



GitHub Copilot

Conseils et astuces Vol. 2

Meilleures pratiques destinées aux développeurs visant à optimiser leur productivité avec GitHub Copilot



Dennis Gassen

Avant-propos

Lors de mon travail au sein de Microsoft Allemagne, j'ai eu l'occasion de collaborer à l'élaboration de plusieurs e-books. Le tout premier e-book, si ma mémoire est bonne, était la brochure Visual Studio 2013 qui comportait des articles sur Visual Studio, Team Foundation Server, Windows 8 et Windows Azure (comme on l'appelait alors). D'autres e-books ont suivi sur Visual Studio, Visual Studio Code ou sur l'optimisation de la collaboration à distance des équipes de développement peu après le début de la pandémie de Covid. Pour le dernier e-book, Thomas Dohmke, aujourd'hui PDG de GitHub, avait écrit un article consacré au changement de culture au sein de GitHub. Ces changements sont déjà largement engagés. Nous parlons bien entendu ici de GitHub Copilot.

Dès 2015, Satya Nadella, le PDG de Microsoft, déclarait que chaque entreprise deviendrait une entreprise de logiciels. Depuis, ce dernier a réitéré ses propos sous différentes formes. Il s'agit avant tout de donner aux équipes de développement ou aux « Citizen Developers » les moyens de tirer le meilleur parti de leur créativité afin de résoudre des problèmes ou de créer des produits et des solutions apportant une réelle valeur ajoutée. La priorité est accordée, d'une part, à l'innovation technologique et, d'autre part, au recentrage sur l'essentiel.

Et c'est justement le rôle que va jouer GitHub Copilot. Afin de vous faire mieux connaître cette nouvelle offre du point de vue des utilisateurs, nous avons décidé de publier un nouvel e-book comportant des conseils et des astuces concrets pour utiliser GitHub Copilot. L'objectif est d'enrichir progressivement cet e-book avec de nouveaux conseils et astuces. Si vous souhaitez apporter votre contribution dans une prochaine édition, veuillez nous envoyer un e-mail à l'adresse suivante : techwiese@microsoft.com. Nous serons ravis de vous lire.

En attendant, je vous souhaite une agréable lecture de cette première édition de notre e-book.

Dennis Gassen



Malte Lantin

Malte Lantin, ingénieur solutions en chef chez GitHub, aide les clients stratégiques européens à adopter GitHub Enterprise, une plateforme centrale de développement sécurisé et de mise à disposition de logiciels.

Introduction : l'avenir du développement de logiciels avec GitHub Copilot

par *Malte Lantin*

Trois ans à peine après le premier livre blanc interne sur GitHub publié en 2020 et le lancement de GitHub Copilot en 2022, le développement de logiciels assisté par l'IA s'est déjà imposé comme un standard incontournable pour de nombreux développeurs et entreprises. Il faut d'ailleurs rappeler que beaucoup considéraient encore récemment que l'intervention de l'IA dans le développement de logiciels n'était que pure fiction. Aujourd'hui, GitHub Copilot représente un véritable tournant dans l'histoire des outils de développement de logiciels.

Grâce aux améliorations permanentes et à des fonctionnalités spécifiquement destinées à une utilisation professionnelle, GitHub Copilot s'est déjà hissé au rang des outils incontournables. Nous allons d'abord vous présenter le fonctionnement de base de GitHub Copilot et la valeur ajoutée qu'il apporte. Puis vous découvrirez dans les chapitres suivants de nombreuses études de cas.

Innovations et fonctions technologiques

GitHub Copilot bénéficie de nombreuses années de recherches dans le domaine des modèles de langage, de leur déploiement sécurisé dans Azure OpenAI Services et d'une intégration parfaite au processus de développement. Les développeurs disposent désormais d'une complétion de code basée sur l'IA dans Visual Studio Code, Visual Studio, les IDE de JetBrains, et d'autres éditeurs, qui récupère le contexte à partir des fichiers de code source en cours de traitement et les informations à partir de chaque éditeur. L'IA peut ainsi générer des suggestions de code adaptées et très pertinentes dans le processus de développement en cours. Le modèle FIM (fill-in-the-middle) est utilisé pour cette complétion de code. Il permet de proposer en permanence les suggestions les mieux adaptées au contexte, tout en prenant en compte le style de chaque projet.

Avec le lancement de GitHub Copilot Chat, dès la fin 2023, les possibilités de développement assisté par l'IA ont été à nouveau considérablement étendues. Grâce à l'interface de conversation intégrée à l'environnement de développement, il est entre autres possible de donner des instructions plus complexes, d'obtenir des recommandations complètes, de récupérer des commentaires sur le code, de générer des tests ou de créer des environnements de travail avec GitHub Copilot. L'approche itérative basée sur le langage naturel facilite l'accès à GitHub Copilot Chat et permet à tous les développeurs d'accélérer le développement des logiciels, de résoudre les problèmes éventuels et d'apprendre de nouvelles technologies. La connaissance du contexte extrait par GitHub Copilot à partir de l'éditeur de code est un facteur de différenciation essentiel par rapport aux autres modèles de langage IA basés sur les conversations. GitHub Copilot Chat peut accéder à des connaissances extraites des fichiers demandés ou d'instructions de ligne de commande pour proposer à tout moment des suggestions pertinentes.

GitHub Copilot s'enrichit continuellement et, depuis le printemps 2024, il est également disponible comme extension des outils GitHub CLI permettant de commenter ou générer des commandes sur la ligne de commande.

Depuis février 2024, les entreprises qui utilisent GitHub Enterprise Cloud peuvent également bénéficier d'une intégration encore plus poussée de GitHub Copilot dans leurs processus de développement : intégration de GitHub Copilot Chat à l'interface web de GitHub, traitement assisté par l'IA des demandes de tirage (pull request), accès aux connaissances de leurs propres dépôts de code source et de leur propre documentation, et bientôt, ajustement des modèles d'intelligence artificielle selon les bases de code de l'entreprise. Cette prise en charge par GitHub Copilot permet d'enrichir et d'améliorer nettement les parties du processus de développement qui se déroulent en dehors de l'environnement de développement.

Impact sur la productivité et la qualité du code

Avec plus de trois milliards de lignes de code générées, GitHub Copilot a déjà apporté la preuve incontestable de son efficacité et de son adoption massive par la communauté des développeurs. À l'heure actuelle, plus de 1,8 million de développeurs de logiciels utilisent GitHub Copilot et plus de 50 000 organisations ont choisi de mettre en place cette technologie. L'impact de GitHub Copilot sur la productivité des développeurs est à la fois considérable et parfaitement quantifiable. Une étude a révélé que plus de 30 % des suggestions de GitHub Copilot ont été acceptées, ce qui souligne la pertinence et l'utilité de cet outil pour les développeurs au quotidien. Avec une augmentation de la productivité pouvant atteindre 55 %, sa valeur ajoutée est évidente. Toutefois, il est tout aussi important de noter que les développeurs interrogés se disent plus satisfaits et plus productifs lorsqu'ils utilisent GitHub Copilot. Ainsi, des études montrent une amélioration de la qualité du code, ainsi que de l'efficacité et de la rapidité des révisions de code. Les développeurs rapportent pouvoir travailler plus longtemps en restant concentrés grâce à GitHub Copilot en raison de la suppression des fréquents changements de contexte et autres distractions.

L'expérience de développement nettement améliorée offre des avantages qui vont bien au-delà des gains de productivité. En automatisant les tâches routinières et répétitives, GitHub Copilot permet aux développeurs de se concentrer sur des aspects plus complexes et créatifs de leurs projets et améliore ainsi leur satisfaction au travail.

Conclusion

En résumé, GitHub Copilot représente une étape majeure dans l'évolution des outils de développement. Il symbolise le début d'une nouvelle ère de la programmation assistée par l'IA, qui améliore considérablement la productivité des développeurs, la qualité du code et influe de manière positive sur le bien-être général de l'équipe de développement. En raison de son évolution permanente et de son intégration accrue au cours des prochaines années dans le processus de développement de logiciels, GitHub Copilot va devenir un outil indispensable dont l'impact va s'accentuer.

Dans les chapitres suivants, nous vous présentons des applications possibles de GitHub Copilot, qui illustrent son utilité concrète au quotidien. Nous vous souhaitons une lecture agréable et beaucoup de plaisir à la découverte de GitHub Copilot.

Informations complémentaires :

- [Copilot fait de GitHub LA plateforme de développement alimentée par l'IA](#)
- [L'impact économique du cycle de vie de développement alimenté par l'IA et les leçons tirées de GitHub Copilot](#)
- ¹ [Mesurer l'impact de GitHub Copilot sur la productivité et la satisfaction des développeurs](#)
- ² [Mesurer l'impact de GitHub Copilot sur la qualité du code](#)

Contenus

Avant-propos	2
Introduction : l'avenir du développement de logiciels avec GitHub Copilot	3
GitHub Copilot : aperçu des versions et fonctionnalités	7
Utilisation de GitHub Copilot	10
Migration d'applications entre les différents langages de programmation avec GitHub Copilot Chat	12
Plus de faux texte	19
Codage intelligent avec GitHub Copilot	23
D'une idée à son déploiement en moins de 30 minutes	28
Écriture de tests avec GitHub Copilot Chat	32
Résolution proactive des problèmes sur les appareils Windows	34
À quoi ressemblait le code à ses débuts ?	38
Automatisation des tâches de développement fastidieuses avec GitHub Copilot Chat	40
Amélioration des invites pour un meilleur code : conseils et astuces pour les développeurs	45
Noms de fonctions et variables pertinents	48
Prise en charge de la génération de cas de test avec GitHub Copilot	50
Trois mois d'évaluation de l'impact de GitHub Copilot	53
GitHub Copilot, un assistant IA pour la création de logiciels accessibles	58
De l'exigence au composant	68
Tirer parti de la syntaxe de conversation : participants au chat, variables de chat et commandes slash	73
Utilisation de GitHub Copilot pour vos projets	78
Ressources complémentaires	79

GitHub Copilot : aperçu des versions et fonctionnalités

GitHub Copilot introduit l'intelligence artificielle dans le développement et aide les développeurs à créer plus rapidement du code de meilleure qualité. L'assistant IA aide les développeurs à écrire des lignes de code ou même des blocs entiers de code et peut proposer des suggestions d'extension ou d'amélioration du code existant. GitHub Copilot permet ainsi d'augmenter de 50 % la vitesse de développement en améliorant considérablement les procédures opérationnelles des développeurs.

Vous utilisez par exemple GitHub Copilot pour écrire un commentaire qui décrit la logique souhaitée et définit vos conventions de style préférées. L'IA se charge du reste pour vous permettre de vous concentrer sur les tâches véritablement complexes. Vous pouvez ainsi améliorer votre productivité et accomplir des tâches répétitives bien plus rapidement et en toute sérénité. Parallèlement, GitHub Copilot aide les développeurs à maîtriser un nouveau langage ou cadre de développement bien plus rapidement que s'ils devaient le faire en lisant des documents ou en effectuant des recherches sur le Web.

Prise en charge de nombreux langages de programmation et développement de code sécurisé

GitHub Copilot prend en charge une multitude de langages de programmation, notamment C#, C++, Python, JavaScript, TypeScript, Go et Ruby. Plus de 50 000 entreprises utilisent déjà l'assistant IA, dont un tiers des entreprises du classement Fortune 500.

GitHub Copilot améliore également le travail en équipe. Le système indexe vos dépôts, comprend le code qui y est enregistré et vous aide à vous lancer plus rapidement dans de nouvelles bases de code. Lorsque vous associez de nouveaux dépôts, vous pouvez vous appuyer sur les travaux réalisés par d'autres. GitHub Copilot permet également aux développeurs de spécifier une invite pour générer des fonctions et des diagrammes. GitHub Copilot évite les vulnérabilités et les failles de sécurité dès la génération du code, car le système est optimisé pour générer du code sécurisé. L'IA bloque les lignes de code dangereuses.

Suggestions basées sur l'IA en temps réel : sécurité accrue et réduction des erreurs

Le point fort de GitHub Copilot réside surtout dans les propositions en temps réel de suggestions basées sur l'IA pour le développement de code. Pendant que vous écrivez des lignes de code, GitHub Copilot analyse ce code et présente automatiquement des suggestions pour le compléter. Il en découle une parfaite harmonie entre le développeur et l'IA. Les développeurs peuvent également créer des messages de validation (commit) automatisés à l'aide de GitHub Copilot. Il existe de nombreuses applications capables de faciliter considérablement le travail quotidien des développeurs.

Github Copilot les aide non seulement à créer du code ou à améliorer des lignes de code existantes, mais l'assistant IA est également un outil très utile pour détecter des erreurs dans le code. La « conversation » en langage naturel est la base de ce processus. Lorsque vous vous trouvez bloqué, il vous suffit de demander à GitHub Copilot. GitHub Copilot agit intelligemment, car l'IA personnalise les réponses en se basant sur vos connaissances techniques et sur la documentation disponible à partir de laquelle les modèles IA poursuivent leur apprentissage.

L'utilisation de GitHub Copilot n'implique pas l'abandon de votre environnement de développement préféré. Vous pouvez intégrer directement GitHub Copilot dans votre environnement de développement (Visual Studio, Visual Studio Code, Vim, Neovim, JetBrains Suite ou Azure Data Studio) via des extensions. GitHub Copilot vous permet également d'obtenir de l'aide au niveau de l'interface de ligne de commande (CLI). Depuis 2024, GitHub Copilot est également disponible pour GitHub Mobile, ce qui permet aux développeurs de l'utiliser sur leur smartphone ou leur tablette comme ils le feraient sur leur ordinateur de bureau.

GitHub Copilot : des versions adaptées à tous les domaines d'application

Il existe une version de GitHub Copilot adaptée à chaque domaine d'application. Ainsi, chaque développeur bénéficie des possibilités offertes par GitHub Copilot, au même titre que les petites, moyennes ou grandes entreprises. Pour ces dernières, il existe les versions GitHub Copilot Business ou GitHub Copilot Enterprise. La version GitHub Copilot Individual est proposée aux développeurs individuels. Toutes les versions ont un point en commun : la possibilité de convertir le langage naturel en code directement dans l'environnement de développement intégré (IDE).

GitHub Copilot Business s'adresse avant tout aux entreprises. Pour 19 dollars par utilisateur et par mois, les développeurs de l'entreprise ont accès à GitHub Copilot. Cet abonnement comprend la complétion de code, la fonctionnalité de conversation avec GitHub Copilot via l'IDE et GitHub Mobile, la prise en charge de la CLI, des filtres pour les failles de sécurité et le code public, la sécurité et la protection des données au niveau de l'entreprise et le référencement du code. Lorsque GitHub Copilot détecte des modèles de codage vulnérables, il bloque les lignes correspondantes. Il applique la même procédure pour l'utilisation de codes publics.

GitHub Copilot Enterprise offre des fonctionnalités supplémentaires et encore plus intéressantes pour les entreprises que la version Business, notamment des fonctionnalités de conversation améliorées spécifiquement adaptées aux dépôts de l'entreprise, et de prise en charge de l'analyse différentielle des demandes d'extraction pour renforcer la précision des révisions de code. Pour la compléction de code, les utilisateurs bénéficient de modèles sur mesure qui renforcent encore l'efficacité et la précision de la création de code. GitHub Copilot Enterprise peut être utilisé non seulement dans l'IDE, la CLI et GitHub Mobile, mais aussi sur GitHub.com.

GitHub Copilot Individual s'adresse aux développeurs désireux de compléter leur code. Proposant également la fonctionnalité de conversation avec GitHub Copilot, cette version est avant tout destinée aux développeurs et aux indépendants. Elle est également disponible pour les étudiants, les enseignants et les responsables de formation.

