Quick tip: Review the prerequisites before you run the lab

End Lab 00:31:34

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Open Google Cloud Console

Username

student-01-88860ac9a24a

Password

dsd92uHkzISg

Project ID

qwiklabs-gcp-01-8d96af7

Build a RAG Solution with Firebase Genkit

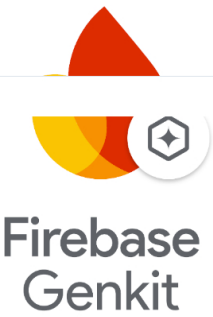
Lab 1 hour No cost Intermediate

Rate Lab

This lab may incorporate AI tools to support your learning.

Overview

Firebase Genkit is an open source framework that helps you build, deploy, and monitor production-ready AI-powered apps.



Genkit is designed for app developers. It helps you easily integrate powerful AI capabilities into your apps with familiar patterns and paradigms. Use Genkit to create apps that generate custom content, use semantic search, handle unstructured inputs, answer questions with your business data, autonomously make decisions, orchestrate tool calls, and much more!

Previous Next

Objective

parsing the data from that vector store to respond to a user query.

You learn how to:

- Create a new Firebase Genkit project.
- Create flows with Genkit and Gemini.
- Create prompts with Dotprompt to provide structured input.
- Use Genkit's indexers and embeddings to create vectors and store them in a local data store.
- Use Genkit's retrievers to query vector databases.
- Use the local Genkit developer UI.
- Explore traces in the Genkit Development UI.

Setup and requirements

Before you click the Start Lab button

Note: Read these instructions.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

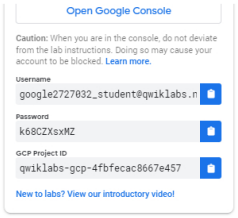
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

Note: If you are using a Pixelbook, open an Incognito window to run this lab.

How to start your lab and sign in to the Console

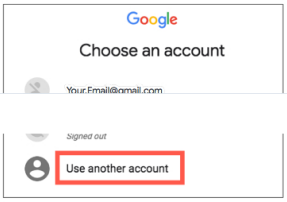
1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

Note: Open the tabs in separate windows, side-by-side.

3. On the Choose an account page, click **Use Another Account**. The Sign in page opens.



4. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

Note: You must use the credentials from the Connection Details panel. Do not use your Google Cloud Skills Boost credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

5. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

After a few moments, the Cloud console opens in this tab.

Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.

Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.

2. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:

gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

- You can list the active account name with this command:

```
gcloud auth list
```

Output:

```
Credentialed accounts:
 - <myaccount>@<mydomain>.com (active)
</mydomain></myaccount>
```

Example output:

```
Credentialed accounts:
 - google1623327_student@qwiklabs.net
```

- You can list the project ID with this command:

```
gcloud config list project
```

Output:

```
[core]
project = <project_id>
</project_id>
```

Example output:

```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

Note: Full documentation of **gcloud** is available in the gcloud CLI overview guide .

Task 1. Setup your Genkit Project

In this task, you set up the **Genkit CLI**, and create an application to access the Gemini model deployed in Google Cloud's Vertex AI.

Initialize your Genkit environment

1. To initialize your environment for Google Cloud, run the following commands:

```
export GCLOUD_PROJECT=qwiklabs-gcp-01-8d96af7f6764
export GCLOUD_LOCATION=us-west1
```

2. To authenticate to Google Cloud and set up credentials for your project and user, run the following command:

```
gcloud auth application-default login
```

3. When prompted, type **Y**, and then press **Enter**.

4. To launch the Google Cloud sign-in flow in a new tab, press Control (for Windows and Linux) or Command (for MacOS) and click the link in the terminal.

5. In the new tab, click the student email address.

7. To let the Google Cloud SDK access your Google Account and agree to the terms, click **Allow**.

   Your verification code is displayed in the browser tab.

8. Click **Copy**.

9. Back in the terminal, where it says **Enter authorization code**, paste the code, and press **Enter**.

   You are now authenticated to Google Cloud.

10. Create a directory for your project, and initialize a new **Node** project:

