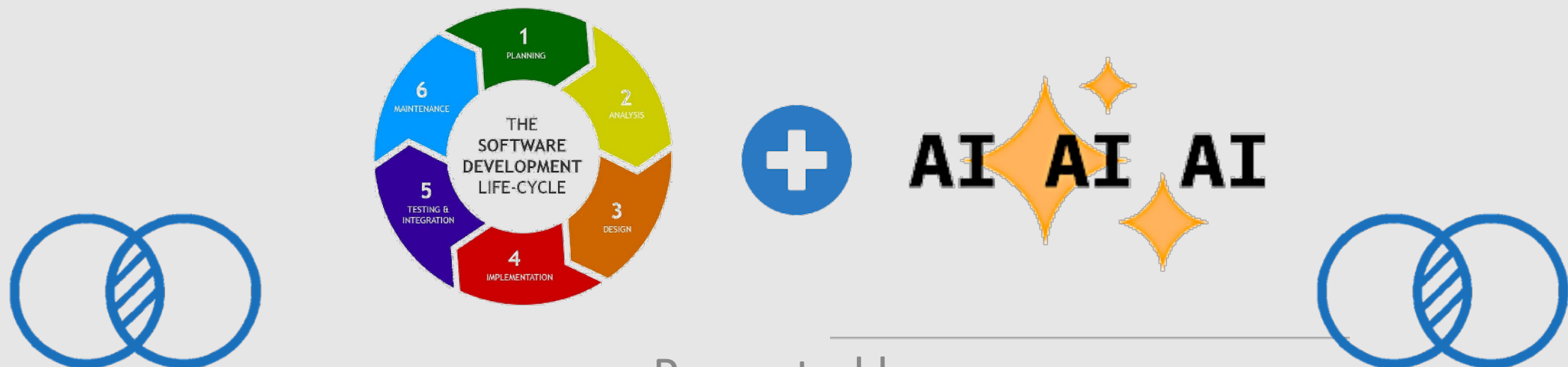




# Incorporating AI into your SDLC

Leveraging AI tooling across the phases of your software development lifecycle



**AI in Production**

Presented by  
Brent Laster  
Tech Skills Transformations LLC

**AI in Production**

© 2025 Brent C. Laster & Tech Skills Transformations LLC



All rights reserved



# About me

- Founder, Tech Skills Transformations LLC
- <https://getskillsnow.com>
- [info@getskillsnow.com](mailto:info@getskillsnow.com)
- Long career in corporate as dev, manager, and director in DevOps and other areas

## Author

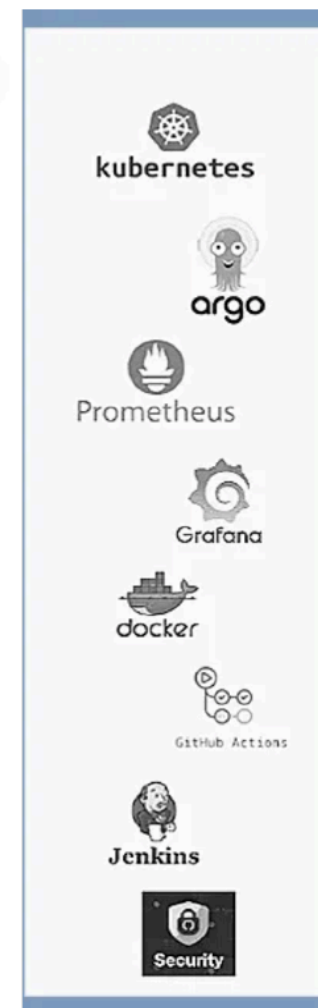
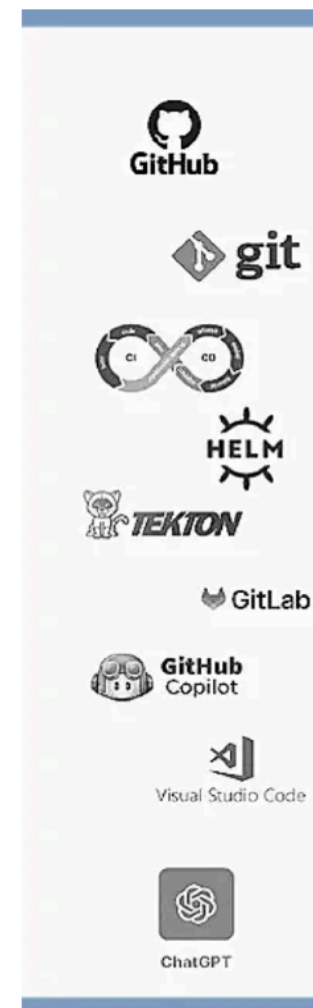
- O'Reilly "reports"
- Books
  - Professional Git
  - Jenkins 2 – Up and Running
  - Learning GitHub Actions
  - AI-Enabled SDLC
  - Learning GitHub Copilot

## Speaker

- Social media
  - ❑ LinkedIn: [brentlaster](#)
  - ❑ X: [@BrentCLaster](#)
  - ❑ Bluesky: [brentclaster.bsky.social](#)
  - ❑ GitHub: [brentlaster](#)

LISTED BELOW ARE A FEW OF THE TECHNOLOGIES FOR WHICH WE OFFER HANDS-ON TRAINING. THESE CAN BE CUSTOMIZED FOR ANY SIZE TEAM FROM 1-100 AND FOR ANY LEVEL FROM BEGINNER TO ADVANCED.