Copilot ne copie aucun code, il le crée

En général, aucune version de GitHub Copilot ne copie le code directement ; elles le génèrent à partir de calculs de probabilités. Bien que les modèles utilisés par GitHub Copilot soient entraînés à l'aide du code issu de dépôts publics accessibles, ils ne contiennent eux-mêmes aucun code. Lors de la génération du code, GitHub Copilot utilise le code déjà disponible dans l'IDE, notamment les lignes de code situées au-dessus et en dessous du curseur. À cela s'ajoutent les informations issues des données d'entraînement et des autres fichiers que vous avez associés à chaque dépôt à l'aide de GitHub Copilot. En se basant sur ces données, GitHub Copilot génère les lignes de code suivantes ou propose des suggestions.

Perspectives d'avenir pour GitHub Copilot

Parce que les fonctionnalités de GitHub Copilot sont enrichies en permanence, vous pouvez compter sur l'ajout régulier de nouvelles possibilités innovantes. Ainsi, des [GitHub Copilot Extensions](#) ont été présentées lors de la conférence Microsoft Build 2024. Celles-ci permettent aux technologies partenaires d'être directement intégrées à GitHub Copilot Chat. Parmi les premiers partenaires concernés, on peut citer DataStax, Docker, LambdaTest, Microsoft Azure, MongoDB, Octopus Deploy, Sentry et Stripe.

Exemple d'utilisation des extensions : Vous devez corriger une erreur de base de données dans votre projet de développement. Grâce à GitHub Copilot Chat, vous pouvez utiliser divers outils et technologies tels que Sentry ou Azure pour analyser et résoudre le problème sans avoir à changer de contexte.

Ceci n'est qu'un exemple parmi tant d'autres des améliorations apportées à GitHub Copilot. Chaque mise à jour est présentée sur le [blog GitHub](#).



Martin Brandl

Martin (@martin_jib) est directeur de la technologie et directeur général de l'entreprise white duck GmbH. Il a plus de dix ans d'expérience dans l'utilisation de Microsoft Azure, sa passion étant le développement d'applications natives cloud. Grâce à ses contributions à la communauté Azure, il a déjà reçu à plusieurs reprises la distinction Microsoft MVP pour Azure

Conseil n° 1 :

Utilisation de GitHub Copilot

Commandes slash dans GitHub Copilot

par Martin Brandl

Les **commandes slash** de GitHub Copilot améliorent l'efficacité des interactions avec l'assistant. Lorsque vous saisissez un simple slash (« / ») dans la conversation, le système affiche une liste de commandes utiles vous permettant d'exécuter directement différentes actions sans avoir à effectuer trop de saisies

Ces commandes contextuelles se réfèrent toujours à un agent signalé par le symbole @. Actuellement, il en existe deux :

- **@workspace** dispose des informations de contexte sur le code de votre environnement de travail et peut vous aider au niveau de la navigation en recherchant des fichiers ou des classes pertinents.
- **@vscode** connaît les commandes et les fonctions de l'éditeur de code VS et peut vous aider à les utiliser de manière efficace.

Prenons un exemple : dans l'extrait de code suivant, nous exécutons une division par un nombre entier dans laquelle les décimales sont tronquées. Le résultat est 2,0, au lieu du résultat 2,5 attendu :

```
int x = 5;
int y = 2;
double result = x / y;
```

Toutefois, cette erreur est simple à corriger en exécutant la commande /fix dans GitHub Copilot Chat. @workspace est ici automatiquement spécifié comme agent :

mjisaak

@workspace /fix

GitHub Copilot used @workspace



Used 1 reference >

The issue with your code is that you are performing integer division, which will not yield a fractional result. To fix this, you should convert `x` or `y` to `double` before performing the division. Here's the corrected code:

```
int x = 5;
int y = 2;
double result = (double)x / y;
```

This will ensure that `result` holds the correct decimal value of the division.

Hormis la commande **/fix**, vous disposez de toute une série d'autres commandes utiles que vous pouvez appeler à l'aide de la commande **/help** :

- **@workspace /explain** : cette commande explique pas à pas le code sélectionné.
- **@workspace /fix** : cette commande suggère une solution pour les erreurs éventuelles détectées dans le code sélectionné.
- **@workspace /new** : cette commande vous permet de créer un projet à partir d'une description en langage naturel.
- **@workspace /newNotebook** : cette commande permet de créer un notebook Jupyter sur la base de votre description.
- **@workspace /terminal** : cette commande vous permet d'obtenir des explications sur l'utilisation du terminal intégré pour les différentes tâches.
- **@workspace /tests** : cette commande génère automatiquement des tests unitaires pour le code sélectionné.
- **@vscode /api** : cette commande vous permet de répondre à vos questions relatives au développement d'extensions de code VS.



David Losert

David est ingénieur principal de solutions chez GitHub. Fort de son expérience de plus de 10 ans en tant que développeur et consultant cloud, il aide les clients professionnels à mettre en place de manière optimale GitHub comme plateforme DevSecOps assistée par l'IA.

Conseil n° 2 :

Migration d'applications entre les différents langages de programmation avec GitHub Copilot Chat

par David Losert

Qu'il s'agisse de mener des projets de standardisation ou de modernisation en comblant les lacunes en compétences spécialisées des développeurs ou d'améliorer la collaboration entre les différentes équipes d'une entreprise (mot clé : « innersource »), il existe sans doute beaucoup de bonnes raisons de migrer une application existante ou une partie de cette dernière d'un langage de programmation vers un autre. Après plus d'une décennie de diversification technologique, découlant notamment du développement des microservices, de la conteneurisation et de la simplification des activités agnostiques sur le plan technologique dans le cloud public, les équipes de nombreuses entreprises aspirent à nouveau à une standardisation et une harmonisation accrues. Heureusement, la conversion ou migration entre deux langages de programmation a été considérablement simplifiée grâce aux progrès de l'intelligence artificielle dans le développement des logiciels avec, en tête de pioche, GitHub Copilot. L'exemple ci-dessous illustre le fonctionnement de ce processus.

Préparation

Avant de démarrer la conversion, vous devez définir la technologie et l'architecture cibles, car même avec GitHub Copilot, la cible du processus doit être clairement définie et la voie à suivre rigoureusement planifiée.

De même, il est recommandé d'avoir une compréhension générale de l'application source pour éviter de réaliser une migration à l'aveugle. GitHub Copilot Chat peut déjà s'avérer très utile avec l'agent @workspace à ce stade. Cet agent intègre automatiquement les fichiers non ouverts dans le contexte traité par GitHub Copilot. Vous bénéficiez ainsi d'explications et d'une meilleure compréhension d'un plus grand nombre de fichiers.

The screenshot shows a GitHub Copilot Chat window. At the top, a user named "davelosert" asks: "@workspace Kannst du mir erklären, was die Applikation im JavaScript Ordner genau tut?". Below this, GitHub Copilot responds with "Used 9 references" and lists the following files and their locations:

- 👉 .gitignore javascript
- 🌐 Dockerfile javascript
- JS index.js javascript
- npm package.json javascript
- M README.md:1-22 javascript
- JS api.js javascript/src

Below the references, there is a summary of the application's purpose and components:

Die Applikation im JavaScript-Ordner ist ein Backend-Server für eine Buchhandlung. Sie stellt eine API bereit, die mit einer Datenbank interagiert, um Informationen über Bücher abzurufen. Hier sind die Hauptkomponenten und ihre Funktionen:

- index.js : Dies ist der Einstiegspunkt der Anwendung. Es importiert die Server -Klasse aus server.js und startet den Server.

Comme le montre la capture d'écran, il est également possible de communiquer avec GitHub Copilot en allemand. Puisque le code est généralement défini en anglais et pour assurer la cohérence de l'ensemble, les captures d'écran suivantes ont été créées avec des textes en anglais (remarque : cet article a d'abord été publié en allemand, ce qui explique l'exemple en allemand). La pile cible peut également être initialisée avec GitHub Copilot Chat et la commande slash **/new**.

Migration

The screenshot shows a GitHub Copilot workspace interface. At the top, there is a message from user 'davelosert' asking for a Java SpringBoot Application for Web using a RESTAPI Controller with a Gradle Build File. Below this, GitHub Copilot has proposed a directory structure:

```
src
└── main
    ├── java/com/example/myapp
    │   ├── controller
    │   │   └── MyRestController.java
    │   └── MySpringBootApplication.java
    └── resources
        └── application.properties
└── test/java/com/example/myapp
    └── MySpringBootApplicationTests.java
build.gradle
settings.gradle
README.md
```

At the bottom left of the workspace, there is a 'Create Workspace' button.

Les développeurs réalisent ensuite la migration directement dans l'IDE. Ce processus est en principe très simple :

1. Vous ouvrez un des fichiers à convertir.
2. Dans la fenêtre de conversation de GitHub Copilot, vous demandez la conversion : « Translate this file into <Target-Programming-Language>, please! » (même avec l'IA, un peu de politesse ne fait pas de mal 😊).

The screenshot shows a GitHub Copilot interface. On the left, a code editor displays a file named 'api.js' with the following content:

```
src > JS api.js > ...
22 module.exports = class API {
21   .database;
20
19   .constructor(database) {
18     .this.database = database;
17   }
16
15   .async.register(app) {
14     .app.get('/books', .async.(req, .res) => {
13       .const.books = await .this.database.getAllBooks();
12       .res.json(books);
11     });
10
9     .app.get('/books/author/:author', .async.(req, .res) => {
8       .const.author = req.params.author;
7
6       .const.books = await .this.database.getBooksByAuthor(author);
5
4       .res.json(books);
3     );
2   }
1 }
```

On the right, a message from user 'davelosert' asks for a translation into Java. GitHub Copilot responds with the equivalent Java code using Spring Boot annotations:

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;
@RestController
public class API {
    private final Database database;
    public API(Database database) {
        this.database = database;
    }
    @GetMapping("/books")
    public List<Book> getAllBooks() {
        return database.getAllBooks();
    }
    @GetMapping("/books/author/{author}")
    public List<Book> getBooksByAuthor(@PathVariable String author) {
        return database.getBooksByAuthor(author);
    }
}
```

3. Copiez le code généré à l'aide de la fonctionnalité Copy & Paste ou insérez-le avec l'option « Insert into new File » dans la cible.

The screenshot shows the GitHub Copilot interface again, this time generating Java code for a Spring Boot application. The code is identical to the one shown in the previous screenshot:

```
import org.springframework.web.bind.annotation.*;
import java.util.List;
```

A context menu is open at the bottom right, showing options: 'Insert Into New File' (which is highlighted in blue), 'Run in Terminal', and a separator line.

Répétez cette opération pour tous les fichiers du système source pour finaliser la migration. Souvent, tout n'est pas si simple et vous devrez surveiller certains facteurs. Nous y reviendrons en détail par la suite.

Revue et ajustements

Après la conversion des différents fichiers individuels, il est souvent nécessaire d'apporter des améliorations plus ou moins importantes au fichier cible. En général, GitHub Copilot vous aide à créer 80 à 90 % de la structure de base, mais au début le code cible est souvent imparfait.

Dans l'exemple précédent, il existe dans Spring Boot un meilleur chemin pour injecter les dépendances dans les classes avec l'annotation `@Autowired`. Dans ce cas, vous pouvez également utiliser la fonctionnalité de conversation pour améliorer le code cible :

davelosert Could you use `@Autowired` rather than a constructor for the Dependency Injection?

GitHub Copilot Used 1 reference >

Yes, you can use `@Autowired` to inject the `Database` dependency. Here is the equivalent Java code:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
public class API {

    @Autowired
    private Database database;
```

Pour ne pas avoir à effectuer à nouveau cet ajustement à chaque nouvelle conversion, vous pouvez également affiner l'invite directement pour le fichier suivant à convertir :

davelosert Please Translate this file into a Java SpringBoot RESTAPI Controller. Take the following requirements into consideration:

- Use `@Autowired` for the Dependency Injection directly with field and not constructors.

Ainsi, vous améliorerez continuellement au cours de la migration les conversions réalisées par GitHub Copilot sur la base de la première invite. Un conseil : avec la touche « flèche vers le haut », vous pouvez afficher les invites précédentes à améliorer dans le champ de saisie, comme sur un terminal.

En raison des différences de conception des langages de programmation, d'autres modifications sont souvent nécessaires. Par exemple, lorsque vous réalisez une conversion comme ici de JavaScript sans déclaration de type vers Java, vous devez créer une classe de données en Java pour les structures des données définies en JS en ligne. Pour ce faire, vous pouvez également utiliser GitHub Copilot Chat pour vous éviter un gros travail de saisie, notamment pour les classes de données les plus importantes :

The screenshot shows a dark-themed chat window. At the top, a user named "davelosert" asks, "Can you generate the Book class?". Below, GitHub Copilot responds, "Used 1 reference > Sure, here is a simple Book class in Java:" followed by the generated code:

```
public class Book {  
    private String title;  
    private String author;  
  
    // Constructor, getters and setters  
    public Book(String title, String author) {  
        this.title = title;  
        this.author = author;  
    }  
}
```

Stratégie de migration : diviser pour mieux régner

Lors d'une migration, il est recommandé de traiter les fichiers en passant d'un état compilé et opérationnel à un autre et d'effectuer entre-temps des archivages réguliers. Cela évite les changements de contexte et une sollicitation excessive due à de trop nombreuses modifications et erreurs simultanées. Vous perdrez ainsi moins facilement le fil de vos idées.

Comme un programme est souvent un réseau de fichiers interdépendants, nous vous recommandons de traiter cette structure de manière ascendante en commençant par les fichiers ne présentant aucune ou très peu de dépendances, puis de continuer à travailler « en remontant » la structure. GitHub Copilot peut également convertir plusieurs fichiers simultanément. Toutefois, dans ce cas, l'ajustement est plus difficile et vous courez le risque de vous déconcentrer.

Tests

Existe-t-il des tests unitaires dans la source ? Super ! Ces tests peuvent bien entendu être également convertis avec GitHub Copilot.

Il n'y en a pas ? Aucun souci. Grâce à la commande slash /test de GitHub Copilot Chat, le problème est vite résolu :

The screenshot shows a GitHub Copilot interface. At the top, there are tabs for 'JS api.js' and 'API.java'. The main area shows a Java file named 'APITest.java' with code generated by Copilot. The code imports various JUnit and Mockito annotations and classes, and defines a test class 'APITest' with a setup method and a test method. The Copilot interface includes a search bar ('/tests'), an 'Accept' button, and a 'Discard' dropdown.

```
1 import org.springframework.web.bind.annotation.*;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.mockito.Mockito.mock;
5 import static org.mockito.Mockito.when;
6
7 import java.util.Arrays;
8 import java.util.List;
9
10 public class APITest {
11     private Database database;
12     private API api;
13
14     @BeforeEach
15     public void setUp() {
16         database = mock(Database.class);
17         api = new API(database);
18     }
19
20     @Test
21     public void testGetAllBooks() {
22         List<Book> expectedBooks = Arrays.asList(new Book("Book 1", "Author 1"), new Book("Book 2", "Author 2"));
23     }
24 }
```

Un simple clic sur « Accept » génère le fichier test correspondant avec les premiers scénarios de test vous servant de base. Les tests E2E agnostiques sur le plan technologique ou en « boîte noire » peuvent en outre représenter un complément important afin de vérifier le comportement général de l’application tant dans l’ancien que dans le nouveau code et de garantir uniformément la fonctionnalité de base des deux côtés.

Le tour est joué

Cette approche vous permet de migrer des applications plus rapidement que jamais entre différentes technologies en utilisant GitHub Copilot Chat. Le code suggéré par GitHub Copilot n'est peut-être pas toujours parfait du premier coup, mais avec un peu de travail sur les fichiers cibles et les invités, vous pourrez réaliser rapidement d'importants progrès. GitHub Copilot Chat vous aide non seulement à mieux comprendre le code source, mais vous évite aussi un gros travail de saisie dont vous n'avez pas à vous préoccuper.

