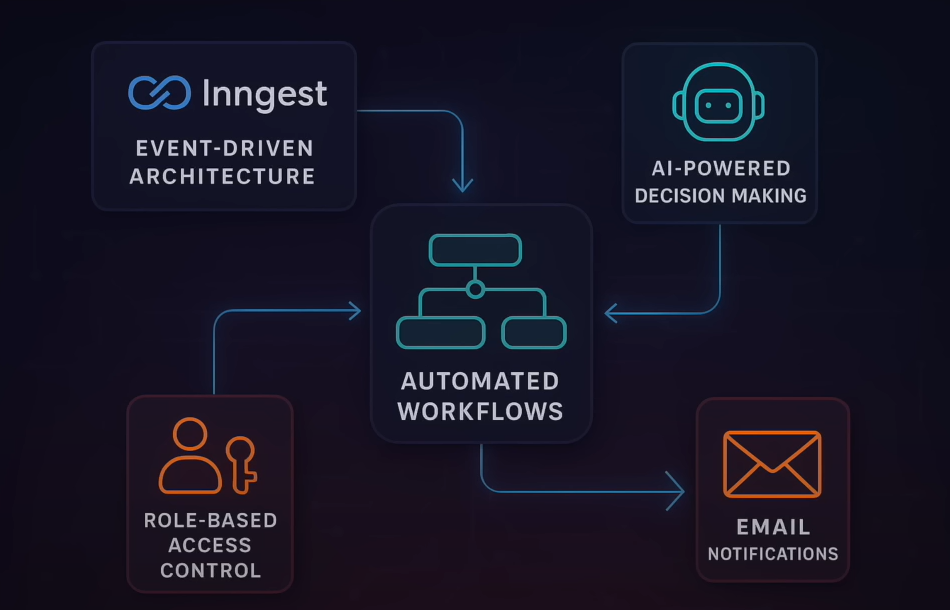
# Full Stack Ai-Agent



## AgentKit

* **AgentKit** is a **toolkit** that helps developers build **AI agents** easily — especially ones that can perform multiple steps, remember things, and act like assistants or bots.
*  Build **AI workflows** that remember what the user said earlier.
*  Connect your app to **LLMs** like **Gemini**, **OpenAI (ChatGPT)**, or **Claude**.
*  Handle multi-turn conversations and memory.
*  Integrate with APIs and other services (like calendars, emails, etc.).

Let’s say you want to build an agent that helps users book appointments:

* AgentKit helps you manage the conversation:  
  "I want a dentist appointment" → "What day?" → "Friday" → "Morning or afternoon?"
* It remembers each step, passes data to Gemini/OpenAI, and completes the task.

## Inngest

* **Inngest** is a **workflow automation tool** that lets your app **react to events** and **run background tasks**.
* Think of Inngest like a smart assistant behind the scenes that says:  
  “Oh, the user submitted a form? Okay, now let me start this workflow.”

## TechStack

* **Backend**: Node.js with Express
* **Database**: MongoDB
* **Authentication**: JWT
* **Background Jobs**: Inngest
* **AI Integration**: Google Gemini API
* **Email**: Nodemailer with Mailtrap
* **Development**: Nodemon for hot reloading

## Objectives

1. The system uses Ai(Gemini) to:
   * 1. Determine required skills
     2. Categorize the ticket
     3. Set priority
     4. Generate helpful Notes
2. Event Driven Architecture (via Inngest)
3. Ai-Based Decision-Making
4. Automated Workflows
5. Role-Based access control
6. Email Notifications

**Background Worker**

**Mail**

**Signup**

**MongoDB**

**Meta Data**

**LLM Gemini**

**AI Agent**

**Ticket Created**

**🔧 1. AI-Powered Ticket Processing**

This means AI is helping manage **support tickets** (problems reported by users).

**🔸 a. Automatic Ticket Categorization**

* AI reads what the user says in the ticket.
* It **decides what type of issue** it is — like "Login Issue", "Payment Failed", or "Bug Report".

✅ *Why it matters:* So you don’t need a human to read every ticket and tag it manually.

**🔸 b. Smart Priority Assignment**

* The AI checks how **urgent** or **important** the ticket is.
* For example:
  + “Payment failed” → High priority
  + “Can I change my profile color?” → Low priority

✅ *Why it matters:* High-priority issues are handled first.

**🔸 c. Skill-based Moderator Matching**

* The AI **assigns the ticket** to the right **moderator** based on the type of issue and the skills the moderator has.
* If it’s a billing problem, it won’t send it to a tech person — it sends it to a finance expert.

✅ *Why it matters:* Problems go to the right person directly, saving time.

**🔸 d. AI-generated Helpful Notes for Moderators**

* AI reads the ticket and writes a **summary or suggestions** for the moderator.
  + Example: “User’s payment failed during checkout on Chrome browser.”

✅ *Why it matters:* Moderators don’t need to read every word — they can act faster.

**👨‍🔧 2. Smart Moderator Assignment**

This is about sending the ticket to the best possible **support agent (moderator)**.

