# THE AI STUDIO APP

## WIRED TO YOUR CRM



MICHAEL REIMER

HIGHLEVEL GUIDES · VOLUME 10 · AI STUDIO

# BUILD AN APP IN AI STUDIO **WIRED TO YOUR CRM.**

This build turns one business process into a working application inside HighLevel. You describe the app, keep the private logic on the server, give it a database and user accounts, take payment through your HighLevel catalog, and send every form submission into a workflow. At the end you have a published app on your own domain with a backup of the code.

| 6 | 45 | 2 hr |
|---|---|---|
| PARTS | STEPS | TO BUILD |

**Michael Reimer**, Founder of CRM Pros · HighLevel Commander and Certified Admin

Updated November 3, 2026

WHAT YOU ARE BUILDING

## WHAT YOU ARE BUILDING **AND WHAT IT DOES.**

The finished system is a customer facing application built in AI Studio. The public pages render on the server so search engines can read them. Private work such as account lookups runs in Server Functions, and any API key stays in Secrets instead of the browser. The cloud backend holds the app's data, its user accounts and uploaded files. Checkout is handled by HighLevel using products from your catalog. Every form submission fires a HighLevel workflow, so leads land in the CRM and follow up starts on its own.

- ✓ A published AI Studio application on a custom domain
- ✓ Private credentials stored in Secrets and used only on the server
- ✓ A database, user sign in and file storage running in Cloud
- ✓ A checkout that sells products from your HighLevel catalog
- ✓ A workflow that fires on every AI Studio form submission
- ✓ A GitHub repository and a ZIP backup of the full codebase

BEFORE YOU START

## WHAT TO HAVE READY **BEFORE YOU START.**

Most of the delays in this build come from things that live outside AI Studio, so set them up first.

- A HighLevel subaccount with AI Studio enabled and the View & Manage AI Studio permission on your user
- The Manage Git & download code permission under **Settings**, then **Team**, then **Permissions**, then **AI Studio**
- A written description of the process: the public pages, the private steps, and any outside service it talks to
- The API keys or tokens for any outside service, ready to paste into Secrets
- Products and prices already created in the HighLevel product catalog
- A payment provider connected in HighLevel payment settings
- A domain you control, with access to its DNS, and a GitHub account

Do the parts in order. Secrets and Server Functions come before Cloud, and Cloud and payments both need a project built on **TanStack Start**, so start the project fresh rather than reusing an old React project.

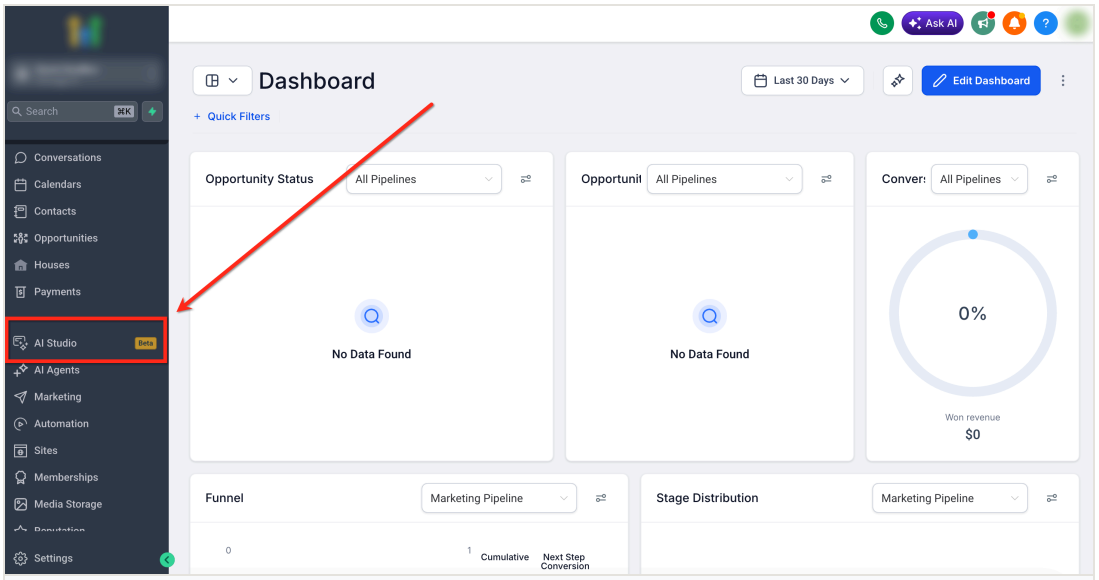
PART 01 OF 06

# 01 CREATE THE PROJECT **AND DESCRIBE THE WHOLE APP.**

AI Studio builds what you describe. If you only describe the look, you get a website. Describe the behavior, the private steps and the outside services, and you get an application with the right architecture from the first draft.