```
mkdir genkit-intro && cd genkit-intro
npm init -y
```

```
npm init -y
```

This command creates a `package.json` file in the `genkit-intro` directory.

11. To install the **Genkit CLI**, run the following command:

```
npm install -D genkit-cli@1.0.4
```

following command:

```
npm install genkit@1.0.4 --save
npm install @genkit-ai/vertexai@1.0.4 @genkit-ai/google-cloud@1.0.4 @genkit-ai/express@1.0.4 --save
npm install @genkit-ai/dev-local-vectorstore@1.0.4 @genkit-ai/evaluator@1.0.4 --save
```

Create the Genkit app code

1. Create the application source folder and main file:

```
mkdir src && touch src/index.ts
```

2. To open the Cloud Shell editor, click **Open Editor** (✎) on the menu bar.

3. In **Explorer**, expand the `genkit-intro` folder, and open the `package.json` file, and review all the dependencies that were added.

   The dependency list should look similar to this:

```
    "@genkit-ai/dev-local-vectorstore": "^1.0.4",
    "@genkit-ai/evaluator": "^1.0.4",
    "@genkit-ai/express": "1.0.4",
    "@genkit-ai/google-cloud": "1.0.4",
    "@genkit-ai/vertexai": "1.0.4",
    "genkit": "^1.0.4"
}
```

4. Go to the `src` folder and open the `index.ts` file. Add the following import library references to the index.ts file:

```
import { z, genkit } from 'genkit';
import { vertexAI } from '@genkit-ai/vertexai';
import { gemini20Flash001 } from '@genkit-ai/vertexai';
import { textEmbedding004 } from '@genkit-ai/vertexai';
import { Document, index, retrieve } from '@genkit-ai/ai/retriever';
import { devLocalIndexerRef, devLocalRetrieverRef, devLocalVectorstore } from '@genkit-ai/dev-local-vectorstore';
import { logger } from 'genkit/logging';
import { enableGoogleCloudTelemetry } from '@genkit-ai/google-cloud';
import { startFlowServer } from '@genkit-ai/express';
```

```
const ai = genkit({
  plugins: [
      vertexAI({ location: 'us-west1' }),
      devLocalVectorstore([
      {
          indexName: 'menu-items',
          embedder: textEmbedding004,
          embedderOptions: { taskType: 'RETRIEVAL_DOCUMENT' },
      }
      ]),
  ]
});

logger.setLogLevel('debug');
enableGoogleCloudTelemetry();
```

The local vector store plugin will use the **textEmbedding004** embedder to create embeddings that allow retrieval of menu items.

6. Add code to define a flow named **menuSuggestionFlow**, which prompts the model to suggest an item for a menu of a themed restaurant, with the theme provided as input:

```
export const menuSuggestionFlow = ai.defineFlow(
{
  name: 'menuSuggestionFlow',
  inputSchema: z.string(),
  outputSchema: z.string(),
},
async (subject) => {
  const llmResponse = await ai.generate({
    prompt: `Suggest an item for the menu of a ${subject}
```

```
themed restaurant`,
    model: gemini20Flash001,
    config: {
      temperature: 1,
    },
  });

  return llmResponse.text;
}
);
```

7. Add the following line in the `index.ts` file, which starts the flow server and exposes your flows as **HTTP** endpoints:

```
startFlowServer({
    flows: [menuSuggestionFlow],
        origin:  * ,
    },
});
```

8. In your Cloud Shell terminal, copy the `package.json`, and `index.ts` files to a Cloud Storage bucket:

```
gcloud storage cp ~/genkit-intro/package.json
gs://qwiklabs-gcp-01-8d96af7f6764
gcloud storage cp ~/genkit-intro/src/index.ts
gs://qwiklabs-gcp-01-8d96af7f6764
```

> **Note:** It is important to copy the files to the Cloud Storage location. If you've made updates after moving the files earlier, please upload the updated files again by running the same commands to avoid any errors.

Click **Check my progress** to verify the objective.

Setup your Genkit Project

Check my progress

Assessment completed!

Task 2. Create a RAG solution

In this task, you create a flow to index data using embeddings and a local vector database. You also create a flow to query the local store with a user query and return the results after they have been processed and formatted by the LLM.

Develop the code

1. In the `index.ts` file above the `startFlowsServer` line, define the input and output objects for the prompts:

```
export const MenuItemSchema = z.object({
  title: z.string()
    .describe('The name of the menu item'),
  description: z.string()
    .describe('Details including ingredients and
preparation'),
});

export type MenuItem = z.infer<typeof MenuItemSchema>;

// Input schema for a question about the menu
export const MenuQuestionInputSchema = z.object({
    question: z.string(),
});

// Output schema containing an answer to a question
export const AnswerOutputSchema = z.object({
  answer: z.string(),
});

// Input schema for a question about the menu where the
menu is provided in JSON data
export const DataMenuQuestionInputSchema = z.object({
  menuData: z.array(MenuItemSchema),
  question: z.string(),
});

// Input schema for a question about the menu where the
menu is provided as unstructured text
export const TextMenuQuestionInputSchema = z.object({
  menuText: z.string(),
  question: z.string(),
});
```

- **MenuItemSchema:** Describes the menu items, where each menu item contains 3 fields (title, description, and price).
- **MenuItem:** A type for the MenuItemSchema definition.
- **MenuQuestionInputSchema:** A string that contains the expected input string from the user.
- **AnswerOutputSchema:** A string that contains the expected response from the LLM that is sent back to the user.
- **DataMenuQuestionInputSchema:** An object that contains the input question and an array of menu items.
- **TextMenuQuestionInputSchema:** An object that contains the input question and the menu items as a string.

2. Define a prompt that instructs the LLM to answer questions about the food menu. Add the following code above the `startFlowsServer` line:

```
export const ragDataMenuPrompt = ai.definePrompt(
{
    name: 'ragDataMenu',
    model: gemini20Flash001,
    config: { temperature: 0.3 },
},
`
You are acting as Walt, a helpful AI assistant here at the
restaurant.
You can answer questions about the food on the menu or any
other questions customers have about food in general.

Here are some items that are on today's menu that are
relevant to helping you answer the customer's question:
{{#each menuData~}}
-       {{this.title}} \${{this.price}}
    {{this.description}}
    {{~/each}}

Answer this customer's question:
{{question}}?
`
);
```

The prompt text defined above uses the `#each` helper to iterate over the `menuData` array of menu items. A helper is an expression from the Handlebars templating language that is supported by Dotprompt. You can use these expressions to add conditional portions to a prompt or iterate through structured content.

```
export const ragMenuQuestionFlow = ai.defineFlow(
{
  name: 'ragMenuQuestion',
  inputSchema: MenuQuestionInputSchema,
  outputSchema: AnswerOutputSchema,
},
async (input) => {
  // Retrieve the 3 most relevant menu items for the
question
  const docs = await ai.retrieve({
    retriever: devLocalRetrieverRef('menu-items'),
    query: input.question,
    options: { k: 3 },
  });

  const menuData: Array<MenuItem> = docs.map(
    (doc) => (doc.metadata || {}) as MenuItem
  );

  // Generate the response
  const response = await ragDataMenuPrompt({
        menuData: menuData,
        question: input.question,
  });

''
```

4. Define a flow that will index items using a vector store. This includes creating embeddings for each item and storing them in the vector store. Add the following code above the `startFlowsServer` line:

```
export const indexMenuItemsFlow = ai.defineFlow(
  {
    name: 'indexMenuItems',
    inputSchema: z.array(MenuItemSchema),
    outputSchema: z.object({ rows: z.number() }),
  },
  async (menuItems) => {
    // Store each document with its text indexed,
    // and its original JSON data as its metadata.
    const documents = menuItems.map((menuItem) => {
      const text = `${menuItem.title} ${menuItem.price} \n
```

