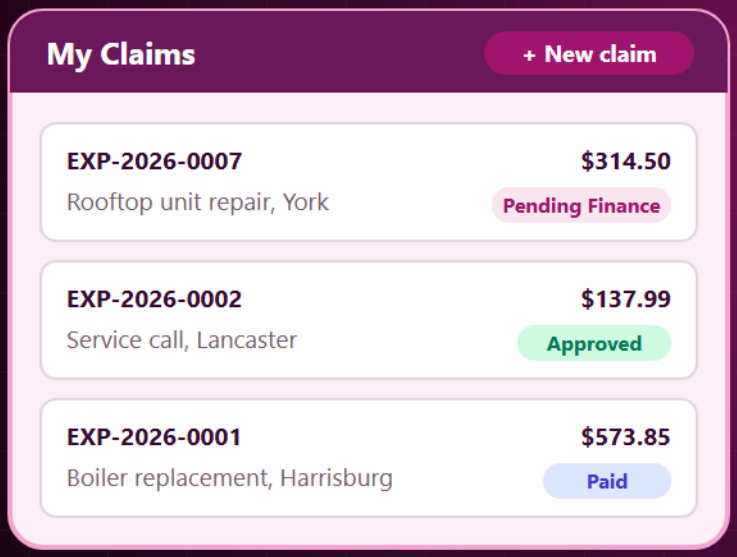





```
Patch(ExpenseLines,
  Defaults(ExpenseLines),
  { Amount: Round(
    Value(txtMiles.Text) * 0.65,
    2) })
```

```
formatNumber(
  314.5, 'C2',
  'en-US')
```

A COMPLETE, PRACTICAL GUIDE

Power Apps and Power Automate

Build low-code business apps and workflows your organization will rely on

Power Fx · Canvas Apps · SharePoint · Delegation · Cloud Flows
Approvals · Dataverse · Model-Driven Apps · ALM · Governance

APPS · FLOWS · APPROVALS · DATAVERSE · PRODUCTION

THENEXBOOKS

Power Apps and Power Automate

Build Low-Code Business Apps and Workflows with Power Fx, SharePoint, Dataverse, and Approvals From your first list and screen to a governed, multi-stage approval system in production

First Edition

THENEXBOOKS

Power Apps and Power Automate
Build Low-Code Business Apps and Workflows with Power Fx, SharePoint, Dataverse, and Approvals

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Formulas, expressions, and code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Power Apps, Power Automate, Dataverse, and the rest of the Microsoft Power Platform are cloud services that change frequently; menu labels, designer screens, feature availability, preview status, and licensing terms can differ between versions, regions, and tenants. Licensing statements in this book are general guidance only — always confirm with Microsoft's current licensing documentation and your administrator. Test every app and flow in a non-production environment before using it with real data.

Microsoft, Microsoft 365, Power Apps, Power Automate, Power BI, Power Platform, Power Fx, Dataverse, Copilot, SharePoint, Microsoft Lists, Microsoft Teams, Outlook, Excel, OneDrive, Microsoft Entra ID, Azure, and Windows are trademarks of the Microsoft group of companies. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by Microsoft Corporation.

Kestrel Field Services, the HVAC company used as the running example in this book, is fictional, as are its employees, expense claims, policies, approval rules, and all sample data. Email addresses and web addresses use reserved example domains. Any resemblance to real organizations or people is coincidental.

Preface

Every organization runs on small processes that nobody ever got around to building properly. Expense claims travel as spreadsheets attached to emails. Approvals happen in reply-all threads. Someone in finance keeps a personal list of who still owes a receipt, and when that person goes on vacation, the process stops. Buying a dedicated system for each of these is expensive, and asking a development team to build one is a request that waits at the bottom of the backlog for a year.

Power Apps and Power Automate exist for exactly these processes. Power Apps lets you build real business applications — screens, forms, galleries, validation, and logic — using formulas that feel like Excel. Power Automate connects those apps to everything else: it routes approvals, sends notifications, moves data between systems, and runs scheduled jobs, all without servers to manage. Together with SharePoint lists or Dataverse as the data store, they let one motivated person, or a small team, deliver in weeks a solution that is secure, maintainable, and genuinely used.

