

VIBE CODING TO PRODUCTION

Complete Beginner-Friendly Masterclass Guide

TABLE OF CONTENTS

****GETTING STARTED****

- What is Vibe Coding (Easy Explanation)
- Tools We'll Use
- How Web Apps Work (Simple Overview)
- Important Words You'll Learn

****SETTING UP YOUR WORKSPACE****

- Install Cursor (Your AI Coding Assistant)
- Understanding Cursor (Chat, Quick Edit, Terminal, Files)
- Install Postman (For Testing)
- Setup Conda (Python Environment)
- Setup NPM and VITE (For Web Pages)
- Install DBeaver (Database Viewer)
- Your Project Template Walkthrough

****PLANNING YOUR APP****

- Choosing What to Build
- **Workshop Activity 1:** Coming Up with App Ideas
- Creating Your App Plan (PRD)
- **Workshop Activity 2:** Writing Your App Plan with AI
- What your app will do
- What pages it needs
- What tools we'll use (Flask + PostgreSQL + React)
- How it will look (colors, style)
- How users will use it (step by step)

- Breaking work into small tasks

****BUILDING YOUR APP****

- **Workshop Activity 3:** Creating Your Home Page
- **Workshop Activity 4:** Building All Your App Pages (with fake data)
- **Workshop Activity 5:** Setting Up Your Database
- **Workshop Activity 6:** Building Your Backend (the brain of your app)
- **Workshop Activity 7:** Connecting Everything Together
- Tips for Fixing Problems

****PUTTING YOUR APP ONLINE****

- **Workshop Activity 8:** Getting Ready to Launch
- Creating Docker Files (packaging your app)
- Uploading to GitHub (code storage)
- Deploying Your App (making it live)
- Setting Up Your Website Address
- Buying a Domain Name
- Connecting Your Domain
- Testing Everything

****BONUS MATERIALS****

- Templates and Examples
- Cursor Tips and Tricks
- Ready-to-Use Prompts
- App Planning Template
- Best Practices for Vibe Coding
- Common Mistakes to Avoid

****ADDITIONAL RESOURCES & CERTIFICATION****

- **Get More Materials:** Login to curatelearn.com
- **Organization Resources:** Go to Organization tab in the Spaces page
- **Vibe Coding Space:** Find all resources in the "VIBE CODING TO PRODUCTION" space
- **Download Certificate:** Available in the Vibe Coding space after completion

GETTING STARTED

What is Vibe Coding (Easy Explanation)

Vibe Coding is a fun, modern way to build apps where you:

- **Build by Feel:** Create apps that feel good to use, not just look good
- **Use AI Help:** Let AI tools like Cursor help you write code faster
- **Build Quick Versions:** Make a working version fast, then improve it
- **Focus on Users:** Always think about what users want and need
- **Keep Improving:** Make small changes based on feedback
- **Do One Thing at a Time:** Work on one feature, test it, then move to the next

Think of it like cooking - you taste as you go, adjust the recipe, and focus on making something people will enjoy eating!

Tools We'll Use

Main Tool:

- **Cursor:** Your AI coding buddy that helps you write code by chatting with it

Helper Tools:

- **Postman:** Tests if your app's backend is working
- **DBeaver:** Lets you see and manage your app's data
- **Git/GitHub:** Saves your code and lets others see it
- **DigitalOcean:** Puts your app on the internet
- **Aiven.io:** Manages your app's database

What We'll Build With:

- **Frontend** (What users see): React + TypeScript + Vite
- **Backend** (The brain): Flask + Python
- **Database** (Where data lives): PostgreSQL
- **Styling** (How it looks): CSS

How Web Apps Work (Simple Overview)

Imagine a restaurant:

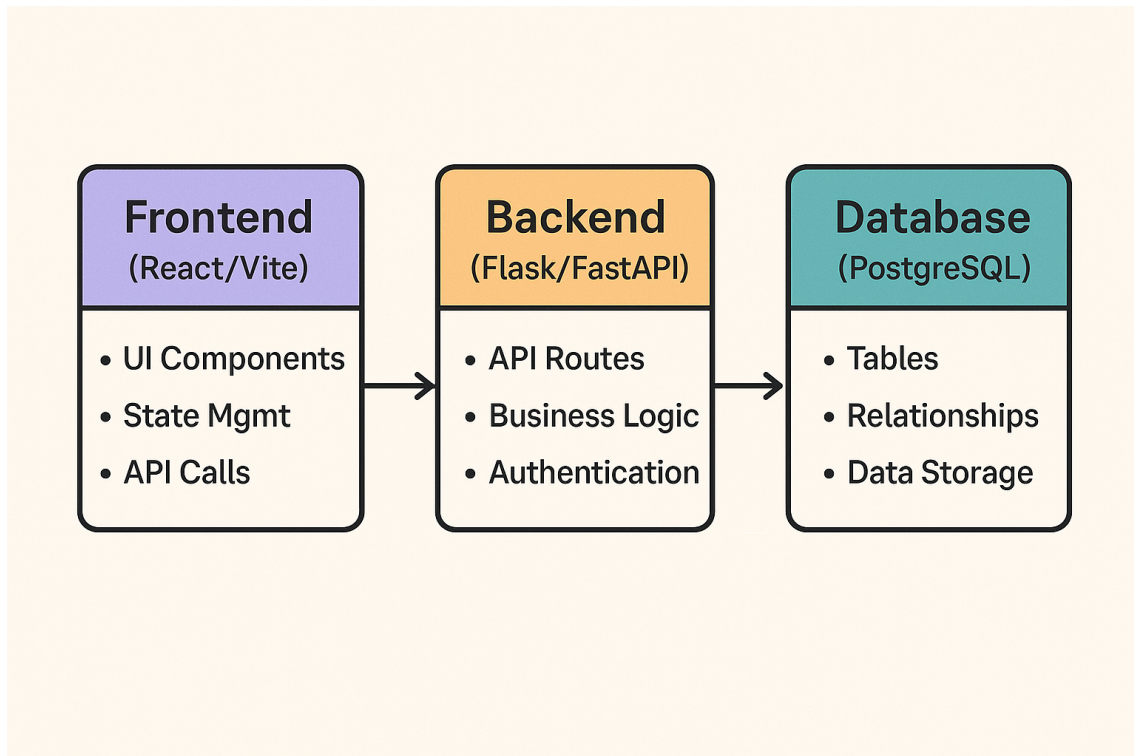
...

■ CUSTOMER (User) ■■ WAITER (Frontend) ■■■ CHEF (Backend) ■ RECIPES (Database)

Orders food → Takes order to kitchen → Cooks the food → Looks up recipes

Gets food ← Brings food back ← Gives finished dish ← Uses stored recipes

...



Web App Diagram

In web apps:

- **Frontend**: What you see and click (like the waiter)
- **Backend**: Processes requests and data (like the chef)
- **Database**: Stores information (like recipe book)

Your Project Setup

When you start a new project, your folders will look like this:

...

my-awesome-app/

■■■■ .cursor/ # Settings for AI helper

■■■■ backend/ # Server code (the brain)
■ ■■■■ blueprints/ # Different features (like login, user profile)
■ ■■■■ docs/ # Instructions for testing
■ ■■■■ config.py # App settings
■ ■■■■ database.py # Database setup
■ ■■■■ main.py # Main app file
■ ■■■■ requirements.txt # List of tools needed
■■■■ frontend/ # User interface (what people see)
■■■■ resources/ # Helpful documents and templates
■■■■ README.md # Project description
...

Why This Setup is Great:

- **Organized:** Everything has its place
- **AI-Friendly:** Cursor can easily understand and help
- **Easy to Grow:** Simple to add new features
- **Ready for Internet:** Can be put online easily

Important Words You'll Learn

Frontend: The part of your app that users see and interact with (buttons, forms, pages)

Backend: The behind-the-scenes part that handles data and logic

API: How the frontend and backend talk to each other (like passing notes)

Endpoint: A specific address where your app can receive requests

Database: Where your app stores information permanently

Authentication: Making sure users are who they say they are (login/signup)

Deployment: Putting your app on the internet so others can use it

Version Control: Keeping track of changes to your code over time

SETTING UP YOUR WORKSPACE

Install Cursor (Your AI Coding Assistant)

1. Go to cursor.sh in your web browser
2. Click the download button for your computer (Windows/Mac/Linux)
3. Run the installer and follow the steps
4. Create an account when it opens
5. Sign in to unlock AI features

Why Cursor is Amazing:

- It's like having a coding expert sitting next to you
- You can ask it questions in plain English
- It can write code for you
- It helps fix mistakes
- It explains what code does

Understanding Cursor (Your New Best Friend)

Chat with AI (`Cmd+L` on Mac, `Ctrl+L` on Windows):

- Ask questions like "How do I create a login page?"
- Request code: "Make me a button that changes color when clicked"
- Get help with errors: "This code isn't working, can you help?"
- Use '@' to mention files: "Look at @app.py and help me add a new feature"

Quick Edit (`Cmd+K` on Mac, `Ctrl+K` on Windows):

- Select any code
- Tell it what to change
- Perfect for small tweaks and improvements

Built-in Terminal:

- Run commands without leaving Cursor
- Test your code
- Install new tools

File Explorer:

- See all your project files
- Create new files and folders
- Organize your work

Install Postman (For Testing Your Backend)

1. Open Cursor
2. Press `Cmd+Shift+X` (Mac) or `Ctrl+Shift+X` (Windows) to open Extensions
3. Search for "Postman"
4. Click "Install" on the official Postman extension
5. Sign in with a free Postman account

What Postman Does:

- Tests if your backend is working correctly
- Sends fake requests to your app
- Shows you what responses you get back
- Helps debug problems

Setup Conda (Python Environment Manager)

What is Conda?

