannot retrieve the other account’s records.

Your action

List the data your app collects. For each field, explain why it is needed, who can access it, how long it is kept, and how it can be deleted.

Prompt to adapt

> Create a privacy and authorization checklist for this app. Identify what must be enforced server-side and propose tests for cross-account access.

Evidence-based checkpoint

Record a failed test with exact reproduction steps and the smallest repair.

LESSON 11 Use AI features only where helpful

What you need to know

An app can be useful without a model call. If AI improves the main workflow, specify its inputs, outputs, costs, fallback behavior, and safety checks. Do not put provider secrets in client code.

Example

A request tracker might summarize a long customer note. The owner must still be able to read and correct the original. If the AI service is unavailable, saving the request should continue to work.

Your action

Identify one optional AI feature that saves time. Describe how to measure its value and how users can recover from a wrong answer.

Prompt to adapt

> Assess whether my app needs AI. Compare a simple rules-based feature with an AI-powered alternative, including cost, accuracy, privacy, and failure modes.

Evidence-based checkpoint

Confirm private data stays private across signed-in and signed-out sessions.

Important limitation

Sign-in proves identity; server-side authorization must still restrict which records each user can access.

LESSON 12 Fix bugs systematically

What you need to know

A bug report should say what you expected, what actually happened, the exact steps to reproduce it, and the device or browser. Change one thing at a time and retest the original problem plus nearby features.

Example

Bad report: “The save button is broken.” Better: “On a phone, tap Add, enter a name, tap Save; the spinner stops but no item appears after refresh.”

Your action

Keep a bug log with severity, steps, expected result, actual result, and status. Prioritize data loss and security before visual polish.

Prompt to adapt

> Here is a reproducible bug report. Propose the three most likely causes, the safest first diagnostic, a minimal fix plan, and regression tests. Do not claim the bug is fixed without evidence.

Evidence-based checkpoint

Test one realistic task on both a phone and a desktop, including an error path.

LESSON 13 Test on a real phone

What you need to know

Desktop previews can hide mobile problems. Test narrow screens, large text, touch targets, virtual keyboards, slow connections, and both portrait and landscape. Try a fresh browser session to expose missing assumptions.

Example

Open the app on your phone, complete the primary workflow, and record where you hesitate. Watch for fixed buttons hidden behind the keyboard and forms that are difficult to scroll.

Your action

Use a test matrix: phone/desktop, new/returning user, valid/invalid input, fast/slow network, and signed-in/signed-out state. Record pass or fail.

Prompt to adapt

> Write a practical mobile acceptance test plan for this app. Prioritize user-visible tasks and include clear pass/fail conditions.

Evidence-based checkpoint

Verify current pricing and integration requirements before adding external services.

LESSON 14 Improve the interface without breaking it

What you need to know

Visual polish comes after reliable behavior. Use a small design system: two or three colors, consistent type sizes, spacing, and button styles. Keep accessibility in mind: contrast, readable labels, keyboard focus, and descriptive errors.

Example

Improve the dashboard heading, card spacing, and form hierarchy while leaving database fields and working flows unchanged. Compare screenshots before and after.

Your action

Make a before-and-after checklist and verify all original actions still work. Avoid redesigning every screen at once.

Prompt to adapt

> Propose a visual refinement pass that preserves all working features. Give a change list, an accessibility checklist, and regression tests.

Evidence-based checkpoint

Separate prototype preview from published release and test the actual public URL.

Important limitation

Publishing is an explicit action. Code visible in preview may not be the code on the public site until you publish.

LESSON 15 Prepare a launch checklist

What you need to know

Launching means more than clicking Publish. Verify the URL, HTTPS, navigation, forms, data storage, privacy information, mobile layout, analytics choices, and a way for users to report problems. Check current hosting and plan terms.

Example

For a small app, invite three people to try the main task without coaching. Observe where they get stuck, fix critical issues, and retest before wider promotion.

Your action

Write a launch checklist and mark each item verified, unverified, or not applicable. Do not treat “looks good in preview” as a production test.

Prompt to adapt

> Build a release checklist for this project covering security, privacy, accessibility, backups, mobile usability, monitoring, and rollback.

Evidence-based checkpoint

Confirm that any shared link is appropriate for its audience and contains no private test data.

LESSON 16 Make a lead-magnet website

What you need to know

A lead-magnet page needs a clear promise, a preview of the guide, a short signup form, a working delivery path, and a privacy explanation. Collect only the information needed. Marketing consent should be separate from delivery consent.