Low-code does not mean low quality, though. Apps that are thrown together grow slow, break when the list passes a few hundred rows, leak data to the wrong people, and stop working the day their creator leaves. This book teaches the platform the way professionals use it: we design the data first, write formulas that delegate, handle errors deliberately, keep secrets and settings out of our logic, package everything in solutions, and deploy through test to production with governance in place.

Who This Book Is For

- **Business analysts and power users** who know Excel and SharePoint and want to replace email-and-spreadsheet processes with real apps and automated workflows.
- **Team leads and operations staff** who own a process — requests, approvals, inspections, onboarding — and want to automate it themselves.
- **IT professionals and developers** who support Power Platform makers, review their solutions, or need to take an app from prototype to governed production.
- **Beginners** with no programming background: every formula and expression is explained, and the book builds one complete system step by step.

What You Will Learn

- How to plan a business app: requirements, data design, statuses, policy rules, and approval routing.
- How to use SharePoint lists as a data source correctly — columns, choices, internal names, indexes, and permissions.
- How to build canvas apps with Power Fx: galleries, forms, Patch, variables, collections, named formulas, responsive layouts, components, and accessibility.

- How delegation works, why it matters, and how to write formulas that keep working as your data grows.
- How to build Power Automate cloud flows: triggers, actions, expressions, conditions, loops, variables, error handling with scopes, and trigger conditions.
- How to build a multi-stage approval process with the Approvals connector, and how to call flows from apps and run scheduled jobs.
- How to model data in Dataverse, add server-side logic, build a model-driven app, and secure it with security roles.
- How to work with receipts and documents, build dashboards, integrate other systems, and tune performance.
- How to ship to production: solutions, environment variables, connection references, pipelines, DLP policies, testing, monitoring, and a go-live plan.

How This Book Is Organized

Part I — Foundations (Chapters 1–5) introduces the platform, plans the running example, builds the SharePoint lists, creates a first canvas app, and teaches Power Fx fundamentals. **Part II — Canvas Apps** (Chapters 6–10) covers variables and named formulas, galleries and delegation, forms and Patch, layout and components, and navigation, validation, and error handling. **Part III — Power Automate** (Chapters 11–15) covers cloud flows, expressions, control flow and error handling, the multi-stage approval flow, and connecting flows to apps. **Part IV — Dataverse and Model-Driven Apps** (Chapters 16–20) moves the solution to Dataverse and adds server-side logic, a model-driven admin app, and role-based security. **Part V — Advanced Building** (Chapters 21–24) covers files and PDFs, reporting, advanced Power Fx and performance, and integration. **Part VI — Production** (Chapters 25–28) covers application lifecycle management, governance, testing and monitoring, and a complete go-live case study. The appendices provide a cheat sheet, templates and reference tables, a production checklist, and a glossary.

Each chapter ends with exercises and a **Key Takeaways** box for quick review.

The Running Example: Kestrel Field Services

Throughout the book we build **Kestrel Expenses** for **Kestrel Field Services**, a fictional heating and air-conditioning company in Philadelphia whose technicians, sales staff, and managers travel to job sites and trade shows. Today they file expense claims on a spreadsheet template and chase approvals by email. We replace that with a canvas app where employees create claims with line items and receipts, policy limits that flag exceptions automatically, and a flow that routes each claim to the employee's manager, then to finance when the claim is large or breaks a policy, and finally to the CFO for the biggest claims. Finance gets a weekly export of approved claims, administrators get a model-

driven app on Dataverse, and the whole solution is deployed through development, test, and production environments. The sample data is small and every amount is exact, so you can check each result by hand.

Versions and Conventions

Power Apps and Power Automate are cloud services, so there is only one current version — and it changes every few weeks. This book describes the platform as it works in **recent versions**. Menus, pane names, and the layout of the flow designer move from time to time, and Microsoft sometimes offers a new and a classic designer side by side. If a command is not exactly where the book says, look for it nearby or use the search box in the studio. Features that are still in **preview** are labeled as such; preview features can change or be withdrawn and are generally not recommended for production. Licensing is described in general terms only, without prices, because plans and entitlements change; check with your administrator before you depend on a premium feature.

Formulas assume an English locale, where a comma separates function arguments and a semicolon chains behavior formulas. Dates are shown in US format (`3/16/2026`) and amounts in US dollars.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, preview status, or a useful aside.

