演習 5: GitHub Copilotコーディングエージェントの成果をレビューする

このラボの最初に、いくつかの Issue を GitHub Copilotコーディングエージェントに割り当てました。コードへのドキュメント追加や、別チームが続けて作業できるようにいくつかのエンドポイント生成を依頼しました。ここでは、Copilot が提案したコード変更を確認し、必要に応じてフィードバックを与えて改善していきます。

シナリオ

これまで繰り返し強調してきたとおり、生成 AI を導入しても、ソフトウェア設計と DevOps の基本は変わりません。生成されたコードは必ずレビューし、通常の DevOps プロセスを踏みます。その前提で、チーム全体にレビューを依頼する前に、ドキュメント作成と新規エンドポイント作成に関する GitHub Copilot の提案を見ていきましょう。

セキュリティと GitHub Copilotコーディングエージェント

Copilotコーディングエージェントは非同期かつ無監督でタスクを実行するため、安全性を確保する制約がいくつか設けられています。たとえば次のとおりです。

- Copilot はあなたのリポジトリに対して読み取り権限のみを持ち、**コード用の専用ブランチに対してのみ**書き込み権限を持ちます。
- コーディングエージェントは GitHub Actions 内で動作し、作業用の分離された一時環境を作成します。
- いかなる GitHub Actions ワークフローも、実行前に人間の承認が必要です。
- 外部リソース（MCP サーバーを含む）へのアクセスはデフォルトで制限 されています。

生成されたドキュメントのレビュー

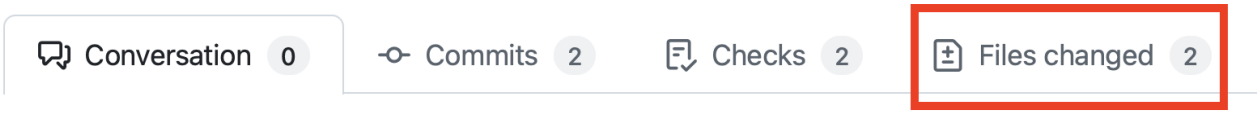
まず、GitHub Copilotコーディングエージェントが生成した最初の Pull Request（PR）—コードへのドキュメント追加—を確認します。ここでは GitHub.com の標準的な PR インターフェイスを使います。

> [!NOTE] PR を閲覧すると、ファイアウォールにより GitHub Copilot がブロックされている旨の警告が出ることがあります。これは**想定どおり**です。デフォルトでは、Copilot は外部リソース（外部 MCP サーバーへの呼び出しを含む）へのアクセスが制限されています。必要に応じて、Copilotコーディングエージェント向けのファイアウォールをカスタマイズまたは無効化できます。

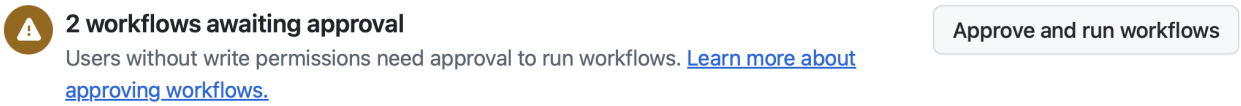
1. github.com 上のあなたのリポジトリに戻ります。
2. **Pull Requests** を選択して PR 一覧を開きます。
3. **Add missing documentation** のようなタイトル（もしくはそれ以上に詳細なタイトル）の PR を開きます。

> [!NOTE] Copilot がまだ作業中であれば、Issue に **[WIP]** フラグが付いています。その場合は Copilot の作業完了を待ちます。数分かかることがあるので、これまで学んだ内容を振り返ってもよいでしょう。

4. PR の準備ができたら **Files changed** タブを選び、変更点を確認します。



5. 新しく追加された docstring やその他ドキュメントを含む更新コードを確認します。具体的な変更内容は状況によって異なります。
6. 更新内容をレビューして問題がなければ、**Conversation** タブに戻ってスクロールします。
7. 承認待ちのワークフローがあることを示すインジケーターが表示されるはずです。
8. **Approve and run workflows** ボタンをクリックし、ワークフローの実行を許可します。



9. PR のチェック欄にワークフローがキューイングされるのが見えるはずです。うまくいけば、バックエンドとフロントエンドの両方でワークフローが成功します。完了まで数分かかる場合があります。

GitHub Copilot に変更を依頼する

Copilot との PR 作業は一方通行ではありません。PR にコメントを追加したり、コードにインラインコメントを付けたりできます。Copilot はこれらのコメントを読み取り、対応のための新しいセッションを開始します。結果は非決定的であるため、具体的に何を求めるべきかの定型文は提示できませんが、依頼のアイデアとしては次のようなものがあります。

- 各コードファイルの先頭に、そのファイルの役割を簡潔に説明するコメントヘッダーを追加する。
- TypeScript と Svelte のファイルに docstring（または相当）を追加する。
- server と client の各フォルダーに、それぞれのコードベースの概要を説明した README を作成する。