- ArgoCD
- Containers
- Docker
- Gerrit
- Git
- GitHub Actions
- GitHub Codespaces
- GitHub Copilot
- GitHub Foundations
- GitHub Security
- GitLab
- GitOps
- Gradle
- Grafana
- Helm
- Jenkins
- Kubernetes
- Kustomize
- LLMs
- Prometheus
- Tekton
- VS Code





Background





# Rise of AI Coding Assistants (Software Dev Asst's = SDA)<sup>4</sup>

- LLMs and AI apps have exploded since OpenAI added chat to its models in November 2022
- Code creation and related are excellent fits for Gen AI due to structure, syntax, and wealth of examples
- At core, provide inline code suggestions as well as NL chat interfaces
- But also provide additional powerful functions
  - Repository indexing (RAG)
  - Summarizing (PRs, Issues, etc.)
  - Agent workflows (From big ideas to complete implementation)
- Users can leverage them in three modes:
  - Acceleration
  - Exploration
  - Directive



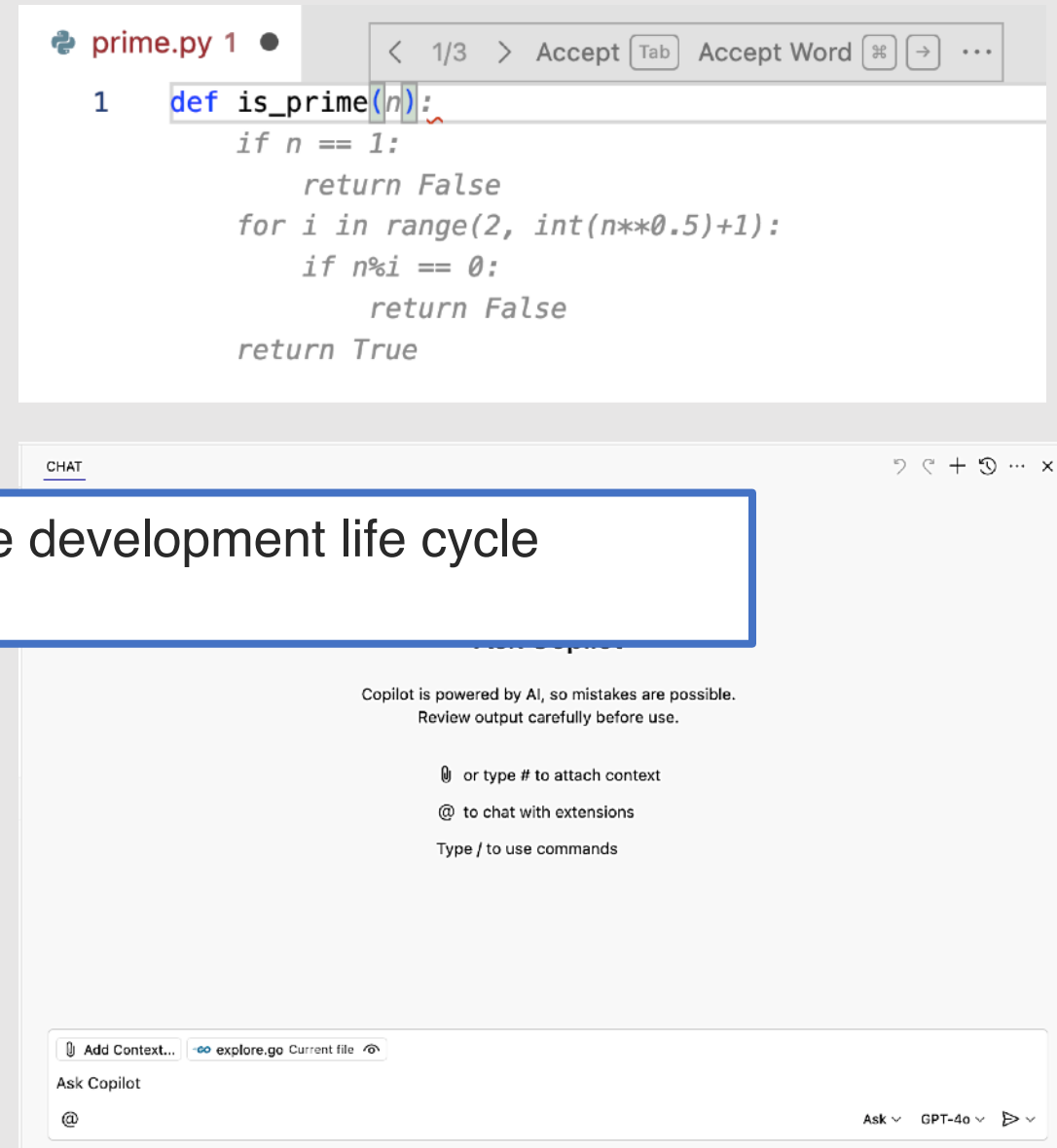




# Uses and Roles

- **Accelerators/Acceleration** - code completion - working in the flow and AI provides completion suggestions (usually in the editor)
- **Explorers/Exploration** - using the chat interface to ask how to code something? What is the API?
- **Directors/Directing** - We can apply these across the software development life cycle (SDLC)!  
specific instructions to create something via the chat interface or in the editor (via comments)
- All of these *can* lead to increased developer productivity

