

Introduction to Amazon Q Developer IDE, CLI, and Kiro

Michael Lin

Sr. Solutions Architect
Amazon Web Services



AI Coding Assistants' Adoption

By 2028, 75% of enterprise software engineers will use AI code assistants, up from less than 10% in early 2023.

Gartner® Press Release, Gartner Says 75% of Enterprise Software Engineers Will Use AI Code Assistants by 2028, April 11, 2024,
<https://www.gartner.com/en/newsroom/press-releases/2024-04-11-gartner-says-75-percent-of-enterprise-software-engineers-will-use-ai-code-assistants-by-2028>. GARTNER is a registered trademark and service mark of Gartner, Inc. and/or its affiliates in the U.S. and internationally and is used herein with permission. All rights reserved.

Amazon Q Developer



Helps developers and IT professionals build faster across the entire software development lifecycle (SDLC)

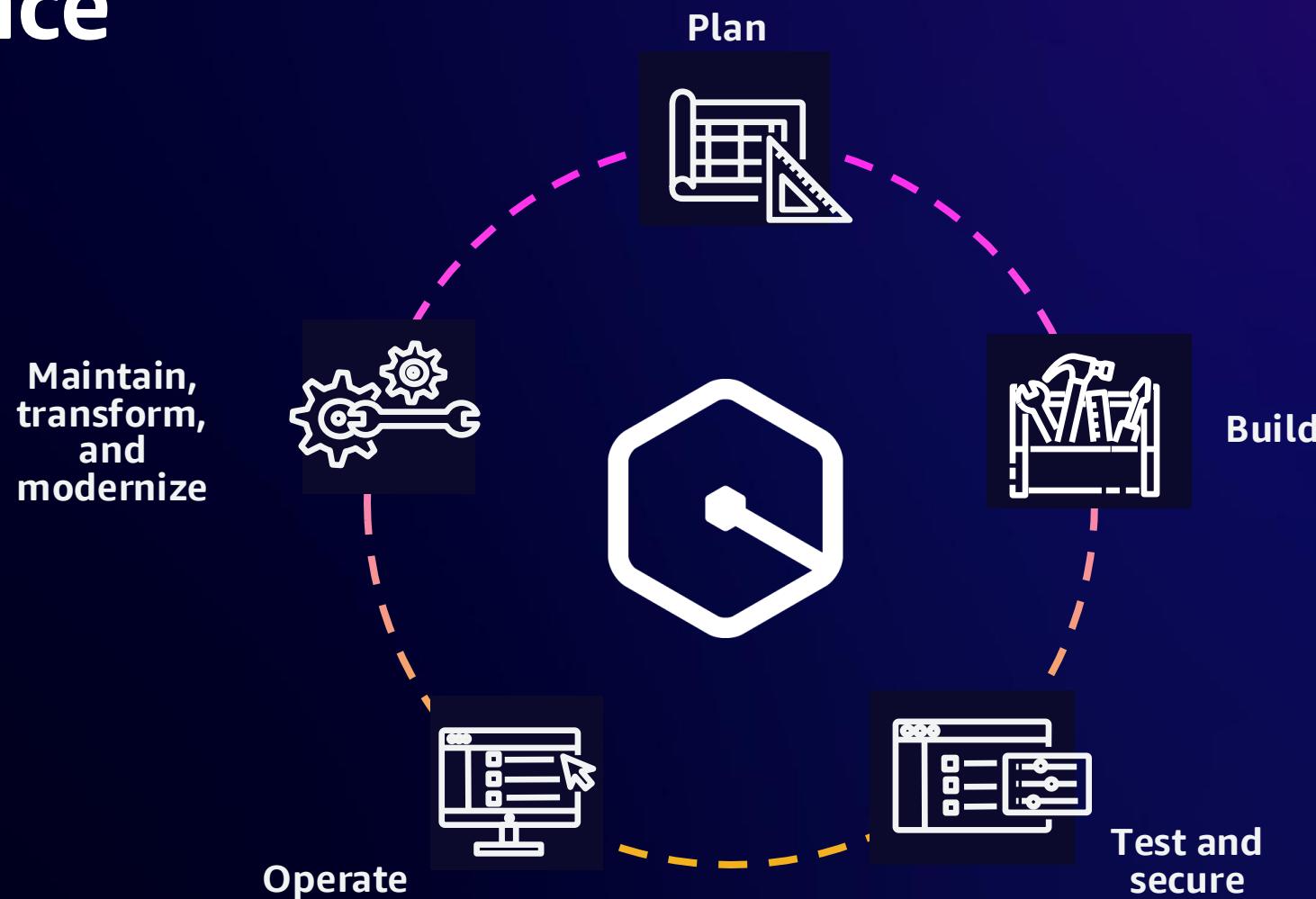
Most accurate coding recommendations

Agents can autonomously help you implement features, refactor code, perform software upgrades and more

