Chapter 5: Common Building Blocks

> *"Most apps are built from the same set of pieces. Learn these once, use them everywhere."*

This chapter is your reference guide. You don't need to read it start to finish - come back when you need a specific feature. Each section tells you what it is, when you need it, and exactly what to tell AI.

Authentication (User Accounts)

What It Is

The system that lets users sign up, log in, and stay logged in. Includes:

- Creating accounts (registration)
- Verifying identity (login)
- Staying logged in (sessions)
- Resetting passwords
- Social login (Google, GitHub, etc.)

When You Need It

You need authentication if:

- Users have personalized data
- Users create content
- You need to know who's doing what
- Different users see different things
- You have any concept of "my stuff"

You don't need authentication if:

- Same content for everyone
- No user-specific features
- Landing pages, blogs, portfolios
- Pure content sites

How to Describe It to AI

Basic setup:

```
Set up authentication using Supabase Auth where users can:
- Sign up with email and password
- Log in with email and password
- Stay logged in even after closing the browser
- Log out when they want
```

Social login:

```
Add Google authentication so users can sign up/log in with their Google account. Configure the OAuth flow with Supabase.
```

Password reset:

```
Add password reset functionality where:
- Users can request a reset email
- They click a link in the email
- They enter a new password
- They're automatically logged in
```

Protected routes:

```
Protect the /dashboard route so only logged-in users can access it. Redirect non-authenticated users to /login.
```

Common Patterns

Email verification: "Require users to verify their email before they can use the app. Send a confirmation email on signup."

Remember me: "Add a 'Remember me' checkbox that keeps users logged in for 30 days instead of the default session duration."

Profile completion: "After users sign up, redirect them to a profile setup page where they add their name and profile picture before accessing the app."

Session management: "Check if the user's session is still valid on page load. If expired, log them out and redirect to login."

What AI Handles for You

✅ Password hashing (bcrypt) ✅ Session token generation ✅ CSRF protection ✅ OAuth flows ✅ Secure cookie management ✅ Token refresh logic

You describe: "Users should be able to log in" AI implements: All the security best practices

---

Databases (Storing Data)

What It Is

Where all your app's data lives permanently. Think of it as organized filing cabinets where each cabinet is a "table" and each file is a "row."

Key concepts:

- **Tables:** Collections of similar data (users table, posts table)
- **Columns:** Properties of each item (email, name, created_at)
- **Rows:** Individual items (one row = one user)
- **Relationships:** How data connects (post belongs to user)

When You Need It

You need a database if:

- Data needs to persist (survive page refresh)
- Multiple users have their own data
- You're creating, reading, updating, or deleting anything

- Users need to save their work

You don't need a database if:

- All data can be calculated on the fly
- Nothing needs to be saved
- Pure display/presentation sites

How to Describe It to AI

Creating tables:

```
Create a Supabase table called "recipes" with these columns:
- id (auto-generated UUID)
- user_id (links to auth.users)
- title (text, required)
- ingredients (text array)
- instructions (long text)
- prep_time (integer, minutes)
- is_public (boolean, default false)
- created_at (timestamp, auto)
```

Relationships:

```
Create a "comments" table where:
- Each comment belongs to one post (foreign key to posts.id)
- Each comment belongs to one user (foreign key to auth.users.id)
- Store the comment text and timestamp
```

Querying data:

```
Fetch all recipes where:
- user_id matches the current user
- OR is_public is true
Order by created_at descending
Limit to 20 results
```

Updating data:

```
When user clicks "Make Public", update that recipe's is_public field to true in the database.
```

Deleting data:

```
Add a delete button that removes the recipe from the database and updates the UI immediately.
```

Common Patterns

Soft deletes: "Instead of actually deleting recipes, add a 'deleted_at' timestamp column. Hide rows where deleted_at is not null. This lets users undo deletion."

Pagination: "Load 20 recipes at a time. Add a 'Load More' button that fetches the next 20 using offset."