Votre charge cognitive étant réduite lors de la migration, vous pouvez ainsi vous concentrer sur les parties importantes du projet, comme par exemple la vérification des cas d'utilisation concrets et de la fonctionnalité de l’application, ainsi que de la qualité du nouveau code.



Daniel Meixner

Spécialiste de l'innovation applicative chez Microsoft, Daniel étudie les architectures cloud, les outils de développement, l'IA et les applications intelligentes. Dans ses différentes fonctions, de développeur à évangéliste, il a expérimenté sous différents angles toutes les facettes du développement de logiciels.

Conseil n° 3 :

Plus de faux texte

par Daniel Meixner

GitHub Copilot étant un outil de développement, il est logique que l'assistant IA ne se concentre, de préférence, que sur les réponses relatives au développement. Par exemple, si vous interrogez GitHub Copilot pour obtenir une recommandation d'achat concernant un produit spécifique, l'assistant vous répondra qu'il ne peut pas vous donner de réponse.

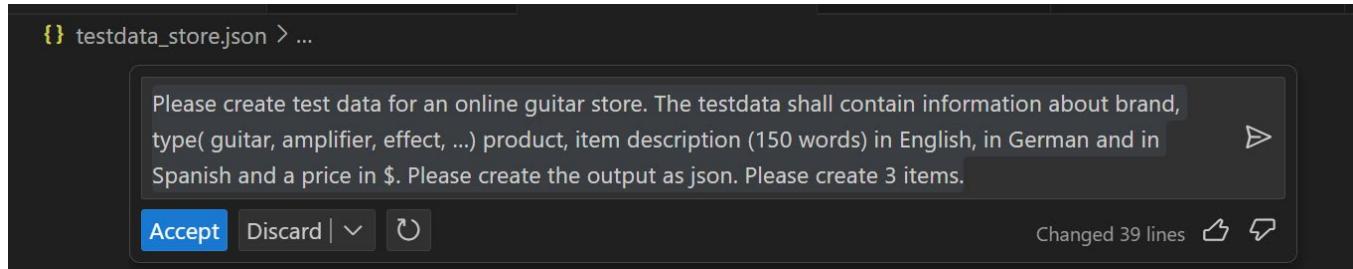
DanielMeixner
Which guitar should I buy?

GitHub Copilot
Used 1 reference >
Sorry, but I can only assist with programming related questions.

Mais cela ne signifie pas que GitHub Copilot ne peut pas utiliser de connaissances générales de différents domaines en dehors du développement de logiciels. Vous pouvez par exemple utiliser ces connaissances pour générer des données de test pertinentes et adaptées à une application au niveau technique qui nécessitent des connaissances d'un domaine spécifique.

Par exemple, si vous programmez une boutique en ligne, vous avez besoin de données de test afin de décrire les produits. Vous pouvez bien évidemment compiler les données de test à partir de différentes sources. Cette méthode permet souvent d'obtenir des données très utiles. Toutefois, cette approche

chronophage exige un assez gros travail. GitHub Copilot vous permet d'utiliser l'IA pour générer très facilement ces données, éventuellement au format JSON, afin de les traiter de façon pertinente. Dans cet exemple, vous créez un fichier nommé *testdata.json* et demandez en ligne les données de test pour votre application dans ce fichier à l'aide du raccourci Ctrl + I. Prenons à présent l'exemple d'une boutique en ligne qui vend des guitares et des accessoires :



Comme vous pouvez le constater, le résultat est déjà très réaliste. GitHub Copilot semble avoir de très bonnes connaissances en la matière. Les descriptions sont justes et également adaptées au test. C'est bien mieux que d'utiliser un faux texte vide de sens comme le Lorem Ipsum. Si vous utilisez plusieurs variantes de langue, vous pouvez également développer l'application en version multilingue. Vous pouvez donc vous lancer très rapidement dans la programmation sans dépendre de tiers, ni perdre de temps.

```
{
  "items": [
    {
      "brand": "Fender",
      "type": "guitar",
      "product": "Stratocaster",
      "description": {
        "english": "The Fender Stratocaster is a classic electric guitar known for its versatile sound and iconic design.",
        "german": "Die Fender Stratocaster ist eine klassische E-Gitarre, die für ihren vielseitigen Klang und ihr ikonisches Design bekannt ist.",
        "spanish": "La Fender Stratocaster es una guitarra eléctrica clásica conocida por su sonido versátil y su diseño icónico."
      },
      "price": "$999"
    },
    {
      "brand": "Gibson",
      "type": "guitar",
      "product": "Les Paul",
      "description": {
        "english": "The Gibson Les Paul is a legendary electric guitar with a rich, warm tone and a solid mahogany body.",
        "german": "Die Gibson Les Paul ist eine legendäre E-Gitarre mit einem warmen, vollen Klang und einem massiven Mahagonikorpus.",
        "spanish": "La Gibson Les Paul es una guitarra eléctrica legendaria con un tono rico y cálido y un cuerpo de caoba maciza."
      },
      "price": "$1499"
    },
    {
      "brand": "Marshall",
      "type": "amplifier",
      "product": "JCM800",
      "description": {
        "english": "The Marshall JCM800 is a classic guitar amplifier known for its powerful, high-gain sound and iconic British tone.",
        "german": "Der Marshall JCM800 ist ein klassischer Gitarrenverstärker, der für seinen kraftvollen, hochverzerrten Sound und seinen ikonischen britischen Klang bekannt ist.",
        "spanish": "El Marshall JCM800 es un amplificador de guitarra clásico conocido por su potente sonido de alta ganancia y su icónico tono británico."
      },
      "price": "$799"
    }
  ]
}
```

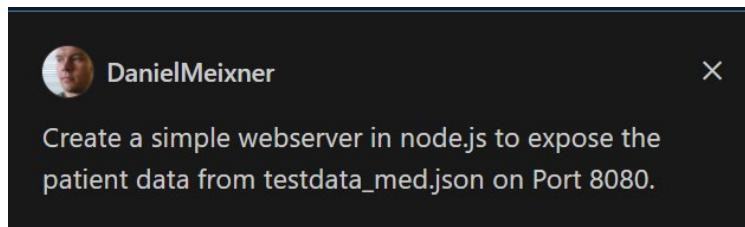
Loin de se limiter à un seul domaine, les connaissances de GitHub Copilot peuvent également être utilisées pour d'autres cas d'utilisation. Supposons que vous travailliez à une application médicale et que vous ayez besoin de données de test de patients. GitHub Copilot peut vous aider à créer des données de patients fictives utilisables dans votre application :

```
Please create testdata for my medical application. I need information on patients. Please provide json containing patientname, patientid (following the scheme of XX-1111XX11), dob, date of last surgery, modality (like CT or MR etc), sex, hospital and physician and patient address (an address in Germany). Please create 5 records.
```

Voici un extrait du résultat :

```
{
  "patients": [
    {
      "patientname": "John Doe",
      "patientid": "AB-1234CD56",
      "dob": "1980-05-15",
      "lastsurgerydate": "2021-10-10",
      "modality": "CT",
      "sex": "Male",
      "hospital": "ABC Hospital",
      "physician": "Dr. Smith",
      "address": "123 Main Street, Berlin, Germany"
    },
    {
      "patientname": "Jane Smith",
      "patientid": "EF-5678GH90",
      "dob": "1992-09-20",
      "lastsurgerydate": "2021-09-05",
      "modality": "MR",
      "sex": "Female",
      "hospital": "XYZ Hospital",
      "physician": "Dr. Johnson",
      "address": "456 Elm Street, Munich, Germany"
    }
  ]
}
```

Ici, la possibilité d'indiquer des formats dans les chaînes de caractères (patientid) sans expressions régulières peut également s'avérer utile. Par ailleurs, vous noterez que GitHub Copilot a automatiquement interprété « dob » correctement comme la date de naissance. Lorsque vous souhaitez transmettre des données, vous pouvez demander à GitHub Copilot de créer un serveur Web. Vous pourrez ainsi interroger les données via une requête HTTP.





George Kosmidis

Spécialiste des technologies Microsoft, George est architecte principal chez Slalom Deutschland et a été récompensé par le programme Microsoft MVP. En tant que fondateur et organisateur du site Munich .NET Meetup, il a réussi à développer une communauté dynamique de près de 1 700 professionnels .NET et Azure. Il s'est vu décerner la distinction Microsoft MVP pour ses contributions à Azure, .NET et Azure DevOps. George est un orateur hors pair, qui partage souvent ses connaissances et son expérience avec d'autres et exerce une forte influence sur la communauté technologique.

Conseil n° 4 :

Codage intelligent avec GitHub Copilot

par George Kosmidis

Le développement de logiciels a considérablement évolué en raison des avancées technologiques et des demandes variables du marché, depuis la programmation manuelle traditionnelle jusqu'à l'approche actuelle plus agile et innovante. L'intégration de l'intelligence artificielle (IA) et du Machine Learning (ML) accélère et révolutionne cette évolution, tant au niveau des processus de développement que des résultats. GitHub Copilot illustre ce changement et exploite le potentiel de l'IA pour aider les développeurs à écrire, tester et optimiser le code afin de privilégier la créativité et la réflexion stratégique au codage habituel.

GitHub Copilot a été développé par GitHub, Microsoft et OpenAI. Il s'agit d'un outil de complétion de code assisté par l'IA capable d'améliorer considérablement l'expérience de programmation. GitHub Copilot exploite une énorme base de données afin de proposer des suggestions intelligentes et contextuelles, accélère le processus de codage et améliore simultanément la précision et la réduction des erreurs, un atout inestimable dans un contexte commercial soumis aux contraintes de temps. En plus de ces fonctions, GitHub Copilot offre une fonction interactive appelée GitHub Copilot Chat. Cette dernière permet aux développeurs de poser des questions liées au code et d'obtenir des réponses directement dans les environnements de développement intégrés (IDE) pris en charge comme Visual Studio, par exemple. En offrant un dialogue direct, GitHub Copilot Chat enrichit davantage l'environnement de développement, tout en accélérant la résolution des problèmes et l'acquisition de connaissances. Cet outil favorise également la recherche de nouveaux modèles de programmation, ainsi que l'élaboration de solutions innovantes et privilégie la réflexion créative et stratégique. À cet égard, GitHub Copilot est bien plus qu'un outil de développement. C'est une ressource essentielle pour les développeurs, capable d'améliorer leur productivité et la qualité globale de leur travail.

Dans cet environnement en évolution, Slalom a été l'une des premières entreprises à utiliser GitHub Copilot et a pu en tirer une expérience inestimable. La section suivante vous présente les connaissances glanées par Slalom et l'impact de GitHub Copilot sur les méthodes de développement. L'expérience de Slalom souligne le rôle de GitHub Copilot, qui s'avère être bien plus qu'un « simple » outil de code : une ressource incontournable pour l'optimisation de la productivité et de la qualité des travaux de développement.

Étude de cas de Slalom illustrant l'avantage offert par GitHub Copilot

Réputé pour ses solutions innovantes, Slalom a collaboré avec un des leaders des fournisseurs de soins de santé pour développer une plateforme moderne d'évaluations de santé basée sur Azure à l'aide des services Azure AI. L'objectif était d'intégrer différentes fonctionnalités Azure AI, notamment Azure Machine Learning, Azure Cognitive Services et Azure Bot Service, afin d'améliorer la prise en charge des patients et l'efficacité opérationnelle.

Le défi

Le projet avait pour objectif d'intégrer parfaitement les fonctions natives des services Azure AI aux modèles Azure ML, déployés sous forme de conteneurs ou utilisés directement par les services IA. Cette intégration représentait un véritable défi à plusieurs niveaux : d'une part, elle nécessitait une expérience approfondie afin d'utiliser efficacement toute l'étendue des fonctionnalités d'Azure et d'autre part, il fallait tenir compte de la nécessité de gérer efficacement les coûts, tout en optimisant la prestation. En raison des délais restreints, la recherche d'un tel équilibre représentait une énorme épreuve au niveau des capacités techniques et de la gestion des ressources de Slalom.

La solution : GitHub Copilot

Nous avons intégré GitHub Copilot au workflow de développement afin de soutenir les efforts de l'équipe. Cela a changé la donne :

1. Accélération du développement

Lors du développement du moteur d'analyse principal avec Azure Machine Learning, l'équipe a exploité GitHub Copilot pour suggérer des modèles de code et des algorithmes optimisés. Le développement a été non seulement accéléré, mais l'équipe a pu également éprouver la résilience et l'efficacité des solutions.

2. Optimisation de l'intégration

Pour l'intégration d'Azure Cognitive Services, notamment au niveau de l'adaptation de modèles IA pour le traitement automatique des langues (NLP), GitHub Copilot a assuré la prise en charge du code en temps réel. Il a proposé des fragments de code et des exemples d'intégration simplifiant considérablement le processus d'implémentation.

3. Amélioration du développement du bot

Lors du développement d'un bot pour les interactions avec les patients via Azure Bot Service, les suggestions de Copilot ont été très utiles à l'équipe afin de parcourir et d'intégrer facilement des processus de conversation complexes dans le backend Azure.

Une analyse plus approfondie

Prenons un exemple concret pour mieux illustrer l'impact de GitHub Copilot sur ce projet. Une des tâches principales consistait à intégrer les Azure Cognitive Services pour le traitement des langues naturelles dans une fonction Azure. GitHub Copilot a joué un rôle décisif. Il s'agissait d'accélérer le développement tout en s'assurant que le code correspondait aux meilleures pratiques. Le fragment de code Python ci-dessous montre comment Copilot a aidé l'équipe de développement à implémenter l'analyse de sentiment avec le client Analyse textuelle d'Azure.

```
# Importing necessary Azure libraries
from azure.ai.textanalytics import TextAnalyticsClient
from azure.core.credentials import AzureKeyCredential

# Initializing the Text Analytics Client
# Github Copilot suggested the use of AzureKeyCredential
client = TextAnalyticsClient(endpoint="<endpoint>", credential=
AzureKeyCredential(<key>))

# Analyze patient feedback using Azure Cognitive Services
# GitHub Copilot helped in crafting this function structure
def process_patient_feedback(feedback):

    # Using the sentiment analysis feature of Text Analytics
    # Copilot suggested the use of 'client.analyze_sentiment' for
    # processing natural language
    response = client.analyze_sentiment(documents=[feedback])
    if not response:
        return { "error": "No response from sentiment analysis!" }
    if response[0].is_error:
        return { "error": "Error in sentiment analysis!" }
    sentiments = response[0]
    # Additional insights such as key phrases can also be extracted
```

```

# This suggestion by Copilot enhances the depth of analysis
response = client.extract_key_phrases(documents=[feedback])
if not response:
    return { „error“: „No response from key phrases extraction!“ }
if response[0].is_error:
    return { „error“: „Error in key phrases extraction!“ }

return {„sentiment“: sentiments, „key_phrases“: response[0].key_phrases}

```

Dans le fragment de code ci-dessus, la contribution de GitHub Copilot est évidente à plusieurs égards. GitHub Copilot a suggéré d'utiliser AzureKeyCredential pour une gestion sécurisée des clés API, une méthode éprouvée lors du développement d'applications Azure. Par ailleurs, les suggestions de Copilot ont permis d'optimiser la structure et l'implémentation de la fonction d'analyse de sentiment, de la sélection des méthodes adaptées au traitement automatique du langage naturel à l'amélioration de l'analyse par l'extraction de phrases clés. Cet exemple montre comment Copilot accélère le codage, tout en l'enrichissant avec des fonctions étendues et de bonnes pratiques. Il est donc un atout inestimable pour le développement complexe de solutions natives Azure.

L'avantage de GitHub Copilot

L'évaluation interne récemment effectuée chez Slalom apporte une preuve convaincante de l'impact transformationnel de GitHub Copilot dans le domaine du développement de logiciels. L'étude a été réalisée sur deux sites différents (Londres et Denver) avec au total quatre unités de développement et a offert un aperçu inédit de l'efficacité de GitHub Copilot dans des conditions contrôlées.