> **WARNING**
> A common mistake or something that can cause data loss, broken flows, or security problems.

Menu commands are written in bold with arrows, such as **Settings > General**. Power Fx formulas, flow expressions, and other code appear in `monospace`; long formulas are broken over several lines to fit the page. Object names follow a consistent convention — `scrMyClaims`, `galMyClaims`, `btnSubmit`, `gblClaim`, `colClaimLines`, `ExpenseClaimApproval` — explained in Chapter 1 and summarized in Appendix B.

Contents

Preface

PART I — FOUNDATIONS

1 Power Apps and Power Automate: What They Are and When to Use Them

2 Planning a Business App: Requirements, Data, and the Kestrel Expenses Blueprint

3 SharePoint Lists as a Data Source: Columns, Choices, and Sample Data

4 Your First Canvas App: Screens, Galleries, Forms, and Studio Basics

5 Power Fx Fundamentals: Formulas, Data Types, and Behavior Properties

PART II — CANVAS APPS

6 Variables, Collections, and Named Formulas

7 Galleries, Filtering, Sorting, and Delegation

8 Data Entry Done Right: Edit Forms, Patch, and Line Items

9 Layout, Theming, Components, and Accessibility

10 Navigation, Validation, and Error Handling

PART III — POWER AUTOMATE

11 Cloud Flow Fundamentals: Triggers, Actions, and Connectors

12 Expressions and Dynamic Content: Workflow Definition Language in Practice

13 Conditions, Loops, Variables, and Error Handling in Flows

14 Approvals: Building the Multi-Stage Expense Approval Flow

15 Apps and Flows Together: Calling Flows from Power Apps, Notifications, and Scheduled Flows

PART IV — DATAVERSE AND MODEL-DRIVEN APPS

16 Dataverse Fundamentals: Environments, Tables, Columns, and Relationships

17 Moving Kestrel Expenses to Dataverse: Modeling, Importing Data, and Canvas Apps on Dataverse

18 Server-Side Logic: Business Rules, Formula and Rollup Columns, and Dataverse Triggers

19 Model-Driven Apps: Views, Forms, Navigation, and Dashboards

20 Dataverse Security: Business Units, Security Roles, Teams, and Sharing

PART V — ADVANCED BUILDING

21 Files, Receipts, and Documents: Attachments, Images, and PDF Reports

22 Reporting and Insight: In-App Dashboards, Exports, and Power BI Integration

23 Advanced Power Fx and Performance: With, ForAll, Concurrent, and Monitor

24 Integration: HTTP, Custom Connectors, Child Flows, and Desktop Flows

PART VI — PRODUCTION

25 Application Lifecycle Management: Solutions, Environment Variables, and Pipelines

26 Governance and Administration: Environments, DLP Policies, and the CoE Starter Kit

27 Testing, Monitoring, and Troubleshooting Apps and Flows

28 Going Live: Deploying Kestrel Expenses to Production — A Complete Case Study

APPENDICES

A Power Fx and Expressions Cheat Sheet

B Templates and Reference

C Production Checklist

D Glossary

PART I

Foundations

What Power Apps and Power Automate are and when to use them, how to plan a business app before building it, setting up SharePoint lists as a clean data source, building a first canvas app, and the Power Fx formula language.

1 · What They Are and When to Use Them

2 · Planning a Business App

3 · SharePoint Lists as a Data Source

4 · Your First Canvas App

5 · Power Fx Fundamentals

CHAPTER 1 · PART I — FOUNDATIONS

Power Apps and Power Automate: What They Are and When to Use Them

Before we build anything, we need a clear picture of what the Microsoft Power Platform is, what "low-code" really means, and where these tools shine or struggle. In this chapter you meet Kestrel Field Services, the company whose expense process we will rebuild across this book, and you get a place to practice.

Why This Book Starts With a Question, Not a Click

It is tempting to open Power Apps Studio on page one and start dragging controls onto a screen. You can do that, and you will have something on the screen within minutes. The trouble starts a few weeks later, when the app that began as a quick experiment has thirty users, a tangle of hidden formulas, a data source that slows down every morning, and nobody who remembers why a certain button exists. Low-code tools make the first ten percent of a project very fast. The remaining ninety percent still needs the same thinking that any software project needs: clear requirements, a sensible data model, security, testing, and a plan for change.