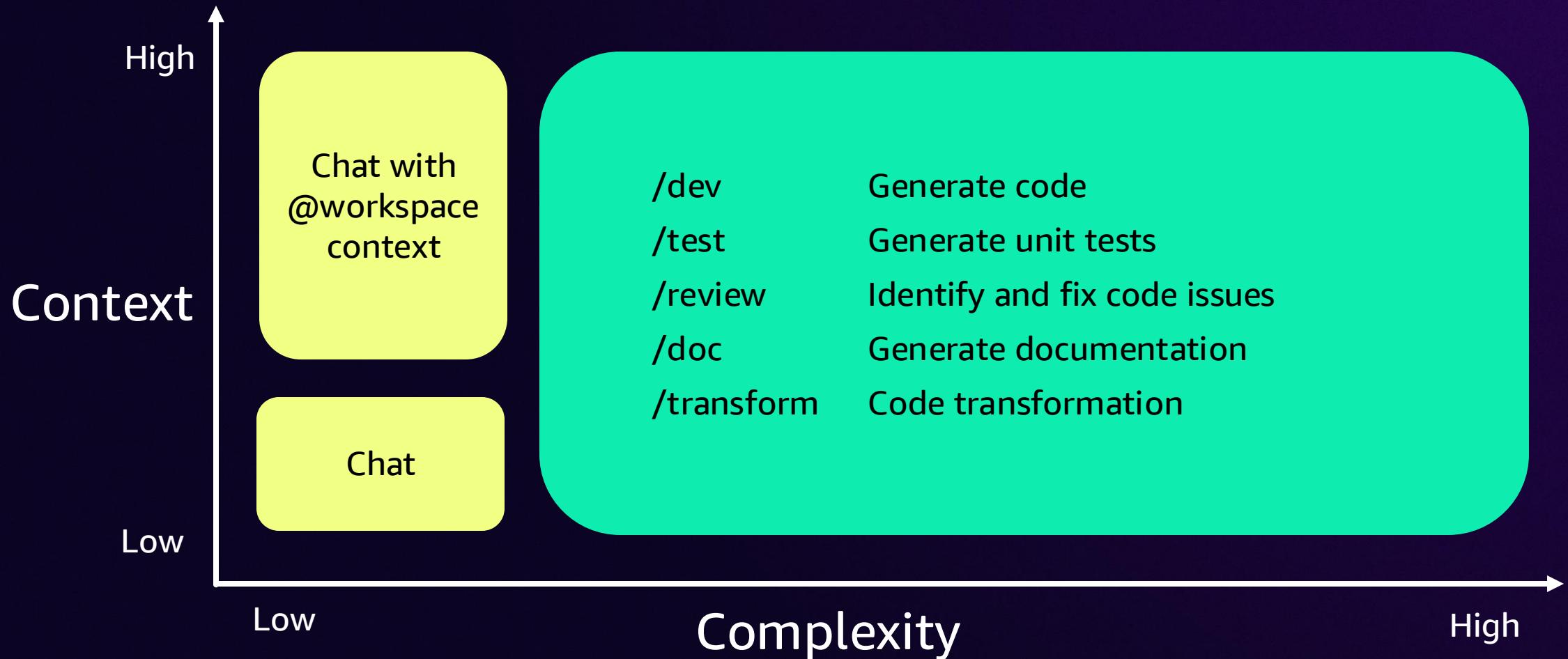
Best-in-class security vulnerability scanning and remediation

Amazon Q is built with security and privacy in mind from the start, making it easier for organizations to use generative AI safely.

Amazon Q Developer transforms the development experience



Positioning of Amazon Q Developer and Agents



Recent Updates in Q Developer IDE, CLI, and Kiro



Recent Updates on Agentic Coding in CLI and IDE

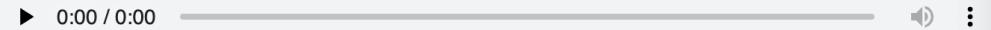
A lightning fast, new agentic coding experience within the Amazon Q Developer CLI

by Brian Beach | on 06 MAR 2025 | in [Amazon Q](#), [Amazon Q Developer](#), [Announcements](#) | [Permalink](#) | [Share](#)

Earlier today, [Amazon Q Developer](#) announced [an enhanced CLI agent](#) within the [Amazon Q command line interface \(CLI\)](#). With this announcement, Q Developer brings the latest agentic experience to the CLI that provide a more dynamic, interactive coding experience that works with you, and iteratively makes changes based on your feedback. Amazon Q Developer can now use the information in your CLI environment to help you read and write files locally, query AWS resources write code, or automatically debug issues.

Amazon Q Developer elevates the IDE experience with new agentic coding experience

by Elizabeth Fuentes | on 02 MAY 2025 | in [Amazon Q Developer](#), [Announcements](#), [AWS CLI](#), [Developer Tools](#), [Featured](#), [Launch](#), [News](#) | [Permalink](#) | [Comments](#) | [Share](#)



Today, [Amazon Q Developer](#) introduces a new, interactive, agentic coding experience that is now available in the [integrated development environments \(IDE\)](#) for [Visual Studio Code](#). This experience brings interactive coding capabilities, building upon existing prompt-based features. You now have a natural, real-time collaborative partner working alongside you while writing code, creating documentation, running tests, and reviewing changes.

<https://aws.amazon.com/blogs/devops/introducing-the-enhanced-command-line-interface-in-amazon-q-developer/>
<https://aws.amazon.com/blogs/aws/amazon-q-developer-elevates-the-ide-experience-with-new-agentic-coding-experience/>

Latest MCP Support in CLI and Context-Aware IDE

Extend the Amazon Q Developer CLI with Model Context Protocol (MCP) for Richer Context

by Brian Beach | on 29 APR 2025 | in [Amazon Aurora](#), [Amazon Q Developer](#), [Announcements](#), [PostgreSQL compatible](#), [RDS for PostgreSQL](#) | [Permalink](#) |  Share

Earlier today, [Amazon Q Developer](#) announced [Model Context Protocol \(MCP\) support](#) in the command line interface (CLI). Developers can connect external data sources to Amazon Q Developer CLI with MCP support for more context-aware responses. By integrating MCP tools and prompts into Q Developer CLI, you get access to an expansive list of pre-built integrations or any MCP Servers that support `stdio`. This extra context helps Q Developer write more accurate code, understand your data structures, generate appropriate unit tests, create database documentation, and execute precise queries, all without needing to develop custom integration code. By extending Q Developer with MCP tools and prompts, developers can execute development tasks faster, streamlining the developer experience. At AWS, we're committed to supporting popular open source protocols for agents like Model Context Protocol (MCP) proposed by Anthropic. We'll continue to support this effort by extending this functionality within the Amazon Q Developer IDE plugins in the coming weeks.