**01** Open your subaccount and click **AI Studio** in the left navigation.

**02** Create a new project from the home screen.




Describe the behavior, not just the look, and AI Studio builds the backend pieces with it.

**03** In the prompt box, describe the application, including the public pages and the private functionality, not just the design.

**04** Name every outside service, private credential, incoming webhook and public API the app needs. Example: an order status portal with a public landing page, a private account lookup that runs on the server, and an API endpoint that receives order status webhooks.

**05** Optionally attach files, images or a live URL for inspiration.

**06** Click **Generate** and wait for the first draft.

**07** Refine the draft with follow up prompts, use **Visual Edits** to click elements in the preview, or open the **Code Editor** for direct changes.

i New projects are built on **TanStack Start**, which gives you server side rendering, Server Functions, REST APIs and Secrets. An older React project needs **Project Settings**, then **Migrate to SSR** before Cloud or payments will work.

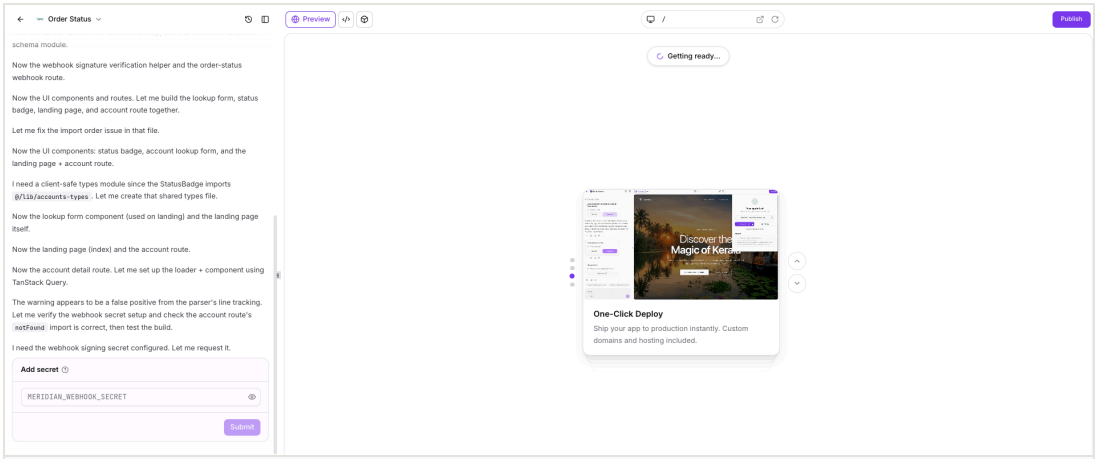
PART 02 OF 06

# 02 KEEP SECRETS SERVER SIDE **AND OFF THE PAGE.**

This is the part most people skip, and it is the part that keeps the app safe. Any API key goes into Secrets. Any work that uses that key runs in a Server Function. The browser only ever sees the result.

**01** Identify every private credential the integration needs.

**02** When AI Studio prompts for it, add each credential through the **Secrets** workflow and give it a clear, recognizable name.




Secrets stay on the server. Server Functions do the private work. REST APIs take calls from outside.

**03** Ask AI Studio to create a **Server Function** for each backend task the frontend calls.

**04** Check that the Server Function reads the credential from Secrets and returns only the information the frontend needs.

**05** For any outside system that must call your app, such as a webhook, ask AI Studio to create a **REST API** route under **/API/** with the HTTP method and expected payload defined.