Think of Conda like separate toolboxes for different projects. Each project gets its own toolbox so tools don't get mixed up.

1. Go to anaconda.com
2. Download Anaconda (it's free)
3. Install it following the setup wizard
4. Open Cursor's terminal and create your toolbox:

```
```bash
conda create -n vcp python=3.11
conda activate vcp
```
```

Setup NPM and VITE (For Your Frontend)

What are NPM and Vite?

- **NPM:** Like an app store for code tools
- **Vite:** Makes your frontend load super fast

1. Go to nodejs.org and download Node.js
2. Install it following the setup wizard
3. Check it worked by typing in terminal:

```
```bash
node --version
```

```
npm --version
```

```
...
```

4. Install Vite:

```
```bash
```

```
npm install -g vite
```

```
...
```

Install DBeaver (Database Viewer)

What is DBeaver?

It's like a file explorer, but for databases. You can see all your app's data in tables.

1. Go to dbeaver.io
2. Download the Community Edition (it's free)
3. Install following the setup wizard
4. We'll connect it to your database later

Setup Local PostgreSQL Database (Using Docker)

What is Docker Compose?

It's a tool that lets you run a PostgreSQL database on your computer without installing it directly. Perfect for development!

Prerequisites:

- Install Docker Desktop from docker.com
- Make sure Docker is running

Start Your Local Database:

1. Open Terminal in Your Project Folder:

```
```bash
```

## Navigate to your project root (where you extracted the template)

```
cd your-project-folder
```

```
...
```

2. Navigate to Backend Directory:

```
```bash
```



```
cd backend/docs
```

```
...
```

Note: The ``docker-compose.yml`` file is located in ``backend/docs/`` directory in your template project.

3. Start the Database:

```
```bash
```

```
docker-compose up -d
```

```
...
```

This creates:

- **PostgreSQL Database** on port 5432
- **pgAdmin** (web interface) on port 8080

### 4. Verify It's Running:

```
```bash
```

```
docker-compose ps
```

```
...
```

You should see both services running

Access Your Database:

Option 1: pgAdmin (Web Interface)

- Open browser: `http://localhost:8080`
- Login: ``admin@example.com`` / ``admin123``
- Add server connection:
- Host: ``postgres``
- Port: ``5432``
- Database: ``vcp_database``
- Username: ``vcp_user``
- Password: ``vcp_password``

Option 2: DBeaver

- Create new PostgreSQL connection
- Host: ``localhost``
- Port: ``5432``
- Database: ``vcp_database``
- Username: ``vcp_user``
- Password: ``vcp_password``

■■■ ****Database Connection Details****

Development Database (Local with Docker):

- **Database Type:** PostgreSQL
- **Host:** localhost
- **Port:** 5432
- **Database Name:** vcp_database
- **Username:** vcp_user
- **Password:** vcp_password
- **Connection URL:** `postgresql://vcp_user:vcp_password@localhost:5432/vcp_database`

pgAdmin Web Interface:

- **URL:** http://localhost:8080
- **Email:** admin@example.com
- **Password:** admin123

Environment Configuration (.env file):

```
```env
```

## Database Configuration

```
DATABASE_URL = "postgresql://vcp_user:vcp_password@localhost:5432/vcp_database"
```

## Flask Configuration

```
SECRET_KEY = "your-secret-key-here"
```

```
JWT_SECRET_KEY = "your-jwt-secret-here"
```

```
DEBUG = "False"
```

## CORS Configuration

```
CORS_ORIGINS = "http://localhost:3000,http://localhost:5173"
```

## JWT Configuration

```
JWT_ACCESS_TOKEN_EXPIRES = "3600"
```

```
JWT_REFRESH_TOKEN_EXPIRES = "2592000"
```

## Port Configuration

```
PORT = "5002"
```

```
```
```

Quick Connection Test:

```
```bash
```

## Test database connection

```
conda activate vcp
cd backend
python -c "
from config import Config
print('Database URL:', Config.SQLALCHEMY_DATABASE_URI)
from main import create_app
app = create_app()
print('■ Database connection successful!')
"
```
```

Useful Commands:

```
```bash
```

## Navigate to the docker-compose file location first