So we start with a question: *what are these tools, and when are they the right choice?* This chapter has no formulas yet. Instead, it gives you the map: the Power Platform family, the kinds of apps and flows you can build, licensing (described carefully and without prices), environments, the two web portals you will use every day, and the running example that ties the whole book together.

What Low-Code Is, and What It Is Not

Low-code means building software mostly by configuring components and writing short formulas instead of writing large amounts of traditional program code. In Power Apps, you place controls such as buttons, text inputs, and galleries on a screen, then set their properties. Some properties take simple values (a color, a width), and others take formulas written in **Power Fx**, the formula language of the Power Platform. If you have written `=SUM(B2:B10)` or `=IF(C2>500,"Review","OK")` in Excel, you already understand the core idea: Power Fx is the Excel formula language applied to apps. A label whose `Text` property is a formula recalculates automatically whenever the data it depends on changes, exactly as a spreadsheet cell does.

In Power Automate, you build **flows**: a trigger (something that happens, such as "a new item was created in a SharePoint list") followed by actions (send an email, update a record, start an approval). You assemble flows in a visual designer, and when you need logic that the designer cannot express with a click, you write short **expressions** in the Workflow Definition Language, a small function-based language we cover in Chapter 12.

Low-code is **not** "no thinking". It is also not "no code" in the strict sense: a serious app contains dozens or hundreds of formulas, and a serious flow contains expressions, error handling, and retry logic. What low-code removes is the plumbing. You do not write HTTP clients, authentication code, user interface rendering, hosting scripts, or database connection pools. Microsoft runs all of that for you. What remains is the part that is truly specific to your business: the data, the rules, the screens, and the process.

> **NOTE**
>
> Throughout this book we use the phrase "citizen developer" for people who build solutions without a software engineering background. It is not a lesser title. Many of the most valuable Power Platform solutions are built by analysts who understand the business process better than any outside developer could. The goal of this book is to help you build like a professional, whatever your job title says.

The Power Platform Family

Power Apps and Power Automate are two members of a larger family called the **Microsoft Power Platform**. The products share connectors, environments, security, and administration, so knowing the whole family helps you pick the right tool for each part of a problem.

Product	What it does	Role in this book
Power Apps	Builds canvas apps and model-driven apps	Main focus (Parts I, II, IV)
Power Automate	Builds cloud flows, desktop flows, and business process flows	Main focus (Part III and later)
Dataverse	Relational data platform with security and server logic	Part IV
Power BI	Reports and dashboards for analytics	Chapter 22 (overview)
Copilot Studio	Builds conversational agents (formerly Power Virtual Agents)	Mention only
Power Pages	Builds external-facing websites (formerly Power Apps portals)	Mention only

Underneath these products sit three shared ideas. First, **connectors**: prebuilt wrappers around services such as SharePoint, Office 365 Outlook, Microsoft Teams, Approvals, SQL Server, and hundreds of others. A connector turns an external API into actions and data sources that you can use without writing code. Second, **environments**: containers that hold apps, flows, data, and connections, with their own security and policies. Third, the **Power Platform admin center** at `admin.powerplatform.microsoft.com`, where administrators manage environments, data policies, capacity, and analytics. You may never need admin rights yourself, but knowing that the admin center exists helps you have productive conversations with your IT department.

You will also hear about **AI Builder**, which adds prebuilt and custom AI models (for example, extracting text from documents) to apps and flows. We mention it for completeness only; the Kestrel solution does not depend on it.

Kinds of Apps: Canvas, Model-Driven, and Power Pages

Power Apps offers two main styles of app, and Power Pages adds a third option for external audiences. Knowing the difference saves you from forcing a tool to do something it was not designed for.

Canvas apps

A **canvas app** starts as a blank canvas, much like a slide in a presentation. You decide exactly where every control goes, what it looks like, and what happens when the user taps it. Canvas apps can connect to many data sources at once: a SharePoint list, an Excel table, a SQL database, Dataverse, and your Office 365 profile, all in the same app. They are ideal when the user experience matters, when the app runs on phones and tablets in the field, or when the process is task-focused ("submit an expense claim", "inspect a rooftop unit").

