

Lab 016 : GitHub Copilot Agent Mode

Niveau : L100 Parcours : ? GitHub Copilot Durée : ~30 min ? Coût : GitHub Free ? Compte GitHub gratuit (le niveau gratuit inclut le mode agent)

Ce que vous allez apprendre

Ce qui distingue le mode agent du Copilot Chat classique Comment activer et utiliser le mode agent dans VS Code Comment l'agent lit votre code source, planifie et exécute des tâches en plusieurs étapes Comment connecter des serveurs MCP pour étendre les capacités de l'agent Bonnes pratiques et limitations

Introduction

GitHub Copilot dans VS Code dispose de trois modes :

Mode | Ce qu'il fait | Ask | Répond aux questions sur le code ; lecture seule | Plan | Explore le dépôt et propose un plan d'implémentation vérifiable | Agent ? | Explore de manière autonome votre code source, exécute des commandes, utilise des outils et accomplit des tâches en plusieurs étapes |

Le mode agent est le plus récent et le plus puissant. Vous décrivez un objectif, et Copilot agit comme un développeur junior : il lit les fichiers, écrit du code, exécute les tests et itère jusqu'à ce que ce soit terminé ? en vous demandant votre approbation aux points de décision clés.

info "Disponible dans VS Code 1.99+" Le mode agent nécessite VS Code 1.99 ou une version ultérieure et l'extension GitHub Copilot. Vérifiez les mises à jour si vous ne voyez pas le sélecteur de mode.

Configuration des prérequis

1. VS Code 1.99+ avec l'extension GitHub Copilot installée 2. Compte GitHub gratuit avec Copilot activé (github.com/features/copilot) 3. Un projet sur lequel travailler (nous utiliserons un projet Python simple)

tip "Démarrage rapide avec GitHub Codespaces" Open in GitHub Codespaces

Toutes les dépendances sont pré-installées dans le devcontainer.

? Fichiers de support

note "Téléchargez ces fichiers avant de commencer le lab" Enregistrez tous les fichiers dans un dossier lab-016/ dans votre répertoire de travail.

Fichier | Description | Téléchargement | [outdoorgear_api.py](#) | Script Python | ? Télécharger |

Exercice du lab

Étape 1 : Activer le mode agent

1. Ouvrez le panneau Copilot Chat (Ctrl+Shift+I) 2. Cherchez le sélecteur de mode en haut de la zone de saisie du chat 3. Sélectionnez « Agent »

Vous remarquerez que la zone de saisie change ? vous pouvez maintenant décrire des objectifs, pas seulement poser des questions.

Étape 2 : Le projet cassé ? Corrigez-le avec le mode agent ?

Cet exercice vous donne un vrai projet Python cassé à corriger en utilisant le mode agent.

L'objectif est de voir comment l'agent lit les fichiers, identifie les problèmes et les corrige ? étape par étape.

Téléchargez le projet : [diagram or code sample]

Ou copiez le fichier ? outdoorgear_api.py ci-dessous :

[diagram or code sample]

Ouvrez le dossier dans VS Code (important ? l'agent a besoin de voir l'ensemble du projet)

: [diagram or code sample]

Phase 1 : Laissez l'agent trouver et corriger les bugs

Passez en mode agent et tapez exactement ceci :

[diagram or code sample]

Observez ce que fait l'agent :

1. ? Il lit le fichier sans que vous ayez à coller quoi que ce soit 2. ? Il identifie chaque bug et explique pourquoi c'est incorrect 3. ?? Il propose des correctifs et vous demande votre approbation 4. ?? Il exécute le fichier pour vérifier que la correction fonctionne

Après avoir accepté, lancez la vérification : [diagram or code sample] Les tests 1 à 6 devraient passer. Le test 7 échouera (c'est attendu ? la fonction est manquante).

tip "Si l'agent se bloque" Essayez d'être plus précis : « Exécute python outdoorgear_api.py et montre-moi la sortie d'erreur, puis corrige le bug restant »

Phase 2 : Ajouter la fonctionnalité manquante

Demandez maintenant à l'agent d'implémenter la fonction search_by_price_range manquante :

[diagram or code sample]

L'agent devrait : 1. Lire le code existant pour comprendre les structures de données 2.