**🔸 a. Automatic Matching of Tickets to Moderators Based on Skills**

* The system checks:
  + What the ticket is about
  + Which moderator has the right skills
* Then it **automatically assigns** the ticket to that person.

**🔸 b. Fallback to Admin Assignment if No Matching Moderator Found**

* If no one has the needed skills or is available, the system sends it to the **admin** as a backup.

✅ *Why it matters:* No ticket is lost or stuck.

**🔸 c. Skill-based Routing System**

* All moderators have skills saved in the system (e.g., “billing”, “technical”, “product help”).
* The AI uses this list to send tickets to the correct person.

✅ *Why it matters:* Fast and accurate responses.

**👥 3. User Management**

This part handles **who can do what** in the system.

**🔸 a. Role-based Access Control (User, Moderator, Admin)**

* Users can only create tickets and see their status.
* Moderators can solve tickets.
* Admins can manage everything.

✅ *Why it matters:* Keeps the system secure and organized.

**🔸 b. Skill Management for Moderators**

* Admins or moderators can **update their skills**.
  + Example: “Now I can handle payment issues too.”
* The AI uses this info for future ticket assignments.

**🔸 c. User Authentication with JWT**

* JWT = JSON Web Token.
* It’s a secure way to **log users in and keep them authenticated**.
* Prevents unauthorized access.

✅ *Why it matters:* Only logged-in users can use the system.

**⚙️ 4. Background Processing**

This is about doing work **in the background**, without freezing the app.

**🔸 a. Event-driven Architecture using Inngest**

* Whenever something happens (like a new ticket is submitted), an **event** is created.
* Inngest listens to that event and runs background tasks (like assigning the ticket or sending emails).

✅ *Why it matters:* Everything runs smoothly, automatically.

**🔸 b. Automated Email Notifications**

* Users and moderators get **emails automatically** when:
  + A ticket is created
  + A response is added
  + A ticket is closed

✅ *Why it matters:* Keeps everyone informed without manual work.

**🔸 c. Asynchronous Ticket Processing**

* Some tasks (like calling AI, or assigning tickets) may take time.
* These are done in the **background**, without making the user wait.

✅ *Why it matters:* The app stays fast and responsive.

* **Mailtrap** is a tool that lets your app **send fake emails** to a **safe test inbox**, instead of sending real emails to customers.
* You can check how emails look and whether they’re being sent, **without spamming real users**.

**🔁 Common Backend Folder Structure (Node.js / Express)**

| **Folder Name** | **Purpose** |
| --- | --- |
| routes/ | Defines URL endpoints and connects them to controller functions. |
| controllers/ | Contains the business logic for each route (e.g., how to create, read, or delete a ticket). |
| models/ | Defines the data schema and database interaction (usually with Mongoose for MongoDB or Sequelize for MySQL). |
| middlewares/ | Functions that run before the request hits the controller (e.g., auth check, logging). |
| utils/ or helpers/ | Utility functions used across the app (e.g., email sender, date formatter, random ID generator). |
| services/ | Business logic layer—used when separating it from controllers for better abstraction (e.g., payment service, ML model service). |
| config/ | Stores config files like DB credentials, environment configs, third-party API keys. |
| validations/ | Input validation logic (like using Joi, express-validator, etc.). |
| constants/ | Static values/constants used across the project (e.g., role types, error codes). |
| jobs/ or workers/ | Background jobs or scheduled tasks (e.g., auto-closing old tickets, cron jobs). |
| logs/ | Application logs (e.g., Winston or Morgan logs can be saved here). |
| uploads/ or public/ | Static files like images, attachments, etc. |
| tests/ | Unit and integration tests (e.g., using Jest, Mocha). |
| docs/ | API documentation or system design notes. |
| scripts/ | One-time scripts (e.g., seeding DB, migrations). |
| ingest/ | (Optional for ML/AI) Scripts or code for loading and preprocessing data. |
| ml-models/ or ai/ | (Optional) Folder containing AI/ML-related models or inference code. |

**📁 Common Frontend Folder Structure (React/Vue/etc.)**

| **Folder Name** | **Purpose** |
| --- | --- |
| components/ | Reusable UI components (buttons, modals, forms, etc.). |
| pages/ or views/ | Page-level components mapped to routes (e.g., /dashboard, /login). |
| assets/ | Static files like images, icons, fonts, etc. |
| services/ | API calls (e.g., using Axios to communicate with backend). |
| context/ or store/ | For state management (e.g., using React Context, Redux). |
| hooks/ | Custom React hooks for logic reuse. |
| styles/ | Global or modular CSS/SCSS files. |
| constants/ | Frontend constants like roles, color codes, etc. |
| utils/ or helpers/ | Frontend utilities like input validation, formatting. |

**📁 Project Root Files**

| **File Name** | **Purpose** |
| --- | --- |
| .env | Stores environment variables (DB credentials, API keys). |
| package.json | NPM metadata, scripts, and dependencies. |
| README.md | Project description and setup guide. |
| .gitignore | Specifies what files/folders Git should ignore. |
| index.js / server.js | Entry point of the backend server. |
| webpack.config.js or vite.config.js | Build config for frontend (optional). |