Example

Offer a free starter guide. After a successful submission, show a thank-you page with a genuine PDF download. If an email is promised, verify the delivery service and sender configuration first.

Your action

Write the headline, three benefits, form labels, privacy note, thank-you message, and download-button text. Test the entire flow with a test address.

Prompt to adapt

> Draft a high-converting but accurate landing page for a free beginner app-building guide. Include optional marketing opt-in, a download success state, and a clear affiliate disclosure.

Evidence-based checkpoint

Draft a clear offer that describes what the app actually does, not what you hope it will do.

LESSON 17 Sell a paid digital edition

What you need to know

A paid guide should offer more than the free sample: a complete walkthrough, tested prompts, annotated screenshots, worksheets, and updates where feasible. Set expectations about what is included and what buyers must provide.

Example

A free 10–15-page sampler can lead to a paid complete guide. Start with a test price, track conversions, and adjust based on real feedback rather than promising a particular income.

Your action

Write a product description with a precise table of contents, file format, refund policy placeholder, and clear statement that tools may have separate costs.

Prompt to adapt

> Write a factual sales-page description for my paid guide. Avoid guaranteed earnings or guaranteed app success. Highlight practical deliverables and beginner outcomes.

Evidence-based checkpoint

Check the free guide promise against its actual length and contents.

Important limitation

The companion Starter Guide is seven pages in the existing edition; do not advertise it as a longer guide.

LESSON 18 Promote with short videos

What you need to know

A useful 30–45-second video has one hook, one problem, one clear benefit, a proof-oriented example, and one call to action. Use readable captions and footage that matches the narration. Do not invent product demonstrations.

Example

Show a rough app idea, a short planning prompt, a mockup, and a working tested screen. Finish with “Get the free starter guide” and a real landing-page URL once it exists.

Your action

Storyboard six scenes and match every narration sentence to a visual. Confirm music licensing, brand claims, and affiliate disclosures.

Prompt to adapt

> Create a 40-second vertical video script for this guide with a strong hook, six distinct visuals, narration, captions, and a truthful CTA. Do not invent a download URL.

Evidence-based checkpoint

Verify a visitor can understand the page, act, and recover from a broken link.

LESSON 19 Maintain and improve your app

What you need to know

After launch, listen to actual users. Track errors, support requests, and the time it takes to complete the main task. Maintain backups and update dependencies. Add features only when they address observed problems.

Example

If three users ask for filtering by status, test a simple filter before adding a complex reporting dashboard. Document changes so you can reverse a problematic release.

Your action

Create a monthly review sheet: top user problem, critical bugs, new feature requests, costs, and one improvement to ship next.

Prompt to adapt

> Turn this user feedback into a prioritized maintenance plan with effort estimates, impact, risks, and tests.

Evidence-based checkpoint

Measure only data you have permission to collect and avoid invented performance claims.

LESSON 20 Your 30-day app-building plan

What you need to know

Consistency beats ambitious scope. Spend the first week defining the problem and screens, the second building the core workflow, the third testing and repairing, and the fourth preparing a limited launch. This is a suggested practice schedule, not a guarantee.

Example

Days 1–7: research and brief. Days 8–14: first functional slice. Days 15–21: persistence, permissions, and testing. Days 22–30: polish, privacy review, release checklist, and small user trial.

Your action

At the end of each week, write what works, what is blocked, and the next smallest testable step. Keep a list of features postponed for later.

Prompt to adapt

> Create a realistic 30-day schedule for my first app with daily 30–60-minute tasks, clear deliverables, and checkpoints. Adapt it to my available time.

Evidence-based checkpoint

Create a release log with version, tested features, known issues and rollback plan.

PROJECT 01 Service request tracker

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A repair shop owner. Build the smallest useful experience for this person.

Data to model

requester, issue, priority, status

Example fictional record

Mara | mower will not start | High | New

Distinct build walkthrough

1. Define a form for requester and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Owner opens New requests, edits Mara to In Progress, then sees the updated row.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Reject an empty issue; filter by status; refresh and confirm the update.

5. Review privacy and access: Owner-only records; no public request list.

Copy-ready build request

> Create a mobile-friendly service request tracker for a repair shop owner. Store these fields: requester, issue, priority, status. Support this precise journey: Owner opens New requests, edits Mara to In Progress, then sees the updated row. Start with fictional sample data: Mara | mower will not start | High | New. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Reject an empty issue; filter by status; refresh and confirm the update.

Acceptance record

Expected: Reject an empty issue; filter by status; refresh and confirm the update. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Owner-only records; no public request list. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 02 Opt-in lead organizer

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A solo newsletter publisher. Build the smallest useful experience for this person.