```
cd backend/docs
```

## Start database

```
docker-compose up -d
```

## Stop database

```
docker-compose down
```

## View logs

```
docker-compose logs postgres
```

## Reset database (deletes all data)

```
docker-compose down -v
```

```
docker-compose up -d
```

```
...
```

### Benefits:

- **Easy Setup:** One command starts everything
- **Isolated:** Doesn't affect your system
- **Consistent:** Same setup for all developers
- **Includes pgAdmin:** Web-based database management

### ■ Using This in Workshop Activities:

#### Workshop Activity 5 (Database Setup):

1. Download and extract the project template
2. Start the local database: ``cd backend/docs && docker-compose up -d``
3. Your Flask app will automatically connect to the local database
4. Use pgAdmin or DBeaver to view your tables and data

#### Workshop Activity 6 (Backend Development):

- The database runs in the background while you build your API
- Test your endpoints knowing the database is always available
- View real data being created through pgAdmin interface

#### Workshop Activity 7 (Integration):

- Frontend and backend connect seamlessly with the local database
- Perfect for testing the complete application flow
- No external dependencies or internet connection required

### During Development:

- Keep the database running throughout your workshop
- Use ``docker-compose logs postgres`` to debug connection issues
- Reset data between tests with ``docker-compose down -v && docker-compose up -d``

### ■■ Important Notes:

- Start the database BEFORE running your Flask application
- The database persists data between sessions
- Use the reset command if you want to start with a clean database

## Your Project Template Walkthrough

### What is the Project Template?

It's a pre-built starting structure for your web app that includes all the basic files and setup you need. Instead of starting from scratch, you get a head start!

## Download the Template:

### 1. Get the Template as ZIP:

- Go to: <https://github.com/Amazingct/MASTERCLASS-VCTP.git>
- Click the green **"Code"** button
- Select **"Download ZIP"**
- Save it to your Desktop or Projects folder

### 2. Extract the Files:

- Find the downloaded `MASTERCLASS-VCTP-main.zip` file
- Double-click to extract it
- Rename the folder to your app name (e.g., `my-awesome-app`)

### 3. Open in Cursor:

- Open Cursor
- Click **"Open Folder"**
- Select your extracted project folder
- You'll see the complete project structure!

## What's Inside the Template:

...

your-app-folder/

■■■■ backend/ # Server-side code (Python/Flask)

■ ■■■■ config.py # App settings

■ ■■■■ requirements.txt # Python packages needed

■ ■■■■ utils/ # Helper tools (file upload, email)

■ ■■■■ docs/ # Development tools

■ ■■■■ docker-compose.yml # Local database setup

■■■■ resources/ # Project documents and guides

■ ■■■■ masterclass.md # This guide!

■ ■■■■ scripts.sh # Helpful commands

■■■■ .cursor/rules/ # AI assistant guidelines

...

## Why Use This Template:

- ■ **Saves Time:** No need to set up basic structure
- ■ **Best Practices:** Follows professional coding standards
- ■ **AI-Ready:** Includes rules to help Cursor assist you better
- ■ **Production-Ready:** Built for real-world deployment

- ■ **Tested:** All basic functionality works out of the box

#### **Next Steps:**

- Once you have the template, we'll customize it for your specific app
  - The AI will help you modify it based on your app idea
  - Everything is organized to make building easier!
- 

## PLANNING YOUR APP

### Choosing What to Build

#### **\*\*WORKSHOP ACTIVITY 1: Coming Up with App Ideas\*\***

**Goal:** Think of a simple, useful app that you can build in 2 days

#### **What Makes a Good Beginner App:**

- ■ Solves a real problem you or others have
- ■ Can be built in 2 days with AI help
- ■ Has 3-5 main features (not too complicated)
- ■ Needs user accounts (login/signup)
- ■ Uses our tech stack (React + Flask + PostgreSQL)

#### **Step-by-Step Process:**

##### **1. Think of Ideas (10 minutes)**

- Write down 1-2 app ideas
- Think about problems you face daily
- Consider what would make your life easier

##### **2. Vote on Best Idea (10 minutes)**

- Share ideas with the group
- Vote on the most interesting/useful one
- Pick the winner!

#### **Great Example App Ideas:**

- **Personal Expense Tracker:** Track where you spend money
- **Study Buddy:** Create and share flashcards

- **Recipe Organizer:** Save and organize your favorite recipes
- **Workout Logger:** Track your exercise progress
- **Book Reading List:** Keep track of books you want to read

**Remember:** Start simple! You can always add more features later.

---

## Creating Your App Plan (PRD - Product Requirement Document)

### What is a PRD?

Think of it as a detailed recipe for your app. It tells you exactly what to build, how it should work, and what it should look like.

## **\*\*WORKSHOP ACTIVITY 2: Writing Your App Plan with AI\*\***

**Goal:** Use AI to create a complete plan for your app, including a step-by-step task list

### What Your Plan Will Include:

- **App Description:** What your app does and why people will love it