Chaque unité devait terminer le plus rapidement possible le test en démarrant avec des bases de code et des backlogs de projet identiques, mais avec une variable cruciale : la moitié de l'équipe utilisait GitHub Copilot, tandis que l'autre moitié devait travailler sans outil piloté par l'IA. Cette approche a permis de donner un aperçu authentique de l'impact de GitHub Copilot sur le développement, car tous les développeurs des deux équipes étaient relativement inexpérimentés, que ce soit avec la base de code ou avec GitHub Copilot.

Les résultats ont été remarquables. Les développeurs peu expérimentés qui ont utilisé GitHub Copilot ont vu leur productivité augmenter de 88 %. Cette statistique à elle seule en dit long sur la capacité de Copilot à créer des conditions de développement identiques pour tous les développeurs et à améliorer nettement l'efficacité et la confiance des programmeurs moins expérimentés.

De plus, GitHub Copilot a joué un rôle évident dans la simplification de la documentation. Ses suggestions adaptées au contexte et associées aux exemples de code précis et aux corrections grammaticales ont permis d'accélérer le processus de documentation, tout en améliorant son intuitivité. Les développeurs ont pu se concentrer davantage sur les aspects créatifs de la programmation, au lieu de perdre du temps avec les subtilités de la documentation.

Parmi les résultats les plus frappants, l'étude a montré que les équipes utilisant Copilot ont accéléré de 50 % la production de code. Cette augmentation de leur rendement est d'une grande importance dans un domaine où le temps est souvent la ressource la plus importante. Concernant les tâches répétitives, l'impact de Copilot a été encore plus marqué, avec une amélioration spectaculaire de la vitesse d'exécution de 96 %, ce qui vient souligner sa capacité d'automatisation et d'optimisation des aspects plutôt ennuyeux de la programmation.

Enfin, la réduction de 77 % du temps consacré à la recherche d'informations ou de solutions met également en lumière un autre avantage important de GitHub Copilot : son utilité en tant que base de connaissances accessible à la demande. Grâce à l'accès immédiat aux informations et à l'aperçu du code, Copilot réduit significativement le temps consacré par les développeurs à la recherche de solutions aux différents problèmes.

L'évaluation interne de Slalom montre clairement que GitHub Copilot est bien plus qu'un simple outil et constitue une évolution radicale du développement de logiciels. Son impact sur la productivité, la courbe d'apprentissage, la qualité de la documentation et l'efficacité globale du codage représente un progrès considérable, notamment pour les équipes souhaitant préserver leur agilité et leur capacité d'innovation dans un monde technologique en évolution rapide. À mesure que nous découvrons et adoptons les nouvelles fonctionnalités offertes par les outils assistés par l'IA tels que GitHub Copilot, l'avenir du développement logiciel semble plus prometteur et passionnant que jamais.



Julia Kordick

Postes déjà occupés par Julia : ingénieur logiciel en chef, chef d'équipe informatique, architecte de solutions cloud, spécialiste technique. Hobbies de Julia : nettoyage du code, architectures logicielles distribuées, résolution de problèmes complexes, JavaScript, « hackathons », féminisme et football américain.

Conseil n° 5 :

D'une idée à son déploiement en moins de 30 minutes

par Julia Kordick

« Hé, Julia, peux-tu montrer aux techniciens du client la mise en œuvre de ce point sur Azure ? »

« Salut Julia, sais-tu si le processus conçu par le client peut fonctionner ? »

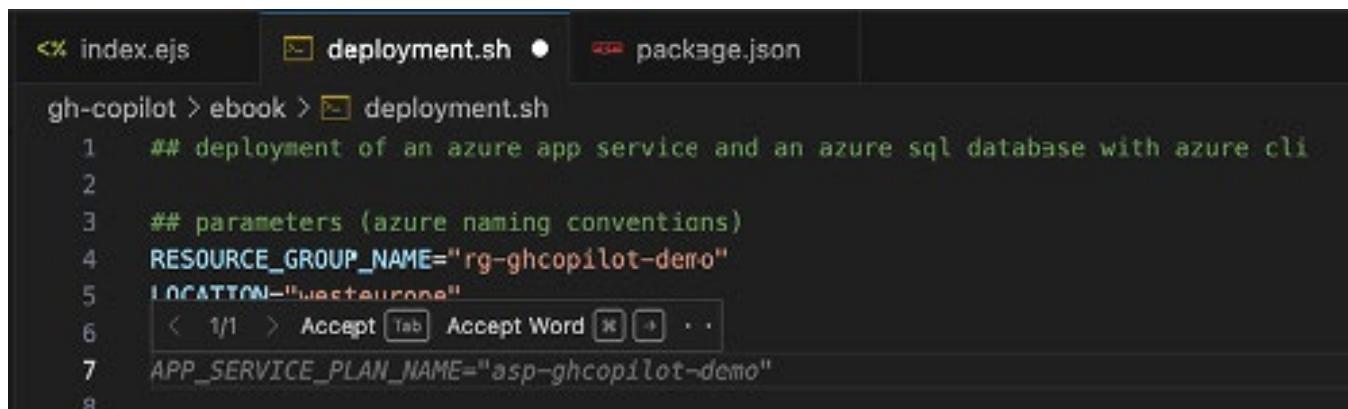
« Bonjour Julia, peux-tu réaliser une démonstration sur un sujet technique pour le client ? »

Je reçois ce genre de demandes presque tous les jours. Preuves de concept, explications et autres démonstrations de concepts techniques rythment mon quotidien. Je me suis créé, au fil du temps, une compilation de divers dépôts que j'utilise désormais comme base pour les différents scénarios. Mais, dans la pratique, les différentes questions abordées pour chaque client sont uniques, tant au niveau technique, qu'au niveau du cas d'utilisation concret. GitHub Copilot m'aide non seulement à ajuster le code et les déploiements existants, mais aussi à créer rapidement un code inédit et adapté au groupe cible.

Cet outil m'offre une très grande liberté de travail en termes d'infrastructure en tant que code et de langages de programmation. En effet, bien que je n'aie jamais utilisé .NET dans un cadre professionnel, mes connaissances pro-code issues d'autres langages associées à la puissance inégalée de GitHub Copilot me suffisent amplement pour créer un code opérationnel dans presque tous les langages. Je peux ensuite le déployer rapidement et facilement dans la CLI Azure sur laquelle je me concentrerai dans cet exemple.

Prenons le scénario suivant : le client a une application monolithique Node.js non conteneurisée, qui nécessite une base de données (relationnelle) en arrière-plan. Nous souhaitons donc provisionner un service Azure App Service et une base de données Azure SQL, la méthode la plus simple consistant à utiliser la CLI Azure.

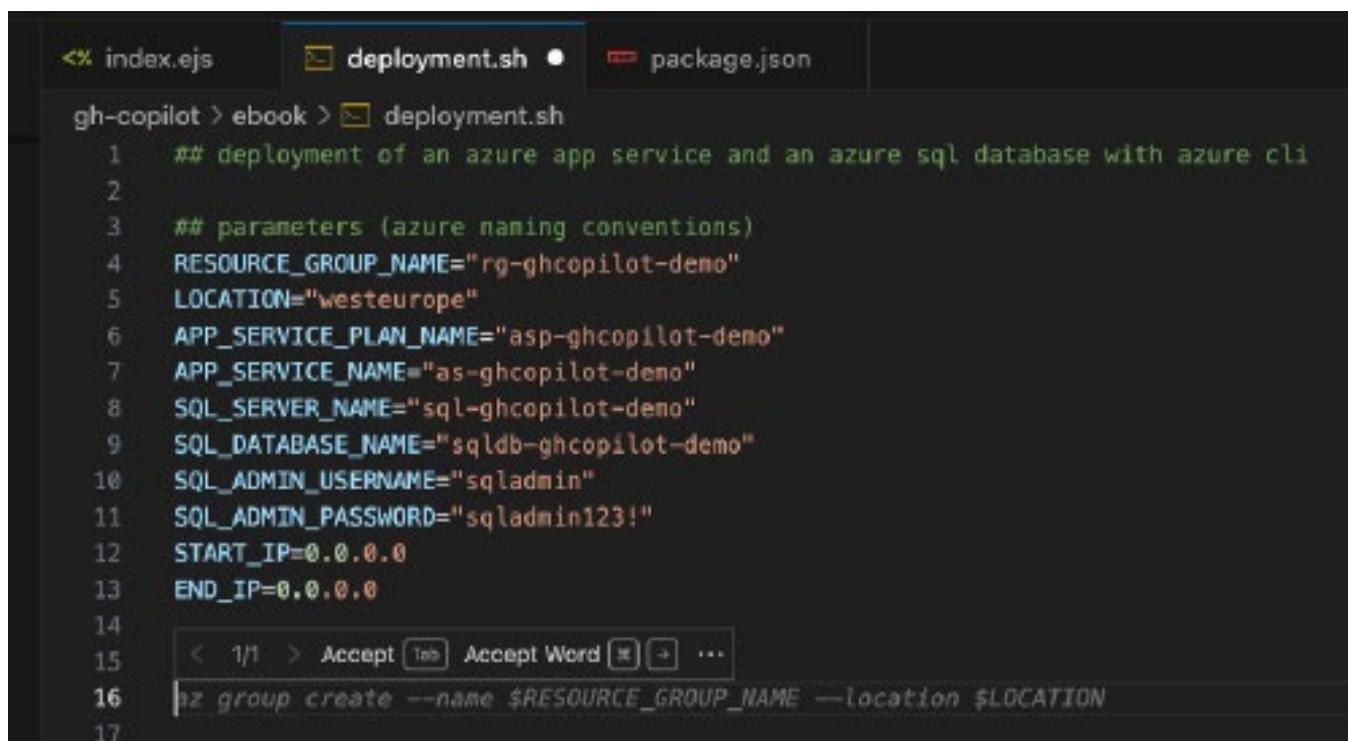
Tout d'abord, je crée un script deployment.sh, puis je décris rapidement le résultat attendu dans un premier commentaire, avant de commencer à définir des paramètres pour mon déploiement. Après la définition des premiers paramètres, GitHub Copilot reconnaît mon format de nom et me propose d'autres suggestions adéquates que je peux accepter très simplement en quasi-totalité.



A screenshot of a terminal window showing a file named deployment.sh. The terminal title is "gh-copilot > ebook > deployment.sh". The code in the terminal is:

```
1 ## deployment of an azure app service and an azure sql database with azure cli
2
3 ## parameters (azure naming conventions)
4 RESOURCE_GROUP_NAME="rg-ghcopilot-demo"
5 LOCATION="westeurope"
6 APP_SERVICE_PLAN_NAME="asp-ghcopilot-demo"
7
```

J'utilise ensuite la CLI Azure pour commencer à provisionner les ressources et GitHub Copilot me fait des propositions qui correspondent à mes paramètres.



A screenshot of a terminal window showing the same deployment.sh script. The terminal title is "gh-copilot > ebook > deployment.sh". The code is identical to the previous screenshot, but the last line has a suggestion at the bottom:

```
16 az group create --name $RESOURCE_GROUP_NAME --location $LOCATION
```

GitHub Copilot me suggère pas à pas les étapes descriptives, y compris les commentaires, nécessaires à la création d'un Azure App Service et d'une base de données Azure SQL. Il me suffit d'appuyer sur Entrée pour obtenir une nouvelle suggestion de GitHub Copilot et sur la touche Tab pour accepter intégralement chaque suggestion. Je décide de ne pas valider intégralement les éléments particulièrement importants ou pertinents pour la sécurité (comme la configuration des règles du pare-feu, par exemple) mais de les accepter mot après mot à l'aide des touches Command et Tab afin de vérifier attentivement les suggestions pour chaque élément.

```
## configure sql server firewall
< 1/1 > Accept [x] Accept Word [x] + ... /source-group $RESOURCE_GROUP_NAME --server $SQL_SERVER_NAME \
--name AllowYourIp --start-ip-address $START_IP --end-ip-address $END_IP
```

Comme je connais bien Azure, la description des ressources nécessaires ne me prend que quelques minutes. Je lance ensuite le script shell que je viens de créer et, en l'espace de quelques minutes seulement, je peux voir le résultat sur ma console ou sur le portail Azure.

The screenshot shows the Azure portal interface for the resource group "rg-ghcopilot-demo". The top navigation bar includes "Home", "rg-ghcopilot-demo", "Create", "Manage view", "Delete resource group", "Refresh", "Export to CSV", "Open query", "Assignments", "Move", "Delete", and "Export". Below the navigation is the "Essentials" section with "Subscription (move) : komidick MCAP! subscription", "Deployment : No deployments", "Location : West Europe", and a "Tags (edit) : Add tags" button. The "Resources" tab is selected, showing a table of resources:

Name	Type	Location
as-ghcopilot-demo	App Service	West Europe
asp-ghcopilot-demo	App Service plan	West Europe
sql-ghcopilot-demo	SQL server	West Europe
sql-ghcopilot-demo (sql-ghcopilot-demo-1)	SQL database	West Europe

Malgré mes connaissances approfondies d'Azure, je ne connais pas par cœur toutes les commandes de la CLI Azure. Par conséquent, lorsque GitHub Copilot me propose une suggestion, j'ai souvent une idée approximative du résultat, mais cela ne me suffit pas.

C'est là que GitHub Copilot Chat entre en jeu. Dans le fichier, il me suffit de marquer à droite de l'écran l'emplacement exact pour lequel j'ai besoin d'une explication avant d'obtenir des précisions dans la fenêtre de discussion à gauche de l'écran.

The screenshot shows a GitHub Copilot interface with a code editor and a chat window.

Chat:

- Hi @jkondick, how can I help you?
- Can you explain me, what this Azure CLI command?

GitHub Copilot: Used Inference

Code Editor: deployment.sh

```
deployment.sh
1  # deployment of an azure app service and an azure sql database with azure cli
2
3  # parameters (azure naming conventions)
4  RESOURCE_GROUP_NAME="rg-github-lab-demos"
5  LOCATION="westeurope"
6  APP_SERVICE_PLAN_NAME="app-github-lab-demos"
7  APP_SERVICE_NAME="ac-github-lab-demos"
8  SQL_SERVER_NAME="sql-github-lab-demos"
9  SQL_DATABASE_NAME="sql01sql01lab-demos"
10  SQL_ADMIN_USERNAME="sqladmin1m"
11  SQL_ADMIN_PASSWORD="PwS$M9D-SaND0H-SaND0H"
12  START_IP="8.8.8.8"
13  END_IP="8.8.8.8"
14
15  # create resource group
16  az group create --name $RESOURCE_GROUP_NAME --location $LOCATION
17
18  # create app service plan
19  az appservice plan create --name $APP_SERVICE_PLAN_NAME --resource-group $RESOURCE_GROUP_NAME --location $LOCATION
20
21  # create app service
22  az webapp create --name $APP_SERVICE_NAME --plan $APP_SERVICE_PLAN_NAME --resource-group $RESOURCE_GROUP_NAME
23
24  # deploy app
25  az webapp up --name $APP_SERVICE_NAME --resource-group $RESOURCE_GROUP_NAME
26
27  # create sql server
28  az sql server create --name $SQL_SERVER_NAME --resource-group $RESOURCE_GROUP_NAME --location $LOCATION
29
30  # configure sql server firewall
31  az sql server FirewallRule create --resource-group $RESOURCE_GROUP_NAME \
32      --server $SQL_SERVER_NAME --name AllowAllIP --start-ip-address $START_IP --end-ip-address $END_IP
33
34  # create sql database
35  az sql db create --resource-group $RESOURCE_GROUP_NAME --server $SQL_SERVER_NAME --name $SQL_DATABASE_NAME
36
37
38
39
40
41
42
43
44
45
```

Bottom Bar: Ask Copilot or type '?' for commands



Thomas Pentenrieder

Thomas (@th_p) est ingénieur logiciel en chef chez Medialesson, spécialisé dans le domaine du développement cloud et Web. Il est également co-organisateur de l'événement « Azure Dev Meetup » à Munich. Microsoft lui a d'ailleurs décerné en 2023 la distinction de MVP pour Azure.

