# FinTrack Development - Phase 1 & 2 Recap

\*\*Date:\*\* July 26, 2025

This document summarizes the foundational steps taken to set up the FinTrack project, explaining the purpose and necessity of each action.

---

## Phase 1: Building the Workshop (Sprint 0: Setup)

The initial phase focused on creating a professional, organized, and safe development environment before writing any product-specific code.

\* \*\*Tool Installation (Python, VS Code, PostgreSQL, Git):\*\* We installed our core machinery. Python is our main power tool for logic, VS Code is our workbench for writing code, PostgreSQL is our secure filing cabinet for data, and Git is our "time machine" for protecting our work.

\* \*\*Project Folder (`fintrack`):\*\* We created a dedicated, empty folder to keep our project organized and separate from all other files on the computer.

\* \*\*Git Initialization (`git init`):\*\* We activated our "time machine" within the project folder. This command tells Git to start tracking all changes, allowing us to save checkpoints and revert to previous versions if we make a mistake.

\* \*\*.gitignore File:\*\* We created this special file to give our "time machine" one crucial instruction: ignore the "digital sawdust." It tells Git to disregard temporary cache files (`\_\_pycache\_\_`) and our library folder (`venv`), keeping our project's saved history clean and professional.

\* \*\*Virtual Environment (`venv`):\*\* We created an isolated "bubble" just for this project. All Python libraries (like Flask) are installed inside this bubble, which prevents conflicts with other Python projects on the system. This is a critical practice for modern development.

\* \*\*"Hello, FinTrack!" App Run:\*\* This was our final system check. By running a minimal web app, we confirmed that Python, Flask, the terminal, and our virtual environment were all communicating correctly.

---

## Phase 2: Laying the Foundation (Sprint 1: Database)

With the workshop ready, we began building the foundational structure of the application itself.

\* \*\*Database Schema Design:\*\* We acted as architects first. We designed the blueprints for our two main "filing cabinets" (the `users` and `assets` database tables), defining what information each would hold \*before\* writing any code.

\* \*\*Installing Database Libraries:\*\* We installed `Flask-SQLAlchemy` and `psycopg2`, which act as the specialized connectors between our Python code and our PostgreSQL database.

\* \*\*Configuring the Database Connection:\*\* We updated `app.py` to give our application the exact address and password for our database. During this step, we performed real-world debugging to solve two common problems:

1. A `connection timeout` error, which taught us how to check and start the PostgreSQL server service on Windows.

2. An authentication error caused by the `@` symbol in a password, which taught us that special characters in connection strings must be handled correctly (URL encoding).

\* \*\*Creating Tables with Code (`db.create\_all()`):\*\* This was a key step demonstrating a modern development practice. Instead of manually creating tables in a database tool, we defined our table blueprints as Python classes in `app.py`. Then, we used a single command (`db.create\_all()`) to have our code automatically build the tables in PostgreSQL.

\* \*\*Verification in pgAdmin:\*\* We acted as quality inspectors. We used the pgAdmin tool to look inside our `fintrack\_db` database and visually confirm that the `user` and `asset` tables were created exactly according to our blueprints.