Use Model Context Protocol with Amazon Q Developer for context-aware IDE workflows

by Ritik Khatwani | on 12 JUN 2025 | in [Amazon Q](#), [Amazon Q Developer](#), [Announcements](#), [Developer Tools](#), [Generative AI](#) | [Permalink](#) |  Share

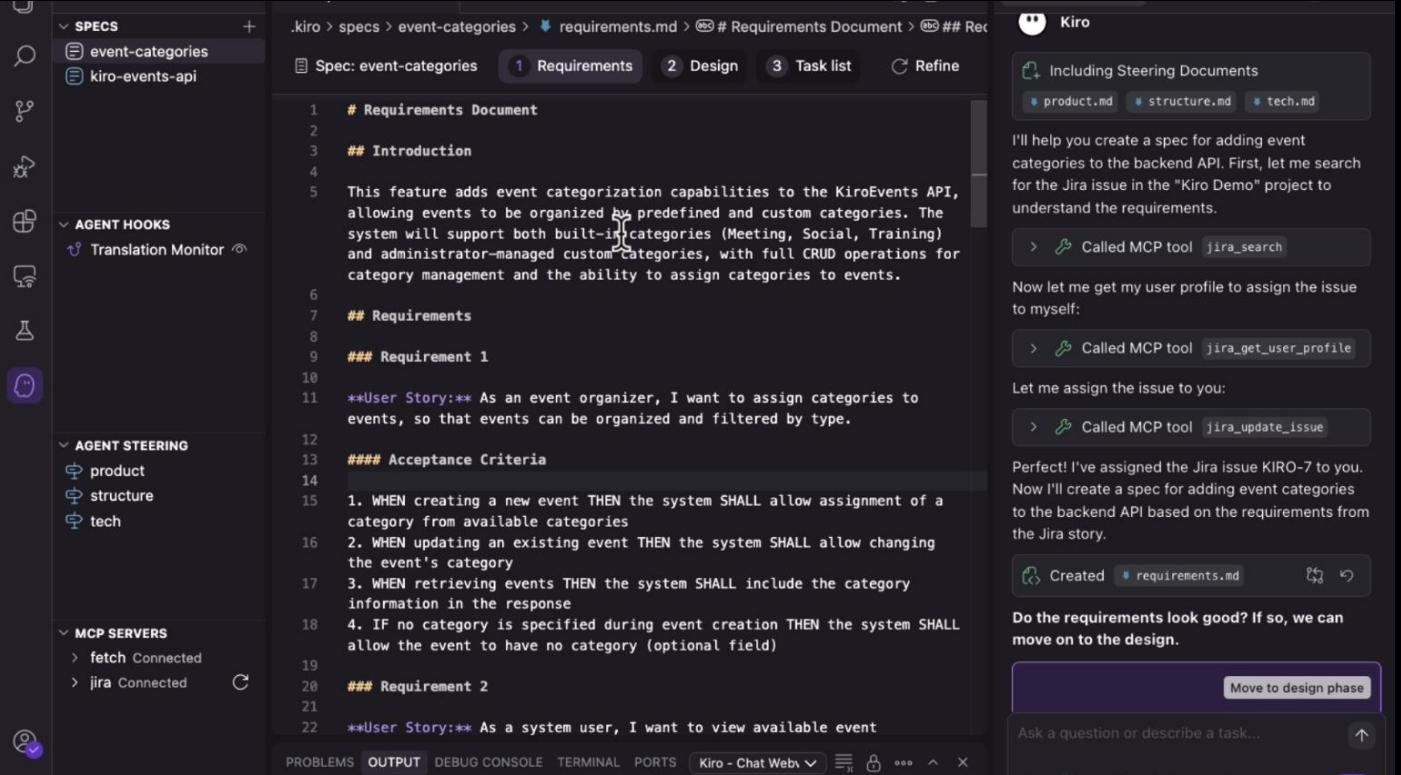
Earlier today, [Amazon Q Developer](#) announced Model Context Protocol (MCP) support in their Integrated Development Environment (IDE) plugins for Visual Studio Code and JetBrains. This allows developers to connect external tools or MCP servers to Q Developer, enabling more context-aware responses and complex workflows. MCP support has already been available in [Amazon Q Developer for Command Line](#) since April 29, 2025.

<https://aws.amazon.com/blogs/devops/extend-the-amazon-q-developer-cli-with-mcp/>
<https://aws.amazon.com/blogs/devops/use-model-context-protocol-with-amazon-q-developer-for-context-aware-ide-workflows/>



Kiro with Spec-Driven Development

- Kiro turns your prompt into clear requirements, system design, and discrete tasks
- Iterate with Kiro on your spec and architecture
- Kiro agents implement the spec while keeping you in control



The screenshot shows the Kiro IDE interface. On the left, there's a sidebar with sections for 'SPECs' (containing 'event-categories' and 'kiro-events-api'), 'AGENT HOOKS' (with 'Translation Monitor'), 'AGENT STEERING' (with 'product', 'structure', and 'tech'), and 'MCP SERVERS' (with 'fetch Connected' and 'jira Connected'). The main area is a code editor titled '.kiro > specs > event-categories > requirements.md'. The code is a Requirements Document:

```
1 # Requirements Document
2
3 ## Introduction
4
5 This feature adds event categorization capabilities to the KiroEvents API, allowing events to be organized by predefined and custom categories. The system will support both built-in categories (Meeting, Social, Training) and administrator-managed custom categories, with full CRUD operations for category management and the ability to assign categories to events.
6
7 ## Requirements
8
9 ### Requirement 1
10
11 **User Story:** As an event organizer, I want to assign categories to events, so that events can be organized and filtered by type.
12
13 #### Acceptance Criteria
14
15 1. WHEN creating a new event THEN the system SHALL allow assignment of a category from available categories
16 2. WHEN updating an existing event THEN the system SHALL allow changing the event's category
17 3. WHEN retrieving events THEN the system SHALL include the category information in the response
18 4. IF no category is specified during event creation THEN the system SHALL allow the event to have no category (optional field)
19
20 ### Requirement 2
21
22 **User Story:** As a system user, I want to view available event
```

On the right, there's a 'Kiro' panel with a message: 'I'll help you create a spec for adding event categories to the backend API. First, let me search for the Jira issue in the "Kiro Demo" project to understand the requirements.' Below it, a series of log entries show the tool interacting with MCP tools and Jira:

- > Called MCP tool jira_search
- Now let me get my user profile to assign the issue to myself:
- > Called MCP tool jira_get_user_profile
- Let me assign the issue to you:
- > Called MCP tool jira_update_issue
- Perfect! I've assigned the Jira issue KIRO-7 to you. Now I'll create a spec for adding event categories to the backend API based on the requirements from the Jira story.
- Created requirements.md
- Do the requirements look good? If so, we can move on to the design.