Data to model

name, email, source, consent_at, next_action

Example fictional record

Avery | avery@example.test | ebook | 2026-11-01 | 2026-11-14

Distinct build walkthrough

1. Define a form for name and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Add Avery with recorded permission; find leads due this week; mark a follow-up done.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Reject malformed email; define duplicate email handling; confirm consent timestamp persists.

5. Review privacy and access: Consent is not implied by downloading a guide.

Copy-ready build request

> Create a mobile-friendly opt-in lead organizer for a solo newsletter publisher. Store these fields: name, email, source, consent_at, next_action. Support this precise journey: Add Avery with recorded permission; find leads due this week; mark a follow-up done. Start with fictional sample data: Avery | avery@example.test | ebook | 2026-11-01 | 2026-11-14. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Reject malformed email; define duplicate email handling; confirm consent timestamp persists.

Acceptance record

Expected: Reject malformed email; define duplicate email handling; confirm consent timestamp persists. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Consent is not implied by downloading a guide. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 03 Appointment inquiry board

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A mobile service business. Build the smallest useful experience for this person.

Data to model

name, service, requested_day, note, state

Example fictional record

Noah | tune-up | Friday | after 2 pm | Requested

Distinct build walkthrough

1. Define a form for name and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Visitor sends an inquiry; owner reviews it privately and changes state to Contacted.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Show receipt without promising a booking; reject missing service; test visitor isolation.

5. Review privacy and access: Requested date is not a confirmed appointment.

Copy-ready build request

> Create a mobile-friendly appointment inquiry board for a mobile service business. Store these fields: name, service, requested_day, note, state. Support this precise journey: Visitor sends an inquiry; owner reviews it privately and changes state to Contacted. Start with fictional sample data: Noah | tune-up | Friday | after 2 pm | Requested. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Show receipt without promising a booking; reject missing service; test visitor isolation.

Acceptance record

Expected: Show receipt without promising a booking; reject missing service; test visitor isolation. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Requested date is not a confirmed appointment. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 04 Home maintenance planner

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A homeowner. Build the smallest useful experience for this person.

Data to model

task, interval_days, completed_on, next_due

Example fictional record

Check smoke alarms | 180 | 2026-11-01 | 2027-04-30

Distinct build walkthrough

1. Define a form for task and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Mark an item completed and recalculate its due date based on the chosen interval.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Test leap-year and month-boundary dates; verify overdue filtering.

5. Review privacy and access: Dates are reminders, not professional safety advice.

Copy-ready build request

> Create a mobile-friendly home maintenance planner for a homeowner. Store these fields: task, interval_days, completed_on, next_due. Support this precise journey: Mark an item completed and recalculate its due date based on the chosen interval. Start with fictional sample data: Check smoke alarms | 180 | 2026-11-01 | 2027-04-30. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Test leap-year and month-boundary dates; verify overdue filtering.

Acceptance record

Expected: Test leap-year and month-boundary dates; verify overdue filtering. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Dates are reminders, not professional safety advice. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 05 Content idea library

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A short-form creator. Build the smallest useful experience for this person.

Data to model

title, hook, channel, stage, notes

Example fictional record

3 ways to save time | Try this before editing | Shorts | Draft

Distinct build walkthrough

1. Define a form for title and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Save a draft; search for save; revise hook; move to Ready.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Search case-insensitively; preserve punctuation and long notes after refresh.

5. Review privacy and access: Do not copy protected scripts without permission.

Copy-ready build request

> Create a mobile-friendly content idea library for a short-form creator. Store these fields: title, hook, channel, stage, notes. Support this precise journey: Save a draft; search for save; revise hook; move to Ready. Start with fictional sample data: 3 ways to save time | Try this before editing | Shorts | Draft. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Search case-insensitively; preserve punctuation and long notes after refresh.

Acceptance record

Expected: Search case-insensitively; preserve punctuation and long notes after refresh. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Do not copy protected scripts without permission. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 06 Digital product catalog

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

An independent e-book seller. Build the smallest useful experience for this person.

Data to model

title, teaser, price, cover, external_checkout

Example fictional record

From Idea to App | beginner workbook | $9.99 | cover | merchant URL

Distinct build walkthrough

1. Define a form for title and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Visitor reads preview and clicks a clearly labeled external checkout.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Verify outbound URL; prevent paid file from being publicly exposed; test broken links.