Conseil n° 6 :

Écriture de tests avec GitHub Copilot Chat

par Thomas Pentenrieder

GitHub Copilot peut générer, le cas échéant, des pans entiers de la logique de programmation à partir de commentaires ou d'invites. Cependant, de mon point de vue, le meilleur atout de cet outil pour les développeurs est sa capacité à faciliter les tâches les plus fastidieuses (mais néanmoins importantes) telles que l'écriture de tests.

Lorsque les méthodes ont déjà été entièrement mises en œuvre, en particulier, GitHub Copilot peut générer avec une très grande précision des tests unitaires adaptés et leurs données de test correspondantes pour les scénarios les plus variés en se basant sur le contexte existant. Ainsi, vous pouvez non seulement vérifier intégralement et rapidement des pans entiers de la logique de programmation, mais aussi examiner les cas limites auxquels vous n'auriez peut-être pas pensé.

Dans l'exemple suivant, nous utilisons GitHub Copilot pour tester de manière approfondie des méthodes d'assistant spécifiques à la date. Ces méthodes sont particulièrement cruciales au niveau du navigateur, car la région et le fuseau horaire dépendent des utilisateurs et les développeurs n'ont aucun contrôle sur ces paramètres.

La méthode à tester, `daysAsArray`, renvoie tous les jours visibles d'un calendrier, par exemple uniquement les jours ouvrables entre une date de début et une date de fin au format ISO standard. En sélectionnant la méthode souhaitée et l'appel de la commande /tests dans GitHub Copilot Chat, vous obtenez en quelques secondes sept scénarios de test différents. Ces derniers sont automatiquement ajoutés au fichier de test existant sans la moindre intervention de votre part.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top bar includes the 'File', 'Edit', 'Selection', and '...>' menus, a back/forward navigation bar, a search bar containing 'TeamPlanner', and window control buttons. The left sidebar features icons for file operations like Open, Save, Find, and Settings.

The main editor area displays a TypeScript file named 'datehelpers.ts'. The code defines two functions: 'daysAsArray' and 'daysOfYearAsArray'. The 'daysAsArray' function takes a start date and an end date as ISO strings, filters them by visible week days (Monday through Friday), and returns an array of Date objects. The 'daysOfYearAsArray' function takes a year and returns an array of Date objects for all days in that year.

A tooltip from the AI Copilot feature is overlaid on the code, reading: '/tests Copilot generated code may be incorrect'. The tooltip has a small icon of a person with a question mark.

```
3 import { WeekDays } from "../features/settings/settings.slice";
4
5
6 // Returns each visible day in the calendar between start and end as an array of Date objects
7 export function daysAsArray(
8   start: string, end: string,
9   visibleWeekDays: WeekDays = [true, true, true, true, true, true, true]): Date[] {
10   return eachDayOfInterval({
11     start: parseISO(start),
12     end: parseISO(end)
13   }).filter((d) => {
14     return visibleWeekDays[d.getDay()];
15   });
16 }
17
18 export function daysOfYearAsArray(year: number): Date[] {
19   return daysAsArray(year + "-01-01", year + "-12-31");
20 }
```

Quoi qu'il en soit, vous devez ensuite corriger les tests. Suite à des erreurs au niveau de l'implémentation, GitHub Copilot pourrait obtenir des prévisions de résultats différentes des vôtres et, de ce fait, formuler des hypothèses erronées dans les tests.



Jannik Reinhard

Ingénieur système en chef, Jannik (@jannik_reinhard) s'est vu décerner la distinction Microsoft MVP dans la catégorie « Enterprise Mobility ». Il travaille comme responsable technique dans la division AIOps de la plus grande entreprise chimique du monde. Sur son temps libre, il tient un blog et intervient lors d'événements.

Conseil n° 7 :

Résolution proactive des problèmes sur les appareils Windows

Utilisation des scripts de remédiation Intune avec GitHub Copilot

par Jannik Reinhard

Microsoft Intune est une plateforme de sécurité et de gestion unifiée des terminaux qui permet aux entreprises de gérer et de protéger leurs terminaux mobiles et leurs smartphones, postes de travail et autres appareils de type HoloLens, etc. Composants essentiels d'Intune, les scripts de remédiation proactifs servent à détecter les problèmes éventuels sur des appareils via un script de détection, à les identifier et, en fonction de la valeur de retour, à exécuter un script de remédiation afin de corriger ces problèmes.

Les scripts sont généralement écrits dans PowerShell et s'appuient les uns sur les autres. C'est là que GitHub Copilot est d'une grande aide. Il suffit simplement d'écrire qu'il s'agit d'un script de remédiation et d'indiquer l'erreur que doit vérifier ce script.

Tous les jours, de nouveaux problèmes apparaissent et sont signalés par les utilisateurs ou via les logs. Ici, l'objectif est d'adopter une approche proactive et de développer directement des scripts pour ces erreurs, capables de détecter et de corriger automatiquement les problèmes rencontrés par les utilisateurs ou lors d'une nouvelle apparition de l'erreur. Cette approche réduit les efforts du Service Desk et le nombre de tickets, tout en offrant aux utilisateurs une meilleure expérience au quotidien.

Comme la création de scripts doit s'effectuer en principe dans un délai restreint, GitHub Copilot est très utile pour gagner un temps précieux. Fini les recherches fastidieuses pour chaque commande, fonction ou ligne de code. Dans bon nombre de cas, il suffit de créer une description détaillée du résultat que je souhaite obtenir à l'aide du script. GitHub Copilot crée ensuite le script à ma place.

Prenons un exemple pour lequel nous allons vérifier s'il existe, dans le journal des événements Windows, des événements affichant l'ID 1000 ou 1001. Il s'agit en l'occurrence de problèmes de stabilité des applications.

Tout d'abord, j'écris un commentaire pour cet exemple, qui décrit les éléments à vérifier :

```
# Intune proactive remediation script to detect if events 1000 and 1001  
are present in the last 24 hours.
```

```
> Detection.ps1 •  
> Detection.ps1  
1 # Intune proactive remediation script to detect if 1000 and 1001 events are present in the last 24 hours  
2  
3 # Get the current date/time  
4 $now = Get-Date  
5
```

Il ne me reste plus qu'à appuyer plusieurs fois sur la touche « Tab ». GitHub Copilot génère ensuite le script à ma place. Je peux ainsi me concentrer sur la commande à exécuter pour lire les journaux d'événements, ou sur la manière de filtrer les données des dernières 24 heures.

```
# Intune proactive remediation script to detect if 1000 and 1001 events are present in the last 24 hours  
# Get the current date/time  
$now = Get-Date  
  
# Get the last 24 hours  
$last24hours = $now.AddHours(-24)  
  
# Get the 1000 and 1001 events from the last 24 hours  
$events = Get-WinEvent -FilterHashtable @{LogName='Application';ID=1000,1001;StartTime=$last24hours}  
  
# If there are 1000 and 1001 events in the last 24 hours, return a non-zero exit code  
if ($events.Count -gt 0) [  
    exit 1  
]
```

Je dois encore affiner l'ensemble pour filtrer uniquement les problèmes concernant le client Teams. Pour ce faire, j'ajoute un nouveau commentaire pour filtrer par Teams :

```

# Intune proactive remediation script to detect if 1000 and 1001 events are present in the last 24 hours

# Get the current date/time
$now = Get-Date

# Get the last 24 hours
$last24hours = $now.AddHours(-24)

# Get the 1000 and 1001 events from the last 24 hours
$events = Get-WinEvent -FilterHashtable @{LogName='Application';ID=1000,1001;StartTime=$last24hours}

# Filter only events where Microsoft Teams crashed
$events = $events | Where-Object {$_.Message -like "*Microsoft Teams*crashed*"}

# If there are 1000 and 1001 events in the last 24 hours, return a non-zero exit code
if ($events.Count -gt 0) {
    exit 1
}

```

Ici, la suggestion n'est pas tout à fait correcte, mais le commentaire n'était pas assez précis. Je vais le corriger :

```

# Intune proactive remediation script to detect if 1000 and 1001 events are present in the last 24 hours

# Get the current date/time
$now = Get-Date

# Get the last 24 hours
$last24hours = $now.AddHours(-24)

# Get the 1000 and 1001 events from the last 24 hours
$events = Get-WinEvent -FilterHashtable @{LogName='Application';ID=1000,1001;StartTime=$last24hours}

# Filter only events where Microsoft Teams crashed (Microsoft Teams)
$events = $events | Where-Object {$_.Message -like "*Microsoft Teams*crashed*"}

# If there are 1000 and 1001 events in the last 24 hours, return a non-zero exit code
if ($events.Count -gt 0) {
    Write-Host "Events found"
    exit 1
}
Write-Host "No events found"
exit 0

```

Maintenant que le script de détection est prêt, nous avons besoin d'un script de remédiation pour vider le cache de Teams dans ce genre de situation. GitHub Copilot va également créer ce script. La procédure est identique. J'écris un commentaire sur le résultat que je souhaite obtenir. Voici mon commentaire :

Intune proactive remediation script to remediate team crashes with cleaning the team cache.

GitHub Copilot va à nouveau reconnaître le résultat à atteindre et il me suffira de valider.

Malheureusement, le résultat n'est toujours pas satisfaisant. J'ai obtenu le même résultat qu'avec le script de détection et je dois le modifier.

```
# Intune proactive remediation script to remediate teams crashes with cleaning the teams cache

# Get the current date/time
$now = Get-Date

# Get the last 24 hours
$last24hours = $now.AddHours(-24)

# Get the 1000 and 1001 events from the last 24 hours
$events = Get-WinEvent -FilterHashtable @{LogName='Application';ID=1000,1001;StartTime=$last24hours}

# Filter only events where Microsoft Teams crashed
$crashes = $events | Where-Object {$_.Message -like "*Microsoft Teams*stopped working*"}

# If there are 1000 and 1001 events in the last 24 hours, return a non-zero exit code
```

Nous allons utiliser ici une petite astuce. J'ai simplement signalé à GitHub Copilot ce qu'il doit faire ici. Il me suffit maintenant de « tabuler » pour obtenir un script parfait.

```
# Intune proactive remediation script to remediate teams crashes with cleaning the teams cache

# Clear teams cache
Remove-Item -Path "$env:LOCALAPPDATA\Microsoft\Teams\*" -Recurse -Force -ErrorAction SilentlyContinue

# Restart teams
Stop-Process -Name Teams -Force -ErrorAction SilentlyContinue
Start-Process -FilePath "$env:LOCALAPPDATA\Microsoft\Teams\current\Teams.exe" -ErrorAction SilentlyContinue

# Wait 10 seconds
Start-Sleep -Seconds 10

# Check if teams is running
$teamsRunning = Get-Process -Name Teams -ErrorAction SilentlyContinue

# If teams is not running, return a non-zero exit code
if ($teamsRunning -eq $null) {
    exit 1
}
```



Holger Sirtl

Holger est architecte de solutions cloud chez Microsoft depuis 2006. Il aide les clients professionnels à créer des architectures applicatives basées sur le cloud. En tant qu'ambassadeur quantique, il est passionné d'informatique quantique.

Conseil n° 8 :

À quoi ressemblait le code à ses débuts ?

par Holger Sirtl

La programmation est ma passion. Depuis mon enfance, je suis fasciné par le monde du code et des algorithmes. C'est vraiment formidable de pouvoir résoudre des problèmes complexes et développer des solutions innovantes à l'aide de quelques lignes de code. En tant qu'architecte de solutions cloud chez Microsoft, mon rôle est de mettre cette passion au service de différents projets pour nos clients et partenaires en programmant des solutions sur mesure.

Dans mes projets clients, je suis souvent confronté au défi de créer de petits exemples de code pour faire la démonstration de nouvelles fonctions ou de meilleures pratiques. Ce faisant, je dois surtout relever deux grands défis : d'une part, mon travail en tant qu'architecte de solutions confronté à des scénarios clients très variables ne me laisse souvent que très peu de temps pour programmer moi-même, et d'autre part, mes projets nécessitent que je développe dans de nombreux langages de programmation différents. Cela signifie que je dois les maîtriser.

Comment itérer en Python via un tableau ? Comment créer une arborescence en C# ou en Java ? Comment obtenir le contenu d'une variable d'environnement dans PowerShell ? Comment faire telle opération sous Linux dans le Bash ? Les exemples ne manquent pas et il est pratiquement impossible de connaître par cœur tous les concepts de programmation dans tous les langages de programmation. C'est là que GitHub Copilot entre en jeu. Cet outil me permet d'élaborer l'exemple de code souhaité via des commentaires. N'ayant plus à mémoriser chaque détail du langage de programmation, je peux me concentrer sur la logique et l'algorithme.

GitHub Copilot permet de créer très facilement des exemples de code et autres petites démonstrations. En fait, je dois simplement savoir comment formuler des commentaires dans la langue souhaitée. J'écris ensuite facilement l'algorithme désiré sous forme de commentaire en allemand ou en anglais, selon les préférences du client. Puis, GitHub Copilot convertit automatiquement ma demande dans le code correspondant.

Voici un aide-mémoire pour les commentaires dans les langages de script et de programmation les plus importants à mes yeux :

Langage	Commentaires sur une seule ligne	Commentaires sur plusieurs lignes
Bash	# comment	:' comment '
C#	// comment	/* comment */
C++	// comment	/* comment */
CSS	/* comment */	/* comment */
HTML	<!-- comment -->	<!-- comment -->
Java	// comment	/* comment */
JavaScript	// comment	/* comment */
PHP	// comment	/* comment */
PowerShell	# comment	<# comment #>
Python	# comment	<>>> comment <>>>

Cette approche a bien évidemment ses limites. Pour créer des solutions logicielles d'envergure, il est indispensable de connaître la structure et les concepts du langage de programmation utilisé. Mais pour des projets plus modestes, GitHub Copilot est très utile pour trouver rapidement des solutions ou réaliser des ajouts à un logiciel existant. GitHub Copilot vous fait gagner du temps et facilite votre travail en vous évitant d'avoir à mémoriser le moindre détail du langage de programmation. Dans l'ensemble, je suis impressionné par les possibilités offertes par GitHub Copilot, un outil très performant qui m'aide à mettre ma passion de la programmation au service des projets de mes clients.

Je ne peux que vous recommander de tester cet outil afin de constater son efficacité et sa simplicité d'utilisation car, en fin de compte, l'essence de la programmation, loin de se résumer à l'écriture de code, est la résolution de problèmes et la mise en pratique d'idées. Et c'est exactement en cela que GitHub Copilot m'est utile.



Robin-Manuel Thiel

Le jour, Robin-Manuel (@robinmanuelt) occupe le poste de « Global Black Belt » chez Microsoft pour les architectures natives cloud et les applications IA. La nuit, il vaque à ses occupations de « nerd », qui consistent à bricoler toutes sortes d'appareils disposant de câbles. Podcast : <http://todocast.io>

Conseil n° 9 :

Automatisation des tâches de développement fastidieuses avec GitHub Copilot Chat

par *Robin-Manuel Thiel*

À l'instar de ChatGPT, GitHub Copilot offre une interface textuelle permettant de communiquer avec l'IA en langage naturel via des invites. Par ailleurs, vous pouvez consulter des fichiers ou des tronçons de code source individuels ou multiples pertinents pour répondre à la question posée. La structure des informations saisies (invites) joue également un rôle essentiel dans GitHub Copilot Chat, comme pour la plupart des instructions en langage naturel données à une IA. L'ingénierie des invites désigne la spécialité consistant à concevoir de manière optimale les invites destinées à l'IA. Les tâches répétitives suivent souvent un modèle spécifique. La clarté de la communication de l'instruction à l'IA pour l'exécution de ces tâches via l'invite influe considérablement sur les chances de réussite de l'opération.