You build canvas apps in **Power Apps Studio**, which runs in your web browser. The Kestrel Expenses app that employees use is a canvas app, and we spend all of Part II on it.

Model-driven apps

A **model-driven app** is built from the data model outward. You define tables, columns, relationships, views, and forms in Dataverse, then assemble an app that presents them with a consistent, responsive layout generated by the platform. You do not position controls; you choose which views and forms appear in the navigation. Model-driven apps are excellent for back-office work with many records and related tables: an administrator reviewing all claims, filtering by status, opening a claim to see its lines and history, and running a bulk update. They require Dataverse. In Chapter 19 we build **Kestrel Expense Admin**, a model-driven app for the finance team.

Power Pages

When the audience is outside your organization (customers, suppliers, the public), **Power Pages** provides websites that work with Dataverse data. Expenses are an internal process, so we will not build a site.

Question	Canvas app	Model-driven app
Who controls layout?	You, pixel by pixel	The platform, from forms and views
Data sources	Many, mixed in one app	Dataverse
Best for	Task-focused, mobile, custom UX	Record management, back office
Logic lives in	Power Fx formulas in the app	Dataverse rules, views, forms
Kestrel example	Kestrel Expenses	Kestrel Expense Admin

The two styles are not rivals. A mature solution often has both: a focused canvas app for the many people who do one task, and a model-driven app for the few people who manage everything. Model-driven apps can even embed canvas-based **custom pages**, a newer capability we mention in Chapter 19.

Kinds of Flows: Cloud, Desktop, and Business Process

Power Automate also comes in several flavors.

- **Cloud flows** run in Microsoft's cloud and talk to services through connectors. There are three trigger styles: **automated** (start when something happens, such as a new list item), **instant** (start when a user presses a button, including a button in a Power App), and **scheduled** (start on a recurrence, such as every Friday afternoon). Almost every flow in this book is a cloud flow.
- **Desktop flows** automate applications on a Windows computer by imitating a person: clicking buttons, typing into fields, reading screens. This is robotic process automation (RPA). It is the right answer when a legacy application has no API. We discuss desktop flows in Chapter 24, including when an API is the better choice.
- **Business process flows** guide users through stages of a process inside model-driven apps (for example, Qualify, Develop, Propose, Close). They require Dataverse. We mention them in Part IV.

Kestrel's approval process is a cloud flow: when a claim's status becomes `Pending Manager`, a flow named `ExpenseClaimApproval` starts, sends approval requests to the right people in order, and records every decision. Chapter 14 builds it step by step.

Where Low-Code Fits, and Where Pro-Code Is Better

Power Apps and Power Automate are powerful, but they are not the right tool for every job. A responsible builder knows the boundaries. The table below summarizes the patterns we see most often.

Good fit for Power Platform	Consider pro-code instead
Internal business processes: requests, approvals, inspections, inventory, onboarding	Public consumer apps with millions of anonymous users
Replacing paper forms, email chains, and shared spreadsheets	Pixel-perfect branded experiences with complex animation
Data that already lives in Microsoft 365 or Dataverse	Heavy computation, real-time streaming, or very high transaction volumes
Teams that need to change the solution often and quickly	Logic that must run offline for long periods with complex conflict resolution
Gluing services together with connectors	Products you plan to sell as software to other companies

The line is not sharp. Many organizations combine both, exposing a custom API written by professional developers to the app through a custom connector (Chapter 24).

Five questions help you decide early:

1. **Who are the users?** Colleagues with Microsoft 365 accounts are a natural fit; anonymous members of the public point to Power Pages or a pro-code site.
2. **Where is the data?** SharePoint, Excel, Dataverse, and SQL Server are easy to reach; an unusual system may need a custom connector or a desktop flow.
3. **How much data?** Thousands of records are routine; millions of rows need careful design.
4. **Who will maintain it?** A solution nobody owns is a risk on any platform.
5. **What does licensing allow?** See the next section.

> **WARNING**
>
> The fastest way to create an unmaintainable app is to skip planning because "it's only low-code". Apps that start as personal experiments often become business-critical without anyone deciding that they should. Treat any app that other people depend on as real software: give it an owner, store it in a solution, document it, and test it. Chapters 25 to 28 show how.