```
${menuItem.description}`;
      return Document.fromText(text, menuItem);
    });
    await ai.index({
      indexer: devLocalIndexerRef('menu-items'),
      documents,
    });
    return { rows: menuItems.length };
```

5. To register the new flows, replace the `ai.startFlowServer` block with the code below:

```
startFlowServer({
    flows: [menuSuggestionFlow, ragMenuQuestionFlow,
indexMenuItemsFlow],
    port: 8080,
    cors: {
        origin: '*',
    },
});
```

Save your work

- In your Cloud Shell terminal, copy the `src/index.ts` file to a Cloud Storage bucket:

```
gcloud storage cp ~/genkit-intro/src/index.ts gs://qwiklabs-
gcp-01-8d96af7f6764
```

Create a RAG Solution

Check my progress

Assessment completed!

Task 3. Test the application

In this task, you interact with the application to test its functionality. You make queries related to the menu, and receive suggestions based on the implemented flows.

1. In the Cloud Shell terminal, start the Genkit developer UI:

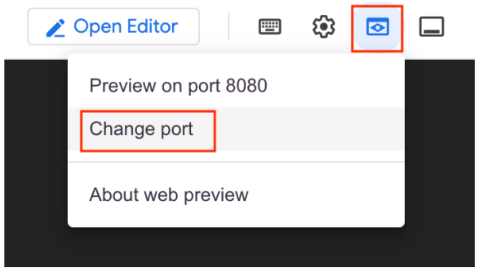
```
cd ~/genkit-intro
npx genkit start -- npx tsx src/index.ts | tee -a
response_output.json
```

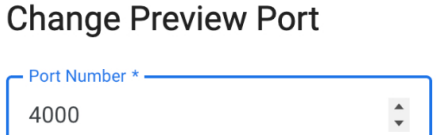
2. Press **Enter** to continue.

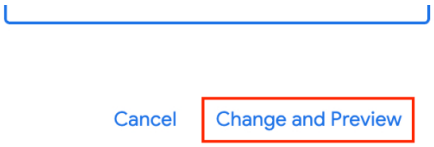
for the flow server to run before continuing to the next step. When ready, you should see output similar to:

```
Flow server running on http://localhost:8080
Reflection server (5969) running on http://localhost:3100
Runtime file written: /home/student_04_b06a2d610476/genkit-
intro/.genkit/runtimes/5969-1736572235048.json
```

3. In the **Web Preview** menu, click **Change port**.







This opens the Genkit developer UI in a separate tab in your browser.

5. In the left panel, check that you have **Flows**, **Prompts**, and **Models**. In the **Flows** and **Prompt** sections, you will see the elements that you defined in code.

6. Click **Flows**, and then click the **indexMenuItems** flow.

7. Delete any existing **Input JSON** contents, and provide these menu items to be indexed:

```
[
{
"title": "Mozzarella Sticks",
"price": 8,
"description": "Crispy fried mozzarella sticks served with
marinara sauce."
},
{
"title": "Chicken Wings",
"price": 10,
"description": "Crispy fried chicken wings tossed in your
choice of sauce."
},
{
"title": "Nachos",
"price": 12,
"description": "Crispy tortilla chips topped with melted
cheese, chili, sour cream, and salsa."
},
{
"title": "Onion Rings",
"price": 7,
"description": "Crispy fried onion rings served with ranch
dressing."
},
{
"title": "French Fries",
"price": 5,
```

```
{
"title": "Mashed Potatoes",
"price": 6,
"description": "Creamy mashed potatoes."
},
{
"title": "Coleslaw",
"price": 4,
"description": "Homemade coleslaw."
},
{
"title": "Classic Cheeseburger",
"price": 12,
"description": "A juicy beef patty topped with melted
American cheese, lettuce, tomato, and onion on a toasted
bun."
},
{
"title": "Bacon Cheeseburger",
"price": 14,
"description": "A classic cheeseburger with the addition of
crispy bacon."
},
{
"title": "Mushroom Swiss Burger",
"price": 15,
"description": "A beef patty topped with sautéed mushrooms,
melted Swiss cheese, and a creamy horseradish sauce."
},
```