Voici quelques éléments pour faciliter la structuration d'une invite afin d'améliorer son traitement par l'IA et la précision de la réponse apportée :

- Inclure des exemples.
- Structurer les invites à l'aide de séparateurs et séparer les demandes d'informations contextuelles.
- Marquer clairement les blocs de code avec des instructions Markdown.

Exemple : transformer le code source en une infrastructure cloud

Récemment, dans le cadre d'un projet client, on m'a demandé de créer une nouvelle table dans une base de données SQL pour chaque valeur d'un type énuméré avec des analyses de sentiments.

```
enum Sentiment {  
    Happy = 'happy',  
    Excited = 'excited',  
    Content = 'content',  
    Joyful = 'joyful',  
    Optimistic = 'optimistic',  
    Amused = 'amused',  
    Pleased = 'pleased',  
    Relieved = 'relieved',  
    Grateful = 'grateful',  
    Inspired = 'inspired',  
    Proud = 'proud',  
    Playful = 'playful',  
    Satisfied = 'satisfied',  
    Hopeful = 'hopeful',  
    Serene = 'serene',  
    Jubilant = 'jubilant',  
    Ecstatic = 'ecstatic',  
    Elated = 'elated',  
    Blissful = 'blissful',  
    Radiant = 'radiant',  
    Merry = 'merry',  
    Enthusiastic = 'enthusiastic',  
    Cheerful = 'cheerful',  
    Whimsical = 'whimsical',  
    Eager = 'eager'  
}  
  
export default Sentiment;
```

Il y a quelques années, cette tâche aurait été fastidieuse, mais aujourd'hui, grâce à GitHub Copilot Chat, elle peut être facilement automatisée en quelques secondes. J'ai donc surligné l'ensemble du code du type énuméré et lancé GitHub Copilot Chat avec cette première invite :

Create an SQL script that creates a new table for each value of the enum. The table name should be the enum value itself. The table consists of the following columns: ID (primary key), Text, Score.

Le résultat fut plutôt décevant :

Here is the SQL script that creates a new table for each value of the enum.

```
CREATE TABLE happy (
    ID INT PRIMARY KEY,
    Text VARCHAR(255),
    Score INT
);
```

J'ai reçu la commande SQL qui, à partir d'une seule valeur de type énuméré, a créé une table selon mes souhaits. Toutefois, le nom de cette table commençait par une lettre minuscule (ce qui ne correspondait pas à nos directives) et ne concernait qu'une seule valeur.

À ce stade, j'aurais pu affiner l'invite initiale en spécifiant une instruction demandant l'utilisation de majuscules pour le nom de la table. De plus, j'aurais pu préciser dans le texte une instruction demandant à ne créer qu'un script avec chaque commande SQL pour la création d'une table par valeur de type énuméré.

L'amélioration des invites dans plusieurs itérations aurait probablement pris autant de temps que l'écriture manuelle du script SQL avec les informations déjà obtenues. Une méthode nettement plus efficace pour optimiser l'obtention du résultat souhaité consiste à ajouter des exemples concrets lors du processus d'ingénierie des invites. J'ai donc utilisé les conclusions de la première réponse pour élaborer une seconde invite enrichie de plusieurs exemples :

Create an SQL script that creates a new table for each value of the enum. The table name should be the enum value itself. The table consists of the following columns: ID (primary key), Text, Score.

Example:

```

```
CREATE TABLE Happy (
 ID INT PRIMARY KEY,
 Text VARCHAR(255),
 Score INT
);
```

```
CREATE TABLE Excited (
 ID INT PRIMARY KEY,
 Text VARCHAR(255),
 Score INT
);
```

```

Le résultat était spectaculaire. Cette fois, il a parfaitement complété l'exemple de script de l'invite avec les valeurs restantes du type énuméré. GitHub Copilot a également tenu compte cette fois de la casse du nom de la table que je n'avais pas évoquée de manière explicite, mais seulement mentionnée dans l'exemple. Grâce à quelques astuces comme l'ajout d'un exemple issu de la première réponse et une délimitation claire entre l'instruction et l'exemple marquée par des tirets, l'invite a pu être orientée correctement, sans reformulation.

Conclusion

Le point fort de GitHub Copilot Chat réside dans sa capacité à comprendre le code existant. Lors de la conversion de type énuméré dans les commandes SQL, GitHub Copilot doit comprendre le contexte correspondant et générer un code non seulement syntaxiquement correct, mais aussi pertinent sur le

plan sémantique. Concernant les tâches répétitives et les conversions sémantiques entre différentes technologies, GitHub Copilot Chat change la donne pour les développeurs. Cela vaut en particulier pour les tâches qui nécessitent une compréhension du code existant et qui doivent être adaptées à une autre technologie. C'est précisément là que GitHub Copilot Chat démontre sa supériorité.

Les interactions avec un « programmeur pair » basé sur les conversations et assisté par l'IA font non seulement gagner du temps, mais réduisent également les erreurs humaines, tout en boostant la productivité des développeurs pour qu'ils puissent concentrer leurs efforts sur les tâches intéressantes, stimulantes et importantes du développement de logiciels. L'IA s'occupe de la partie fastidieuse.



Christian Wenz

Christian (@chwenz) est un pionnier du Web, un spécialiste des technologies et un entrepreneur. Depuis 1999, il a écrit de nombreux livres sur les technologies Web et autres sujets connexes, qui ont été traduits dans 11 langues. Son activité principale consiste à conseiller des entreprises dans le domaine de la digitalisation et de l'industrie 4.0. Depuis 2004, il est Microsoft MVP dans la catégorie Technologies de développement.

Conseil n° 10 :

Amélioration des invites pour un meilleur code : conseils et astuces pour les développeurs

par Christian Wenz

Les démonstrations de produits et les présentations de GitHub Copilot lors de conférences mettent l'accent sur la simplicité avec laquelle cet outil permet de générer du code opérationnel. Les scénarios de test sélectionnés sont similaires et bien préparés. Dans la pratique, il n'est pas toujours si simple de travailler avec GitHub Copilot, car les tâches triviales alternent parfois avec des demandes complexes. GitHub Copilot n'est pas totalement adapté à chaque cas d'utilisation.

Dans les cas où une utilisation est concevable, la génération de code avec GitHub Copilot ne fonctionne pas toujours d'une manière aussi harmonieuse que lors d'une démonstration dans des conditions idéales. Selon le concept « Garbage in, garbage out », une donnée erronée entrant dans un processus ne peut que produire des résultats erronés en bout de chaîne ou, pour le dire plus positivement, la qualité de l'invite a un énorme impact sur la qualité du code généré. D'où l'apparition de la spécialité « ingénierie des invites » il y a quelques années de cela (un article sur le sujet a été créé dans Wikipedia en octobre 2021). Les invites optimisées améliorent la qualité des résultats. Dans GitHub Copilot, cela concerne aussi bien les invites de commandes directes que les commentaires dans le code qui peuvent être traités par GitHub Copilot.

Mais comme toujours en pareille situation, rien n'est figé dans le marbre. Il n'existe pas de solution unique adaptée à chaque application. Toutefois, le respect de certaines règles contribue à améliorer le résultat :

1. Rester clair et non équivoque

Si vous chargez une équipe de développement de créer une application pour une liste de tâches, vous n'obtiendrez sans doute pas le résultat escompté. Par exemple, il peut manquer des informations sur le système cible, la fonctionnalité souhaitée, l'expérience utilisateur, etc. Ici, la précision est essentielle :

en effet, la précision des demandes influe directement sur le niveau de concordance des résultats. Ce constat est d'autant plus vrai pour un outil logiciel comme GitHub Copilot. Formulez précisément le résultat escompté en évitant d'ajouter des détails non pertinents (voir point 2 pour connaître les exceptions).

Par ailleurs, la formulation doit être claire et réalisable. Dans la mesure du possible, évitez les négations et priviliez les instructions et demandes claires.

2. Le contexte est roi

Une invite commence de préférence comme un récit utilisateur : le contexte est expliqué. La question principale est la suivante : que doit permettre de réaliser le code ? L'exemple canonique serait celui d'une application pour une liste de tâches. On commence donc l'invite avec précisément cette déclaration. Cela permet au système, d'une part, de prendre en compte les solutions existantes dans ce domaine et d'autre part, de définir un concept pour la solution à élaborer. Vient ensuite une description des différentes fonctionnalités à implémenter.

3. Vers le succès, étape par étape

À l'instar de la programmation, la prise en charge du code par GitHub Copilot est un processus itératif. Le premier essai correspond rarement au résultat final souhaité. Il est toujours possible d'améliorer le code, soit via GitHub Copilot, soit en intervenant vous-même sur ce code. Il est donc judicieux de spécifier séparément les différentes tâches demandées dans le code. Vous pouvez procéder par étapes successives. Par exemple, lors de la création d'une API, vous pouvez générer en premier le modèle de données avant d'implémenter les points de terminaison.

Vous pouvez bien entendu procéder également en une seule étape, par exemple en spécifiant le contexte (voir point 2), puis une liste à puces avec les fonctionnalités pertinentes. Toutefois, dans ce cas, une intervention n'est possible que lorsque l'ensemble du code est disponible. Avec une procédure itérative, des corrections peuvent être effectuées plus tôt.

4. Utiliser des exemples

Ce qui, pour certains, semble aller de soi peut en laisser d'autres perplexes. Pour l'ingénierie des invites, il est souvent utile spécifier le résultat d'une fonctionnalité à l'aide d'exemples. À cet effet, la structure des données est également définie telle qu'elle doit être traitée ultérieurement.

The screenshot shows a Visual Studio Code window with a JavaScript file named 'copilot.js'. The code is as follows:

```
function getRandomDate() {
    const start = new Date();
    const end = new Date(new Date().getTime() + 86400000);
    return new Date(start.getTime() + Math.floor(Math.random() * (end.getTime() - start.getTime())));
}

function getRandomWeather() {
    const temperature = Math.floor(Math.random() * 100);
    const date = getRandomDate();

    return {
        date: date.toISOString(),
        temperature: `Temperature: ${temperature}°C`
    };
}
```

In the left sidebar, there is a 'Copilot' section with the following content:

- wenz: Create a function to return a random weather forecast. Desired outcome: {temperature: '21° C', date: '2023-11-14'}.
- GitHub Copilot: Used 1 reference > Here's a step-by-step plan:
 - How can I use the `getRandomWeather` function in a React component?
- Ask Copilot or try ▶

Cette spécification d'exemples est importante pour différents aspects de l'hygiène du code : GitHub Copilot peut tenir compte des schémas de nomenclature des identifiants et du style de code déjà appliqué. Au final, le code semblera (presque) avoir été écrit par vous.

5. Faire preuve de politesse

Je ne plaisante pas ! En l'occurrence, je ne fais aucunement référence au dessin animé populaire auprès des passionnés d'informatique, dans lequel des robots qui dominent le monde épargnent une personne, parce qu'elle dit toujours « s'il vous plaît » et « merci » lorsqu'elle s'adresse à un système IA. Toutefois, tout porte à croire qu'entretenir des rapports courtois avec un système informatique améliore l'humeur de l'utilisateur et son comportement personnel. Soyons honnêtes : une utilisation correcte nous permet de compenser le travail supplémentaire de saisie de formules de politesse par la rapidité de génération des résultats, non ?



Suad Wolgram

Suad est développeur informatique et cloud junior chez white duck GmbH et étudie l'informatique à l'université technique de Rosenheim.

Conseil n° 11 :

Noms de fonctions et variables pertinents

par Suad Wolgram

L'utilisation de noms de fonction pertinents est un facteur déterminant pour exploiter efficacement GitHub Copilot. C'est pourquoi je souhaiterais vous montrer comment aider GitHub Copilot à générer des suggestions de code plus précises en utilisant de meilleurs noms de fonction.

Considérons le scénario avec l'exemple de code suivant. Dans la fonction `fetch()`, nous souhaitons appeler tous les personnages Star Wars via l'API Star Wars :

```
export class ApiService {  
  private starWarsUrl = environment.baseUrl;  
  
  constructor(private httpClient: HttpClient) {}  
  
  fetch() {  
    }  
}
```

Dans ce contexte, il apparaît clairement que GitHub Copilot ne comprend pas l'objectif que nous souhaitons atteindre avec la méthode `fetch()`. Nous ne pouvons donc pas obtenir de suggestions appropriées à ce stade. Une solution consiste à remplacer `fetch()` par un nom de fonction pertinent qui décrit précisément la tâche que la méthode doit réaliser. Une fois le contexte clarifié, GitHub Copilot propose immédiatement des suggestions pour atteindre l'objectif escompté :

```
getAllStarWarsCharacters() {  
    return this.httpClient.get(`${this.starWarsUrl}/people`);  
}
```

Dans ce cas, GitHub Copilot a même reconnu notre variable `starWarsUrl` et l'a intégrée dans la solution.

Prenons un second exemple : nous souhaitons calculer le nombre d'habitants d'une planète de l'univers Star Wars et restituer les informations obtenues dans une carte observable.

Comme précédemment, nous choisissons un nom de méthode pertinent et indiquons le résultat que nous souhaitons obtenir. Par ailleurs, nous définissons une variable avec le nom `planetCitizenCount` :

```
calculateCitizenCountForEachPlanet(): Observable<Map<string, number>> {  
    let planetCitizenCount = new Map<string, number>();  
  
    return this.getAllStarWarsCharacters().pipe(  
        map((characters: any) => {  
            characters.forEach((character: any) => {  
                let planetName = character.homeworld;  
                if (planetCitizenCount.has(planetName)) {  
                    planetCitizenCount.set(  
                        planetName,  
                        planetCitizenCount.get(planetName) + 1  
                    );  
                } else {  
                    planetCitizenCount.set(planetName, 1);  
                }  
            });  
            return planetCitizenCount;  
        })  
    );  
}
```

Dès que nous commençons à insérer la logique, GitHub Copilot reconnaît le contexte en se basant sur le nom de la méthode choisie et sur les noms de variables appropriés, puis nous propose l'implémentation correspondante. Nous pouvons également voir que GitHub Copilot utilise correctement la variable grâce à la clarté du contexte.

**Joël Zimmerli**

Développeur informatique Full-Stack expérimenté dans différents langages de programmation comme Java, C# et TypeScript, Joël Zimmerli concentre son activité sur le développement piloté par les tests (TDD).

Conseil n° 12 :

Prise en charge de la génération de cas de test avec GitHub Copilot

par Joël Zimmerli

Pour le développement piloté par les tests, l'écriture des tests unitaires est, parmi les tâches importantes, l'une des plus chronophages. Dans cet article, je souhaite montrer comment j'utilise GitHub Copilot pour écrire mes tests de manière efficace et complète afin de disposer de davantage de temps à consacrer au développement en tant que tel.