**06** Add authentication or signature validation to that route when the outside system supports it.

**07** Test each Server Function and REST endpoint with both expected and unexpected input.

**08** Review the generated code for any extra validation you want.

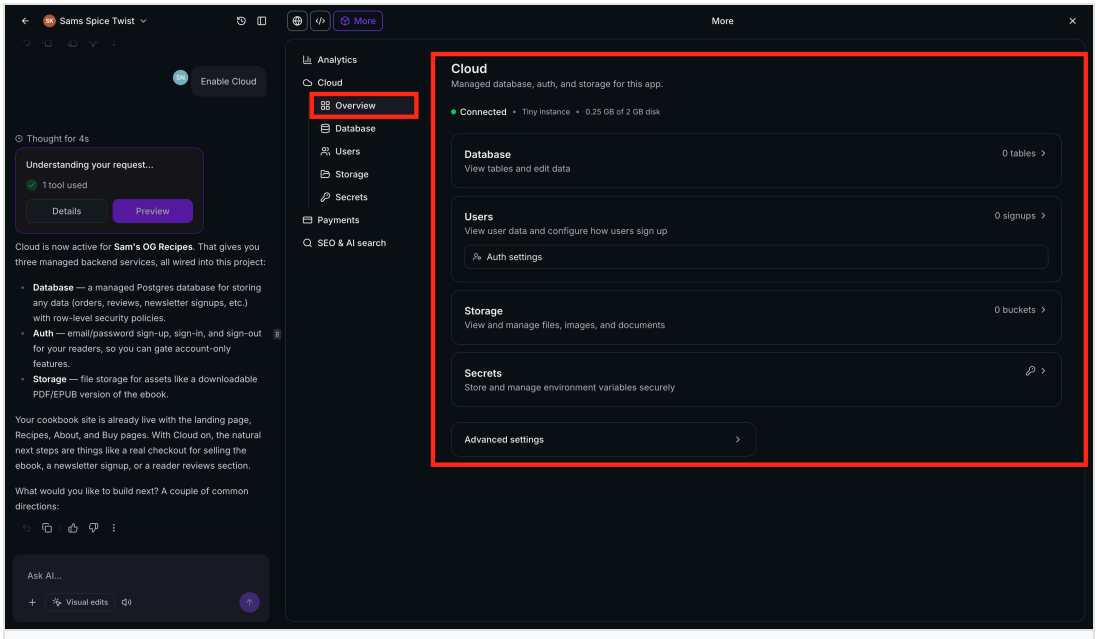
> **i** Never paste a credential into frontend code. If a Secret shows up anywhere in browser visible output, stop and move that logic into a Server Function.

PART 03 OF 06

# 03 TURN ON CLOUD AND GIVE THE APP DATA, USERS AND STORAGE.

Cloud is the managed backend that lives inside the project. Enabling it creates a database, a sign in service and file storage buckets. You manage all three from the Cloud dashboard without writing server code.

**01** Inside the project, click **More**, then **Cloud**.




The Cloud dashboard: database, users and storage in one place.

**02** Open **Overview** and click **Enable Cloud**, then keep working while provisioning finishes.

**03** Go to **Cloud**, then **Database**, click **RLS policies** and review the access policy on each table. Describe any change you need in the chat.

**04** Go to **Cloud**, then **Users**. Set the **Sign in methods** to email and password, set the **Minimum password length**, and leave **Autoconfirm email** off.

**05** Set the **Site URL** and add any **Redirect URLs** users return to after signing in.

**06** Go to **Cloud**, then **Storage** and click **New bucket**. Enter a lowercase name and choose **Private** for user uploads.

**07** Open the bucket's Storage page, enter an **Upload size limit** that fits your files, and click **Save**.

i Every project gets a free Tiny instance, 2 GB of database storage and 1 GB of file storage. Leave the instance size alone until the app proves it needs more. A storage increase cannot be reversed and keeps billing even while Cloud is paused.