Licensing in Plain Language

Licensing is the topic that surprises new builders most often, so we address it now, carefully. We will not print prices or exact plan limits, because Microsoft changes licensing regularly and the details vary by agreement and region. Licensing changes; check Microsoft's current licensing guide and your admin before you commit to a design.

The broad picture at the time of writing looks like this:

- Many Microsoft 365 plans include the use of Power Apps and Power Automate with **standard connectors**. Standard connectors include SharePoint, Office 365 Outlook, Microsoft Teams, Approvals, OneDrive for Business, and Excel Online (Business).
- **Premium** connectors and features need a premium license, such as a Power Apps per-app or premium plan, or a Power Automate premium plan. Premium examples include Dataverse (outside of Teams), the HTTP action, custom connectors, SQL Server, and desktop flows that run unattended.
- **Dataverse for Teams** provides a reduced version of Dataverse inside Microsoft Teams. It is worth knowing it exists, but its capabilities are limited compared with full Dataverse, and we do not use it in this book.

In the designer, premium connectors and actions are labeled with a **Premium** marker, which is a helpful hint while you build. If you add a premium connector to an app, every user who runs that app needs an appropriate license, not only you.

This is exactly why Kestrel Expenses is designed in two phases. Parts I to III use only SharePoint, Outlook, Teams, and Approvals, which are standard connectors, so the app can run under many Microsoft 365 plans. Part IV moves the data to Dataverse, which brings real benefits (relational data, row-level security, better delegation, server-side logic) and also requires premium licensing. You will be able to make that trade-off consciously instead of discovering it after launch.

> **NOTE**
>
> Never promise your stakeholders that a solution is "free" because it was built with tools that came with Microsoft 365. The cost of a solution includes licenses for premium features, the time to build and maintain it, and the capacity it consumes. Get a written answer from your licensing contact before you go live.

Environments in One Paragraph

An **environment** is a container for apps, flows, connections, and (optionally) a Dataverse database. Every tenant has a **default** environment that everyone can use, which is convenient for experiments and a governance headache for production work. Administrators can create additional environments of different types: **sandbox** for development and testing, **production** for live solutions, **developer** environments for individual learning, and **trial** environments that expire. Each environment has its own security, data policies, and region. Kestrel will eventually use three environments, **Kestrel Dev**, **Kestrel Test**, and **Kestrel Prod**, and Chapters 25 and 26 explain how solutions move between them. For now, you only need one environment to practice in.

Getting a Place to Practice

To follow this book you need a Microsoft work or school account with access to Power Apps, Power Automate, and SharePoint. There are several ways to get one, and the right choice depends on your situation.

1. **A sandbox or developer environment at work.** If your organization already uses Microsoft 365, ask your IT department or Power Platform administrator for a sandbox or developer environment where you can experiment. This is the best option, because you work with realistic policies and can show the result to colleagues.
2. **The Power Apps Developer Plan.** Microsoft offers a Developer Plan that gives you a free developer environment for learning and building (check the current terms, as they change). It includes Dataverse, which is useful for Part IV. Your organization's policies may control whether you can sign up with a work account.
3. **The default environment.** If nothing else is available, you can practice in the default environment, but keep your experiments clearly named and clean them up afterward, because everyone in your tenant shares that space.

For the SharePoint parts of the book you also need permission to create a SharePoint site, or an existing site where you can create lists. Chapter 3 creates the **Kestrel Expenses** site at an address of the form `https://<your-tenant>.sharepoint.com/sites/KestrelExpenses`. If you cannot create sites, ask an administrator to create one for you and give you owner permissions.

> **TIP**
>
> Use a test account or a dedicated practice environment for approvals. Approval requests and notification emails from your experiments will land in real inboxes. In the Kestrel data every person uses an address at `example.com`, which is a reserved domain that never delivers mail. When you test flows, replace those addresses with your own address or with test users in your tenant.

A Tour of make.powerapps.com

Power Apps is managed from a web portal, often called the **maker portal**, at `make.powerapps.com`. Sign in with your work or school account. Labels and menu positions may differ slightly in recent versions, because Microsoft updates the portal every few weeks, but the main areas have been stable for a long time.