- **Database Design:** What information your app needs to store
- **Page List:** Every screen/page your app will have
- **Tech Stack:** The tools we'll use to build it
- **Design Style:** Colors, fonts, and overall look
- **User Journey:** Step-by-step how people will use your app
- **Task Breakdown:** Exactly what to build and in what order

### Step-by-Step Process:

#### 1. Write Down Your Idea Clearly (5 minutes)

- Describe your app in 2-3 sentences
- List the main things users can do
- Note who your target users are

#### 2. Expand Your Idea with AI (10 minutes)

- Use prompts from `@resources/prompts.md` to add details
- Ask AI to suggest additional features
- Make sure scope is realistic for 2 days

#### 3. Generate Complete Plan (15 minutes)

- Use AI to create your full PRD document
- Include database design and page layouts
- Get a complete task breakdown organized by phases

#### 4. Review and Approve (15 minutes)

- Read through the entire plan
- Make sure everything makes sense
- Ensure you can realistically build it in 2 days
- Get group agreement on the final plan

#### Development Phases in Your Plan:

1. **Phase 1:** Build all pages with fake data
2. **Phase 2:** Create database and backend
3. **Phase 3:** Connect frontend to backend
4. **Phase 4:** Test and polish everything

#### Success Checklist:

- ☐ App idea is clearly described
  - ☐ Database design is complete
  - ☐ All pages are planned out
  - ☐ User journey is mapped
  - ☐ Tasks are broken into small, doable steps
  - ☐ Timeline fits in 2 days
  - ☐ Everyone agrees on the plan
- 

## BUILDING YOUR APP

**Important Note:** Now that you have your plan, stick to it! Your PRD and task list will guide every step of building your app.

---

### **\*\*WORKSHOP ACTIVITY 3: Creating Your Home Page\*\***

**Goal:** Build a beautiful landing page with working signup/signin forms (using fake data for now)

**What You'll Build:**



- A homepage that explains what your app does
- Sign up form (doesn't save data yet)
- Sign in form (doesn't check passwords yet)
- Professional, clean design
- Mobile-friendly layout

### Step-by-Step:

#### 1. Generate Your Landing Page (20 minutes)

- Use AI prompts from `@resources/prompts.md`
- Ask for a landing page based on your PRD
- Include your app's value proposition
- Add call-to-action buttons

#### 2. Test and Improve (10 minutes)

- Click all buttons and forms
- Check how it looks on different screen sizes
- Make sure the message is clear
- Adjust colors and fonts to match your brand

### Success Criteria:

- [ ] Page explains your app clearly
  - [ ] Forms look professional and work (even with fake data)
  - [ ] Design matches your planned style
  - [ ] Everything works on mobile and desktop
  - [ ] Users would want to sign up after seeing this page
- 

## **\*\*WORKSHOP ACTIVITY 4: Building All Your App Pages (with Fake Data)\*\***

**Goal:** Create every page your app needs using placeholder content, so you can see the complete user experience

### What You'll Do:

Build each page listed in your PRD with fake/sample data. This lets you test the user experience before connecting to real data.

### Step-by-Step:

#### 1. Build Each Main Page (45-60 minutes total)

- Work on one page at a time
- Use fake data for everything (sample text, numbers, images)
- Focus on layout and user experience, not fancy animations
- Make sure every feature from your PRD has a page or section

## 2. Test User Flow (30 minutes)

- Click through your entire app like a real user would
- Make sure navigation makes sense
- Check that all buttons and forms do something
- Ensure the experience feels smooth and logical

### Example Pages You Might Build:

- Dashboard/Home (main app screen)
- Profile page
- Settings page
- Main feature pages (based on your specific app)
- Add/Create pages
- List/View pages

### Keep It Simple:

- Use basic, clean layouts
- No complex animations yet
- Focus on functionality over fancy effects
- Make sure users understand what each page does

### Success Criteria:

- [ ] Every feature from your PRD has a corresponding page
  - [ ] You can navigate through the entire app
  - [ ] User flow feels logical and smooth
  - [ ] Pages look clean and professional
  - [ ] Everything works with fake data
- 

## **\*\*WORKSHOP ACTIVITY 5: Setting Up Your Database\*\***

**Goal:** Create the database structure that will store all your app's information

### **What You'll Do:**

Set up tables and relationships for all the data your app needs, based on your PRD's database design.

## Step-by-Step:

### 1. Create Database Models (20 minutes)

- Build models for each type of data (users, posts, etc.)
- Include relationships between different data types
- Add timestamps (when created, when updated)
- Make sure models support all features in your PRD

### 2. Use the Migration System (10 minutes)

- **Always use our custom migration script** - never use Flask-Migrate directly
- The migration script makes database changes easy and safe