At the bottom, there's a text input field: 'Ask a question or describe a task...' and a button: 'Move to design phase'.

Q Developer IDE



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Create a Flask API for managing a to-do list. The API should have endpoints to add a new task, retrieve all tasks, update a task's status, and delete a task. Each task should have an ID, a description, and a status (completed or not completed). Implement basic error handling for invalid requests.

Ok, let me create a plan. This may take a few minutes.

Generating plan ...

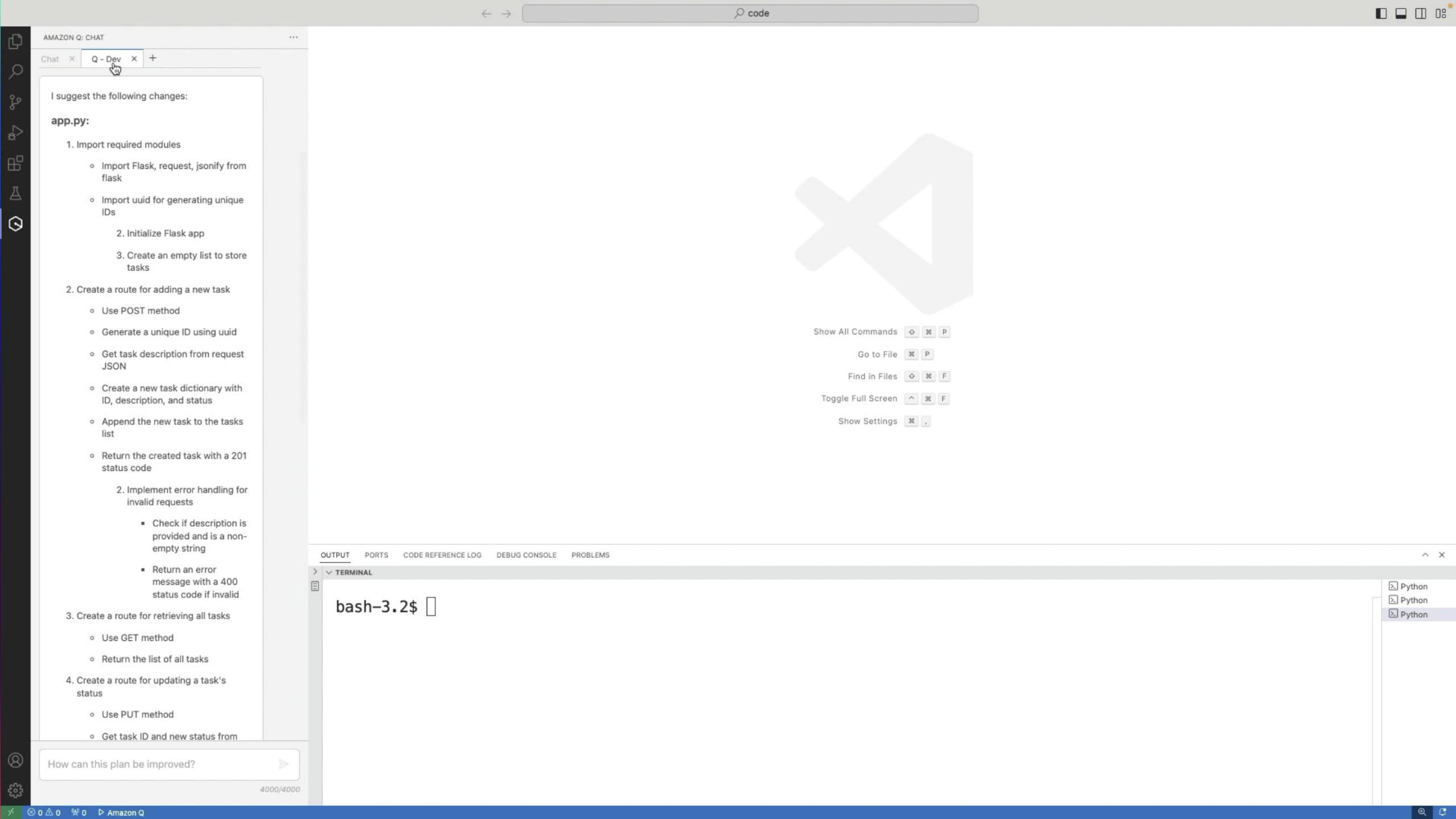
4000/4000

OUTPUT **PORTS** **CODE REFERENCE LOG** **DEBUG CONSOLE** **PROBLEMS**

> ✓ TERMINAL

bash-3.2\$





AMAZON Q: CHAT

Chat X Q - Dev X +

I suggest the following changes:

app.py:

1. Import required modules
 - o Import Flask, request, jsonify from flask
 - o Import uuid for generating unique IDs
 2. Initialize Flask app
 3. Create an empty list to store tasks
2. Create a route for adding a new task
 - o Use POST method
 - o Generate a unique ID using uuid
 - o Get task description from request JSON
 - o Create a new task dictionary with ID, description, and status
 - o Append the new task to the tasks list
 - o Return the created task with a 201 status code
 2. Implement error handling for invalid requests
 - Check if description is provided and is a non-empty string
 - Return an error message with a 400 status code if invalid
3. Create a route for retrieving all tasks
 - o Use GET method
 - o Return the list of all tasks
 4. Create a route for updating a task's status
 - o Use PUT method
 - o Get task ID and new status from