PART 04 OF 06

# 04 CONNECT THE FORM AND FIRE A WORKFLOW ON EVERY SUBMISSION.

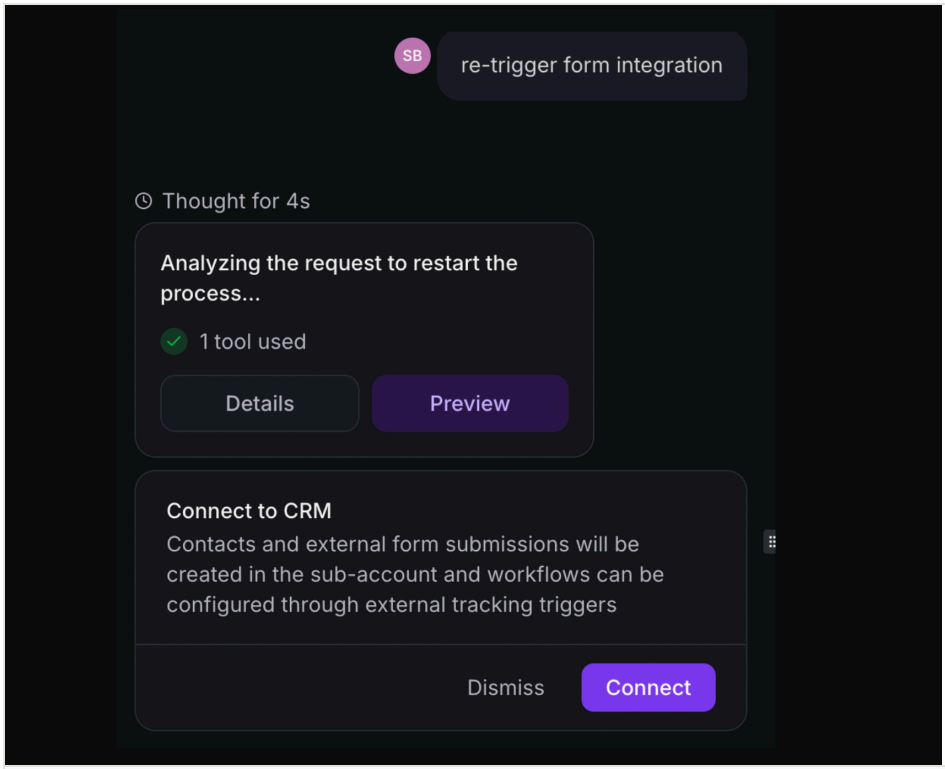
Cloud stores app data. It does not store leads. A form connected to CRM tracking sends each submission into the subaccount, and the AI Studio Form Submitted trigger starts the follow up.

**01** Ask AI Studio to add the lead form your process needs and to connect it to CRM tracking.

**02** If the form existed before, type **Retrigger form integration** in the chat, then connect the form to CRM tracking again.

**03** Go to **Automation**, then **Workflows** and create a new workflow.

**04** Click **Add New Trigger**, search for **AI Studio Form Submitted** and select it.




Filter by project and form so the right workflow fires.

**05** Add the **AI Studio project** filter and the **AI Studio form** filter so only this app's form starts the workflow.

**06** Add the actions you want, such as an email, an SMS, a tag or opportunity creation.

**07** Save the workflow and toggle it to **Published**.

i With no filters, the workflow fires on submissions from every AI Studio project in the subaccount. Add at least the project filter.

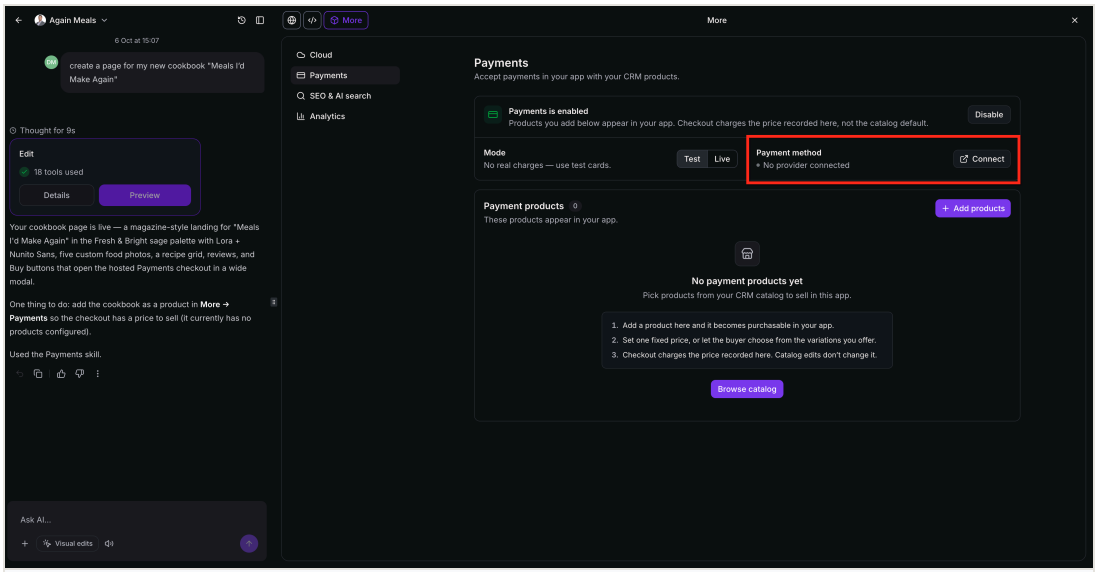
PART 05 OF 06

# 05 ADD PAYMENTS FROM YOUR HIGHLEVEL CATALOG.

HighLevel renders the checkout, so your app never touches card data. The trade is that every product, price and variation must already exist in the HighLevel product catalog. Payments start in Test mode.

