

[Before you start](#)

[Exercise scenario](#)

[Set up the library application in Visual Studio Code](#)

[Use GitHub Copilot to explain the library application codebase](#)

[Create the project documentation for the README file](#)


[Summary](#)

[Clean up](#)

Analyze and document code using GitHub Copilot

GitHub Copilot can help you understand and document a codebase by generating explanations and documentation. In this exercise, you use GitHub Copilot to analyze a codebase and generate documentation for the project.

This exercise should take approximately **20** minutes to complete.

 **IMPORTANT:** To complete this exercise, you must provide your own GitHub account and GitHub Copilot subscription. If you don't have a GitHub account, you can [sign up](#) for a free individual account and use a GitHub Copilot Free plan to complete the exercise. If you have access to a GitHub Copilot Pro, GitHub Copilot Pro+, GitHub Copilot Business, or GitHub Copilot Enterprise subscription from within your lab environment, you can use your existing GitHub Copilot subscription to complete this exercise.

Before you start

Your lab environment must include the following: Git 2.48 or later, .NET SDK 9.0 or later, Visual Studio Code with the C# Dev Kit extension, and access to a GitHub account with GitHub Copilot enabled.

If you're using a local PC as a lab environment for this exercise:

- For help configuring your local PC as your lab environment, open the following link in a browser: [Configure your lab environment resources](#).
- For help enabling your GitHub Copilot subscription in Visual Studio Code, open the following link in a browser: [Enable GitHub Copilot within Visual Studio Code](#).

If you're using a hosted lab environment for this exercise:

- For help enabling your GitHub Copilot subscription in Visual Studio Code, paste the following URL into a browser's site navigation bar: [Enable GitHub Copilot within Visual Studio Code](#).
- Open a command terminal and then run the following commands:

To ensure that Visual Studio Code is configured to use the correct version of .NET, run the following command:

CodeCopy

```
dotnet nuget add source https://api.nuget.org/v3/index.json -n nuget.org
```

Exercise scenario

You're a developer working in the IT department of your local community. The backend systems that support the public library were lost in a fire. Your team needs to develop a temporary solution to help the library staff manage their operations until the system can be replaced. Your team chose GitHub Copilot to accelerate the development process.

Your colleague has developed an initial version of the library application, but due to time constraints, they haven't had a chance to document the code. You need to analyze the codebase and create documentation for the project.

This exercise includes the following tasks:

- Set up the library application in Visual Studio Code.
- Use GitHub Copilot to explain the library application codebase.
- Use GitHub Copilot to create a README.md file for the library application.

Set up the library application in Visual Studio Code

Your colleague has developed an initial version of the library application and has made it available as a .zip file. You need to download the zip file, extract the code files, and then open the solution in Visual Studio Code.

Use the following steps to set up the library application:

1. Open a browser window in your lab environment.
2. To download a zip file containing the library application, paste the following URL into your browser's address bar: [GitHub Copilot lab - Analyze and document code](#)

The zip file named AZ2007LabAppM2.zip will be downloaded to your lab environment.

3. Extract the files from the **AZ2007LabAppM2.zip** file.

For example:

- a. Navigate to the downloads folder in your lab environment.
 - b. Right-click **AZ2007LabAppM2.zip**, and then select **Extract all**.
 - c. Select **Show extracted files when complete**, and then select **Extract**.
4. Open the extracted files folder, then copy the **AccelerateDevGHCopilot** folder to a location that's easy to access, such as your Windows Desktop folder.
 5. Open the **AccelerateDevGHCopilot** folder in Visual Studio Code.

For example:

- a. Open Visual Studio Code in your lab environment.
 - b. In Visual Studio Code, on the **File** menu, select **Open Folder**.
 - c. Navigate to the Windows Desktop folder, select **AccelerateDevGHCopilot** and then select **Select Folder**.
6. In the Visual Studio Code SOLUTION EXPLORER view, expand the solution to show the following solution structure:
 - AccelerateDevGHCopilot
 - src
 - Library.ApplicationCore\
 - Library.Console\
 - Library.Infrastructure\
 - tests
 - UnitTests\
 7. Ensure that the solution builds successfully.

For example, in the SOLUTION EXPLORER view, right-click **AccelerateDevGHCopilot**, and then select **Build**.

You'll see some Warnings, but there shouldn't be any Errors.

Use GitHub Copilot to explain the library application codebase

GitHub Copilot can help you to understand an unfamiliar codebase by generating explanations at the solution, file, and code line levels.