How can this plan be improved? ➤

4000/4000

code

□ □ □



Show All Commands ⌘ ⌥ P

Go to File ⌘ P

Find in Files ⌘ ⌥ F

Toggle Full Screen ⌘ ⌥ F

Show Settings ⌘ ,

OUTPUT PORTS CODE REFERENCE LOG DEBUG CONSOLE PROBLEMS

> V TERMINAL

bash-3.2\$

Python

Python

Python

Q

Amazon Q

The image shows a screenshot of a code editor interface, likely Visual Studio Code, displaying a Python file named `app.py`. The code implements a simple REST API for managing tasks using the Flask framework.

```
1 from flask import Flask, request, jsonify
2 import uuid
3
4 app = Flask(__name__)
5
6 tasks = []
7
8 @app.route('/tasks', methods=['POST'])
9 def add_task():
10     data = request.json
11     if not data or 'description' not in data or not isinstance(data['description'], str) or data
12         ['description'].strip() == '':
13             return jsonify({"error": "Invalid task description"}), 400
14
15     task = {
16         "id": str(uuid.uuid4()),
17         "description": data['description'],
18         "status": "not completed"
19     }
20     tasks.append(task)
21     return jsonify(task), 201
```

Below the code editor, there is a terminal window titled "TERMINAL" showing a bash prompt:

```
bash-3.2$
```

On the right side of the interface, there is a sidebar with several "Python" entries listed under "RECENT FILES".

← → Home Workspaces API Network

Search Postman

Invite ⚙️ Bell Upgrade

My Workspace New Import GET Get Tasks POST Add Task +

HTTP Amazon Q LiL / Add Task

POST http://127.0.0.1:5000/tasks

Params Authorization Headers (8) Body **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "description": "check in code to repo",
3   "status": "not started"
4 }
```

Status: 201 CREATED Time: 5 ms Size: 292 B Save as example

Collections Environments History ChatGPT ChatGPT Action NASA APIs NDC OpenAI API

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "description": "check in code to repo",
3   "id": "d84309ef-8d00-4a1d-940a-8e987ac93b0f",
4   "status": "not completed"
5 }
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

Q CLI with MCP



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

```
~/aws/amazonq » q chat
```

mba@mbas-MacBook-Air

```
✓ calculator loaded in 1.14 s
```

```
✓ sqlite loaded in 1.32 s
```

```
⚠ tavily_mcp has loaded in 2.01 s with the following warning:
```

The following tool names are changed:

- tavily_mcp___tavily-search -> tavily_mcp___tavilysearch

- tavily_mcp___tavily-extract -> tavily_mcp___tavilyextract

```
✓ awslabsaws_documentation_mcp_server loaded in 2.80 s
```

```
✓ awslabscore_mcp_server loaded in 2.92 s
```



Did you know?

You can use `/compact` to replace the conversation history with its summary to free up the context space

`/help` all commands • `ctrl + j` new lines • `ctrl + s` fuzzy search

🤖 You are chatting with `claude-4-sonnet`

>

> find average salary by department using sqlite database on my system. use tool at your disposal.

> I'll help you find the average salary by department using the SQLite database. Let me start by exploring what tables are available in your database.

🛠 Using tool: list_tables from mcp server sqlite

● Running list_tables with the param:

```
{  
  "name": "list_tables",  
  "arguments": {}  
}
```

Allow this action? Use 't' to trust (always allow) this tool for the session. [y/n/t]:

> t

● Completed in 0.54s

> Great! I can see there are two tables: `emp` and `dept`. Let me examine the structure of these tables to understand the data better.

🛠 Using tool: `describe_table` (trusted) from mcp server `sqlite`

- Running `describe_table` with the param:

```
{  
  "name": "describe_table",  
  "arguments": {  
    "table_name": "dept"  
  }  
}
```

- Completed in 0.5s

> Perfect! Now I can see the structure:

- `emp` table has employee data including `sal` (salary) and `deptno` (department number)
- `dept` table has department data including `deptno` and `dname` (department name)

Let me query the average salary by department:

🛠 Using tool: `read_query` from mcp server `sqlite`

- Running `read_query` with the param:

```
{  
  "name": "read_query",  
  "arguments": {  
    "query": "SELECT \n      d.dname as department_name,\n      ROUND(AVG(e.sal), 2) as average_salary,\n      COUNT(e.empno) as employee_count\nFROM emp e\nJOIN dept d ON e.deptno = d.deptno\nWHERE e.sal IS NOT NULL\nGROUP BY d.deptno, d.dname\nORDER BY average_salary DESC"  
  }  
}
```

> Excellent! Here are the results for average salary by department:

Average Salary by Department

Department	Average Salary	Employee Count
ACCOUNTING	\$2,916.67	3 employees
RESEARCH	\$2,175.00	5 employees
SALES	\$1,566.67	6 employees

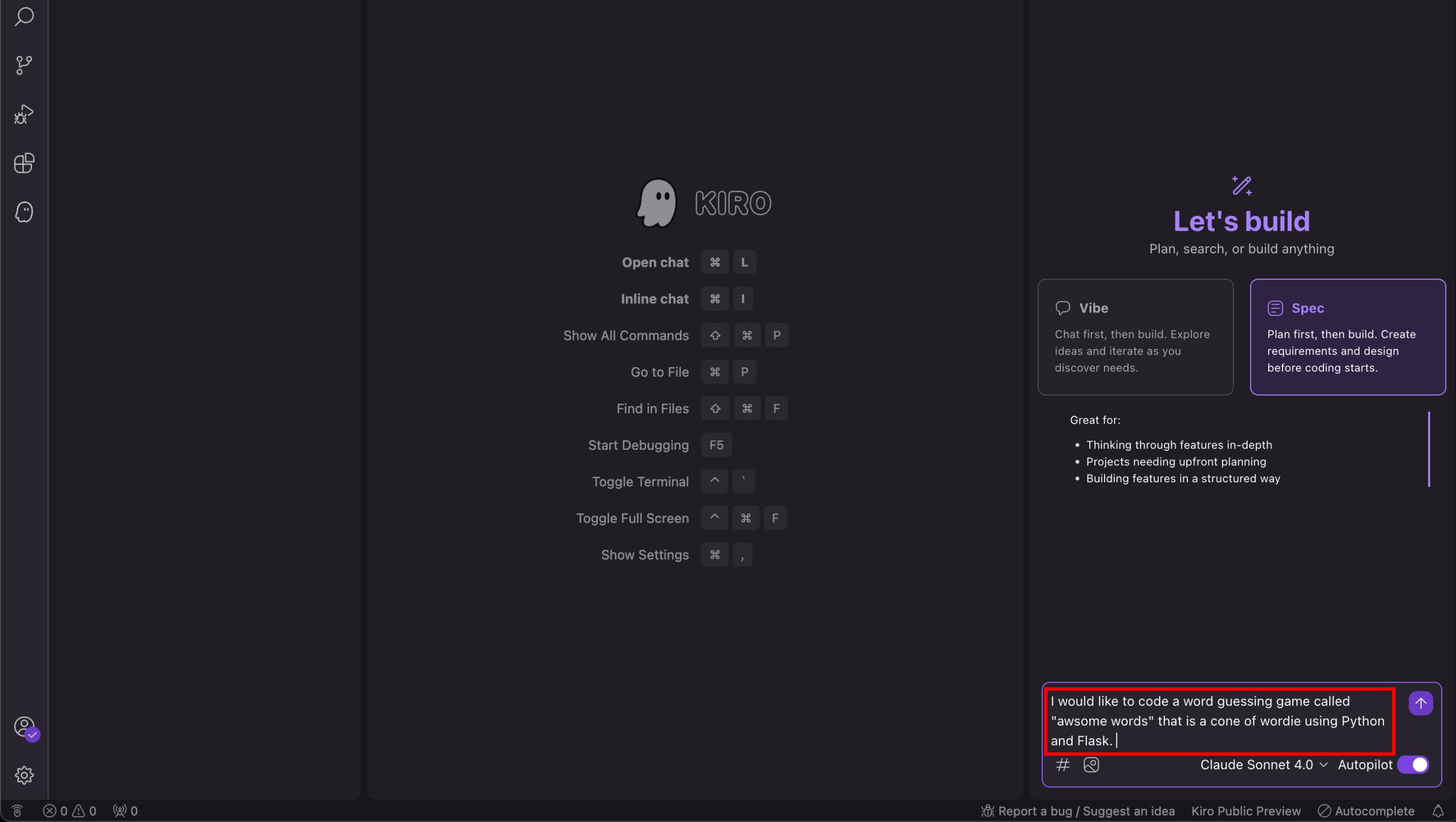
Key Insights:

- ACCOUNTING department has the highest average salary at \$2,916.67
- SALES department has the lowest average salary at \$1,566.67
- RESEARCH department falls in the middle with \$2,175.00
- The SALES department has the most employees (6), while ACCOUNTING has the fewest (3)

Kiro with Spec-Driven Development



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.



EXPLORER: KIRO_SUMMIT... + ⌂ C ⌂ ...

requirements.md x .kiro > specs > awesome-words > requirements.md > # Requirements Document > ## Requirements

Spec: awesome-words 1 Requirements 2 Design 3 Task list Refine

1 # Requirements Document

2

3 ## Introduction

4

5 Awesome Words is a web-based word guessing game inspired by Wordle, built using Python and Flask. Players attempt to guess a 5-letter word within 6 attempts, receiving feedback after each guess to help them narrow down the correct answer. The game provides an engaging, interactive experience with visual feedback and maintains game state throughout a session.

6

7 ## Requirements

8

9 ### Requirement 1

10

11 **User Story:** As a player, I want to start a new word guessing game, so that I can challenge myself with a fresh puzzle.

12

13 ##### Acceptance Criteria

14

15 1. WHEN a player visits the game page THEN the system SHALL display a clean game board with 6 empty rows for guesses

16 2. WHEN a new game starts THEN the system SHALL randomly select a valid 5-letter word from a predefined word list

17 3. WHEN a new game starts THEN the system SHALL

I would like to code a word guessing game called "awsome words" that is a clone of wordle using Python and Flask.

Kiro

Great idea! Let's create a spec for your "Awesome Words" game - a Wordle clone using Python and Flask. I'll start by creating the requirements document based on your concept.

Created requirements.md

Do the requirements look good? If so, we can move on to the design.

Move to design phase

Ask a question or describe a task...

⌂ Claude Sonnet 4.0 Autopilot

requirements.md

Spec: awesome-words Requirements Design Task list Refine