Implémenter la fonction 3. Exécuter les tests pour vérifier

Phase 3 : Écrire une suite de tests

Demandez maintenant à l'agent de créer des tests appropriés :

[diagram or code sample]

Observez l'agent : Il crée le dossier tests/ Il écrit des tests complets en utilisant des fixtures pytest Il exécute pytest dans le terminal Il corrige toutes les erreurs de test qu'il trouve

Phase 4 : Améliorer la qualité du code

[diagram or code sample]

Étape 3 : Exploration du code source

Essayez de demander à l'agent d'analyser ce qu'il vient de créer :

[diagram or code sample]

L'agent lit l'ensemble du code source et synthétise une réponse cohérente ? sans que vous ayez à coller du code.

Étape 4 : Connecter un serveur MCP (bonus)

Le mode agent prend en charge les serveurs MCP. Configurez VS Code pour utiliser le serveur MCP du Lab 020 :

.vscode/mcp.json : [diagram or code sample]

Puis demandez en mode agent : [diagram or code sample]

Étape 5 : Instructions personnalisées

Créez .github/copilot-instructions.md pour que l'agent suive toujours les conventions de votre projet :

[diagram or code sample]

Mode Plan vs. mode Agent : quand utiliser lequel

Utilisez le mode Plan quand | Utilisez le mode Agent quand | Vous devez comprendre la portée avant de modifier les fichiers | Vous avez approuvé l'approche et souhaitez l'implémenter | Les exigences ou les fichiers concernés ne sont pas clairs | La tâche nécessite plusieurs fichiers et plusieurs étapes | Vous voulez un plan vérifiable et des points de contrôle | Vous voulez que l'agent exécute des outils et vérifie le résultat | L'architecture et les compromis restent à décider | Le comportement attendu et les limites sont clairs |

Ce que l'agent a fait (en coulisses)

[diagram or code sample]

Résumé

? Lit votre code source ? pas besoin de copier/coller du code dans le chat ? Exécution en plusieurs étapes ? planifie et accomplit des tâches complexes ? Accès au terminal ? exécute les tests, vérifie les corrections ? Intégration MCP ? connecte des outils personnalisés ?

Approbation à chaque étape ? vous gardez le contrôle

Prochaines étapes

Créer un serveur MCP pour étendre le mode agent : ? Lab 020 ? Serveur MCP en Python Créer un Chat Participant VS Code (@agent personnalisé) : ? Lab 025 ? VS Code Chat Participant Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. Le ZIP contient le guide du lab et tous les fichiers d'accompagnement déclarés.

Télécharger le PDF généré Télécharger le ZIP généré

Supporting file: `outdoorgear_api.py`

`outdoorgear_api.py` ? BROKEN PROJECT ? Use Copilot Agent Mode to fix and extend this SCENARIO: This is supposed to be the OutdoorGear Inc. product management API. It was written in a hurry and has several problems: - 5 bugs that prevent it from running -

Missing features that are referenced but not implemented - No tests - Poor error handling

YOUR MISSION (using Copilot Agent Mode): Phase 1: "Fix all the bugs in this file so the tests pass" Phase 2: "Add the missing `search_by_price_range` function" Phase 3: "Create a tests/ folder with pytest tests for all functions" Phase 4: "Add type hints and docstrings to all public functions" HOW TO USE: 1. Open this FOLDER in VS Code (not just this file) 2.

