# Replit: Build, Run, Share and Deploy an AI App

Replit is a browser-based workspace where you write code, run it, and publish it on the web, with no installation on your computer. This guide goes from your first running page to an AI-powered app with secure keys, saved data, and a deployed address. Plans and prices are deliberately not covered here; check Replit's own pricing page when you need them.

> **Running example.** A small "Dish Description Studio": a form takes a dish name, category and price, calls an AI model, and shows a short description. It is the same food-seller example used in the tokens guide.

## Part A. Basics

### 1. Words you need

| Word | Meaning |
|---|---|
| Repl | A project: its files, its running app and its settings. |
| Workspace | The browser editor: file tree, code editor, console and preview. |
| Console | Where the program prints messages and errors. |
| Preview | The running app shown inside the workspace. |
| Secret | A private value (such as an API key) stored outside the code. |
| Deployment | A published copy of the app at a public address. |
| Replit Agent | An AI that builds or edits the project from your written request. |
| Environment variable | A named setting the program reads at runtime, such as `AI_API_KEY`. |

### 2. Create an account and your first project

1. Sign up at replit.com. For practice, use a test account, not your work or personal main account.
2. Click **Create Repl**. Choose a template for a web app in a language you know (for example Python with a web framework, or Node.js).
3. Give the project a clear name, such as `dish-studio-practice`.

4. Press **Run**.

**You should see:** the console shows the server starting, and the preview pane shows a page. If the preview is blank, look at the console first.

### 3. Learn the workspace in five minutes

| Area | What you do there |
|---|---|
| File tree (left) | Create, rename and delete files. Keep the main file easy to find. |
| Editor (centre) | Write code. Save happens automatically in most setups. |
| Console (bottom) | Read output and errors. Read the first error line, not the last. |
| Preview (right) | See the running page. Refresh after a change if needed. |
| Packages panel | Add libraries your code imports. |
| Secrets panel (lock icon) | Store API keys and private values. |
| Version history | See earlier saved states and roll back a change. |

### 4. Make one change and see it run

1. Open the main file and change the page heading to `Dish Description Studio`.
2. Save, then refresh the preview.
3. If nothing changes, read the console for a red error line.

**Common error:** a missing package. Read the error ("No module named ..." or "Cannot find module ..."), add the package in the Packages panel, and run again.

### 5. Build the input form

Add a form with three fields: dish name, category (a dropdown: Breakfast, Lunch, Dinner, Dessert), and price in rupees. Use a submit button. Give every field a visible label. Labels help screen-reader users and make the form easier to use with a keyboard.

```
<form method="post" action="/describe">
  <label>Dish name <input name="dish" required></label>
  <label>Category
    <select name="category">
      <option>Breakfast</option><option>Lunch</option>
      <option>Dinner</option><option>Dessert</option>
    </select>
  </label>
  <label>Price (Rs) <input name="price" type="number" min="1" required></label>
  <button type="submit">Describe my dish</button>
</form>
```

**You should see:** the form in the preview. Submitting it does nothing useful yet; that is the next part.

## Part B. Intermediate: connect an AI model safely

### 6. Store the key as a secret

An AI call needs an API key from your model provider. Never put the key in the code, in a screenshot, in a chat, or in a public project.

1. Create a key in your AI provider's console, with a monthly spending limit set there.
2. Open the **Secrets** panel and add `AI_API_KEY` with the key as its value.
3. In code, read it from the environment: Python `os.environ["AI_API_KEY"]`; Node `process.env.AI_API_KEY`.

> **If a key leaks,** delete it in the provider console at once and create a new one. Changing the code does not undo an exposed key.

### 7. Write the AI call with limits

Every call should have three guards: a fixed instruction, a capped output length, and a logged usage count. These make cost predictable (see the tokens guide).

```
import os
# Use your provider's official SDK. Names below are placeholders.
from ai_client import Client

client = Client(api_key=os.environ["AI_API_KEY"])

SYSTEM = ("You write short, appetising menu descriptions for local food sellers. "
          "Warm tone. Maximum 50 words. No prices or claims about health.")

def describe(dish, category, price):
    user = f"Dish: {dish}\nCategory: {category}\nPrice: Rs {price}"
    reply = client.generate(system=SYSTEM, user=user, max_tokens=120,
temperature=0.7)
    print("tokens in/out:", reply.usage.input_tokens, reply.usage.output_tokens)
    return reply.text
```

**You should see:** after one submission, the console prints the token counts and the preview shows a description of about 50 words.

### 8. Handle failure gracefully

- If the key is missing, show "AI is not configured" instead of a crash.