5. Review privacy and access: A click to PayPal is not proof of payment.

Copy-ready build request

> Create a mobile-friendly digital product catalog for an independent e-book seller. Store these fields: title, teaser, price, cover, external_checkout. Support this precise journey: Visitor reads preview and clicks a clearly labeled external checkout. Start with fictional sample data: From Idea to App | beginner workbook | $9.99 | cover | merchant URL. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Verify outbound URL; prevent paid file from being publicly exposed; test broken links.

Acceptance record

Expected: Verify outbound URL; prevent paid file from being publicly exposed; test broken links. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: A click to PayPal is not proof of payment. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 07 Project expense log

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A hobby maker. Build the smallest useful experience for this person.

Data to model

date, category, amount, description

Example fictional record

2026-11-03 | Materials | 18.45 | bolts

Distinct build walkthrough

1. Define a form for date and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Add two purchases and show their combined monthly total.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Test decimals, empty category, edits, and consistent currency rounding.

5. Review privacy and access: No banking credentials or tax advice.

Copy-ready build request

> Create a mobile-friendly project expense log for a hobby maker. Store these fields: date, category, amount, description. Support this precise journey: Add two purchases and show their combined monthly total. Start with fictional sample data: 2026-11-03 | Materials | 18.45 | bolts. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Test decimals, empty category, edits, and consistent currency rounding.

Acceptance record

Expected: Test decimals, empty category, edits, and consistent currency rounding. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: No banking credentials or tax advice. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 08 Event RSVP manager

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A local organizer. Build the smallest useful experience for this person.

Data to model

event, attendee, email, guests, response

Example fictional record

Workshop | Casey | casey@example.test | 2 | Yes

Distinct build walkthrough

1. Define a form for event and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Create event capacity 10; accept RSVPs; show remaining seats.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Test 10th and 11th seat; duplicate response; cancellation restores capacity.

5. Review privacy and access: Attendee details visible only to authorized organizers.

Copy-ready build request

Create a mobile-friendly event rsvp manager for a local organizer. Store these fields: event, attendee, email, guests, response. Support this precise journey: Create event capacity 10; accept RSVPs; show remaining seats. Start with fictional sample data: Workshop | Casey | casey@example.test | 2 | Yes. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Test 10th and 11th seat; duplicate response; cancellation restores capacity.

Acceptance record

Expected: Test 10th and 11th seat; duplicate response; cancellation restores capacity. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Attendee details visible only to authorized organizers. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 09 Inventory count board

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A small craft seller. Build the smallest useful experience for this person.

Data to model

sku, name, quantity, reorder_point

Example fictional record

MUG-BLK | black mug | 7 | 5

Distinct build walkthrough

1. Define a form for sku and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Receive 4 units, sell 8, then display low-stock flag at 3.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Reject invalid negative stock; test simultaneous adjustments; verify totals.

5. Review privacy and access: A prototype is not a full accounting system.

Copy-ready build request

> Create a mobile-friendly inventory count board for a small craft seller. Store these fields: sku, name, quantity, reorder_point. Support this precise journey: Receive 4 units, sell 8, then display low-stock flag at 3. Start with fictional sample data: MUG-BLK | black mug | 7 | 5. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Reject invalid negative stock; test simultaneous adjustments; verify totals.

Acceptance record

Expected: Reject invalid negative stock; test simultaneous adjustments; verify totals. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: A prototype is not a full accounting system. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 10 Client feedback collector

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A freelance designer. Build the smallest useful experience for this person.

Data to model

project, rating, comment, publish_permission

Example fictional record

Logo refresh | 4 | easy process | No

Distinct build walkthrough

1. Define a form for project and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Client submits private feedback; owner reviews; only approved quotes appear publicly.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Test no-consent case; unpublish permission withdrawal; require valid rating.

5. Review privacy and access: Never treat feedback as permission to publish a testimonial.

Copy-ready build request

Create a mobile-friendly client feedback collector for a freelance designer. Store these fields: project, rating, comment, publish_permission. Support this precise journey: Client submits private feedback; owner reviews; only approved quotes appear publicly. Start with fictional sample data: Logo refresh | 4 | easy process | No. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Test no-consent case; unpublish permission withdrawal; require valid rating.

Acceptance record

Expected: Test no-consent case; unpublish permission withdrawal; require valid rating. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Never treat feedback as permission to publish a testimonial. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 11 Study session planner

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A learner. Build the smallest useful experience for this person.

Data to model

topic, planned_at, minutes, completed

Example fictional record

Data basics | 2026-11-15 19:00 | 25 | No

Distinct build walkthrough

1. Define a form for topic and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Schedule a session; mark it complete; show completed minutes.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Test timezone, zero duration, rescheduling, and incomplete-session totals.

