**1. Initialize Your GitHub Repository First**

1. **Create a new repo.**
   * **Add a descriptive name (e.g., WahyBot-studio) and a README.md..**
   * **Include a .gitignore (PowerPlatform, Node, or Python templates as needed).**
   * **Choose an appropriate license.**
2. **Define your branching strategy.**
   * **Main branch for stable code.**
   * **Feature branches (feature/ai-flow, feature/powerapp-ui) for isolated work.**
3. **Scaffold your project structure.**
   * **/azure-functions for serverless code**
   * **/powerapps for Canvas app exports**
   * **/docs for design docs and mockups**
   * **/powerbi for your future analytics files**

**Initializing GitHub first ensures version control, smooth collaboration, and an audit trail of every change.**

**2. Set Up Your Development Environment**

**Once your repo is live:**

* **Clone it locally and install CLI tools (Azure CLI, PAC CLI).**
* **Commit a basic “Hello World” Azure Function or Canvas app stub.**
* **Push to GitHub to verify CI/CD (you can add GitHub Actions later).**

**Establishing this foundation keeps your code organized and shareable from Day 1.**

**3. Integrate Power BI for Insights**

**After the core AI and Power Apps bits are underway:**

1. **Define the metrics you want to track (e.g., conversation count, user satisfaction).**
2. **Build a simple Power BI report.**
   * **Connect to your Azure Table, Cosmos DB, or Dataverse where you log chat transcripts.**
   * **Create visuals (line charts, KPIs) to monitor usage.**
3. **Store your Power BI .pbix file under /powerbi in GitHub.**

**By adding Power BI second, you tap into real-time insights without delaying your core development milestones.**

**Summary**

* **Start with GitHub to set up version control, collaboration, and your project scaffold.**
* **Next, build your AI/Power Apps stack (Azure Functions, Canvas App, custom connectors).**
* **Finally, layer on Power BI to analyze and visualize user interactions.**

**This sequence keeps your project organized and ensures you don’t circle back to set up essential infrastructure later.**

**i need day by day get hub step by step please and same task same day with pwr app]**

----------------------------------------------------------------------------------------------------------- **Day 1: Repo & Canvas App Initialization**

GitHub

* Navigate to https://github.com/new..
* Choose owner, set repo name WahyBot-studio.
* Initialize with README.md, select .gitignore (PowerPlatform), add MIT license.
* Clone locally and push the first commit.

Power Apps

* In Power Apps, select **Canvas app from blank**, name it “WahyBot Chat.”
* Add a **Text input** (txtPrompt) and a **Button** (btnSend).
* In the app’s **OnStart**, add:

powerfx

Set(chatHistory, []);

Deliverable: Public repo URL + Canvas app stub with controls.

**Day 2: UI Layout & Branching**

GitHub

* Create a branch:

bash

git checkout -b feature/ui-layout

* Scaffold directories:
* /powerapps
* /docs
* Commit and push.

Power Apps

* Sketch your main screen (use Figma or paper).
* Build layout in Canvas: add a **Gallery** (galChat), placeholders for messages.
* Add basic styling: background, cards, fonts.

Deliverable: feature/ui-layout branch + UI mockup screenshot.

**Day 3: Azure Function Stub & AI Resource**

GitHub

* In /azure-functions, run:

bash

func init chatHandler --dotnet

cd chatHandler

func new --template "HTTP trigger" --name ChatHandler

git add .

git commit -m "Initialize Azure Function stub"

git push origin feature/ui-layout

Power Apps

* In Azure Portal, create **Azure OpenAI** or **AI Builder** resource.
* Copy endpoint & key.
* Use Postman to send a test POST to your Function’s local URL.

Deliverable: Function stub + screenshot of successful API call.

**Adding a License in GitHub**

1. Go to **github.com/new** and fill in your repo details.
2. Under **“Initialize this repository with”**, select **“Choose a license”**.
3. Pick from the dropdown (e.g., MIT, Apache 2.0, GPL 3.0, BSD-3-Clause), then **Create repository**.
4. A LICENSE file is added at the root—no manual copy-pasting needed.

**2. Permissive vs. Copyleft Licenses**

| **License** | **Type** | **Main Terms** |
| --- | --- | --- |
| MIT | Permissive | Use, modify, distribute—even commercially—only requirement: include the original copyright & license text【11^】. |
| Apache 2.0 | Permissive | Similar to MIT + explicit patent grant and patent-use protection; requires stating changes in NOTICE file【11^】. |
| BSD-3-Clause | Permissive | Like MIT, but adds a clause forbidding use of your name/logo to promote derived products without permission【11^】. |
| GPL 3.0 | Strong Copyleft | You can modify and distribute, but any distributed derivative must use GPL 3.0 too (source must stay open)【11^】. |

**3. Which License Is “Better”?**

* **MIT**: Best for maximum reuse with minimal fuss. Great for hackathon or early-stage projects.
* **Apache 2.0**: If you’re concerned about patent claims, this adds extra protection.
* **GPL 3.0**: If you want to ensure all derivatives remain open source.
* **BSD-3-Clause**: When you want permissive terms plus a simple name-use restriction.

For most rapid-prototyping and hackathon demos, **MIT** is the go-to choice: it’s short, clear, and widely understood.

**4. After Choosing a License**

* Review any external dependencies’ licenses to ensure compatibility.
* Commit your LICENSE file and push.
* Mention the license in your README under a **License** section to make it unmistakable for contributors and users【7^】.

gitignore?

**Setting Up Your .gitignore**

A .gitignore file tells Git which files or folders to skip—keeping your repo clean of local settings, build artifacts, and secrets.

**1. Choose the Right Template**

When you click **Add .gitignore** during repo creation, pick the template that matches your stack:

* **VisualStudio** – for C#/.NET Azure Functions
* **Node** – for JavaScript/TypeScript Functions
* **Python** – for Python Functions or scripts
* **PowerShell** – if you automate with PS scripts

Choosing one ensures common build folders (like bin/, obj/, node\_modules/) are ignored automatically.

**2. Customize for Your Project**

Below is a starter .gitignore you can drop into your repo root **after** repo creation. Tweak paths as needed.

gitignore

# ----------------------

# Azure Functions / .NET

# ----------------------

local.settings.json # secrets & connection strings

bin/ # compiled binaries

obj/ # build output

.Functions.Cli/ # local Functions runtime data

# ----------------------

# Node/TypeScript

# ----------------------

node\_modules/ # dependencies

dist/ # transpiled code

# ----------------------

# Power Apps & Power Platform

# ----------------------

.powerapps.local/ # local app settings

env/ # custom environment files

Solution.zip # exported unmanaged solutions

# ----------------------

# General

# ----------------------

.vscode/ # VS Code settings

\*.log # log files

\*.user # IDE user prefs

.DS\_Store # macOS file metadata

Thumbs.db # Windows file cache

**3. How to Add or Update .gitignore**

1. If you skipped it at creation, in your repo root click **Add file → Create new file**, name it .gitignore.
2. Paste your chosen template + custom rules.
3. Commit directly to main (or open a PR if you have protected branches).