**Contents**

[**✅ Roadmap** 2](#_Toc198813980)

[1. Step 1: Core Architecture Mapping 2](#_Toc198813981)

[2. Step 2: Dev Environment Setup Plan 2](#_Toc198813982)

[3. Step 3: MVP Flow Mapping & Feature Prioritization 2](#_Toc198813983)

[#1 Core Architecture Mapping 3](#_Toc198813984)

[**🎛️ 1. Architecture Overview** 3](#_Toc198813985)

[**🔍 2. Module Breakdown** 4](#_Toc198813986)

[**🔄 3. Data & Control Flow** 4](#_Toc198813987)

[**🌐 4. Where AI Fits** 5](#_Toc198813988)

[#2 Dev Environment Setup Plan 6](#_Toc198813989)

[📘1. Blueprint & Scaffold for GitHub repo (folder structure) 7](#_Toc198813990)

[📘 2. Setup & Project Scaffold / Setup Instructions (README\_SETUP.md) 9](#_Toc198813991)

[#3 MVP Flow Mapping & Feature Prioritization 11](#_Toc198813992)

[“Risks & Mitigations” Section 11](#_Toc198813993)

[Append Next Roadmap Steps (4–6) 12](#_Toc198813994)

**✅ Roadmap**

## Step 1: Core Architecture Mapping

We’ll map the **internal system architecture** for the Copilot software—breaking it into modules aligned with your 3 core capabilities:

1. **Statistical Test Guidance Engine** (Decision Tree + Wizard).
2. **SPSS Code Generator** (AI Copilot for syntax).
3. **Plain-Language Output Explainer** (Narrative engine for interpretation).

This includes:

* Module breakdown
* How they connect (data flow, UI, AI logic)
* Where AI models come into play (prompt generation, result parsing)
* Which stack tools power each module. The

## Step 2: Dev Environment Setup Plan

Based on that architecture, I’ll sketch out the folder structure and package scaffolding we’ll use—React/Electron + Python backend integration.

## Step 3: MVP Flow Mapping & Feature Prioritization

To Create a flowchart of how a user goes from:

* Selecting a test → understanding it → generating SPSS syntax → interpreting results.

This flow will directly map to:

* Buttons, input fields, dropdowns, tabs, etc.
* What each UI component triggers behind the scenes.

# #1 Core Architecture Mapping

**🎛️ 1. Architecture Overview**

┌─────────────┐ ┌──────────────────┐ ┌──────────────┐

│ SPSS (or │◀────▶│ SPSS-Bridge │ ◀──▶ │ Python │

│ RStudio, │ │ Automation Layer │ │ Engine │

│ Jamovi) │ └──────────────────┘ └──────────────┘

│ (UI) │ ▲ ▲

└─────────────┘ │ │

▲ │ │

│ │ calls SPSS syntax │

│ UI-Automation │ │

│ overlay & clicks │ │

▼ ▼ │

┌───────────────────┐ ┌──────────────────┐ ┌─────────────────┐

│ Embedded Chat-Bar │ │ Node.js Backend │ │ AI Engine │

│ (Electron + │ │ & License Server│ │ (OpenAI/Claude)│

│ React + TS) │ └──────────────────┘ └─────────────────┘

└───────────────────┘ ▲ ▲

▲ ▲ │ prompts / parses │

│ │ │ │

│ │ ▼ │

│ │ ┌─────────────────┐ │

│ │ │ SQLite DB │◀─────────┘

│ │ │ (Project State, │

│ │ │ Chat History, │

│ │ │ Snapshots) │

│ │ └─────────────────┘

│ │

│ └─── local state(mode, user context, UI flags)