```
1 # Requirements Document
7 ## Requirements
50 ### Requirement 5
54 ##### Acceptance Criteria
57 2. WHEN a player uses all 6 attempts without
guessing correctly THEN the system SHALL display a
game over message and reveal the correct word
58 3. WHEN the game ends THEN the system SHALL
provide an option to start a new game
59
60 ### Requirement 6
61
62 **User Story:** As a player, I want an intuitive
web interface, so that I can easily interact with
the game.
63
64 ##### Acceptance Criteria
65
66 1. WHEN the game loads THEN the system SHALL
display a responsive web interface that works on
desktop and mobile devices
67 2. WHEN interacting with the game THEN the system
SHALL provide clear visual indicators for
interactive elements
68 3. WHEN using the interface THEN the system SHALL
support both keyboard input and on-screen controls
for letter entry
69 4. WHEN the page loads THEN the system SHALL
display the game title "Awesome Words" prominently
```

I would like to code a word guessing game called "awsome words" that is a clone of wordle using Python and Flask.

Kiro

Great idea! Let's create a spec for your "Awesome Words" game - a Wordle clone using Python and Flask. I'll start by creating the requirements document based on your concept.

Created requirements.md

Do the requirements look good? If so, we can move on to the design.

Move to design phase

I would like to support multiple languages. for start, let's use English, French, and German.

Claude Sonnet 4.0 Autopilot

The image shows a developer's workflow integrated with AI assistance. On the left, the VS Code interface displays a requirements document for a game named "Awesome Words". The document includes sections like "# Requirements Document", "## Introduction", and "## Requirements". A specific section of the introduction is highlighted in green, detailing the game's mechanics: "Awesome Words is a web-based word guessing game inspired by Wordle, built using Python and Flask. Players attempt to guess a 5-letter word within 6 attempts, receiving feedback after each guess to help them narrow down the correct answer. The game provides an engaging, interactive experience with visual feedback and maintains game state throughout a session." This highlighted text is also repeated below it in a larger, semi-transparent box. A red arrow points from the bottom of this box towards the AI interface on the right.

EXPLORER: KIRO_SUMMIT... ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋

requirements.md ↗ requirements.md ./kiro/specs/awesome-words ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋

.kiro > specs > awesome-words > requirements.md # Requirements Document ## Introduction

1 1 # Requirements Document
2 2
3 3 ## Introduction
4 4
5 - Awesome Words is a web-based word guessing game
- inspired by Wordle, built using Python and Flask.
- Players attempt to guess a 5-letter word within 6
- attempts, receiving feedback after each guess to
- help them narrow down the correct answer. The game
- provides an engaging, interactive experience with
- visual feedback and maintains game state
- throughout a session.
5+ Awesome Words is a web-based word guessing game
+ inspired by Wordle, built using Python and Flask.
+ Players attempt to guess a 5-letter word within 6
+ attempts, receiving feedback after each guess to
+ help them narrow down the correct answer. The game
+ supports multiple languages (English, French, and
+ German) and provides an engaging, interactive
+ experience with visual feedback and maintains game
+ state throughout a session.
6 6
7 7 ## Requirements
8 8
9 9 ### Requirement 1
10 10
11 11 **User Story:** As a player, I want to start a new
word guessing game, so that I can challenge myself
with a fresh puzzle.
12 12
13 13 ##### Acceptance Criteria
14 14

New Session × + ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋

Kiro

Retry

Kiro

Great idea! Let's create a spec for your "Awesome Words" game - a Wordle clone using Python and Flask. I'll start by creating the requirements document based on your concept.

Created requirements.md ⌂ ⌃

Do the requirements look good? If so, we can move on to the design.

I would like to support multiple languages. for start, let's use English, French, and German.

I'll update the requirements to include multi-language support for English, French, and German.

3 edits to file requirements.md ⌂ ⌃

Accepted edits to requirements.md ⌂ ⌃

Accepted edits to requirements.md ⌂ ⌃

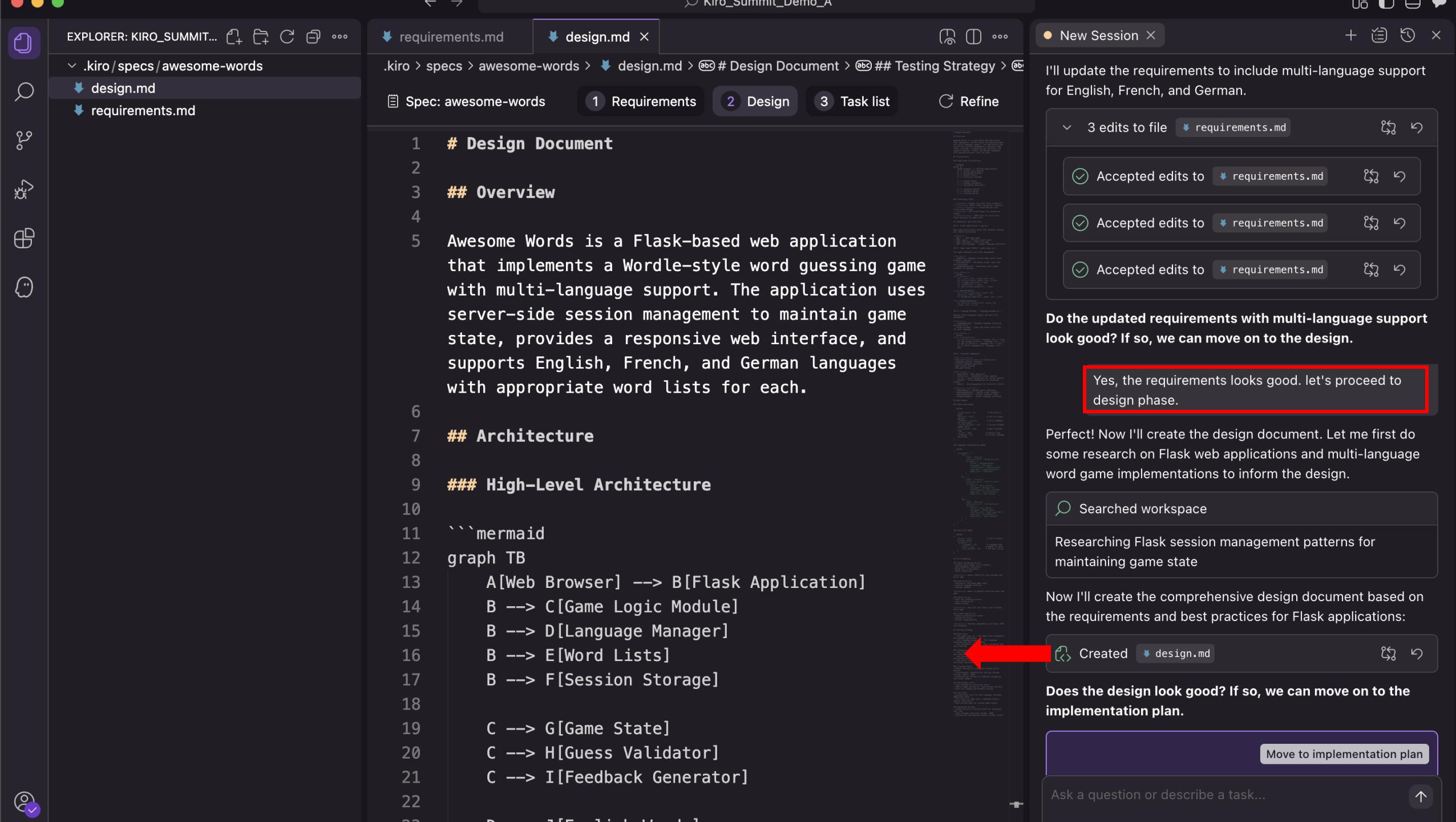
Accepted edits to requirements.md ⌂ ⌃

Do the updated requirements with multi-language support look good? If so, we can move on to the design.

Move to design phase

Ask a question or describe a task...

⌂ Claude Sonnet 4.0 ⌂ Autopilot ⌂



design.md
requirements.md

Spec: awesome-words 1 Requirements 2 Design 3 Task list Refine