**01** Inside the project, click **More**, then **Payments**.

**02** Click **Enable Payments**. If AI Studio already offered **Add payments to this app** during the build, you can click **Enable Payments** there instead.




Products come from the HighLevel catalog. The provider comes from HighLevel payment settings.

**03** In **Payment products**, click **Add products** or **Browse catalog**.

**04** Search for the product by name or SKU and select it.

**05** Click **Continue to pricing** and choose **Fixed price** or **Customer chooses variation**.

**06** Review the configuration and click **Add product**.

**07** Check the **Payment method** area shows your connected provider.

**08** Leave the project in **Test mode** until you have run a full checkout.

i Manual payment methods such as Cash on Delivery, shipping and pickup options, and product personalization fields are not supported in AI Studio payments.

PART 06 OF 06

# 06 PUBLISH ON YOUR DOMAIN **AND BACK UP THE CODE.**

Publishing puts the app on a preview URL first, then on your own domain. Git sync and the ZIP download give you a copy of the code outside HighLevel, which matters the day you want to hand the project to a developer.

**01** Click **Publish** in the top right corner, review the generated address, and confirm publication.

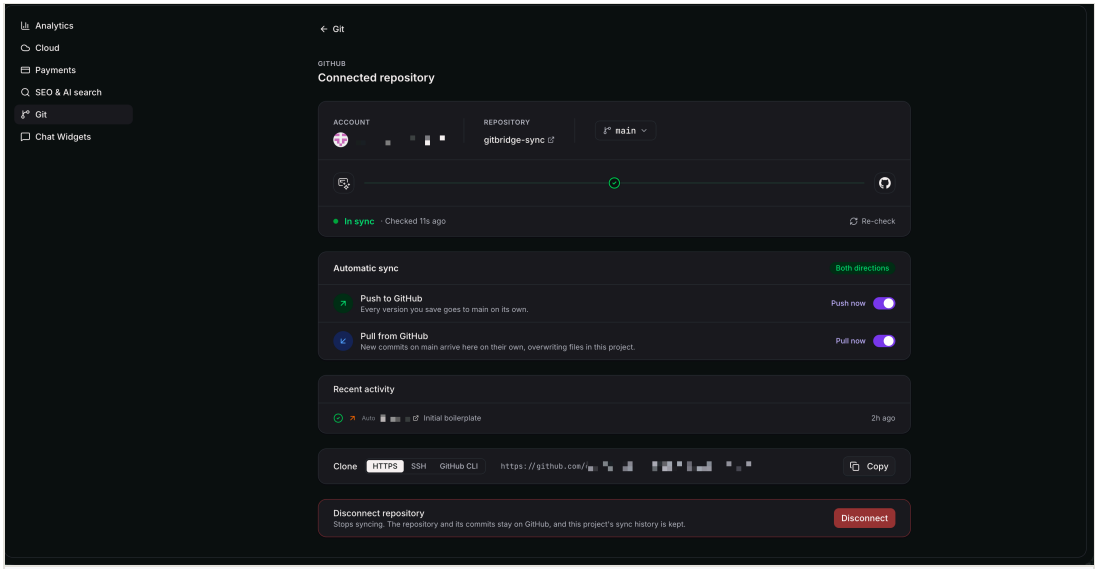
**02** Click **Publish** again, select **Add Custom Domain**, and enter your domain.

**03** Add the DNS records AI Studio shows at your domain provider, click **Verify DNS**, and wait for certificate provisioning to finish.

**04** Click **More**, then **Git**, select **GitHub**, click **+ Add connection**, and click **Continue on GitHub**.

**05** Install and authorize Lead Connector on GitHub, then return to AI Studio and let the connection finish. A private repository is created and the first version is pushed.

**06** Confirm **Automatic Push** and **Automatic Pull** are both enabled.




Every saved version pushes to GitHub on its own.