```
"price": 13,
"description": "A crispy chicken breast on a toasted bun
with lettuce, tomato, and your choice of sauce."
},
{
"title": "Pulled Pork Sandwich",
"price": 14,
"description": "Slow-cooked pulled pork on a toasted bun
with coleslaw and barbecue sauce."
},
{
"title": "Reuben Sandwich",
"price": 15,
"description": "Thinly sliced corned beef, Swiss cheese,
sauerkraut, and Thousand Island dressing on rye bread."
},
{
"title": "House Salad",
"price": 8,
"description": "Mixed greens with your choice of dressing."
},
{
"title": "Caesar Salad",
"price": 9,
"description": "Romaine lettuce with croutons, Parmesan
cheese, and Caesar dressing."
},
{
"title": "Greek Salad"
```

```
tomatoes, cucumbers, and red onions."
},
{
"title": "Chocolate Lava Cake",
"price": 8,
"description": "A warm, gooey chocolate cake with a molten
chocolate center."
},
{
"title": "Apple Pie",
"price": 7,
"description": "A classic apple pie with a flaky crust and
warm apple filling."
},
{
"title": "Cheesecake",
"price": 8,
"description": "A creamy cheesecake with a graham cracker
crust."
}
]
```

8. Click **Run**.

   The flow response should show the number of documents indexed.

```
npx genkit start -- npx tsx src/index.ts | tee -a response_output.json
```

9. In the Genkit Developer UI, under **Flows**, click the **ragMenuQuestion** flow. Test the flow by providing the following question, and click **Run**.

```
{
  "question": "What's on the menu for today?"
}
```

   View the response that is generated.

View trace and other information

1. Click **View trace** at the bottom. Notice the time it took to run the steps in the flow, and the metadata associated with the request.
2. To view the retrieved menu items, click **devLocalVectorstore/menu-items**.
3. To view the embeddings generated for the question, click **vertexai/text-embedding-004**.

Re-run the flow with different prompts

1. In the left panel, click **Flows**, and then click **ragMenuQuestion**.
2. Re-run the flow providing these questions in the prompts, and observe the results.

```
{
  "question": "Are there any chicken options in today's
menu?"
}
```

```
{
  "question": "What burgers do you have on today's menu?"
}
```

Test the retriever

1. In the left panel, click **Retrievers**, and then click **devLocalVectorstore/menu-**
2. For **Query**, replace the input JSON with the following JSON:

```
{
  "content": [
    {
      "text": "burgers"
    }
  ]
}
```

3. For **Options**, replace the input JSON with the following JSON:

```
{
  "k": 5
}
```

4. Click **Run**, and scroll to view the retrieved menu items.

5. To view the schema of the expected JSON for the indexer, in the left panel, click **Indexers**, and then click **devLocalVectorstore/menu-items**.

6. To view the schema of the expected JSON for the embedder, in the left panel, click **Embedders**, and then click **vertexai/text-embedding-004**.

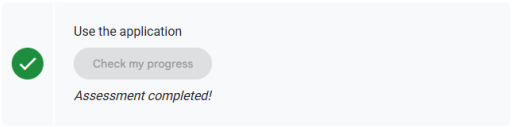
Save your work

1. Close the UI tab, and type **CTRL+C** in your Cloud Shell terminal to stop the Genkit application.

2. In the Cloud Shell terminal, copy the `response_output.json` file to a Cloud Storage bucket:

```
gcloud storage cp ~/genkit-intro/response_output.json
gs://qwiklabs-gcp-01-8d96af7f6764
```

Click **Check my progress** to verify the objective.



Task 4. Evaluate the application

Firebase Genkit supports third-party evaluation tools through plugins, paired with powerful observability features that provide insight into the runtime state of your LLM-powered applications. Genkit tooling helps you automatically extract data including inputs, outputs, and information from intermediate steps to evaluate the end-to-end quality of LLM responses, and understand the performance of your system's building blocks.

For your RAG flow, Genkit will extract the set of documents that were returned by the retriever so you can evaluate the quality of your retriever while it runs in the context of the flow using the Genkit faithfulness and answer relevance metrics.

Add plugins to your Genkit configuration

1. In the Cloud Shell editor, at the top of the `src/index.ts` file, add the **evaluator** library import:

```
import { GenkitMetric, genkitEval } from '@genkit-
ai/evaluator';
```

```
genkitEval({
  judge: gemini20Flash001,
  metrics: [GenkitMetric.FAITHFULNESS,
GenkitMetric.ANSWER_RELEVANCY],
  embedder: textEmbedding004, //
GenkitMetric.ANSWER_RELEVANCY requires an embedder
}),
```

The configuration should look like this:

```
const ai = genkit({
  plugins: [
    // Load the Vertex AI plugin. You can optionally specify your
project ID
    // by passing in a config object; if you don't, the Vertex AI
plugin uses
    // the value from the GCLOUD_PROJECT environment variable.
    vertexAI({ location: 'us-west1' }),
    devLocalVectorstore([
      {
        indexName: 'menu-items',
        embedder: textEmbedding004,
        embedderOptions: { taskType: 'RETRIEVAL_DOCUMENT' },
      }
    ]),
```

```
      metrics: [GenkitMetric.FAITHFULNESS,
GenkitMetric.ANSWER_RELEVANCY],
      embedder: textEmbedding004, // GenkitMetric.ANSWER_RELEVANCY
requires an embedder
    }),
  ]
});
```

Run the evaluation

1. In your Cloud Shell terminal, create a `data` directory:

```
mkdir ~/genkit-intro/data
```

2. Create a file with the input prompts that you would like to use for the evaluation:

```
cat <<EOF > ~/genkit-intro/data/questions.json
[
  {
    "input": {
        "question": "What's on the menu today?"
    }
  }
```

```
  {
    "input": {
        "question": "Are there any burgers in the menu
today?"
    }
  }
]
EOF
```

3. In the Cloud Shell terminal, start the Genkit developer UI:

```
cd ~/genkit-intro
npx genkit start -- npx tsx src/index.ts | tee -a
response_output.json
```

Wait for the command to return the **Genkit Developer UI** URL in the output, and for the flow server to run before continuing to the next step.

4. In a separate Cloud Shell terminal, run the eval function on your **ragMenuQuestion** flow, using the `questions.json` file that you just created.

```
cd ~/genkit-intro
npx genkit eval:flow ragMenuQuestion --input
data/questions.json --output application_result.json
```

View the evaluation results

1. To access the Genkit Developer UI from your browser, use **Web Preview** to access the UI on port **4000**.
2. In the left panel of the UI, click **Evaluations**, and then click the **ragMenuQuestion** evaluation.
3. Inspect the two questions and evaluations. For each question, note the **Faithfulness** and **Answer Relevancy** scores.
4. To expand the result, in the first row, click **Expand** ( ).
5. View the input, output, and context that provide the rationale for the scores.
6. Repeat the previous steps to analyze the second result.

Save your work

1. Close the UI tab, and type **CTRL+C** in your original Cloud Shell terminal to stop the Genkit application.

```
gcloud storage cp -r ~/genkit-intro/data gs://qwiklabs-gcp-
01-8d96af7f6764
gcloud storage cp ~/genkit-intro/application_result.json
gs://qwiklabs-gcp-01-8d96af7f6764
gcloud storage cp ~/genkit-intro/src/index.ts
gs://qwiklabs-gcp-01-8d96af7f6764
```

Click **Check my progress** to verify the objective.

Evaluate the Application

Check my progress

Assessment completed!

Congratulations!

In this lab, you created a Genkit project and leveraged indexers and retrievers from Genkit to build a RAG solution. You created a prompt, added an indexer to generate

embeddings for your menu items, stored them locally, and used a retriever to access them. You used Gemini to generate the responses and tested the application using the Genkit developer UI. Finally, you evaluated your RAG flow using the Genkit evaluation toolkit.

Manual Last Updated January 16, 2025

Lab Last Tested January 16, 2025

Course Badge