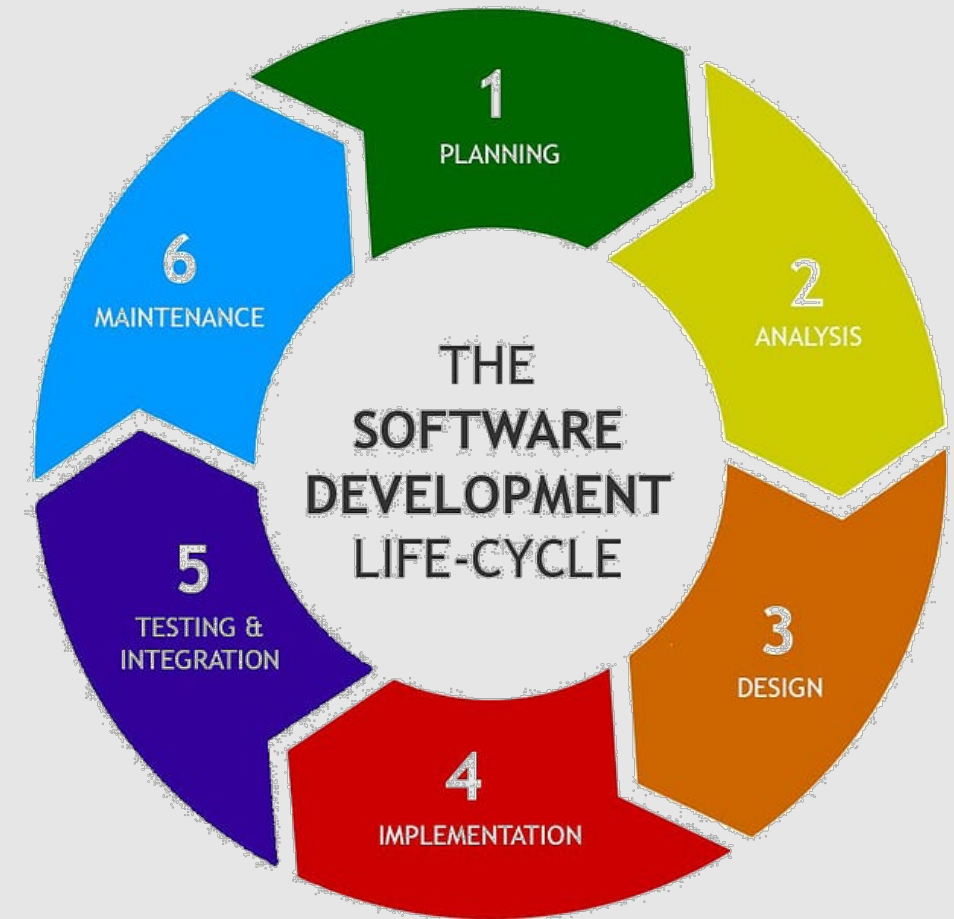
source: "<https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai>





# The Software Development Lifecycle (SDLC)

- Structured process used to create high-quality software efficiently and consistently
- Outlines and defines the steps to plan, develop, test, and deploy
- Typical phases include
  - Planning
  - Analysis
  - Design
  - Implementation
  - Testing and integration
  - Maintenance
- Defined phases may vary across organizations/projects





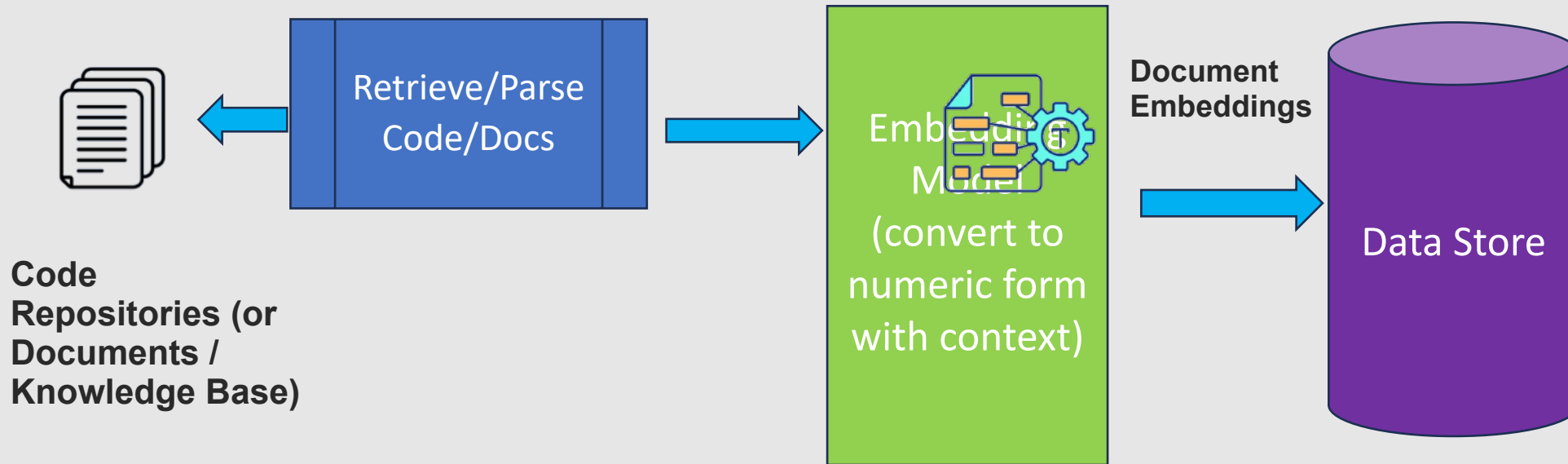
# Planning





# Background : RAG

- Goal: Search your local data (code, docs, etc.) to give additional context for LLM to work with and give you more targeted, useful responses (considers your data along with its training)