1. 生成されたドキュメントに対し、変更を求めるコメントを 1 つ追加します。上記のアイデアを使っても、独自のリクエストでも構いません。
2. Copilot が変更を行う様子をそのまま見ていて問題ありません。
3. **View session** を選択すると、Copilot の作業の様子を確認できます。Copilot が更新のために新しいセッションを開始する点に注目してください。
4. **Back to pull request** を選択すると、PR 画面に戻れます。

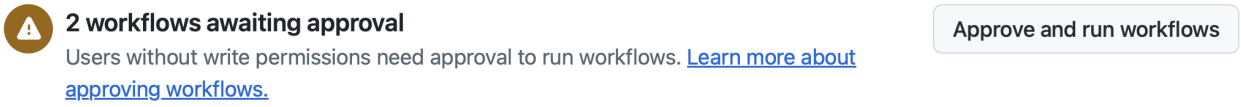
5. Copilot の変更が完了すると、PR に新しいコミットが追加されます。
6. **Files changed** タブで変更内容を再度レビューします。

満足できるまで反復して構いません。準備ができたら、PR をドラフトから Ready に変更し、main ブランチへマージします。

新しいエンドポイントをレビューする

次に、ゲームの作成・更新・削除のためにゲーム API にエンドポイントを追加する Issue を解決するために、Copilot が生成した PR に戻りましょう。

1. GitHub.com のあなたのリポジトリに戻ります。
2. **Pull Requests** タブを選びます。
3. **Add CRUD endpoints for games API** のようなタイトル（またはより詳細なタイトル）の PR を選びます。
4. **Files changed** タブで生成されたコードを確認します。
5. レビューを終えて問題なければ、**Conversation** タブに戻ってスクロールします。
6. 承認待ちのワークフローがあることを示すインジケーターが表示されます。
7. **Approve and run workflows** ボタンをクリックし、ワークフローの実行を許可します。



8. PR のチェック欄にワークフローがキューイングされるはずです。うまくいけば、バックエンドとフロントエンドの両方で成功します。完了まで数分かかることがあります。
9. **オプション:** このブランチに Codespace から切り替えて、新しいエンドポイントを手動テストすることもできます。Codespace に移動してターミナルを開き、次のコマンドを実行します（**<branch-name>** は Copilot が作成したブランチ名、例：**copilot/fix-8** に置き換え）。

```
git fetch origin
git checkout <branch-name>
```

Copilot は新しいエンドポイントを作成しました！先ほどと同様に、Copilot コーディングエージェントに更新を依頼しながら反復的に作業できます。たとえば、重複を減らすためにエラーハンドリングを集約する、コメントや docstring の追加を徹底する（—これは**カスタムインストラクションを更新する前**に割り当てたタスクでした—）などです。これまでと同様、**Conversation** タブで新規コメントを追加すれば、Copilot がそれを検知して新しいセッションを開始します。

オプション演習 - さらに Issue を追加してエージェントの能力を試す

コードベースを探索し、非同期に変更を加えられるペアプログラマにアクセスできるのは強力です。各タスクの最初のイテレーションまで素早く到達し、あなたはレビューと舵取りに集中したり、エディタで引き継いで作業を続けたりできます。

ここまで大きく前進しましたが、ぜひリポジトリに追加の Issue を作成し、Copilot に解決させてみてください。アイデア例：

- ゲーム詳細ページに出資者（backer）向け関心フォームを作成する
- ゲーム一覧エンドポイントにページングを実装する
- Flask API に入力検証とエラーハンドリングを追加する
- その他？あなたなら何を検討しますか？

まとめ

おめでとうございます！ラボを完了しました。IDE からリポジトリまで、GitHub Copilot が提供するさまざまな機能を体験しました。特に次の点に取り組みました。

- **GitHub Copilot と Model Context Protocol（MCP）を活用して開発を効率化**：Copilot がリポジトリとやり取りできるよう GitHub MCP サーバーを設定し、エージェントモードで詳細なバックログを作成しました。
- **カスタムインストラクションやプロンプトファイルで Copilot をプロジェクト標準に従わせる方法を探究**：Copilot にコンテキストを与えるカスタムインストラクションファイルを作り、ガイドライン準拠のコード生成を促し、反復タスクや確立済みプラクティスのためのプロンプトファイルを使いました。
- **エージェントモードで新機能を実装し、バックエンド／フロントエンドにまたがる変更を連携・自動化**：ゲーム一覧ページにカテゴリ／出版社フィルタを導入し、クライアント・バックエンド・テストにまたがる変更を実施しました。
- **Issue の割り当てと PR で、ペアプログラマとしての Copilot を体験**：バックログの Issue を Copilot に割り当て、PR 作成・計画立案・実装・フィードバックに基づく反復を行いました。

これは始まりに過ぎません。あなた自身のプロジェクトで Copilot をどう活用するのか、楽しみにしています。ラボを楽しんでいただけたなら幸いです。Happy coding!

リソース

- GitHub Copilot
- Copilot エージェントについて
- Copilot への Issue の割り当て
- Copilot コーディングエージェントのセットアップ／ワークフローベストプラクティス
- Copilot コーディングエージェントのファイアウォール設定