Prenons un exemple concret avec une application .NET dans laquelle des biens immobiliers ont été enregistrés avec leurs prix de vente. La fonction de recherche permet un filtrage par fourchette de prix. Pour implémenter ce filtre, j'ai écrit une Factory qui crée une fonction de comparaison sur la base des prix minimums et maximums. La figure suivante présente un extrait de cette fonction :

```
static Expression<Func<HouseDocument, bool>> PriceIsInRangeFilter(
    int? minPrice, int? maxPrice){
    var expressionBuilder = PredicateBuilder.New<HouseDocument>(true);
    if (minPrice.HasValue)
        expressionBuilder.And(PriceHigherThanFilter(minPrice.Value));
    if (maxPrice.HasValue)
        expressionBuilder.And(PriceLowerThanFilter(maxPrice.Value));

    return expressionBuilder;
}
```

Créer le nom du test est la première étape pour utiliser GitHub Copilot pour la génération automatique de tests. Ce nom décrit l'objet à tester, la fonction à appeler et le résultat à obtenir. Cette structure de nom est largement répandue dans le monde des tests. GitHub Copilot peut ainsi générer automatiquement l'ensemble du test comme l'illustre cette figure.

```
[Fact]
public void HouseQueryToExpression_ShouldEvaluateCorrectly(){
    // Arrange
    var query = new HouseQuery()
    {
        UpperPriceLimit = 100,
        LowerPriceLimit = 0
    };
    var house = new HouseDocument()
    {
        Id = „house“,
        Name = „house“,
        Price = 50
    };

    // Act
    var expression = query.ToExpression();
    var result = expression.Invoke(house);

    // Assert
    result.Should().BeTrue();
}
```

Ce test doit être ensuite mis à l'épreuve avec les différents prix des biens immobiliers. Pour ce faire, le test peut être paramétrisé de manière spécifique, comme illustré dans la figure suivante.

```
[Theory]
public void HouseQueryToExpression_ShouldEvaluateCorrectly (int?
upperLimit, int? lowerLimit, int housePrice, bool expected){
    // Arrange
    var query = new HouseQuery()
    {
        UpperPriceLimit = upperLimit,
        LowerPriceLimit = lowerLimit
    };
    var house = new HouseDocument()
    {
        Id = „house“,
        Name = „house“,
        Price = housePrice
    };

    // Act
    var expression = query.ToExpression();
    var result = expression.Invoke(house);

    // Assert
    result.Should().Be(expected);
}
```

GitHub Copilot peut également générer les paramètres de test, comme illustré dans la figure suivante.

```
[Theory]
[InlineData(100, 0, 50, true)]
public void HouseQueryToExpression_ShouldEvaluateCorrectly(int?
upperLimit, int? lowerLimit, int housePrice, bool expected){
```

Après plusieurs générations des paramètres, on obtient une liste complète des possibilités, comme illustré dans la figure suivante. Il est toutefois recommandé de vérifier et de revoir attentivement cette liste.

```
[Theory]
[InlineData(100, 0, 50, true)]
[InlineData(100, 0, 99, true)]
[InlineData(100, 0, 1, true)]
[InlineData(100, 0, 150, false)]
[InlineData(100, 50, 150, false)]
[InlineData(100, 150, 200, false)]
[InlineData(100, 0, 0, false)]
[InlineData(100, 100, 100, false)]
[InlineData(100, 100, 0, false)]
[InlineData(null, null, 0, true)]
[InlineData(100, null, 0, true)]
[InlineData(null, null, 101, false)]
[InlineData(null, 100, 101, true)]
[InlineData(null, 100, 99, false)]
public void HouseQueryToExpression_ShouldEvaluateCorrectly (int?
upperLimit, int? lowerLimit, int housePrice, bool expected){
```

En suivant ces étapes, GitHub Copilot peut vous aider à créer efficacement des tests dans différents scénarios.

**Tobias Deekens**

Ancien basketteur et guitariste médiocre, Tobias Deekens est aussi un développeur, un enseignant passionné et un orateur naturel qui possède une solide expérience du développement et de l'architecture front-end. Il prend un immense plaisir à travailler avec des équipes diversifiées dans des environnements agiles et à les former.

Conseil n° 13 :

Trois mois d'évaluation de l'impact de GitHub Copilot

par Tobias Deekens

En février dernier, GitHub a annoncé le lancement de [GitHub Copilot for Business](#). Cette annonce a immédiatement attiré notre attention et suscité notre intérêt, et les ingénieurs de toute l'organisation ont fait part de leur souhait d'utiliser ce produit. Après avoir défini en interne une stratégie d'adoption, nous avons décidé d'évaluer GitHub Copilot pendant trois mois pour voir s'il pouvait améliorer notre productivité. Nous décrirons dans cet article notre démarche d'évaluation et d'adoption de GitHub Copilot.

Chez commercetools, nous utilisons des langages de programmation et des outils divers pour construire nos différents produits, de Scala à TypeScript et de PHP à Go et Rust. Nous avons pour principe de choisir nos technologies en connaissance de cause, afin de sélectionner des outils qui améliorent notre productivité. En outre, nous voulons conserver notre esprit d'équipe, qui est bien ancré dans les valeurs de notre entreprise, à mesure que celle-ci se développe. Nous constatons que la diversité des outils construits avec l'IA générative a un impact énorme sur la collaboration des ingénieurs logiciels et nous sommes impatients d'intégrer ces outils à nos routines quotidiennes.

Pourquoi évaluer avant d'adopter ?

Vous vous demandez peut-être pourquoi nous avons évalué pendant trois mois un produit aussi omniprésent et performant que GitHub Copilot au lieu de le proposer directement à tous les ingénieurs. C'est que, par principe, commercetools pense que l'adoption de l'IA doit être le fruit d'une démarche pragmatique. Même si l'IA fait plutôt l'unanimité, les choix et les décisions d'utilisation ne doivent pas être imposés d'en haut. Nous voulons que nos équipes évaluent et décident elles-mêmes comment l'IA peut améliorer la productivité et apporter des fonctionnalités. Dans notre cas, le service d'ingénierie a étudié GitHub Copilot comme nous le faisons pour n'importe quel nouvel outil. Ainsi, les personnes qui ont participé à l'évaluation sont celles qui auront à utiliser l'outil pour le meilleur et pour le pire, et dont nous souhaitions obtenir un avis réaliste.

En outre, l'abondance des produits nouveaux et des outils améliorés basés sur l'IA générative rend indispensable une évaluation approfondie de l'impact de chaque outil individuellement, ainsi que de l'association de différents outils. Par exemple, est-il préférable d'utiliser Replit Ghostwriter, Codeium, CodeComplete et GitHub Copilot en tandem, ou faut-il envisager d'associer l'un ou l'autre à un outil différent tel que Mintlify ou Wrap ? Pour répondre à cette question, il ne suffit pas de parcourir les sites Web commerciaux de ces outils, il faut les avoir pratiqués.

Pour étayer notre décision d'adoption, nous voulions comprendre clairement quel était l'impact attendu et l'impact réel de GitHub Copilot au sein de notre service d'ingénierie, autrement dit pour nos ingénieurs front-end, nos ingénieurs back-end, nos ingénieurs en fiabilité des sites, nos ingénieurs en automatisation des tests et pour les personnes travaillant sur la documentation.

Méthode d'évaluation de GitHub Copilot

Après avoir décidé de réaliser une évaluation contrôlée, nous avons commencé par fixer une durée pertinente : une période de trois mois, répartie sur deux trimestres, semblait idéale. Nous espérions que cette durée nous permettrait d'obtenir un aperçu du cycle d'ingénierie complet, incluant une fin de trimestre. C'est en effet le moment où les équipes déploient souvent de nouvelles fonctionnalités sur l'ensemble de nos produits.

Après avoir fixé la durée, nous devions déterminer la taille de l'échantillon. Avec seulement cinq à dix ingénieurs sur 150, l'évaluation risquait de produire des résultats faussés. Par conséquent, nous souhaitions intégrer 30 à 35 ingénieurs à l'équipe d'évaluateurs, soit une représentation de 20 à 25 %. Enfin, nous avons souhaité impliquer le plus grand nombre de spécialisations possible afin de constituer un groupe hétérogène utilisant des outils et des langages différents.

Nous étions désormais prêts à communiquer sur notre projet via un article de blog interne. Nous avons expliqué le processus et mis en lien un formulaire Google pour ceux qui souhaitaient s'inscrire. Au bout d'une semaine, nous avions 34 inscriptions réparties sur l'ensemble de l'organisation. Cela correspondait à peu de chose près à la taille de l'échantillon que nous souhaitions et, heureusement, nous avons pu intégrer tous les candidats. Tout le groupe a ensuite été ajouté à une liste de diffusion et à un canal Slack destinés à communiquer de nouvelles informations. Pour les accès, tous les membres ont été ajoutés à une équipe dédiée sur GitHub, ce qui leur a permis d'accéder à GitHub Copilot.

Une fois tout en place, nous nous sommes mis en retrait et nous avons laissé chaque personne intégrer GitHub Copilot dans son workflow et poursuivre son travail. Au bout d'une semaine, nous avons fait un point rapide pour nous assurer que chaque participant avait bien installé et intégré GitHub Copilot dans

l'éditeur de son choix. Au cours des semaines suivantes, les évaluateurs ont partagé leurs impressions et leurs échantillons de code dans Slack ou sur Pull Requests. De notre côté, nous préparions en coulisses une vaste enquête finale.

Pendant toute la durée de l'évaluation, nous sommes restés connectés à GitHub en arrière-plan. Les évaluateurs nous ont communiqué des statistiques intéressantes, telles que le taux moyen d'acceptation du code. Par ailleurs, nous avons obtenu GitHub Copilot Chat pour les deux dernières semaines de notre évaluation, ce qui nous a permis d'avoir un aperçu des perspectives futures offertes par GitHub Copilot pour une collaboration renforcée. Nous sommes impatients de connaître l'avenir de GitHub Copilot et des possibilités créées par les différentes propositions.

Résultats et conséquences

Nous avions prévu que GitHub Copilot s'intègrerait parfaitement aux workflows quotidiens et serait facile à utiliser. Nous espérions que les suggestions seraient utiles pour tous les langages de programmation et qu'elles ne gêneraient pas les utilisateurs. Tout au long de notre évaluation, aucune de ces attentes n'a été déçue. En revanche, nous avons découvert des possibilités d'amélioration et remarqué que la qualité des suggestions variait considérablement en fonction du type de travail effectué.

Ces conclusions sont détaillées dans notre enquête principale, qui comportait finalement 15 questions et s'articulait autour de trois grands axes :

1. GitHub Copilot a-t-il été utilisé en permanence ?
2. GitHub Copilot nous rend-il plus productifs ?
3. GitHub Copilot présente-t-il des risques ou des inconvénients majeurs pour nous ?

**« Il rédige les notes de mise à jour à ma place !
C'est absolument génial ! »**

UN ÉVALUATEUR, APRÈS SON PREMIER CAFÉ DU MATIN

Nous avons approfondi ces trois grands axes avec les questions suivantes :

1. Combien de fois avez-vous utilisé GitHub Copilot pendant l'évaluation ?
2. Votre utilisation de GitHub Copilot a-t-elle évolué au cours de ces trois mois ?
3. Combien de fois avez-vous dû modifier des suggestions faites par GitHub Copilot ?
4. Où avez-vous réalisé les gains de productivité les plus importants ?
5. Devrions-nous évaluer d'autres outils utilisant l'IA générative cette année pour améliorer notre productivité ?

« Il y avait une rivalité permanente entre GitHub Copilot et l'habituel IntelliSense. »

UN ÉVALUATEUR OBSERVANT UN COMBAT QUOTIDIEN

Voici les principaux enseignements tirés des réponses obtenues :

- 57 % des évaluateurs ont utilisé GitHub Copilot tous les jours ; tous les autres l'ont utilisé un jour sur deux.
- 95 % des participants affirment que Copilot a amélioré leur productivité.
- 63 % déclarent que leur utilisation de Copilot a augmenté au fil du temps.
- 67 % déclarent que les suggestions étaient utiles.
- 82 % déclarent que les suggestions étaient rarement problématiques.
- 60 % déclarent que GitHub Copilot est un assistant de programmation IA suffisant.
- 80 % ne pensent pas que d'autres outils seraient nettement meilleurs.
- 100 % des participants souhaitent continuer d'utiliser GitHub Copilot.

**« Parfois, GitHub Copilot semble en sommeil,
avec de nombreuses fenêtres VS Code ouvertes.
Puis il vous jette à la figure 50 lignes de code. »**

UN ÉVALUATEUR AYANT REÇU DES SUGGESTIONS EN RAFALE

Outre ces chiffres, nous avons également réussi à recueillir des informations plus qualitatives sur les points forts et les points faibles de GitHub Copilot :

- Réussit l'écriture de tests (72 %).
- Aide à la refonte du code (42 %).
- Est particulièrement performant pour la compléction automatique, l'écriture de texte réutilisable et la génération de modèles automatiques (~60 %).
- A des difficultés avec les logiques métier complexes (82 %).
- Manque de puissance lorsque le contexte du code est important (43 %).
- Doit être envisagé avec prudence lorsque la performance ou la sécurité sont en jeu (27 %).
- Est sans utilité avec les cadres de développement hautement spécialisés ou modernes (14 %).

**« GitHub Copilot est tellement intelligent
qu'il en est parfois dangereux. »**

UN ÉVALUATEUR APRÈS LA SUGGESTION PAR GITHUB COPILOT DE CHARGER UNE PAR UNE
40 000 ENTITÉS D'UNE BASE DE DONNÉES

L'évaluation de GitHub Copilot sur une période relativement longue nous a permis d'identifier des points à améliorer :

- Actuellement, il n'est pas possible de donner son avis sur les suggestions proposées.
- GitHub Copilot ne peut pas être configuré pour être inactif dans des dossiers ou des situations spécifiques.
- Il ne fonctionne pas très bien avec des fichiers différents.
- Il a du mal à effectuer des refontes homogènes sur les bases de code les plus vastes.

Tous ces chiffres nous ont permis de comprendre l'utilité de GitHub Copilot au sein de notre organisation et nous ont aidés à prendre la décision d'adopter cet outil. Une fois mis en service, GitHub Copilot a été utilisé en continu, et son utilisation a même augmenté. Les suggestions, de bonne qualité, étaient souvent acceptées. Les utilisateurs ont pu intégrer facilement l'outil dans leur environnement de travail existant et ont constaté des gains de productivité significatifs. Pour toutes ces raisons, nous poursuivrons le déploiement dans l'ensemble de notre organisation au cours des semaines à venir.

Cet article a été publié à l'origine sur <https://commercetools.com/blog/how-we-evaluated-the-impact-of-github-copilot-for-3-months>. Merci à Tobias et à commercetools de nous avoir permis de le reprendre dans cet e-book.



Dennis Gassen

Dennis Gassen est le responsable « Go to Market » pour l'innovation numérique et applicative chez Microsoft Allemagne. Lorsqu'il n'est pas profondément plongé dans son travail (quel qu'il soit), il soutient la communauté des développeurs. Pendant son temps libre, on a toutes les chances de le croiser sur son vélo quelque part dans le sud de Munich.

Conseil n° 14 :

GitHub Copilot, un assistant IA pour la création de logiciels accessibles

par Dennis Gassen

Je n'expliquerai pas en détail dans cet article comment il serait possible ou préférable de développer des logiciels accessibles car ce n'est pas le propos de cet e-book. Ce que je voudrais montrer, c'est que GitHub Copilot est déjà capable de prendre en compte et de suivre certaines normes fondamentales qui rendent les produits logiciels plus accessibles et utilisables par de nombreuses personnes en situation de handicap.

Environ 1,3 milliard de personnes dans le monde vivent avec un handicap, ce qui représente environ 16 % de la population mondiale. Une grande partie d'entre elles résident dans des pays à faible revenu où les systèmes de santé et d'éducation sont insuffisants, et ces personnes sont en moyenne plus pauvres que les personnes valides. Pour cette population, l'accès à l'information est d'une importance capitale.

Les [Règles pour l'accessibilité des contenus web](#) (WCAG) font office de norme internationale pour l'accessibilité des offres web. Dans l'Union européenne, elles ont un caractère obligatoire pour les institutions publiques depuis septembre 2019 pour les nouveaux sites web, depuis le 23 septembre 2020 pour les sites web existants, et depuis le 23 juin 2021 pour les applications mobiles. Le World Wide Web Consortium (W3C) a publié la version 1.0 des WCAG en 1999. Depuis décembre 2008, la version 2.0 est reconnue comme [Norme ISO](#). Actuellement, la version 2.2 est recommandée par le W3C. Bien que les recommandations du W3C n'aient pas de valeur juridique universelle, elles ont néanmoins un fort caractère contraignant pour le développement de sites web et d'applications accessibles.