5. Review privacy and access: Avoid unnecessary personal profiling.

Copy-ready build request

> Create a mobile-friendly study session planner for a learner. Store these fields: topic, planned_at, minutes, completed. Support this precise journey: Schedule a session; mark it complete; show completed minutes. Start with fictional sample data: Data basics | 2026-11-15 19:00 | 25 | No. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Test timezone, zero duration, rescheduling, and incomplete-session totals.

Acceptance record

Expected: Test timezone, zero duration, rescheduling, and incomplete-session totals. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Avoid unnecessary personal profiling. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

PROJECT 12 FAQ publishing board

Practice brief — proposed design, not a tested or published Floot application.

User and purpose

A product creator. Build the smallest useful experience for this person.

Data to model

question, answer, category, published

Example fictional record

Where is my download? | Check receipt email | Delivery | No

Distinct build walkthrough

1. Define a form for question and the remaining listed fields; choose required fields and validation.

2. Build a screen that lets the intended user complete this workflow: Draft an FAQ, preview privately, publish it, and find it via public search.

3. Add persistence, a confirmation state and a useful failure message; use the example record above.

4. Check the exact behavior: Confirm unpublished drafts never appear publicly; test blank answers and search.

5. Review privacy and access: Do not publish private support messages.

Copy-ready build request

Create a mobile-friendly faq publishing board for a product creator. Store these fields: question, answer, category, published. Support this precise journey: Draft an FAQ, preview privately, publish it, and find it via public search. Start with fictional sample data: Where is my download? | Check receipt email | Delivery | No. Add field validation, empty and error states, persistence and appropriate server-side access control. Do not claim integrations or features are working until tested. Provide a test checklist based on: Confirm unpublished drafts never appear publicly; test blank answers and search.

Acceptance record

Expected: Confirm unpublished drafts never appear publicly; test blank answers and search. Actual result: _____. Evidence: _____. Result: PASS / FAIL / NOT TESTED.

Security and scope: Do not publish private support messages. If your app collects personal information, verify applicable privacy and consent requirements before using real records.

Reusable worksheets and prompt templates

Product brief

User: ____ Problem: ____ Main task: ____ Three must-have features: ____ Explicit non-goals: ____.

Bug report

Device/browser: ____ Steps: ____ Expected: ____ Actual: ____ Error message: ____ Retest result: ____.

Release log

Version/date: ____ Published URL: ____ Features tested: ____ Known limitations: ____ Rollback approach: ____.

Planning prompt

> Turn [idea] into one small app brief with roles, screens, fields, validation and six acceptance tests. Ask about assumptions.

Debugging prompt

> Given [reproduction steps], [expected] and [actual], propose one minimal fix and a regression test. Do not claim success without evidence.

Marketing prompt

> Write an accurate landing-page headline and three benefits for [audience] using only these verified capabilities: [list].

Floot platform notes — checked November 2026

Floot builds TypeScript/React apps and provides managed Postgres, sign-in, file storage, email and hosting. Available capabilities and usage limits depend on current plan and documentation.

Build through a connected assistant such as ChatGPT. A description is a request, not proof that the feature was implemented or tested.

Preview and published versions use the same project database. Test writes may affect live records. For isolation, use a genuinely separate project and database.

Publishing is explicit: verify the publish status and then test the public URL. Code changes in preview are not automatically a public release.

Use server-side authorization in addition to sign-in. Never put private API credentials into client code.

Floot documents integrated payments; check its current Payments page for supported processors. PayPal links in LeadsLeap are an external checkout workflow, not evidence of a built-in Floot PayPal integration.

Do not promise a permanently free plan or a particular quota. Verify terms and prices on the day you build.

Official references: How Floot works · Database · Publishing · Sign-in · Documentation home

Sales, delivery and release checklist

Before activating a $9.99 PayPal button, verify the customer receives exactly the advertised PDF after a confirmed payment. A success redirect or button click is not payment verification. Do not put the paid PDF at a public guessable URL.

Test the free Starter Guide signup and delivery separately. Record email marketing consent separately from delivery; make opt-out possible and publish appropriate privacy information.

Test PayPal checkout with a permitted sandbox/test transaction, failure/cancel paths, refund contact details, and a customer download on a phone. The merchant must configure these systems; this PDF does not perform payment or lead capture.

The twelve projects are educational briefs; none are represented as verified live applications. No earnings, traffic or app-store acceptance is guaranteed.

Floot referral: https://floot.com/r/N3VJF3 (affiliate link; possible commission).