Search: "Add a search input that filters recipes by title or ingredients using Supabase's text search."

Real-time updates: "Use Supabase real-time subscriptions so when one user adds a comment, other users see it appear immediately without refreshing."

Data validation: "Before inserting a recipe, check that title is not empty and prep_time is a positive number. Show error if invalid."

What AI Handles for You

✅ SQL query syntax ✅ Indexing for performance ✅ Connection pooling ✅ Type safety ✅ Error handling ✅ Race conditions

You describe: "Store user posts with title and body" AI implements: Proper table structure with relationships and constraints

---

File Uploads (Images, PDFs, Documents)

What It Is

Letting users upload files from their device to your app. Files are stored separately from your database (usually in "blob storage" or "object storage").

Common use cases:

- Profile pictures
- Document attachments
- Product images
- Resume uploads
- CSV imports

When You Need It

You need file uploads if:

- Users share images or documents
- You have media-heavy features
- Users need to back up files
- Profile pictures, cover photos, etc.

You don't need file uploads if:

- Text-only content
- URLs to external images are fine
- No user-generated media

How to Describe It to AI

Basic upload:

```
Add a profile picture upload where:
- Users can select an image file (JPG, PNG)
- Max size 5MB
- Preview the image before uploading
```

```
- Upload to Supabase Storage in "avatars" bucket
- Save the public URL to user's profile in database
```

Multiple files:

```
Let users upload multiple images for a product listing:
- Allow selecting multiple files at once
- Show preview thumbnails
- Upload to "products" bucket with unique filenames
- Store array of URLs in products table
```

File restrictions:

```
Only allow PDF uploads with:
- Max size 10MB
- Show error if wrong file type
- Display file name and size before upload
- Show upload progress bar
```

Direct camera access:

```
On mobile, show "Take Photo" button that:
- Opens device camera
- Lets user take picture
- Uploads directly without saving to device
```

Common Patterns

Image resizing: "After upload, automatically resize images to 800px width for display and 100px width for thumbnails. Store both versions."

File validation: "Check file type on both client and server. Reject files that aren't images with a helpful error message."

Upload progress: "Show a progress bar while files upload. Display percentage complete."

Delete functionality: "Add a delete button that removes the file from storage and updates the database URL to null."

Drag and drop: "Create a dropzone where users can drag files from their desktop instead of clicking a file input."

What AI Handles for You

✅ Multipart upload handling ✅ File type validation ✅ Unique filename generation ✅ Public URL generation ✅ CORS configuration ✅ Bucket permissions

You describe: "Users can upload profile pictures" AI implements: Complete upload flow with preview and validation

---

Payments (Stripe Integration)

What It Is

Collecting money from users. Stripe is the industry standard for online payments - it handles credit cards, subscriptions, invoices, and compliance.

Common use cases:

- One-time purchases
- Monthly subscriptions
- Pay-per-use billing
- Marketplace commissions

When You Need It

You need payments if:

- You're charging money (obviously)
- Users buy products or services
- Subscription-based access
- Freemium with paid upgrades

You don't need payments if:

- Free product forever
- Monetizing through ads
- Not ready to charge yet

How to Describe It to AI

One-time payment:

```
Integrate Stripe for one-time payments:
- Add a "Buy Now" button ($29 price)
- Open Stripe Checkout when clicked
- After successful payment, save purchase to database
- Grant user access to premium features
- Send confirmation email
```

Monthly subscription:

```
Set up Stripe subscriptions:
- Create a $9/month subscription plan
- Show "Subscribe" button
- Handle Stripe Checkout
- Listen for Stripe webhooks
- Update user's subscription status in database
- Handle cancellations and failed payments
```

Usage-based billing:

```
Implement pay-per-use with Stripe:
- Track API calls per user in database
- Calculate bill at end of month
- Charge users based on usage tier
- Send invoice
```

Common Patterns

Free trial: "Offer 14-day free trial before charging. Capture card details but don't charge until trial ends. Send reminder email 3 days before charging."