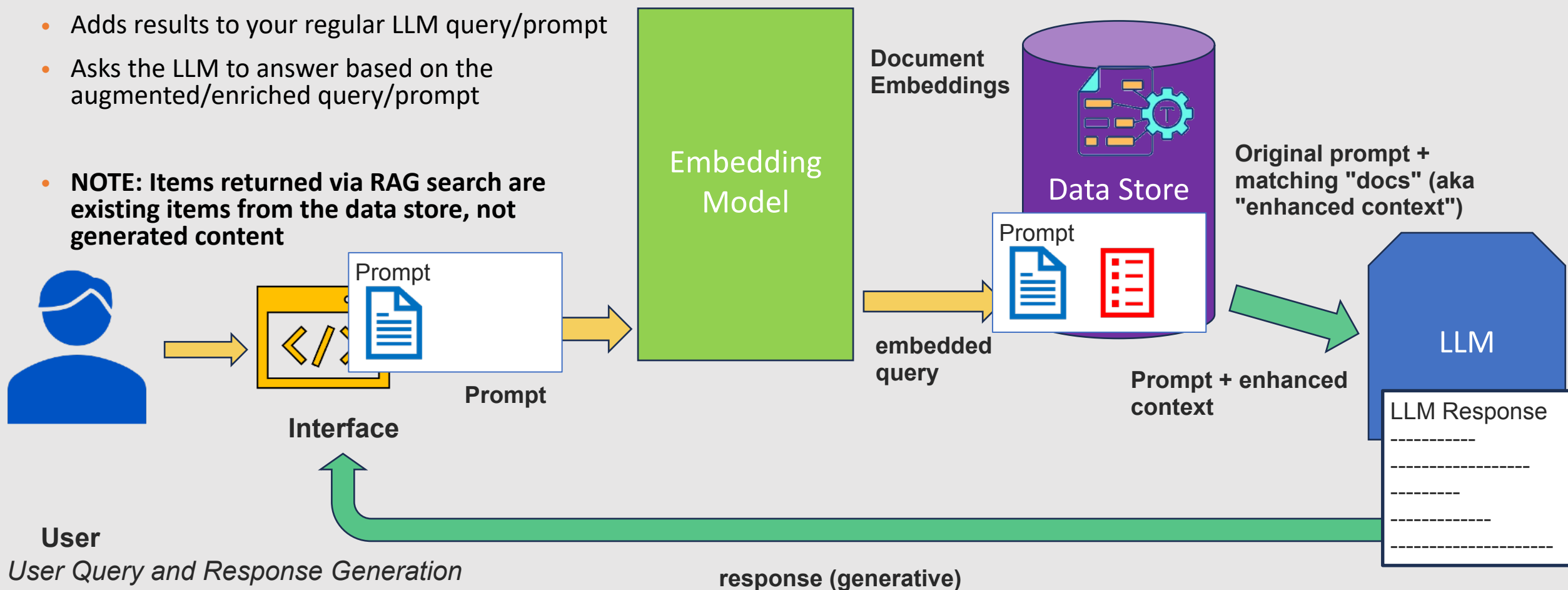


- Your data is parsed and stored with information about other data it's related to in a data store
  - parsing = tokens, storing as numeric data = embeddings, information about other data = vectors (set of numbers that represent how much a term is related to others in the data store)



# Background: RAG

- For queries/prompts, application gathers results (most relevant ones) from the vector database with your data
- Adds results to your regular LLM query/prompt
- Asks the LLM to answer based on the augmented/enriched query/prompt
- **NOTE: Items returned via RAG search are existing items from the data store, not generated content**





# Planning with AI (approaches)

- Index your code (targeted RAG) – usually automatic now
- Make code repositories accessible for AI (via RAG/indexes)
  - Easily find and identify reusable code
- Ask questions of your code to simplify estimates, avoid rework, ground knowledge
  - How did we implement function X?
  - Does our code use dependency Y that has this security vulnerability? If so, where?
  - We want to migrate from one platform/cloud/language / framework to another - where do we need to focus our efforts?
  - The open-source library we use has changed licensing terms - how are we using that?
  - Do we have code that does Z?







# Planning: Prototyping

- For brainstorming/considering requirements
- Can quickly draft pseudo code in comments
- AI can generate "draft" code
- Doesn't have to be perfect, just enough for estimation/planning

```
# Given a list of app features and their dependencies as (feature, depends_on) pairs
# Determine a valid order in which the user can unlock and access features
# Raise an error if a cycle exists (invalid dependency loop)
# Example input:
#     features = [
#         ("upload", "signup"),
#         ("share", "upload"),
#         ("analytics", "share"),
#         ("signup", None)
#     ]
# Expected output:
#     ["signup", "upload", "share", "analytics"]

# Define a function to accept the feature dependency list
# Build a directed graph using adjacency lists
# Use topological sort to determine feature unlock order
# Validate and raise an error if dependencies form a cycle
# Return an ordered list of unlockable features
```

```
from collections import defaultdict, deque

def determine_feature_unlock_order(dependencies):
    # Build graph and in-degree count
    graph = defaultdict(list)
    in_degree = defaultdict(int)
    features = set()

    for feature, depends_on in dependencies:
        features.add(feature)
        if depends_on:
            graph[depends_on].append(feature)
            in_degree[feature] += 1
            features.add(depends_on)
        else:
            in_degree[feature] += 0 # Ensure entry in in-degree map

    # Start with features that have no dependencies
    queue = deque([f for f in features if in_degree[f] == 0])
    unlock_order = []

    while queue:
        current = queue.popleft()
        unlock_order.append(current)

        for dependent in graph[current]:
            in_degree[dependent] -= 1
            if in_degree[dependent] == 0:
                queue.append(dependent)

    if len(unlock_order) != len(features):
        raise ValueError("Circular feature dependency detected")

    return unlock_order

# Example usage
if __name__ == "__main__":
    features = [
        ("upload", "signup"),
        ("share", "upload"),
        ("analytics", "share"),
        ("signup", None)
    ]
    order = determine_feature_unlock_order(features)
    print("Feature unlock order:", order)
```





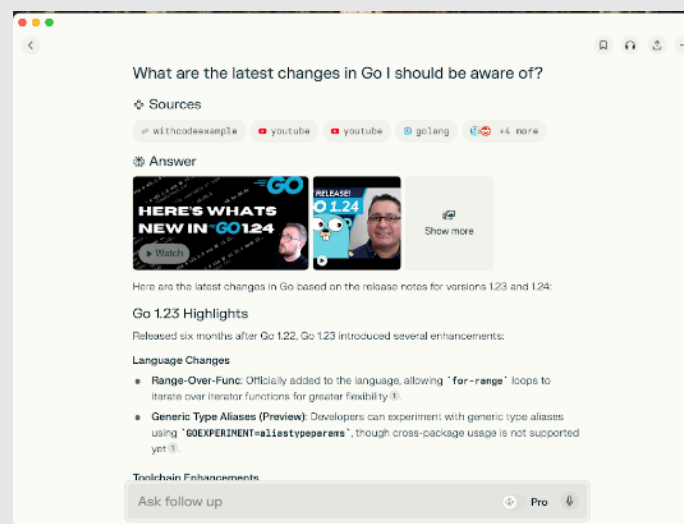
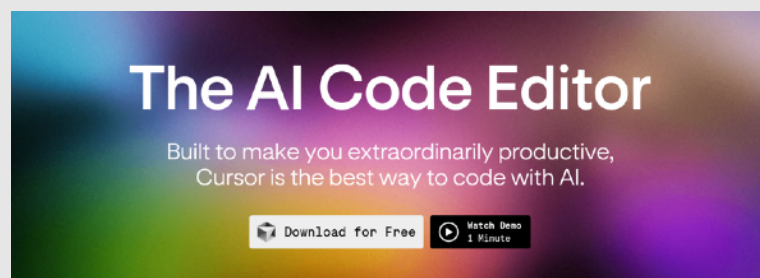
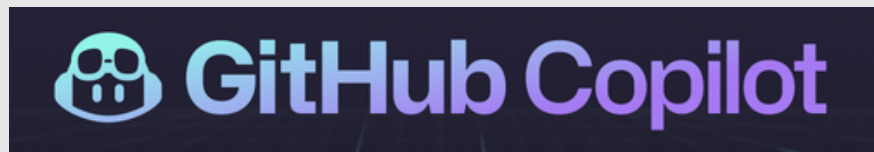
# Development