```
```bash
```

Create and apply your database changes

```
python migrate.py "Create initial database tables"
```

Check what migrations exist

```
python migrate.py --status
```

Apply any pending changes

```
python migrate.py --upgrade
```

```
```
```

### 3. Check Your Database (10 minutes)

- Run your app to make sure database starts correctly
- Open DBeaver to see your tables
- Verify all tables and relationships are correct

## Database Best Practices:

- Every table should have an ID number
- Include created\_at and updated\_at dates
- Use clear, descriptive names
- Plan for future growth

## Why We Use Our Migration System:

- Safer than standard Flask migrations
- Clearer error messages
- One command does everything

- Easy to understand what's happening

**Success Criteria:**

- [ ] All data types from PRD have database tables
  - [ ] Relationships between tables are set up correctly
  - [ ] Database starts without errors
  - [ ] You can see tables in DBeaver
  - [ ] Migration system is working
- 

## **\*\*WORKSHOP ACTIVITY 6: Building Your Backend (The Brain of Your App)\*\***

**Goal:** Build the server-side logic that handles all your app's functionality, organized by feature groups

**What You'll Do:**

Create the backend endpoints (API routes) that your frontend will use to get and save data. You'll work on related features together as "collections."

**The Step-by-Step Pattern:**

**For Each Collection of Features:**

**1. Build Related Endpoints (30-45 minutes per collection)**

- Start with User Management (register, login, profile)
- Create all CRUD operations (Create, Read, Update, Delete)
- Add proper error handling and validation
- Use consistent response formats

**2. Test in Postman (15 minutes per collection)**

- Create a Postman collection for these endpoints
- Test every endpoint thoroughly
- Try good data and bad data
- Make sure error messages are helpful
- Save examples of successful requests

**3. Connect to Frontend (30 minutes per collection)**

- Replace fake data with real API calls
- Add loading messages while waiting for data
- Handle errors gracefully (show user-friendly messages)

- Test the complete user experience

#### 4. Move to Next Collection

- Repeat the same pattern for next group of features
- Keep going until all PRD features are built

#### Example Collections:

- **User Management:** Register, login, profile, logout
- **Main Features:** Based on your specific app (posts, expenses, recipes, etc.)
- **Data Management:** Create, edit, delete your app's main content
- **Extra Features:** Search, filtering, user preferences

#### Development Pattern:

...

Build Endpoints → Test in Postman → Connect to Frontend → Repeat for Next Collection

...

#### Key Success Principles:

- **One Collection at a Time:** Finish each group completely before moving on
- **Test Everything:** Every endpoint must work perfectly in Postman first
- **Connect Immediately:** Link frontend to backend after each collection
- **Test User Experience:** Make sure the complete flow feels good

#### Success Criteria for Each Collection:

- [ ] All endpoints work correctly in Postman
  - [ ] Frontend successfully uses the new endpoints
  - [ ] Error handling works (shows helpful messages)
  - [ ] Loading states work (users see progress)
  - [ ] User experience feels smooth and professional
- 

## **\*\*WORKSHOP ACTIVITY 7: Connecting Everything Together\*\***

**Goal:** Make sure your complete app works perfectly from start to finish

#### What You'll Do:

Test your entire app like a real user would, fix any remaining issues, and polish the experience.

#### Step-by-Step:

##### 1. End-to-End Testing (30 minutes)

- Go through every user journey from your PRD