**07** Click **Code** in the top navigation, scroll to the bottom, and click **Download Codebase** to keep a ZIP snapshot.

**08** Open **More**, then **Payments** and switch from **Test mode** to **Live mode** once the test checkout in the next section passes.

i Git sync scans every push for credentials. A recognized secret blocks the push, which is one more reason to keep keys in Secrets and out of the code.

TEST IT

# TEST THE APP
# **BEFORE YOU CALL IT DONE.**

Run these on the published URL, not the preview, so you test what customers will see.

**01** Open the published domain in a private browser window. The public pages load and the page titles and headings read correctly.

**02** Trigger the private action, such as the account lookup. You get the expected result, and view source shows no API key anywhere in the page.

**03** Send a test request to the REST endpoint with a wrong payload. The app returns a clean error, not a crash.

**04** Sign up as a new user with a short password. The app rejects it until the password meets the minimum length. Then sign up correctly and confirm the user appears under **Cloud**, then **Users**.

**05** Upload a file as that user. It appears in the private bucket under **Cloud**, then **Storage**.

**06** Submit the lead form. The contact appears in the subaccount, and the workflow's **Execution Logs** tab shows the AI Studio Form Submitted trigger fired and the actions ran.

**07** Run a checkout in Test mode. Pricing matches the catalog and the order completes without a real charge. Then switch to Live mode.

WHEN A STEP FAILS

# WHEN SOMETHING FAILS **AND HOW TO FIX IT.**

Most failures in this build trace back to one of these.

### Cloud or Payments will not enable on an existing project

Both need a TanStack Start project. Open **Project Settings**, then **Migrate to SSR**, or start a new project.

### The form submits but the workflow never fires

Confirm the form is connected to CRM tracking, the project is published, and the test entry was sent after that publish. If the form predates the trigger, type **Retrigger form integration** in the chat and reconnect it. Then check the Execution Logs tab.

### The Git push is blocked

Git sync found a credential in the code. Move it into Secrets, reference it from a Server Function, and push again.

### The GitHub branch shows as diverged

Both sides have new commits. Select **Review**, then choose **Force push** to keep the AI Studio version or **New branch** to keep both sets of commits.

### Cloud paused on its own

The wallet balance ran out. The app keeps running for 48 hours while the agency owner is notified, then Cloud pauses without deleting data. Fund the wallet and Cloud wakes up automatically.

### The project environment goes stale or stops responding

Ask AI Studio in the chat to recover the environment. It restores the project and verifies the result before finishing.

FROM THE FIELD

# MICHAEL'S **TAKE.**



**Michael Reimer**

FOUNDER OF CRM PROS · HIGHLEVEL COMMANDER

The first thing I do on an app build is write the process down before I touch the prompt box. Which pages are public, which steps must stay private, and which outside system the app talks to. That one paragraph decides whether AI Studio gives you a brochure or an application. Then I add the Secrets before I ask for a single Server Function, because the architecture follows the credentials.

Where people go wrong is the frontend. They paste an API key into the page because it worked in the preview. It works right up until someone opens view source. Keep every key in Secrets, let a Server Function do the work, and test the endpoint before you publish. Git sync will block the push if you get this wrong, and that block is a gift.

This build pays off when the process you turned into an app used to need a person. An order lookup, a client portal, an intake with a paid step. On the backend side I stay on the free Tiny instance until the app earns an upgrade, and I watch Cloud usage, because the reserved database storage does not shrink and keeps billing while paused. For a prototype I would rather remove Cloud than pay for idle capacity.

GO DEEPER

# THE FEATURES **BEHIND THIS BUILD.**

Every feature this guide touches has its own page in the library, with the settings, limits and Michael's notes.

AI EMPLOYEE

AI Studio in HighLevel: Build sites and experiences with AI

AI EMPLOYEE

AI Studio full stack features: SSR, server functions, REST APIs & secrets

AI EMPLOYEE

AI Studio cloud backend. Manage data, users and storage in HighLevel

AI EMPLOYEE

Accept payments in AI Studio in HighLevel

AI EMPLOYEE

AI Studio Cloud pricing in HighLevel

AI EMPLOYEE

Download AI Studio codebase as ZIP in HighLevel

AI EMPLOYEE

How to use Git sync in HighLevel AI Studio

AI EMPLOYEE

AI Studio form submitted trigger for HighLevel workflows