Analyze code using prompts in the Chat view

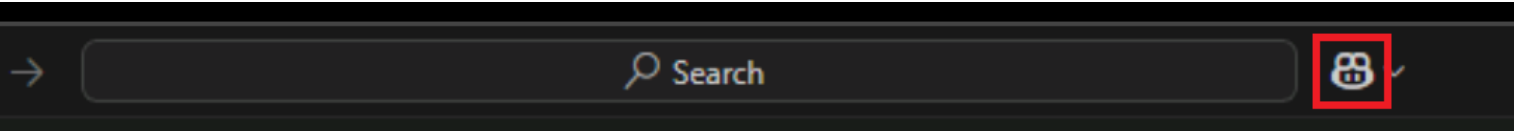
GitHub Copilot’s Chat view includes a chat-based interface that allows you to interact with GitHub Copilot using natural language prompts. When evaluating an existing codebase for the first time, you can create prompts that generate an explanation at the workspace or project level, or at the code block or code line level. To assist you in specifying the context of your prompt, GitHub Copilot provides chat participants, chat variables, and slash commands.

- Chat participants are used to scope your prompt to a specific domain.
- Chat variables are used to include specific context in your prompt.
- Slash commands are used to avoid writing complex prompts for common scenarios.

Use the following steps to complete this section of the exercise:

1. Ensure that the AccelerateDevGHCopilot solution is open in Visual Studio Code.
2. Open GitHub Copilot’s Chat view.

To open the Chat view, select the **Toggle Chat** button at the top of the Visual Studio Code window.



You can also open the Chat view using the **Ctrl+Alt+I** keyboard shortcut.

3. In the Chat view, enter a prompt that uses GitHub Copilot’s [@workspace](#) Chat participant to generate a description of the project.

For example, enter the following prompt in the Chat view:

Code	Copy
@workspace describe this project	

Use Chat participants, such as the [@workspace](#), to improve the responses generated by GitHub Copilot. Chat participants work like domain experts who can help you in their specialized areas. Use [@workspace](#) when you want GitHub Copilot to consider the structure of your project, how different parts of your code interact, or design patterns in your project.

To see a list of all available chat participants, type @ in the chat prompt box.

4. Take a minute to compare GitHub Copilot’s response with the actual project files.

You should see a response that describes each of the projects in the solution:


- **Library.ApplicationCore**
- **Library.Console**
- **Library.Infrastructure**
- **UnitTests**

5. Use the SOLUTION EXPLORER view to expand the project folders.
6. Locate and then open the **ConsoleApp.cs** file.

The ConsoleApp.cs file is located in the **src/Library.Console** folder.


7. Take a moment to review the code file.
8. Enter a prompt in the Chat view that generates a description of the **ConsoleApp** class.

For example, enter the following prompt in the Chat view:

Code  Copy

@workspace #usages How is the ConsoleApp class used?

Use chat variables, such as **#usages**, to include specific context in your prompt. To see a list of the chat variables, type **#** in the chat prompt box.

 **NOTE:** GitHub Copilot considers your chat history and the code files you have open in Visual Studio Code when constructing a context for your prompt and generating a response.


9. Take a minute to verify the accuracy of GitHub Copilot’s response.

You should see a response that the describes where the **ConsoleApp** class is defined and how it’s used in the codebase. The ConsoleApp.cs and Program.cs files are referenced in the response, along with line numbers

10. Open the **Program.cs** file and examine the code.

11. Enter a prompt in the Chat view that generates an explanation of the Program.cs file.

For example, enter the following prompt in the Chat view:

Code  Copy

@workspace /explain Explain the Program.cs file

Use Slash commands, such as **/explain**, to avoid writing complex prompts for common scenarios. To see a list of all available slash commands, type **/** in the chat prompt box. Available slash commands may vary, depending on your environment and the context of your chat.

12. Take a minute to review the detailed response generated by GitHub Copilot.

You should see a response that includes an overview and a breakdown that explains how the file is used within the application.

13. Close the Program.cs file.

Improve chat responses by adding context

GitHub Copilot uses context to generate more relevant responses.

Opening files in the code editor is one way to establish context, but you can also add files to the Chat context using drag-and-drop operations or by using the **Attach Context** button in the Chat view.

Use the following steps to complete this section of the exercise:

- 1. Expand the **Library.Infrastructure** project, and then expand the **Data** folder.
- 2. Use a drag-and-drop operation to add the following files from the SOLUTION EXPLORER view to the Chat context: **JsonData.cs**, **JsonLoanRepository.cs**, and **JsonPatronRepository.cs**.

GitHub Copilot uses the Chat Context to understand the code files that are relevant to your prompt. You can add files to the Chat context using drag-and-drop operations, or you can use the **Attach Context** button in the Chat view.

Instead of adding individual files manually, you can let Copilot find the right files from your codebase automatically. This can be useful when you don’t know which files are relevant to your question.

To let Copilot find the right files automatically, add **#codebase** in your prompt or select Codebase from the list of context types.