# Types of AI Assistance for Coding

- Extensions that add AI code completions and chat
- AI Coding IDEs
  - Based on VS Code
  - AI-first design
  - Tend to focus more fully on project-wide insights and application building
  - Think “collaborate like Copilot and handle complex tasks like an agent”
- Research-focused AI assistants
  - Offer answers based in current content
  - Synthesize info from multiple sources (cited) into concise, natural language answers
  - Uses LLM on back end
  - Great at real-time info retrieval
  - Can also do most things that ChatGPT can do (but with up-to-date info and sources)





# How coding assistants can help and what to remember

14

- Code suggestions for nearly any current language
- Comment driven code creation
- Automatic test generation
- SQL generation (for queries, tables, indices, stored procs)
- Regular expression generation
- Pattern/mappings generation
- Document code
- Explain code
- Fixing code
- Reviewing code
- Translating code
- Generate specs/configs
- What is the API to do...

- Reduce cognitive load
- Let programmers focus on the parts that need their expertise

- AI needs good context
- AI needs good guidance or queries (prompts)
- AI can hallucinate
- AI can be out-of-date
- AI doesn't do real-time validation of generated data
- AI doesn't care about tech debt
- **BOTTOM LINE:** AI-generated code intended for production must ALWAYS be subject to same oversight and review





# Leveraging Agents

- Most AI assistants have agentic functions built in
- Agent = AI w/ agency
  - ability to gather input
  - act
  - make changes
- End-to-end change
- Autonomous with oversight
- Goes beyond typical AI assist functions

skillrepos / copilot-adv

Code Issues 1 Pull requests Actions Projects Wiki Security Insights Settings

Files

main

Go to file

app

app.py

auth.py

datastore.py

extra

images

scripts

.gitignore

LICENSE

README.md

labs.md

requirements.txt

copilot-adv / app /

brentlaster Update datastore.py 47f5f1b · last month History

Name	Last commit message	Last commit date
..		
app.py	Update app.py	last month
auth.py	Add initial app files	2 months ago
datastore.py	Update datastore.py	last month





## Resolving Issues







# Resolving issues with AI

17

- Shortcut fix commands available
- Prompts to help find and fix issues (or potential issues)
  - What are some places in this repository where logic/performance/security errors can occur?
  - What are the most recent significant changes to the codebase?
  - What are the code changes that affected this code in branch x from date y to date z?
  - How can I make this code be more efficient with system resources?
  - Where else might this problem occur?
  - What other parts of the codebase use this routine?

```
/fix-code
```

Tabnine Tabnine Protected

```
opts := zap.Options{
    Development: true,
}
opts.BindFlags(flag.CommandLine)
flag.Parse()
```

Apply Insert Copy

In the provided code, the errors were caused by incorrect variable declaration and incorrect usage of the `zap.Options` and `flag` packages. The `opts` variable should be declared as a `zap.Options` type, and the `flag.CommandLine` should

```
prime.py > is_prime
1 def is_prime(n):
3     if n <= 1 or (n > 3 and (n % 2 == 0 or n % 3 == 0)):
4         return False
```

/fix "x" is not defined

> Used 1 reference

The issue is that the variable `x` is undefined; it should be replaced with `n`, which is the parameter of the function.

Ask Copilot @ GPT-4o

Accept Close

```
5     if n <= 3:
6         return True
7
```



# Working with tracking mechanisms (Copilot examples)

- Given an issue, can prompt Copilot for next steps
- Prompt to provide summary of issue
  - Includes overview of issue based on what Copilot can gather from the system using tools
- Generate pull request summaries

**Add a description**

Write **Preview**

This pull request includes significant changes to improve security by fixing SQL injection vulnerabilities and adds a new GitHub Actions workflow for creating issues. The most important changes are listed below:

**Security Improvements:**

- `models/models.go` : Fixed SQL injection vulnerabilities by parameterizing queries in `NameQuery` , `AuthorQuery` , and `ReadQuery` functions. [1] [2] [3]

**GitHub Actions:**

- `.github/workflows/create-issue.yml` : Added a new workflow to create issues manually from the Actions tab, including steps to echo inputs and create an issue using the GitHub REST API.

**Create pull request**





# AI review – Having Copilot review code in VS Code



19

- AI-powered feedback on code as you write it
  - Review selection:
    - » select code in VS Code and ask for initial review
    - » Available now for all Copilot subscribers
- Review changes: ask Copilot for deeper review of all changes before pushing from “Source Control” tab
  - » Can also be done in pull requests on GitHub.com
  - » Currently must be on waitlist

117 `if numAvailable > instance.Spec.Replicas {`

Code Review Comment (1 of 5)

**GitHub Copilot**

The condition `numAvailable > instance.Spec.Replicas` could be simplified for better readability. Consider using a variable to store the difference.