│

└─── user clicks / chat commands

**🔍 2. Module Breakdown**

|  |  |  |
| --- | --- | --- |
| Module | Responsibility | Tech Stack |
| 1. Embedded Chat-Bar UI | • Presents chat-style prompts & buttons• Displays progress, errors, interpretations | Electron + React + TypeScriptTailwind CSS, Framer Motion |
| 2. Node.js Backend & License | • Orchestrates user requests → AI → SPSS bridge• Validates subscription (Stripe & JWTs)• Routes data to/from SQLite | Node.js (TypeScript)Stripe API, JWT license server |
| 3. AI Engine Interface | • Builds & sends prompt to LLM• Parses LLM response into commands or narrative | OpenAI API (GPT-4) or Anthropic Claude via REST |
| 4. SPSS-Bridge Automation Layer | • Executes SPSS syntax (via Python Essentials)• Falls back to GUI automation (PyAutoGUI) | Python + SPSS Python EssentialsPyAutoGUI / AutoHotKey |
| 5. Python Engine | • Loads & runs SPSS scripts• Reads/writes .sav files• Returns raw tables & logs | Python 3.x |
| 6. SQLite Project Store | • Persists user context, chat history, cleaned-data snapshots, logs | SQLite (embedded) |

**🔄 3. Data & Control Flow**

1. **User Launch**
   * SPSS starts → Copilot icon appears → user clicks “Launch Copilot.”
   * Electron UI loads chat-bar, authenticates via Node.js license server.
2. **Context Collection**
   * Chat-bar prompts collect research info → stored in SQLite.
   * Node.js retains context and selected tests.
3. **Mode Selection**
   * User chooses Automated or Guided mode → state updated in UI + SQLite.
4. **Automated Assistant**
   * Node.js sends “run test X” command to AI interface.
   * AI builds prompt (“Based on context A, generate SPSS syntax for regression…”).
   * Python Engine executes syntax via SPSS Python Essentials.
   * Python returns raw output tables → Node.js formats & styles → UI displays.
   * Repeat for each test.
5. **Interactive Visual Guide**
   * Node.js instructs Python/UI to overlay highlights.
   * UI animates pulsing on target SPSS buttons.
   * User clicks → UI sends event back to Node.js → verifies via automation listener → next step prompt.
6. **Result Interpretation**
   * After all tests, Node.js sends tables + context to AI interface.
   * AI returns narrative (“The model explains 60% variance…”).
   * UI shows write-ups; on approval, Node.js can export to Word or insert into SPSS report.
7. **Export & Save**
   * User clicks “Export,” Node.js bundles tables + narratives into .docx/.pdf.
   * All chat & execution logs remain in SQLite for re-run or audit.

**🌐 4. Where AI Fits**

* **Prompt Generation**: Node.js → AI Interface
* **Result Parsing**: AI Interface → structured JSON
* **Narrative Output**: AI Interface → plain-text explanations
* **Error Clarifications**: AI Interface → real-time chat guidance

# #2 Dev Environment Setup Plan

The folder structure:

stat-copilot/

├── electron-app/ # Front-end UI (React + Electron)

│ ├── public/

│ └── src/

│ ├── components/

│ ├── styles/

│ └── index.tsx

├── backend/ # Node.js orchestration & license server

│ ├── src/

│ │ ├── controllers/

│ │ ├── routes/

│ │ └── server.ts

│ └── package.json

├── python-bridge/ # SPSS-integration scripts

│ ├── scripts/

│ └── requirements.txt

├── docs/ # Architecture diagrams & setup guides

│ ├── ARCHITECTURE.md

│ └── README\_SETUP.md

└── .gitignore

***When this is pushed:***

* **You** will have immediate access to each piece.
* **Future devs** can jump right in.
* **CI/CD** (build, lint, test) will be wired up automatically.

**Steps To Be Done:**

1. **Initialize the repo** and share the link with you.
2. **Write the setup guide** (docs/README\_SETUP.md) so your machine is ready in one command.
3. **Commit the empty folders** and configuration files—Electron boilerplate, Node.js starter, Python env file.