3. Enter a prompt in the Chat view that generates an explanation of the data access classes.

For example, enter the following prompt in the Chat view:

Code

@workspace /explain Explain how the data access classes work

Copy

4. Take a couple minutes to read through the response.

You should see a response that describes each of the data access classes (**JsonData**, **JsonLoanRepository**, and **JsonPatronRepository**) and how they work together to manage data access in the application. Key methods, such as **LoadData**, **SaveLoans**, and **SavePatrons**, should be mentioned in the response.

5. Take a minute to examine the JSON data files that are used to simulate library records.

The JSON data files are located in the **src/Library.Console/Json** folder.

The data files use ID properties to link entities. For example, a **Loan** object has a **PatronId** property that links to a **Patron** object with the same ID. The JSON files contain data for authors, books, book items, patrons, and loans.

!

NOTE: Notice that Author names, book titles, and patron names have been anonymized for the purposes of this training.

Build and run the application

Running the application helps you understand the user interface, key features of the application, and how app components interact.

Use the following steps to complete this section of the exercise:

1. Ensure that you have the **Solution Explorer** view open.

The Solution Explorer view is not the same as the Explorer view. The Solution Explorer view uses project and solution files as “directory” nodes to display the structure of the solution.

2. To run the application, right-click **Library.Console**, select **Debug**, and then select **Start New Instance**.

If the **Debug** and **Start New Instance** options aren’t displayed, ensure that you’re using the Solution Explorer view and not the Explorer view.

The following steps guide you through a simple use case.

3. When prompted for a patron name, type **One** and then press Enter.

You should see a list of patrons that match the search query.

!

NOTE: The application uses a case-sensitive search process.

4. At the “Input Options” prompt, type **2** and then press Enter.

Entering **2** selects the second patron in the list.

You should see the patron’s name and membership status followed by book loan details.

5. At the “Input Options” prompt, type **1** and then press Enter.

Entering **1** selects the first book in the list.

You should see book details listed, including the due date and return status.

6. At the “Input Options” prompt, type **r** and then press Enter.

Entering **r** returns the book.

7. Verify that the message “Book was successfully returned.” is displayed.

The message “Book was successfully returned.” should be followed by the book details. Returned books are marked with **Returned: True**.

8. To begin a new search, type **s** and then press Enter.

9. When prompted for a patron name, type **One** and then press Enter.

10. At the “Input Options” prompt, type **2** and then press Enter.

11. Verify that first book loan is marked **Returned: True**.

12. At the “Input Options” prompt, type **q** and then press Enter.

13. Stop the debug session.

Create the project documentation for the README file

Readme files provide project contributors and stakeholders with essential information about a code repository. They help users understand the purpose of the project, how to use it, and how to contribute. A well-structured README file can significantly improve the usability and maintainability of a project.

You need a README file that includes the following sections:

- **Project Title:** A brief, clear title for the project.
- **Description:** A detailed explanation of what the project is and what it does.
- **Project Structure:** A breakdown of the project structure, including key folders and files.
- **Key Classes and Interfaces:** A list of key classes and interfaces in the project.
- **Usage:** Instructions on how to use the project, often including code examples.
- **License:** The license that the project is under.

In this section of the exercise, you’ll use GitHub Copilot to create project documentation and add it to a **README.md** file.

Use the following steps to complete this section of the exercise:

1. Add a new file named **README.md** to the root folder of the **AccelerateDevGHCopilot** solution.
2. Open the Chat view.
3. To generate project documentation for your README file, enter the following prompt:

Code Copy

@workspace Generate the contents of a README.md file for a code repository. Use "Library App" as the project title. The README file should include the following sections: Description, Project Structure, Key Classes and Interfaces, Usage, License. Format all sections as raw markdown. Use a bullet list with indents to represent the project structure. Do not include ".gitignore" or the ".github", "bin", and "obj" folders.

NOTE: Using multiple prompts, one for each section of the README file would produce more detailed results. A single prompt is used in this exercise to simplify the process.

4. Review the response to ensure each section is formatted as markdown.

You can update sections individually to provide more detailed information or if they aren't formatted correctly. You can also copy GitHub Copilot's response to the README file and then make corrections directly in the markdown file.

5. Copy the suggested documentation, and then paste it into the README.md file.

To copy the entire response, scroll to the bottom of the response, right-click in the empty space to the right of the "thumbs-up" icon, and then select **Copy**

6. Format the README.md file as needed.

You can update the settings for GitHub Copilot to enable markdown formatting, then use GitHub Copilot to help you update sections of the README.md file.

When you're finished, you should have a README.md file that includes each of the specified sections, with an appropriate level of detail.

Summary

In this exercise, you learned how to use GitHub Copilot to analyze and document a codebase. You used GitHub Copilot to generate explanations for the project structure, key classes, and data access classes. You also used GitHub Copilot to create a README.md file for the project.

Clean up

Now that you've finished the exercise, take a minute to ensure that you haven't made changes to your GitHub account or GitHub Copilot subscription that you don't want to keep. If you made any changes, revert them now.