Suggested change:

```
-     if numAvailable > instance.Spec.Replicas {  
-         log.Info("Scaling down pods", "Currently available", numAvailable, "Required replicas", instance.Spec.  
-             diff := numAvailable - instance.Spec.Replicas  
+     diff := numAvailable - instance.Spec.Replicas  
+     if diff > 0 {  
+         log.Info("Scaling down pods", "Currently available", numAvailable, "Required replicas", instance.Spec.
```

Apply and Go to Next ▾ Discard and Go to Next ▾

118 `log.Info("Scaling down pods", "Currently available", numAvailable, "Required replicas", instance.Spec.Replicas)`

119 `diff := numAvailable - instance.Spec.Replicas`





# Testing





# Testing with AI

21

- Create tests for ...
- What is the process for testing and debugging the project's code?
- Generate unit tests / integration tests for ...

TABNINE AI: CHAT

Tabnine Protected

What is the process for testing and debugging the project's code?

Tabnine Tabnine Protected

To test and debug the project's code, you can follow these steps:

**Testing:**

- 1. Unit Tests:** Run unit tests to verify the functionality of individual components. You can use the Ginkgo and Gomega testing frameworks provided by the project. Look for files ending in `_test.go` in the `controllers` directory.
- 2. Integration Tests:** Run integration tests to verify the interaction between different components. These tests may require a running Kubernetes cluster. You can use the `envtest` package provided by the project to set up a local test environment.
- 3. E2E Tests:** Run end-to-end (E2E) tests to verify the overall functionality of the Operator. These tests may require a running Kubernetes cluster. You can use the `envtest` package provided by the project to set up a local test environment.

TABNINE AI: CHAT

Tabnine Protected

Create tests for the `is_prime` function using Python.

Tabnine Tabnine Protected

```
def test_is_prime():  
    assert is_prime(2) == True  
    assert is_prime(3) == True  
    assert is_prime(4) == False  
    assert is_prime(5) == True  
    assert is_prime(6) == False
```

prime.py > is\_prime

/tests

Generate unit tests for the selected code

```
1 def is_prime(n):  
2     if n == 1:  
3         return False  
4     for i in range(2, int(n ** 0.5) + 1):  
5         if n % i == 0:  
6             return False  
7     return True  
8
```





# AI Testing – from the start

- Can help set up a testing framework for your code
- Recommend appropriate frameworks
- Provide step-by-step setup and configuration instructions
- Suggest relevant testing extensions for your dev env
- Can have custom test generation instructions

CHAT

gwstudent2 selected " `testing` (built-in Go testing framework)"

**Workspace**

To set up tests using Go's built-in `testing` package, you can create test files with the `_test.go` suffix. Here's the file tree structure for adding a basic "hello world" test:

```

  controllers
  └─ roarapp_controller_test.go
  
```

**Apply Changes**

**Terminal Commands**

No additional installation is required since `testing` is built into Go. You can run your tests with the following command:

```
go test ./...
```

You can also search for an extension to make tests easy to run in VS Code:

CHAT

@workspace /setupTests

**Workspace**

Based on your workspace structure, it appears you're working on a Go project, likely related to Kubernetes (given the presence of CRD configurations and controller files). For testing in Go, especially for Kubernetes-related projects, you should consider frameworks that support unit testing and integration testing for controllers and CRDs.

**Pick a testing framework**

Pick from these options, or use chat to tell me what you'd prefer:

- ``testing`` (built-in Go testing framework)
- ``envtest`` (from ``controller-runtime``)
- ``Ginkgo`` (for BDD-style testing)

What are the best testing practices?

Add Context...

@workspace /setupTests Set up tests in your project (Experimental)

@ Ask GPT-4o





# Testing – beyond the basics

- What other edge cases should I be testing?
- How do I test X for performance?
- How do I test Y for security?
- Test Driven Development
  - Create dependency file
  - Create test class (fail)
  - Create implementation class (succeed)

CHAT

+ ⌚ ...

1. **Zero and One:** Both are not prime numbers by definition, so it's important to test these explicitly.
2. **Negative Numbers:** While the definition of prime numbers typically does not include negatives, ensuring your function gracefully handles negative inputs is good practice.
3. **Large Prime Numbers:** Testing with large prime numbers can help ensure your algorithm performs well under stress and correctly identifies primes in all cases.
4. **Large Non-Prime Numbers:** Similarly, large non-prime numbers, especially those that are products of two large primes, can be challenging for some algorithms.
5. **Very Small Primes:** The first few primes (2, 3, 5, 7) are often corner cases in prime number algorithms, especially if any optimizations or shortcuts are taken.
6. **Sequential Primes:** Testing sequences of prime numbers (like 11, 13, 17, 19) can help identify issues with algorithms that might have patterns or periodic errors.
7. **Primes That Are Products of Small Primes:** Numbers like 49 (7x7), 121 (11x11), or 169 (13x13) can test if the algorithm correctly identifies squares of primes as non-prime.
8. **Floating Point Numbers:** While prime numbers are inherently integers, ensuring your function correctly handles or rejects floating-point numbers is a good practice.
9. **Data Types Other Than Integers:** Passing strings, lists, or other non-integer types can help ensure your function is robust against type errors.

Here's how you might extend the test suite to cover some of these cases:

```
def test_is_prime_zero():
    assert is_prime(0) == False

def test_is_prime_one():
    assert is_prime(1) == False
```

What are some common optimization techniques for prime number checking algorithms?

are there any other edge cases that should be tested?