Payment plans: "Let users choose between monthly ($9) and annual ($90, two months free). Show savings for annual plan."

Failed payment handling: "When payment fails, retry automatically 3 times. Email user after each failed attempt. Downgrade to free plan after 3 failures."

Upgrade/downgrade: "Let users upgrade from $9/month to $19/month plan. Prorate the difference and charge immediately."

Refunds: "Add admin interface to issue refunds. When refunded, downgrade user and send confirmation email."

What AI Handles for You

✅ Stripe API integration ✅ Checkout session creation ✅ Webhook signature verification ✅ Subscription lifecycle ✅ Payment intent handling ✅ PCI compliance setup

You describe: "Users pay $29 to unlock features" AI implements: Complete Stripe integration with webhooks and database updates

Important Note

Start without payments. Get users first, add billing later. Many successful products launched free and added payments months later once they proved value.

Email (Sending Messages)

What It Is

Programmatically sending emails from your app. Used for notifications, updates, marketing, and transactional messages.

Common use cases:

- Welcome emails
- Password reset links
- Purchase confirmations
- Weekly digests
- Team invitations

When You Need It

You need email if:

- Users need notifications
- Onboarding flows
- Password resets
- Purchase confirmations
- Any off-app communication

You don't need email if:

- In-app notifications are enough
- No user accounts
- Early MVP stage

How to Describe It to AI

Transactional email (Resend recommended):

```
Set up Resend for transactional emails:
- Install Resend SDK
- Configure with API key
- Send welcome email when user signs up
- Include: greeting, what to do next, link to dashboard
- Use plain HTML template with good styling
```

Password reset email:

```
When user requests password reset:
- Generate secure reset token
- Store token with expiry in database
- Send email with reset link containing token
- Link goes to /reset-password?token=xyz page
```

Notification email:

```
When someone comments on user's post:
- Get post author's email from database
- Send notification email
- Include: commenter name, comment text, link to post
- Add unsubscribe link at bottom
```

Common Patterns

HTML templates: "Create reusable email templates with variables for user name, link URL, etc. Use inline CSS for styling."

Bulk emails: "Send weekly digest to all users with new features. Queue emails to avoid hitting rate limits. Track open rates."

Email verification: "Send verification email with 6-digit code on signup. User must enter code before accessing app."

Drip campaigns: "After signup, send day 1 welcome, day 3 tips, day 7 case study. Track which emails user has received."

Unsubscribe: "Add unsubscribe link to all marketing emails. Store preference in database. Don't send to unsubscribed users."

What AI Handles for You

✅ SMTP configuration ✅ HTML email formatting ✅ Rate limiting ✅ Retry logic ✅ Error handling ✅ Template rendering

You describe: "Send welcome email when users sign up" AI implements: Complete email flow with proper formatting and error handling

Recommended Services

Transactional (automated emails):

- **Resend** - $20/month, 50k emails, excellent DX
- **SendGrid** - Free tier: 100 emails/day
- **Postmark** - Pay per email, great deliverability

Marketing (newsletters):

- **ConvertKit** - $29/month, designed for creators
- **Mailchimp** - Free tier: 500 contacts
- Keep it separate from app emails

Deployment (Going Live)

What It Is

Making your app accessible on the internet. Deployment platforms host your code, serve it to users, and handle scaling.

Key concepts:

- **Build:** Converting your code to optimized production files
- **Deploy:** Uploading to hosting platform
- **Environment:** Production vs development settings
- **Domain:** Custom URL (yourapp.com)

When You Need It

You deploy when:

- App is functional enough to test
- Ready for first users
- Want feedback from others
- Need a public URL

Don't wait for:

- Perfect features
- Beautiful design
- Zero bugs
- Complete functionality

Ship early, iterate publicly.