- Try to break things on purpose (enter weird data, click rapidly, etc.)
- Test on different devices and screen sizes
- Make sure error messages are helpful and friendly

## 2. Polish User Experience (15 minutes)

- Check loading times (should feel fast)
- Ensure loading messages appear when needed
- Verify responsive design works on mobile
- Make final tweaks to colors, spacing, and text

## 3. Final Bug Fixes (15 minutes)

- Fix any issues you discovered during testing
- Clean up any messy code
- Make sure everything follows your design standards

### Things to Test:

- **User Registration:** Can new users sign up successfully?
- **User Login:** Do existing users get logged in correctly?
- **Main Features:** Does every key feature work as planned?
- **Error Handling:** What happens when things go wrong?
- **Mobile Experience:** Does it work well on phones?
- **Data Persistence:** Is information saved correctly?

### Success Criteria:

- ☐ All PRD features work correctly
  - ☐ Complete user journeys feel smooth
  - ☐ App looks professional and polished
  - ☐ Works well on different devices
  - ☐ Error handling is user-friendly
  - ☐ App feels fast and responsive
  - ☐ You'd be proud to show this to others
- 

## Tips for Fixing Problems

### When Your Frontend Has Issues:

- Open browser developer tools (F12)
- Look at the Console tab for error messages

- Check Network tab to see if API calls are working
- Try refreshing the page
- Ask Cursor: "This error appeared, can you help me fix it?"

#### **When Your Backend Has Issues:**

- Check the terminal where your Flask app is running
- Look for error messages in red text
- Test individual endpoints in Postman
- Make sure your database is running
- Ask Cursor to review your code

#### **When Frontend and Backend Don't Talk:**

- Check that API URLs are correct
- Verify request and response formats match
- Look for CORS errors (Cross-Origin Resource Sharing)
- Make sure authentication is working
- Test the same request in Postman first

#### **General Problem-Solving:**

1. **Read the Error Message:** Error messages usually tell you what's wrong
2. **Ask Cursor for Help:** Copy the error and ask for assistance
3. **Test Small Parts:** Isolate the problem to find the exact issue
4. **Take Breaks:** Sometimes stepping away helps you see the solution
5. **Use Google:** Many programming problems have been solved before

---

## PUTTING YOUR APP ONLINE

**Final Step:** Once everything works perfectly on your computer, it's time to share it with the world!

---

### **\*\*WORKSHOP ACTIVITY 8: Getting Ready to Launch\*\***

**Goal:** Package your app and deploy it to the internet so others can use it

**What You'll Do:**

Create Docker containers (packages) for your app, upload to GitHub, and deploy to a cloud service.

### Step-by-Step:

#### 1. Create Docker Packages (15 minutes)

- Use AI to generate Docker files for both frontend and backend
- These files tell the internet server how to run your app
- Test that packages build correctly on your computer

#### 2. Upload to GitHub (15 minutes)

- Create a GitHub repository (online code storage)
- Upload all your code
- This lets deployment services access your code

#### 3. Deploy to DigitalOcean (30 minutes)

- Connect your GitHub to DigitalOcean App Platform
- Configure environment settings
- Deploy both frontend and backend
- Monitor deployment progress

#### 4. Test Everything (15 minutes)

- Visit your live app
  - Test all features work online
  - Check performance and speed
  - Celebrate your success! ■
- 

## Creating Docker Files (App Packages)

### What is Docker?

Think of Docker like a shipping container for your app. It packages everything your app needs to run, so it works the same everywhere.

### Backend Docker File (Flask App):