```
1 # Design Document
7 ## Architecture
27
28 ### Technology Stack
29
30 - **Backend**: Python 3.8+ with Flask framework
31 - **Frontend**: HTML5, CSS3, JavaScript (vanilla)
32 - **Session Management**: Flask-Session with server-side storage
33 - **Styling**: CSS Grid/Flexbox for responsive layout
34 - **Data Storage**: JSON files for word lists, Flask sessions for game state
35
36 ## Components and Interfaces
37
38 ### 1. Flask Application (`app.py`)
39
40 Main application entry point that handles routing and request processing.
41
42 **Routes:**
43 - `GET /` - Main game page
44 - `POST /guess` - Process player guess
45 - `POST /new-game` - Start new game
46 - `POST /set-language` - Change language preference
47
48 ### 2. Game Logic Module (`game_logic.py`)
49
50 Core game mechanics and state management.
```

3 edits to file requirements.md

Accepted edits to requirements.md

Accepted edits to requirements.md

Accepted edits to requirements.md

Do the updated requirements with multi-language support look good? If so, we can move on to the design.

Yes, the requirements looks good. let's proceed to design phase.

Perfect! Now I'll create the design document. Let me first do some research on Flask web applications and multi-language word game implementations to inform the design.

Searched workspace

Researching Flask session management patterns for maintaining game state

Now I'll create the comprehensive design document based on the requirements and best practices for Flask applications:

Created design.md

Does the design look good? If so, we can move on to the implementation plan.

Move to implementation plan

Ask a question or describe a task...

Claude Sonnet 4.0 Autopilot

Ln 245, Col 50 Spaces: 4 UTF-8 LF Markdown Report a bug / Suggest an idea Kiro Public Preview Autocomplete

.kiro > specs > awesome-words > tasks.md > # Implementation Plan

Spec: awesome-words 1 Requirements 2 Design 3 Task list Update tasks

1 # Implementation Plan

92 ↳ Start task

93 - [] 6. Integrate components and test end-to-end functionality

94 ↳ Start task

95 - [] 6.1 Wire together backend and frontend

96 - Connect Flask routes with game logic classes

97 - Integrate session management with game state

98 - Connect language manager with UI localization

99 - Test complete game flow from start to finish

100 - Requirements: All requirements integration

101 ↳ Start task

102 - [] 6.2 Implement complete game scenarios

103 - Write integration tests for winning game scenarios

104 - Write integration tests for losing game scenarios

105 - Test language switching during active games

106 - Test session persistence across page refreshes

107 - Requirements: 5.1, 5.2, 6.4

108 ↳ Start task

109 - [] 6.3 Add final polish and optimization

110 - Implement loading states and user feedback

111 - Add input sanitization and security measures

112 - Optimize word list loading and caching

113 - Test cross-browser compatibility

114 - Requirements: 7.1, 7.2

Do the updated requirements with multi-language support look good? If so, we can move on to the design.

Yes, the requirements looks good. let's proceed to design phase.

Perfect! Now I'll create the design document. Let me first do some research on Flask web applications and multi-language word game implementations to inform the design.

Searched workspace

Researching Flask session management patterns for maintaining game state

Now I'll create the comprehensive design document based on the requirements and best practices for Flask applications:

Created [design.md](#)

Does the design look good? If so, we can move on to the implementation plan.

ok. let's move to implementation.

Perfect! Now I'll create the implementation plan with actionable coding tasks based on the requirements and design.

Created [tasks.md](#)

Do the tasks look good?

Finalize task list

Ask a question or describe a task...

Claude Sonnet 4.0 Autopilot

Kiro_Summit_Demo_A

EXPLORER... ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

requirements.md design.md tasks.md

.kiro > specs > awesome-words > tasks.md > @@ # Implementation Plan

Spec: awesome-words 1 Requirements 2 Design 3 Task list Update tasks

1 **# Implementation Plan**

2 **Task in progress** | View changes

3 - [] 1. Set up project structure and core dependencies

- Create Flask application directory structure with templates, static, and data folders
- Create requirements.txt with Flask and necessary dependencies
- Initialize basic Flask app with configuration
- Requirements: 7.1, 7.4

4 ↴ Start task

5 - [] 2. Implement core game logic classes

6 ↴ Start task

7 - [] 2.1 Create GameState class with state management

- Write GameState class to track word, guesses, attempts, and completion status
- Implement methods for adding guesses and checking game completion
- Create unit tests for GameState functionality
- Requirements: 1.3, 4.1, 4.2, 5.1, 5.2

8 ↴ Start task

9 - [] 2.2 Implement GuessValidator class

- Write GuessValidator to validate 5-letter alphabetic input
- Implement word existence checking against word lists
- Create unit tests for input validation scenarios

10 11 12 13 14 15 16 17 18 19

Would like to code... × Execute task: 1. Set up project structure and core dependencies 1. Set up project structure and core dependencies

Kiro

Working

Ask a question or describe a task...

Getting Started



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Deployment Decisions for Q Developer

Free tier



Code faster with code suggestions in the IDE and CLI



Use where you work: Your IDE, CLI, the AWS Management Console, etc.



Limited monthly access of advanced features

Pro tier

(\$19/mo. per user)



Manage users and policies with enterprise access controls



Customize Amazon Q to your code base



High limits of advanced features



Access to Kiro (no wait-list)

Deployment Decisions for Kiro

KIRO FREE

COMING SOON

\$0/mo.

per user

✓ 50 Vibe requests

✓ 100 Spec requests

2-WEEK TRIAL

PRO TIER

COMING SOON

\$20/mo.

per user

✓ 225 Vibe requests

✓ 125 Spec requests

PRO+ TIER

COMING SOON

\$40/mo.

per user

✓ 450 Vibe requests

✓ 250 Spec requests

POWER TIER

COMING SOON

\$200/mo.

per user

✓ 2,250 Vibe requests

✓ 1,250 Spec requests

<https://kiro.dev/pricing/>



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!



Please complete the session
survey in the mobile app

Michael Lin

linmicht@amazon.com