- If the call times out, show a retry message and do not retry in a tight loop.
- If the output is empty, show a fallback message and log the input.

### 9. Save results to a database

To keep a history of generated descriptions, add a small database. Replit offers a built-in database option in the workspace; alternatives include a hosted database service. Store only the fields you need: dish, category, price, description, time, and the token counts. Do not store personal data for a practice app.

| Table: descriptions | Type | Note |
|---|---|---|
| id | number | Primary key, automatic |
| dish, category | text | From the form |
| price | number | Validated: greater than zero |
| description | text | From the AI call |
| input_tokens, output_tokens | number | For cost tracking |
| created_at | time | Automatic |

## Part C. Advanced: deploy, monitor and collaborate

### 10. Deploy the app

1. Open the **Deploy** panel and choose the type that fits: a website or web app (always-on address) is the usual choice for this example. Scheduled or background types suit other jobs.
2. Check the build command and run command the panel shows. They should match how you ran the app.
3. Deploy, then open the public address in a private browser window.
4. Test a normal submission, an empty form, a negative price, and a simulated AI failure.

**You should see:** the same page as in the workspace, at a public address, with secrets working. Deployment options and limits change; confirm them on Replit's documentation before you rely on them.

### 11. Logs, errors and rollback

- **Logs:** check the deployment logs after each change. Log the request type, the status and the token counts, never the API key.
- **Rollback:** use version history to return to an earlier saved state if a change breaks the app.

- **Health check:** keep one page that returns "ok" so you can see at a glance that the app is up.

### 12. Collaboration

Invite a teammate to a project only for practice work with fake data. Agree who edits which file. Use version history and comments to explain changes. Do not share secrets with collaborators; they should add their own key for their own testing.

### 13. Using the Replit Agent well

1. Describe the app in one paragraph: who uses it, what the inputs are, what the outputs are, and what it must never do (for example: never store customer phone numbers).
2. Ask for one feature at a time, then test it.
3. Read every file the Agent changes. Ask it to explain anything you do not understand.
4. Never let the Agent paste a real key into code. Check the secrets panel yourself.

## Part D. Other tools you may need

| Tool | Why you need it | Safe practice |
|---|---|---|
| AI provider account (console) | Gives the API key for model calls | Set a spending limit; use a test key |
| GitHub | Stores code history outside Replit, and lets you move projects | Use a practice repository with no secrets in it |
| A database (built-in or hosted) | Stores generated results | Store only the fields you need |
| Spreadsheet (for example Google Sheets) | Quick data sink or log for non-technical teammates | Test sheet only |
| Browser and dev tools | Inspect network calls and errors | Never paste keys into screenshots |
| Email or messaging (optional) | Notifications to a person | Approval before sending; test account only |

## Part E. When Replit is the right choice

| Choose Replit when | Choose another tool when |
|---|---|
| You want to build and deploy a web app from a browser | You only need a no-code workflow between apps (use n8n or Make) |

| | |
|---|---|
| You are learning to code and want instant feedback | Your app needs heavy, specialised infrastructure from day one |
| You need a prototype for users to test quickly | You need strict data residency rules that the platform cannot meet (check first) |

## Part F. Test checklist

| Test | Pass when |
|---|---|
| App runs in the workspace | No red error in the console |
| Key comes from secrets only | Code contains no key; the call works |
| Output is capped | Description stops near 50 words |
| Missing key | A clear message, no crash |
| Negative or empty price | Validation message; no database write |
| Deployed address works | Normal path returns a page |
| Rollback works | An earlier version restores the page |

## Part G. Common problems

| Problem | Fix |
|---|---|
| "Module not found" | Add the package in the Packages panel and run again. |
| Key error at runtime | Check the secret name matches the code exactly. |
| Works in the workspace, fails when deployed | Check the run command and the port setting in deployment. |
| Blank preview | Read the console; the first red line is the cause. |
| Output too long, costs rising | Lower max tokens; ask for a word limit; check usage logs. |

## Part H. Quick check

1. Where should an API key be stored?
2. What is the first thing to read when the preview is blank?
3. Why cap output length on every AI call?
4. What is a safe way to share a practice project?

**Answers:** 1. In Secrets, read from the environment. 2. The first red line in the console. 3. It keeps cost and response size predictable. 4. Share only fake data, and never share secrets.

## Glossary

| Term | Plain meaning |
|---|---|
| Repl | A project in Replit. |
| Secret | A private value kept outside the code. |
| Deployment | A published version at a public address. |
| Environment variable | A named setting the program reads at runtime. |
| Version history | Saved earlier states you can return to. |

- Replit documentation: docs.replit.com.