```
```dockerfile
```

Use Python as the base

```
FROM python:3.11-slim
```


Set working directory

WORKDIR /app

Install Python dependencies

COPY requirements.txt .

RUN pip install -r requirements.txt

Copy your app code

COPY . .

Expose port 5000

EXPOSE 5000

Start your app

CMD ["python", "main.py"]

...

Frontend Docker File (React App):

```dockerfile

## Build stage

FROM node:18-alpine

WORKDIR /app

## Install dependencies

COPY package\*.json ./

RUN npm install

## Build the app

COPY . .

RUN npm run build

# Serve stage

FROM nginx:alpine

COPY --from=0 /app/dist /usr/share/nginx/html

...

## Upload Code to GitHub

### Why GitHub?

GitHub is like Google Drive for code. It stores your code safely and lets deployment services access it.

### Steps:

#### 1. Initialize Git (version control):

```
```bash
```

```
git init
```

```
git add .
```

```
git commit -m "Initial version of my app"
```

...

2. Create GitHub Repository:

- Go to github.com
- Click "New Repository"
- Name it (same as your project)
- Make it public or private

3. Upload Your Code:

```
```bash
```

```
git remote add origin [your-github-url]
```

```
git push -u origin main
```

...

## Deploy to DigitalOcean

### What is DigitalOcean App Platform?

It's a service that takes your code and makes it available on the internet automatically.

### Steps:

### 1. **Connect GitHub Account:**

- Go to [DigitalOcean App Platform](https://cloud.digitalocean.com/apps)
- Create account if needed
- Connect your GitHub account
- Select your repository

### 2. **Configure Your App:**

- Choose your main branch (usually "main")
- Set build settings (DigitalOcean usually detects these automatically)
- Set environment variables (database URLs, secret keys)

### 3. **Deploy:**

- Click "Create Resources"
- Watch the deployment logs
- Wait for "Deployment successful" message
- Get your live app URL!

## Setting Up Your Website Address

Having your own domain name (like myawesomeapp.com) makes your app look professional!

## Buying a Domain Name

### Step-by-Step:

#### 1. **Go to GoDaddy.com:**

- Visit [godaddy.com](https://godaddy.com)
- Create an account or sign in

#### 2. **Search for Your Domain:**

- Type your desired domain name
- Try variations if your first choice is taken
- Good extensions: .com, .app, .dev, .io

#### 3. **Purchase Your Domain:**

- Add to cart
- Choose 1-2 year registration
- Consider domain privacy (hides your personal info)
- Complete payment

## Connecting Your Domain to DigitalOcean

## Why Do This?

Instead of a complicated URL like `app-12345.ondigitalocean.app`, you'll have `yourdomain.com`.

### Steps:

#### 1. Get DigitalOcean DNS Settings:

- In DigitalOcean, go to Networking → Domains
- Click "Add Domain"
- Enter your domain name
- Copy the nameserver addresses (like ns1.digitalocean.com)

#### 2. Update GoDaddy Settings:

- In your GoDaddy account, go to "My Products"
- Click "DNS" next to your domain
- Find "Nameservers" section
- Change to "Custom" nameservers
- Enter the DigitalOcean nameservers
- Save changes

#### 3. Configure DNS Records:

- Back in DigitalOcean DNS manager
- Add A record: `@` points to your app's IP
- Add CNAME record: `www` points to your domain
- Save all settings

#### 4. Wait for Changes to Take Effect:

- DNS changes can take up to 48 hours
- Usually works within a few hours
- Use [whatsmydns.net](https://whatsmydns.net) to check progress

#### 5. Enable HTTPS (Security):

- DigitalOcean automatically provides SSL certificates
- Your site will work with <https://yourdomain.com>
- This makes it secure and trusted

## Testing Your Live App

### Final Checklist:

#### 1. Basic Functionality:

- [ ] App loads at your domain

- ☐ All pages work correctly
- ☐ Users can sign up and log in
- ☐ All main features function properly

## 2. Performance:

- ☐ Pages load quickly (under 3 seconds)
- ☐ Images and content appear correctly
- ☐ No broken links or missing resources

## 3. Security:

- ☐ HTTPS works (lock icon in browser)
- ☐ User data is protected
- ☐ Authentication works correctly

## 4. Mobile Experience:

- ☐ App works on phones and tablets
- ☐ Text is readable without zooming
- ☐ Buttons are easy to tap

## 5. User Testing:

- ☐ Ask friends/family to try your app
- ☐ Get feedback on usability
- ☐ Fix any confusing parts

## Congratulations! ■

Your app is now live on the internet! Share the URL with friends and family.

---

# BONUS MATERIALS

## Templates and Examples

- Ready-to-use code snippets
- Example PRD documents
- Sample app structures
- Common UI components

## Cursor Tips and Tricks

- Best prompts for different tasks
- How to get better AI responses
- Keyboard shortcuts that save time
- Advanced AI features

## Ready-to-Use Prompts

- Landing page generation
- Database model creation
- API endpoint building
- UI component design
- Bug fixing assistance

## App Planning Template

- Step-by-step PRD creation
- Feature prioritization guides
- User story templates
- Technical specification formats

## Best Practices for Vibe Coding

- Start simple, then improve
- Test everything as you build
- Keep user experience in focus
- Use AI effectively
- Organize your code well

## Common Mistakes to Avoid

- Building too many features at once
  - Not testing on mobile devices
  - Forgetting error handling
  - Making things too complicated
  - Skipping user testing
-

# Wrapping Up

## What You've Accomplished:

- ■ Learned vibe coding methodology
- ■ Set up a complete development environment
- ■ Planned and built a full-stack web application
- ■ Deployed your app to the internet
- ■ Set up a professional domain name

## What is Vibe Coding Really About?

Vibe coding isn't just about using AI tools - it's about creating applications that users genuinely enjoy using. You focus on the experience, iterate quickly, and use modern tools to bring ideas to life faster than ever before.

## Your Next Steps:

1. **Keep Building:** Now that you know the process, try building another app
2. **Learn More:** Explore advanced features and techniques
3. **Get Feedback:** Share your app and listen to users
4. **Join the Community:** Connect with other vibe coders
5. **Teach Others:** Share what you've learned

## Remember the Core Principles:

- **User Experience First:** Always think about how it feels to use your app
- **Iterate Quickly:** Build fast, test often, improve continuously
- **Use AI Wisely:** Let AI handle the repetitive work while you focus on creativity
- **Keep It Simple:** Simple solutions often work better than complex ones
- **Have Fun:** Coding should be enjoyable!

## You're Now a Vibe Coder! ■

The skills you've learned here will help you build any web application you can imagine. Start with simple ideas, master the process, and gradually take on bigger challenges.

Happy coding, and remember - the best apps are the ones that make people's lives a little bit better!