📎 ▶ ✓



# Creating Tests Via Prompts

- **Prompt for Unit Tests:**

- *"Develop a comprehensive suite of unit tests for the <method/function> in <language/framework>. Write multiple test methods that cover a wide range of scenarios, including edge cases, exception handling, and data validation."*

- **Prompt for Integration Tests:**

- *"Write integration tests for the <method/function> in the <class>. Use mocks to simulate the <external service> and verify that it works correctly when <condition>."*

Source: <https://docs.github.com/en/copilot/using-github-copilot/example-use-cases/writing-tests-with-github-copilot#writing-unit-tests-with-copilot-chat>





# Refactoring/Updating





# Refactoring with AI

26

- Consider a code example
- Can refactor via prompt
- Resulting code

```
def is_prime(n):  
    return n > 1 and all(n % i for i in range(2, int(n**0.5) + 1))
```

The `is_prime` function should have more meaningful variable names (max of 3 characters) and comments for better readability.

```
# Check if the input is a positive integer greater than 1  
if not isinstance(num, int) or num <= 1:  
    raise ValueError("Input must be a positive integer greater  
than 1")  
  
# Check if the number is greater than 1 and not divisible by  
any number up to its square root  
if num > 1 and all(num % div for div in range(2, int(num**0.5)  
+ 1)):  
    return True  
else:  
    return False
```





# Refactoring for better testability

- Can also ask Copilot to refactor code to be more suitable for testing
- For example: "Refactor the code in #file:<filename> to make it more easily testable"

```
def is_prime(n):  
    if not isinstance(n, int):  
        raise ValueError("Input must be an integer")  
    if n < 0:  
        raise ValueError("Input must be a non-negative integer")  
  
    if n < 2:  
        return False  
    for i in range(2, n):  
        if n % i == 0:  
            return False  
    return True
```

GitHub Copilot

> Used 1 reference

To make the code more testable, we can refactor it by separating the validation logic from the core functionality. This way, we can test the validation and the prime-checking logic independently. Here is the refactored code:

```
def validate_input(n):  
    if not isinstance(n, int):  
        raise TypeError("Input must be an integer")  
    if n <= 1:  
        raise ValueError("Input must be greater than 1")  
  
def is_prime(n):  
    validate_input(n)  
  
    if n == 1:  
        return False  
    for i in range(2, int(n ** 0.5) + 1):  
        if n % i == 0:  
            return False  
    return True
```

Now, we can write separate unit tests for `validate_input` and `is_prime` functions. This separation of concerns makes the code more modular and easier to test.





- Can probe with questions to decide what needs to be updated?
  - Where do we use API v1alpha1?
  - What are the changes between v1alpha1 and v1 API versions?
- Can prompt to do updates such as “Change v1alpha1 API to v1”
- AI-recommended change areas
  - import paths
  - references to API versions
  - scaffold comments
  - Controller code

Tabnine Protected

3. Update the scaffold comments:

- Replace `//+kubebuilder:scaffold:imports` with `//+kubebuilder:scaffold:imports:v1`.
- Replace `//+kubebuilder:scaffold:scheme` with `//+kubebuilder:scaffold:scheme:v1`.

4. Update the controller code:

- If you have any custom logic or code that references the v1alpha1 version of the API, update it to use the v1 version.

Here's an example of how the updated code might look like:

```
package main

import (
    "flag"
    "os"

    // Import all Kubernetes client auth plugins (e.g. Azure, GCP, OIDC, etc.)
    // to ensure that exec-entrypoint and run can make use of them.
    "k8s.io/client-go/plugin/pkg/client/auth"
```







# Augmenting Code with AI

- Augmenting = adding additional functionality to existing code
- Prompt “The is\_prime function should log whether the input is prime or not”
- AI suggested additional code that
  - pulls in a logging framework, initializes a logger instance
  - adds logging calls for each possible output

logging module in Python. Here's an example of how you can do it:

```
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def is_prime(n):
    if not isinstance(n, int) or n <= 1:
        raise ValueError("Input must be a positive integer greater than 1")
    if n > 1 and all(n % i for i in range(2, int(n**0.5) + 1)):
        logger.info(f"{n} is a prime number")
        return True
    else:
        logger.info(f"{n} is not a prime number")
        return False
```

Insert Copy New file

In this updated version, the `logging.basicConfig(level=logging.INFO)` line sets up the





# Changing Implementations with AI

- Can select code and prompt for alternative implementations
- Consider example code
- Prompt “Explain this function in detail and suggest alternative implementations”
- Also - porting
- Prompt “convert to Go”

```
package main

import "fmt"

func fibonacci(n int) []int {
    sequence := []int{0, 1}
    for sequence[len(sequence)-1] + sequence[len(sequence)-2] <= n {
        sequence = append(sequence, sequence[len(sequence)-1] +
sequence[len(sequence)-2])
    }
    return sequence
}

func main() {
    n := 100
    fmt.Println(fibonacci(n))
}
```





# Documentation

# Using AI for general documentation via comments

- Basic methods may just do “header” comments
- By being more specific in prompt, can get more extensive comments

```

1  def is_prime(n):
2      /doc
3      /doc
4      """
5      Check if a number is prime.
6      A prime number is a natural number
7      This function uses a trial divisio
8      Parameters:
9      n (int): The number to check f
10
11     Returns:
12     bool: True if the number is pr
13
14     Examples:
15     >>> is_prime(2)
16     True
17     >>> is_prime(4)
18     False
19     >>> is_prime(17)
20     True
21     """
22     """Check if a number is prime."""
23     if n <= 1:
24         return False

```

```

# Initialize the variable i to 5
# We will use this variable to check for factors of n
i = 5

# Loop to check for factors of n
# The loop continues until i squared is greater than n
while i * i <= n:
    # Check if n is divisible by i or i + 2
    # If it is, then it is not a prime number
    if n % i == 0 or n % (i + 2) == 0:

```

Can you explain the logic used?

Verbosely comment this code so that anyone can easily understand what each line is doing

prime.py Current file

@

GPT 4o



# Generating Framework-compatible documentation

- If the SDA recognizes that the code is in a language that has a standard documentation tool associated with it, it will default to creating comments ready for that tool
- Given example code...
- Have Copilot document it
- Can then generate framework doc from it

The screenshot shows a web browser window with the address bar displaying the file path: `/Users/developer/roarv2/util/src/main/java/com/demo/util/ToJSON.html`. The page content is structured as follows:

- **Constructor Detail**
  - **ToJSON**  
`public ToJSON()`
- **Method Detail**
  - **toJSONArray**  
`public org.codehaus.jettison.json.JSONArray toJSONArray(java.sql.ResultSet rs)  
throws java.lang.Exception`  

Converts a given ResultSet into a JSONArray. Each row in the ResultSet will be converted into a JSONObject, and each column in the row will be added to the JSONObject with the column name as the key. This method provides a straightforward way to convert relational data into a structured JSON format, making it easier to serialize and transmit data over HTTP or store it in a document-oriented database.