- **Environment picker** in the top bar, showing the current environment. Always check it before you create anything; building in the wrong environment is one of the most common beginner mistakes.
- **+ Create**, with options such as **Blank app > Blank canvas app**, starting from data, or starting from a template.
- **Apps**, which lists the apps you own or that are shared with you, with commands to edit, play, share, and view details such as version history.
- **Tables**, showing Dataverse tables when the environment has a Dataverse database.
- **Solutions**, where you package apps, flows, and tables for deployment (central from Chapter 25).

Many makers also see a **Copilot** box on the home page that offers to build an app from a description. Copilot features, where available in your region and tenant, can save time when sketching an idea, but features change quickly. Always review generated formulas and data structures before you rely on them. In this book we build everything deliberately so that you understand each part.

A Tour of make.powerautomate.com

Power Automate has its own portal at `make.powerautomate.com`. It shares environments with Power Apps, so a flow you create in the Kestrel Dev environment here can be used by an app in the same environment there.

- **+ Create**, with the three cloud flow types: **Automated cloud flow**, **Instant cloud flow**, and **Scheduled cloud flow**, plus options for desktop flows and templates.

- **My flows**, with tabs for cloud flows you own, flows shared with you, and desktop flows.
- **Approvals**, the Approvals center where you see approval requests you have received and sent. Chapter 14 returns here often.
- **Connectors**, where you browse connectors and see which are standard or premium.

Inside a flow you work in the **designer**. At the time of writing, two designers exist: a newer designer with a modern layout and Copilot assistance, and the classic designer. You can usually switch between them with a toggle, although some features appear in only one. The screens differ, but the concepts (triggers, actions, dynamic content, expressions, settings, and run history) are the same. We describe steps in a way that works in both, and we point out differences when they matter.

Meet Kestrel Field Services

Every chapter in this book builds on one realistic scenario. **Kestrel Field Services** is a fictional HVAC (heating, ventilation, and air conditioning) installation and maintenance company headquartered in Philadelphia, Pennsylvania. Its technicians travel to job sites in Harrisburg, Lancaster, York, and Allentown. Its sales team attends trade shows across the country. Its finance team reimburses everyone's expenses, and its small IT team supports the Microsoft 365 tenant. All amounts are in US dollars, and the company works in the Eastern time zone.

Nine people appear in our sample data. You will meet them again and again, so here is the cast.

Code	Name	Job title	Department	Manager
E1000	Rachel Kim	Chief Executive Officer	Executive	(none)
E1001	Daniel Reyes	Chief Financial Officer	Finance	Rachel Kim
E1002	Maria Lopez	Finance Manager	Finance	Daniel Reyes
E1003	Aisha Khan	Operations Manager	Field Operations	Rachel Kim
E1004	Tom Becker	Field Technician	Field Operations	Aisha Khan
E1005	Priya Nair	Field Technician	Field Operations	Aisha Khan
E1006	James Carter	Sales Director	Sales	Rachel Kim
E1007	Lena Fischer	Account Executive	Sales	James Carter
E1008	Sam Okoro	IT Support Specialist	IT	Daniel Reyes

Email addresses follow the pattern `firstname.lastname@example.com`, all lowercase, for example `priya.nair@example.com`. Two roles matter for approvals: **Maria Lopez** is the finance approver, and **Daniel Reyes** is the CFO.

The problem Kestrel wants to solve

Today, Kestrel employees fill in an Excel template for each trip, print it, staple paper receipts to it, and hand it to their manager, or they email a scanned copy. Managers sign and forward the paper to finance. Finance retypes every line into the accounting system, chases missing receipts, and argues about meals that went over the limit. Nobody can answer simple questions such as "how many claims are waiting for me?" or "how much did we spend on lodging this month?" without opening a pile of files.

The company wants a simple, mobile-friendly system that does the following:

- lets employees create a claim with line items and photographed receipts from a phone or laptop;
- checks the company's expense policy while the employee types, not weeks later;
- routes each claim to the right approvers automatically (manager, then finance, then the CFO when needed);
- notifies everyone at each step and keeps a complete history;
- gives finance a weekly list of approved claims to pay, and a clear view of everything in progress;
- is secure, governed, and deployable through proper development, test, and production stages.

The Kestrel Expenses plan

The solution we build is called **Kestrel Expenses**. Here is the plan in one picture.