How to Describe It to AI

Initial Vercel deployment:

```
Deploy this Next.js app to Vercel:
- Connect my GitHub repository
- Set up environment variables
- Configure build settings
- Set up automatic deployments on git push
```

Custom domain:

```
Connect my custom domain (myapp.com) to Vercel:
- Add domain in Vercel settings
- Configure DNS records
- Set up SSL certificate
- Redirect www to non-www
```

Environment variables:

```
Set up different environment variables for production vs development:
- Use local database in development
- Use production database in production
- Keep API keys secure
```

Common Patterns

Preview deployments: "Every git branch gets its own preview URL. Test features before merging to main."

Rollback: "If deployment breaks something, roll back to previous version instantly."

Custom build commands: "Run database migrations before each deployment. Fail deployment if migrations fail."

Environment-specific behavior: "Use Stripe test mode in development, live mode in production. Show developer banner in staging."

Analytics: "Add Vercel Analytics to track pageviews, performance, and errors in production."

What AI Handles for You

✅ Build configuration ✅ Optimization settings ✅ Cache configuration ✅ SSL certificates ✅ CDN setup ✅ Edge functions

You describe: "Deploy my app to production" AI implements: Complete deployment pipeline with best practices

Recommended Platforms

For Next.js/React:

- **Vercel** - Best for Next.js, free tier generous
- **Netlify** - Great for static sites
- **Cloudflare Pages** - Fast, global, generous free tier

For Python/Flask:

- **Railway** - Easiest Python deployment, $5/month
- **Render** - Good free tier, easy setup

- **Fly.io** - Good for global deployment

For Databases:

- **Supabase** - Free tier: 500MB, 50k monthly users
- **Railway** - Includes PostgreSQL, $5/month
- **PlanetScale** - Serverless MySQL, generous free tier

How These Pieces Work Together

Real apps combine these building blocks:

Example: Simple SaaS Tool

1. **Authentication** - Users sign up and log in
2. **Database** - Store user data and preferences
3. **File Uploads** - Let users upload CSVs to process
4. **Email** - Send welcome email and weekly reports
5. **Deployment** - Live on custom domain

Example: Marketplace

1. **Authentication** - Buyers and sellers have accounts
2. **Database** - Products, orders, reviews
3. **File Uploads** - Product images
4. **Payments** - Stripe for transactions
5. **Email** - Order confirmations, seller notifications
6. **Deployment** - Scaled to handle traffic

Example: Content Platform

1. **Authentication** - Writers create accounts
2. **Database** - Articles, comments, likes
3. **File Uploads** - Cover images, author photos
4. **Email** - New post notifications
5. **Deployment** - Fast global CDN

Notice the pattern: Most apps need 3-5 of these building blocks. You're not building something completely unique - you're arranging familiar pieces in a new way.

Quick Reference Card

Save this for when you're building:

"I need users to log in" → Authentication section **"I need to save data"** → Database section **"Users upload images"** → File Uploads section **"I need to charge money"** → Payments section **"Send notifications"** → Email section **"Make it live"** → Deployment section

Each section tells you exactly what to tell AI. Come back to this chapter whenever you need a specific feature.

What You Don't See Here

Some building blocks are more advanced and not needed for your first few projects:

Background jobs - Tasks that run periodically (we cover this in the Technical Playbook) **Real-time features** - Live chat, collaborative editing (advanced, we cover this in Technical Playbook) **Search** - Full-text search engines (start with simple database queries) **Analytics** - User behavior tracking (add after you have users) **Testing** - Automated tests (add after you ship)

Focus on the core six building blocks above. They'll get you 90% of the way to a shipped product.

Now let's talk about what happens when things don't work as expected.

Connect & Share

💌 **Newsletter**: Build to Launch - Weekly AI building tips, templates, and real builder stories

✍️ **Medium**: AI Builders - Read more articles and guides

💬 **Reddit**: r/VibeCodingBuilders - Join the community

🦋 **Bluesky**: @jenny-ouyang - Connect

💼 **LinkedIn**: Jenny Ouyang - Connect