Parameters:

    - rs - The ResultSet to be converted. This ResultSet is typically obtained from executing a SQL query against a database.

Returns:

JSONArray containing the data from the ResultSet. Each element in the JSONArray corresponds to a row in the ResultSet, represented as a JSONObject.

Throws:

    - java.lang.Exception - If there is an error during the conversion process. This could be due to issues accessing the ResultSet data, problems with the JSON library, or other unexpected issues.





# Generating Documentation for APIs

34

- If your codebase is setup for an API documentation framework like [Swagger](#), then you can have the SDA generate the corresponding Swagger documentation for your APIs

CHAT

techupskills

create Swagger documentation for the APIS in #file:EmployeeController.java

GitHub Copilot

> Used 1 reference

Given the provided excerpt from `EmployeeController.java`, I'll create a Swagger documentation outline for the APIs within this controller. Since the actual API methods (endpoints) are not included in the provided excerpt, I'll assume a common set of CRUD operations for demonstration purposes.

Swagger Documentation for EmployeeController

```
swagger: '2.0'
info:
  description: This API provides the capability to search Employee from an
  version: 1.0.0
  title: Employee Directory API
  termsOfService: 'http://swagger.io/terms/'
  contact:
    email: support@example.com
  license:
    name: Apache 2.0
```

What are the steps to configure Swagger in a Spring Boot application?

Ask Copilot or type / for commands









# Generating Functional Documentation

- SDAs are also capable of creating functional documentation that is targeted towards external users
- May also produce a set of related info including *Identifying Public APIs, Method Signatures, Endpoint Mapping, etc.*


create functional documentation explaining the various public APIs in `#file:EmployeeController.java`

 EmployeeController.java

---

 **GitHub Copilot**  

▼ Used 1 reference

 EmployeeController.java `#file:EmployeeController.java`

---

## EmployeeController API Documentation

The `EmployeeController` class provides a set of RESTful APIs to search and filter employees from an Employee Directory. These APIs are designed to be consumed by front-end applications or other services. Below is the functional documentation of the various public APIs exposed by the `EmployeeController`.

### API Endpoints

1. Get All Employees
  - **Endpoint:** `/employees/all`
  - **Method:** GET
  - **Description:** Retrieves a list of all employees in the directory.
  - **Produces:** application/json







# Extracting Summary Documentation

- Copilot can also be used to extract a high-level summary of the key documentation from certain types of projects
- This can be extremely useful for quickly understanding the significant details of the code base for a project

The key documentation extracted from the provided code and workspace structure highlights the setup and configuration of a Spring Boot application with Swagger for API documentation. Here's a summary:

## Project Structure and Configuration

- **Project Base:** Spring Boot application designed for demonstrating Swagger integration.
- **Group ID:** `com.techupskills.springboot.swagger`
- **Artifact ID:** `springboot-swagger`
- **Version:** `0.0.1-SNAPSHOT`
- **Java Version:** 17
- **Swagger Version:** 3.0.0
- **Packaging:** JAR
- **Parent:** Spring Boot Starter Parent with version 3.1.0
- **Source Encoding:** UTF-8

## Dependencies

- **Spring Boot Starter Web:** For building web, including RESTful, applications using Spring MVC.
- **Spring Boot Starter Test:** For testing Spring Boot applications with libraries including JUnit, Hamcrest, and Mockito.
- **Spring Boot Starter Data JPA:** For using Spring Data JPA with Hibernate.





## Simplifying Maintenance w/ Explaining/Onboarding



# Simplifying Onboarding

- Onboarding is a key aspect of helping devs get going with code
- Simple to have AI generate an onboarding guide

The screenshot displays a VS Code workspace for a project named 'copilot-adv'. The Explorer sidebar on the left shows the file structure, including 'app.py', 'auth.py', and 'datastore.py'. The main editor window shows the code in 'app.py', which imports Flask, auth, and datastore, and defines a Flask application with a route. Below the code editor is a terminal window showing the current directory as '/workspaces/copilot-adv' and the shell as 'bash'. On the right side, the 'CHAT' panel is open, displaying the 'Ask Copilot' interface. It includes a warning that 'Copilot is powered by AI, so mistakes are possible' and instructions on how to use context (e.g., '@' to chat with extensions). At the bottom of the chat panel, there is a button to 'Add Context...' and a dropdown menu for the model, currently set to 'GPT-4.1'. The status bar at the bottom indicates the workspace is 'copilot-adv', the branch is 'main', and the language is 'Python'.





# Explaining Code

- Key for taking over/assisting with code bases
- Useful for learning new frameworks/languages
- Using built-in “explain” functionality
- Can also have it explain how to see/run functionality

The screenshot shows the Visual Studio Code interface with a Codespace named 'copilot-adv'. The Explorer sidebar on the left shows the file structure of the workspace, including files like 'app.py', 'auth.py', 'datastore.py', and 'requirements.txt'. The main editor displays the 'app.py' file, which contains Python code for a Flask application. The terminal at the bottom shows the command prompt. On the right, the 'CHAT' sidebar is open, displaying the 'Ask Copilot' interface. The interface includes a message input area and a list of context files, with 'app.py' selected as the current file. The status bar at the bottom indicates the current file is 'app.py' and the interpreter is 'Python'.



That's all - thanks!

40

**Contact:** [training@getskillsnow.com](mailto:training@getskillsnow.com)

[techskillstransformations.com](https://techskillstransformations.com)  
[getskillsnow.com](https://getskillsnow.com)