Kestrel Expenses — solution overview



The book follows this plan in six parts: foundations (Chapters 1 to 5), the canvas app (Part II), flows and approvals (Part III), Dataverse and the model-driven admin app (Part IV), files, reporting, performance, and integration (Part V), and production (Part VI).

One claim follows us through the whole journey. In Chapter 8 you create **EXP-2026-0007**, a claim from Priya Nair for a rooftop unit repair at a York job site, with five lines totaling $314.50. One hotel line is over the lodging limit, so the claim has a policy exception, and the approval flow sends it to Aisha Khan (her manager) and then to Maria Lopez (finance) before it is approved. By the end of the book you will be able to trace that claim through every screen, formula, flow run, and table.

Naming Conventions From Day One

Names matter more in low-code than you might expect. Formulas refer to controls by name, flows refer to actions by name, and apps refer to flows by name. When a control is called `Gallery3`, a formula like `Gallery3.Selected.TotalAmount` tells you nothing. When it is called `galMyClaims`, the formula explains itself. Renaming later is possible, but every rename is a chance to break something, so we agree on conventions now and use them throughout.

Object	Convention	Kestrel example
Screen	`scr` + name	`scrHome`, `scrMyClaims`
Label	`lbl` + name	`lblWelcome`, `lblTotal`
Text input	`txt` + name	`txtSearch`, `txtAmount`
Button	`btn` + name	`btnNewClaim`, `btnSubmit`
Gallery	`gal` + name	`galMyClaims`, `galLines`
Form	`frm` + name	`frmClaim`, `frmClaimView`
Drop down	`drp` + name	`drpStatusFilter`, `drpCategory`
Combo box	`cmb` + name	(not used in the core app)
Date picker	`dp` + name	`dpExpenseDate`
Toggle	`tgl` + name	`tglReceipt`
Other controls	`chk`, `rdo`, `ico`, `img`, `con`, `cmp`, `tmr`, `att`, `htm`	`attReceipts`
Global variable	`gbl` + name (Set)	`gblClaim`
Context variable	`loc` + name (UpdateContext)	`locSubmitting`
Collection	`col` + name	`colCategories`
Named formula	PascalCase, no prefix	`MileageRate`

Flow	PascalCase, no spaces	`SubmitExpenseClaim`
Flow variable	`var` + name	`varFinanceRequired`
SharePoint column	No spaces, PascalCase	`TotalAmount`
Dataverse table	Prefix `kfs_`, lowercase	`kfs_expenseclaim`

A few of these rules deserve a short explanation now, and later chapters give the full reasons.

- **Flow names without spaces.** When you add a flow to a canvas app, the app refers to it by name, as in `SubmitExpenseClaim.Run(gblClaim.ClaimNumber)`. A name with spaces or hyphens produces an awkward reference. PascalCase keeps it clean.
- **Renamed flow actions.** Inside a flow we rename actions to short sentences such as "Get employee" or "Manager approval". Expressions refer to them with spaces replaced by underscores, as in `outputs('Get_employee')`. A flow full of actions named "Get item 2" and "Condition 5" is very hard to maintain.
- **SharePoint columns without spaces.** SharePoint fixes a column's internal name at creation and encodes spaces awkwardly (Chapter 3). The `kfs` prefix in Dataverse comes from our publisher in Chapter 16.

> **TIP**
> Write your conventions down and share them with everyone who builds in your environment. A one-page naming standard, agreed before the first app, costs an hour and saves days of confusion. You can start from the table above.

How to Read This Book

Each chapter builds on the previous ones, so the first reading works best in order. Code appears in blocks with a caption that names where it goes. A caption such as `App.Formulas` means "select the App object in the tree view and set its Formulas property". A caption such as `galMyClaims.Items` means "select the gallery named galMyClaims and set its Items property". Flow expressions have captions that name the flow and the action. Menu paths appear in bold with arrows between levels, such as **Settings > General > Data row limit**.

If a label differs from the one in this book, look for the concept rather than the exact word: the property, the trigger, or the setting will still exist, perhaps under a new heading.

You can ask [your AI assistant] to explain a formula, but treat every suggestion as a draft: check function names, argument order, and delegation, and test the result with the Kestrel sample data.