Open Copilot Chat ? switch to AGENT MODE 3. Try Phase 1 first: "Fix all the bugs in `outdoorgear_api.py` so the basic tests at the bottom pass when I run `python outdoorgear_api.py`" 4. Watch the agent: it will read the file, identify bugs, and fix them 5. Continue with Phase 2, 3, 4

```
from datetime import datetime import json
```

```
CATALOG = [ {"id": "BOOT-001", "name": "TrailBlazer X200", "category": "footwear", "price": 189.99, "stock": 42, "active": True}, {"id": "TENT-001", "name": "Summit Pro Tent", "category": "camping", "price": 349.00, "stock": 15, "active": True}, {"id": "HRNS-001", "name": "ClimbTech Pro Harness", "category": "climbing", "price": 129.99, "stock": 28, "active": True}, {"id": "PACK-001", "name": "OmniPack 45L", "category": "packs", "price": 279.99, "stock": 31, "active": True}, {"id": "JACK-001", "name": "StormShell Jacket", "category": "clothing", "price": 349.00, "stock": 8, "active": True}, {"id": "BOOT-002", "name": "Summit Trail Low", "category": "footwear", "price": 99.99, "stock": 55, "active": False}, ]
```

```
ORDERS = []
```

```
def get_all_products(include_inactive: bool = False) -> list: """Return all products, optionally including inactive ones.""" if include_inactive: return CATALOG return [p for p in CATALOG if p["active"] == True]
```

```
def get_product_by_id(product_id: str) -> dict | None: """Return a product by its ID, or None if not found.""" for product in CATALOG: if product["id"] = product_id: # BUG 1: assignment instead of comparison return product return None
```

```
def add_to_cart(cart: dict, product_id: str, quantity: int) -> dict: """Add a product to the cart. Returns updated cart.""" product = get_product_by_id(product_id)
```

```
if product is None: raise ValueError(f"Product {product_id} not found")
```

```
if product["stock"] float: """Calculate total price for all items in cart.""" total = 0 for item in cart: # BUG 3: iterating over keys, not values total += item["product"]["price"] *
```

```

item["quantity"] return round(total, 2)
def apply_promo_code(total: float, code: str) -> float: """Apply a promo code discount to
the total.""" VALID_CODES = { "SUMMER10": 0.10, "OUTDOOR20": 0.20, "WELCOME5": 0.05, }
if code in VALID_CODES: discount = total * VALID_CODES[code] return round(total - discount,
2) else: raise ValueError(f"Invalid promo code: {code}") # BUG 4: should return total
unchanged, not raise
def place_order(cart: dict, customer_name: str, promo_code: str = None) -> dict: """Convert
cart to an order. Returns order confirmation.""" if len(cart) == 0: # BUG 5: should use
"not cart" or "len(cart) == 0" raise ValueError("Cannot place empty order") # actually this
is fine, but...
total = calculate_cart_total(cart)
if promo_code: total = apply_promo_code(total, promo_code)
order = { "order_id": f"ORD-{len(ORDERS) + 1:04d}", "customer": customer_name, "items":
cart, "total": total, "placed_at": datetime.now().isoformat(), "status": "confirmed", }
# Deduct stock for product_id, item in cart.item(): # BUG 5: .item() should be .items() for
product in CATALOG: if product["id"] == product_id: product["stock"] -= item["quantity"]
ORDERS.append(order) return order
--- MISSING FEATURE --- The search_by_price_range function is referenced in the tests below
but hasn't been implemented yet. Parameters: min_price: float, max_price: float Returns:
list of products in that price range (inclusive), sorted by price ascending
===== Basic tests ? these should
pass after Phase 1 ===== if __name__
== "__main__": print("Running basic tests...\n")
# Test 1: get all active products active = get_all_products() assert len(active) == 5,
f"Expected 5 active products, got {len(active)}" print(f"? Test 1: get_all_products() ? 5
active products")
# Test 2: get product by ID boot = get_product_by_id("BOOT-001") assert boot is not None,
"BOOT-001 should exist" assert boot["name"] == "TrailBlazer X200" print(f"? Test 2:
get_product_by_id() ? found BOOT-001")
# Test 3: add to cart and calculate total cart = {} cart = add_to_cart(cart, "BOOT-001", 2)
cart = add_to_cart(cart, "TENT-001", 1) total = calculate_cart_total(cart) assert abs(total
- 728.98) < 0.01, f"Expected $728.98, got ${total}" print(f"? Test 3: cart total =
${total}")
# Test 4: promo code discounted = apply_promo_code(728.98, "SUMMER10") assert
abs(discounted - 656.08) < 0.01, f"Expected $656.08, got ${discounted}" print(f"? Test 4:
promo SUMMER10 applied = ${discounted}")
# Test 5: invalid promo code should NOT raise ? should return original total unchanged =
apply_promo_code(728.98, "INVALID_CODE") assert unchanged == 728.98, f"Invalid code should
return original total, got ${unchanged}" print(f"? Test 5: invalid promo code returns
original total = ${unchanged}")
# Test 6: place order order = place_order(cart, "Alex Chen", "SUMMER10") assert
order["status"] == "confirmed" assert order["customer"] == "Alex Chen" print(f"? Test 6:
order placed ? {order['order_id']}")
# Test 7: search by price range (requires Phase 2) budget_items = search_by_price_range(50,
200) assert len(budget_items) == 2 # TrailBlazer X200 ($189.99) and ClimbTech ($129.99)
assert budget_items[0]["price"] <= budget_items[1]["price"] # sorted ascending print(f"?
Test 7: search_by_price_range(50, 200) ? found {len(budget_items)} products")
print("\n? All tests passed! Ready for Phase 3 (write tests) and Phase 4 (type hints).")

```