## 📘1. Blueprint & Scaffold for GitHub repo (folder structure)

stat-copilot/

├── electron-app/

│ ├── public/

│ │ └── index.html

│ └── src/

│ ├── components/

│ │ └── ChatBar.tsx

│ ├── styles/

│ │ └── tailwind.css

│ ├── index.tsx

│ └── main.ts

├── backend/

│ ├── src/

│ │ ├── controllers/

│ │ │ └── license.ts

│ │ ├── routes/

│ │ │ └── auth.ts

│ │ └── server.ts

│ └── package.json

├── python-bridge/

│ ├── scripts/

│ │ └── spss\_runner.py

│ └── requirements.txt

├── docs/

│ ├── ARCHITECTURE.md

│ └── README\_SETUP.md

├── .gitignore

└── README.md

**File/Code Stubs**

**Electron-app/public/index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Stat Copilot</title>

</head>

<body>

<div id="root"></div>

</body>

</html>

**Eectron-app/src/index.tsx**

import React from 'react';

import ReactDOM from 'react-dom';

import ChatBar from './components/ChatBar';

ReactDOM.render(<ChatBar />, document.getElementById('root'));

**Backend/Package.json**

{

"name": "stat-copilot-backend",

"version": "0.1.0",

"main": "src/server.ts",

"scripts": {

"dev": "ts-node src/server.ts",

"start": "node dist/server.js"

},

"dependencies": {

"express": "^4.17.1",

"stripe": "^8.0.0",

"jsonwebtoken": "^8.5.1"

},

"devDependencies": {

"typescript": "^4.0.0",

"ts-node": "^9.0.0"

}

}

## 📘 2. Setup & Project Scaffold / Setup Instructions (README\_SETUP.md)

**What I Need:**

* A computer (Windows or Mac)
* Internet connection

**Step 1: Install Node.js**

* Visit: [https://nodejs.org](https://nodejs.org/)
* Click "Download for Windows"
* Install using default settings

**Step 2: Install Git**

* Visit: <https://git-scm.com/downloads>
* Download and install Git for your system

**Step 3: Clone the Project (Later when you have GitHub)**

Open your terminal (Command Prompt on Windows, Terminal on Mac):

git clone https://github.com/YOUR\_USERNAME/stat-copilot.git

cd stat-copilot

**Step 4: Install Dependencies for Frontend**

cd electron-app

npm install

**Step 5: Install Dependencies for Backend**

cd ../backend

npm install

**Step 6: Install Python & SPSS Essentials (for developers only)**

* I’ll need Python 3.10+ installed.
* SPSS Python Essentials must be installed with SPSS.
* Add installation guide here later once dev tools are finalized.

**Step 7: Run the App**

**To run the frontend:**

cd electron-app

npm start

**To run the backend server:**

cd ../backend

npm run dev

# #3 MVP Flow Mapping & Feature Prioritization

**1. Deliverables**

* **Scope**: “Automated‑Assistant → t‑test workflow”
* **Artifacts**:
  + docs/MVP\_FLOW.md (flowchart + UI ↔ back‑end mapping)
  + A simple UI sketch (e.g. Chat‑Bar input → “Run t‑test” button → Results pane)
  + List of back‑end calls (e.g. runTTest(data, iv, dv))

**2. Estimated Duration & Dependencies**

* **Flow Mapping**: 1 day (no dependencies)
* **MVP\_FLOW Doc + UI Sketch**: 1 day (follows Flow Mapping)
* **Back‑end Stub Coding**: 2 days (requires Electron shell in place)

**3. Success Criteria**

* Electron app launches and prompts for variables.
* Clicking “Run t‑test” executes through the Python‑SPSS bridge for a sample dataset.
* Results appear in the chat‑bar with an auto‑formatted interpretation.

# “Risks & Mitigations” Section

* **Risks & Mitigations**
* **SPSS Scripting Quirks**
  + *Mitigation:* Build a PyAutoGUI fallback path for menu‑click automation.
* **LLM Rate Limits**
  + *Mitigation:* Cache common prompt templates and batch smaller requests.
* **UI Glitches on Windows**
  + *Mitigation:* Test Electron builds on your Windows 10 VM before each merge.

# Append Next Roadmap Steps (4–6)

**Step 4: User Testing & Feedback (2 weeks)**

* Recruit 5 early‑user students.
* Observe them running the t‑test MVP.
* Collect pain points and bug reports.

**Step 5: Alpha Release & Bug Fixes (1 week)**

* Publish an internal alpha build.
* Triage and fix critical bugs.
* Finalize documentation updates.

**Step 6: Institutional Pilot & Pricing (3 weeks)**

* Approach one university for a small‑scale pilot.
* Integrate their feedback into pricing tiers.
* Prepare marketing materials for student and institutional plans.