Désormais, lorsque nous utilisons GitHub Copilot Chat et que nous demandons simplement comment GitHub Copilot peut nous aider à développer des applications accessibles, nous obtenons la réponse suivante :

Can you support me developing accessible software?

 GitHub Copilot

Absolutely, I can provide guidance on best practices for developing accessible software. Here are some general tips:

1. **Semantic HTML:** Use HTML elements for their given purpose. This helps assistive technologies understand your content.
2. **Alt Text for Images:** Provide alternative text for images. This helps screen readers describe images to visually impaired users.
3. **Keyboard Accessibility:** Ensure that your software can be fully utilized with a keyboard.
4. **Color Contrast:** Ensure that text color contrasts sufficiently with the background color.
5. **Aria Attributes:** Use ARIA (Accessible Rich Internet Applications) attributes where necessary to provide additional context to assistive technologies.
6. **Testing:** Use accessibility testing tools to check your software's accessibility. Tools like aXe or Lighthouse can be very helpful.
7. **User Testing:** Conduct user testing with people who have disabilities. This can provide valuable insights into how accessible your software is in practice.

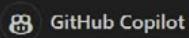
Remember, accessibility isn't a one-time task - it's a continuous aspect of software development.

Ces premières recommandations ne sont pas fausses mais doivent être prises comme de simples conseils généraux. Cependant, l'**ingénierie des invites** nous permet de donner à GitHub Copilot davantage de contexte pour préciser nos besoins et obtenir des réponses plus détaillées.

Pour ce faire, l'invite suivante peut être utilisée :

```
I want to learn more about accessibility standards and need to write code that confirms with WCAG 2.2 defined at https://www.w3.org/TR/WCAG22/. Please act as my accessibility coach to make sure I stick to the most common accessibility standards and guidelines. When you answer accessibility related questions, please also use sources like w3.org and webaim.org. Please also provide links and references in your answers whenever possible. When you suggest code, use semantic HTML and follow the ARIA Authoring Practices Guide and related design patterns.
```

La réponse de GitHub Copilot ressemble alors à ceci :



Absolutely, I can guide you through the process of developing software that adheres to WCAG 2.2 standards and follows the ARIA Authoring Practices Guide. Here are some key principles and coding practices to keep in mind:

1. **Perceivable:** Information and user interface components must be presentable to users in ways they can perceive. This means that users must be able to perceive the information being presented. For example, providing alt text for images, transcripts for audio, and captions for videos.

```

```

2. **Operable:** User interface components and navigation must be operable. This means that users must be able to operate the interface. For example, ensuring that all functionality is available from a keyboard.

```
<button onclick="myFunction()">Click me</button>
```

3. **Understandable:** Information and the operation of user interface must be understandable. This means that users must be able to understand the information as well as the operation of the user interface. For example, using clear labels for form inputs.

```
<label for="fname">First name:</label><br>
<input type="text" id="fname" name="fname"><br>
```

4. **Robust:** Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies. This means that users must be able to access the content as technologies advance. For example, using semantic HTML elements like `<nav>`, `<header>`, `<main>`, and `<footer>` to structure your content.

```
<header>
  <h1>Website Title</h1>
</header>

<nav>
  <a href="#content1">Content 1</a>
  <a href="#content2">Content 2</a>
</nav>

<main id="content1">
  <h2>Content 1</h2>
  <p>This is some content.</p>
</main>

<footer>
  <p>Copyright 2022</p>
</footer>
```

You can find more detailed information on each principle, guideline, and success criteria in the [WCAG 2.2 documentation](#).

For more practical advice on implementing these standards, [WebAIM](#) is a great resource. They provide a wide range of articles, tutorials, and checklists to help you make your software more accessible.

The [ARIA Authoring Practices Guide](#) is a great resource for understanding how to use ARIA to improve accessibility, especially for complex widgets and interaction patterns.

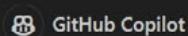
Remember, accessibility is not just about adhering to standards - it's about ensuring that as many people as possible can use and benefit from your software.

L'invite sert de point de départ et contient les éléments suivants :

- Informations contextuelles élargies pour répondre aux requêtes.
- Définition du rôle de GitHub Copilot et du retour d'information attendu.
- Référence à des sources fiables à utiliser pour répondre à nos questions.
- Référence à des ressources et à des sources supplémentaires.

Nous pouvons également demander à GitHub Copilot de donner des exemples précis de cas dans lesquels les normes d'accessibilité n'ont pas été respectées :

Please create a sample website that is not conform with the WCAG 2.2 standards and illustrate on this website what are the most common mistakes when it comes to accessibility.



Sure, here's a simple example of a website that does not conform to WCAG 2.2 standards:

```
<!DOCTYPE html>
<html>
<head>
    <title>Non-accessible Website</title>
    <style>
        .low-contrast {
            color: #888;
            background-color: #aaa;
        }
    </style>
</head>
<body>
    
    <div class="low-contrast">This is a low contrast text.</div>
    <a href="#">Click here</a>
    <input type="text">
    <button onclick="alert('Button clicked!')">Submit</button>
</body>
</html>
```

Here are the most common mistakes in this website:

1. **Missing alternative text:** The image tag `` does not have an `alt` attribute. Screen readers use the `alt` attribute to describe the image to visually impaired users.
2. **Low contrast:** The text inside the `div` with class `low-contrast` has low contrast, making it hard for visually impaired users to read.
3. **Non-descriptive link text:** The link text "Click here" does not provide context about what the link does or where it goes.
4. **Missing form labels:** The input field `<input type="text">` does not have a corresponding label, making it hard for screen reader users to understand its purpose.
5. **Non-accessible JavaScript:** The button uses an `onclick` event handler, which is not accessible to keyboard-only users. It should use a more accessible event like `onkeypress`.

Remember, these are just a few examples of common accessibility mistakes. There are many other aspects of accessibility to consider when developing a website.

Mieux encore, GitHub Copilot suggère immédiatement les solutions adaptées :

Here's how you can fix these issues:

```
<!DOCTYPE html>
<html>
<head>
    <title>Accessible Website</title>
    <style>
        .high-contrast {
            color: #333;
            background-color: #fff;
        }
    </style>
</head>
<body>
    
    <div class="high-contrast">This is a high contrast text.</div>
    <a href="#">More information</a>
    <label for="inputField">Input Field</label>
    <input id="inputField" type="text">
    <button type="button" onclick="alert('Button clicked!')">Submit</button>
</body>
</html>
```

Bien évidemment, les exemples présentés ici sont très simples. Cependant, ils illustrent parfaitement les possibilités offertes par GitHub Copilot pour respecter les normes d'accessibilité et rendre vos logiciels aussi accessibles que possible au plus grand nombre.

Voici d'autres invites qui peuvent vous être utiles :

Invite : Quelles sont les meilleures pratiques pour intégrer des tests d'accessibilité sur l'ensemble du cycle de vie des applications ?

Réponse de GitHub Copilot

Incorporating accessibility testing throughout the application lifecycle is crucial for creating and maintaining accessible software. Here are some best practices:

1. Integrate Accessibility from the Start: Accessibility should be a consideration from the very beginning of the project, not an afterthought. This includes during the design phase, where color contrast, font sizes, and navigation should be considered.
2. Automated Testing: Use automated testing tools like axe, Lighthouse, or WAVE to catch common accessibility issues. These tools can be

integrated into your CI/CD pipeline to ensure accessibility is checked with every build.

3. Manual Testing: Automated testing can't catch everything. Manual testing, including using the software with a screen reader, navigating with a keyboard only, and testing with real users with disabilities, is crucial.

4. Incorporate Accessibility in Code Reviews: Make accessibility a part of your code review process. This can help catch potential issues before they become a part of the codebase.

5. Training and Awareness: Ensure that all members of your team understand the importance of accessibility and how to implement it. This includes developers, designers, content creators, and testers.

6. Regular Audits: Regularly audit your application for accessibility. This can help catch any new issues that may have been introduced.

7. Use WCAG as a Guide: The Web Content Accessibility Guidelines (WCAG) provide a wide range of recommendations for making Web content more accessible. Use these guidelines as a reference throughout your project.

Remember, accessibility is not a one-time task, but a continuous effort throughout the entire application lifecycle.

For more detailed information, you can refer to the WCAG guidelines and WebAIM, which provides a wide range of resources to help you make your software more accessible.

Invite : Comment puis-je vérifier si mon application peut fonctionner avec un lecteur d'écran ?

Réponse de GitHub Copilot

Testing your application for screen reader accessibility involves both automated and manual testing methods. Here are some steps you can follow:

1. **Automated Testing:** Use automated testing tools like axe, Lighthouse, or WAVE to catch common accessibility issues. These tools can help identify issues like missing alt text, incorrect semantic HTML, and missing form labels.

2. **Manual Testing with a Screen Reader:** To truly understand the experience of a screen reader user, you should test your application with a screen reader. There are several screen readers available, including NVDA and JAWS for Windows, VoiceOver for macOS and iOS, and Talk-Back for Android. As you navigate your application with the screen reader, pay attention to whether all content and functionality is accessible and understandable.

3. **Keyboard Navigation:** Screen reader users often navigate using the keyboard, so ensure that all interactive elements are keyboard accessible (can be focused and activated using the keyboard) and that the tab order is logical.

4. **Semantic HTML:** Screen readers rely on semantic HTML to understand the content and structure of the webpage. Ensure that you're using the correct HTML elements for their intended purpose and that you're using ARIA roles and properties where appropriate.

5. **Descriptive Text:** Ensure that all images have alt text, that form inputs have associated labels, and that links and buttons have descriptive text.

Remember, while automated testing can catch many issues, there's no substitute for manual testing with a screen reader. It's also a good idea to involve users with disabilities in your testing process, as they can provide valuable insights based on their lived experiences. For more detailed information, you can refer to the WCAG guidelines

and WebAIM, which provides a wide range of resources to help you make your software more accessible.

Invite : Quelles sont les meilleures pratiques pour créer des tableaux accessibles en langage HTML ?

Réponse de GitHub Copilot

Creating accessible tables in HTML involves ensuring that the table structure is clear and understandable, both visually and for assistive technologies. Here are some best practices:

- 1. Use <th> for Headers:** Use the `<th>` element for table headers, and the `<td>` element for table data. This helps screen readers understand the structure of the table.

```
<table>
  <tr>
    <th>Header 1</th>
    <th>Header 2</th>
  </tr>
  <tr>
    <td>Data 1</td>
    <td>Data 2</td>
  </tr>
</table>
```

- 2. Use scope Attribute:** The `scope` attribute can be used on `<th>` elements to specify whether it's a header for a column (`scope=>col`) or a row (`scope=>row`). This helps screen readers understand the relationship between headers and cells.

```
<table>
  <tr>
    <th scope="col">Header 1</th>
    <th scope="col">Header 2</th>
  </tr>
  <tr>
    <th scope="row">Row Header</th>
    <td>Data</td>
  </tr>
</table>
```

- 3. Use <caption> for Table Description:** The <caption> element provides a description of the table. This can be useful for providing context to screen reader users.

```
<table>
  <caption>A description of the table</caption>
  <!-- rest of table -->
</table>
```

- 4. Use <thead>, <tbody>, and <tfoot>:** These elements can be used to group the header, body, and footer of the table. This can make the table more understandable for screen reader users.

```
<table>
  <thead>
    <!-- header rows -->
  </thead>
  <tbody>
    <!-- body rows -->
  </tbody>
  <tfoot>
    <!-- footer rows -->
  </tfoot>
</table>
```

- 5. Avoid Complex Layouts:** Complex table layouts can be confusing for screen reader users. If possible, stick to simple layouts with clear headers for each row and column.
- 6. Ensure High Contrast:** Ensure that the text and background colors of your table have sufficient contrast to be easily readable.

For more detailed information, you can refer to the WCAG guidelines and WebAIM, which provides a wide range of resources to help you make your tables more accessible.

Résumé

L'ingénierie des invités permet d'améliorer de manière significative l'accessibilité des résultats de GitHub Copilot Chat. Bien entendu, on ne peut pas attendre des résultats parfaits d'un simple « copilote ». Cependant, vous pouvez utiliser GitHub Copilot Chat de manière très efficace pour élargir vos connaissances et votre compréhension des questions d'accessibilité. N'oubliez pas non plus que vous pouvez poser à GitHub Copilot de nouvelles questions sur les réponses obtenues ou demander des références et des exemples supplémentaires. Il appartient aux développeurs de vérifier les suggestions.



#Developer

Syrine Chelly

Syrine Chelly est architecte de solutions cloud et Developer Advocate chez Microsoft où elle travaille depuis plus de 3 ans dans l'unité réussite client dans le domaine du cloud intelligent. Auparavant, elle a travaillé dans le secteur de la robotique, l'automatisation et l'IA pour l'industrie automobile.



Tobias Wittenburg

Tobias Wittenburg est architecte de solutions cloud et développeur chez Microsoft. Il travaille dans l'industrie du logiciel depuis 2005 et a de l'expérience dans le développement de logiciels à grande échelle sur différentes technologies.

Conseil n° 15 :

De l'exigence au composant

par Syrine Chelly et Tobias Wittenburg

L'utilisation du langage naturel dans l'écriture de code est un vieux rêve de l'industrie informatique, que l'on peut faire remonter à l'introduction du COBOL dans les années 1950. Grâce à l'utilisation de grands modèles linguistiques, nous nous rapprochons de plus en plus de cette vision. GitHub Copilot exploite cette capacité pour produire du code.

Lorsque nous avons commencé à utiliser GitHub Copilot et GitHub Copilot Chat, ils nous servaient à produire du code en ligne à partir de commentaires et à poser des questions sur le code, les langages, les procédures ou les cadres de développement dans Copilot Chat. Voici un exemple en C# avec la suggestion en gris.

```
//create a method to calculate the simple interest
public double Calculate(double principal, double rate, double time)
{
    if (principal <= 0)
    {
        throw new ArgumentException("Principal must be greater than zero");
    }

    if (rate <= 0)
    {
        throw new ArgumentException("Rate must be greater than zero");
    }

    if (time <= 0)
    {
        throw new ArgumentException("Time must be greater than zero");
    }

    Result = principal * rate * time / 100;
    return Result;
}
```

Voici un exemple des premières utilisations de GitHub Copilot Chat :

The screenshot shows a conversation in GitHub Copilot Chat. A user named **tobiaswittenburg** asks: "How do exceptions in C# work?". GitHub Copilot responds with a link: "1 Verweis verwendet >". The response text is: "In C#, exceptions are a type of object that represents an error or a condition that prevents the normal flow of program execution. They are created and then thrown with the `throw` keyword." Below this, another message says: "Here's a basic example of how to throw and catch an exception:" followed by a code block:

```
try
{
    // Code that could potentially throw an exception
    throw new Exception("An error occurred");
}
catch (Exception ex)
{
    // This block will execute if an exception of type Exception
    Console.WriteLine(ex.Message);
}
finally
{
    // This block will execute regardless of whether an exception
    // It's often used for cleanup code
}
```

In this example, the `try` block contains code that might throw an exception. The `catch` block contains code that will execute if an exception is thrown. The type of the exception caught can be specified as a parameter to the `catch` block. In this case, it's catching `Exception`, which is the base class for all exceptions in C#. The `finally` block contains code that will execute regardless of whether an exception was thrown.

You can also define your own custom exceptions by creating a new class that inherits from `Exception` or one of its subclasses.

GitHub Copilot tient toujours compte du contexte. Le contexte inclut le fichier actuellement ouvert, les métadonnées (par exemple, le langage de programmation) ainsi que les onglets voisins.

Cependant, il est aussi possible d'améliorer le contexte en ajoutant des informations supplémentaires à nos invites. L'idée est de faire connaître notre intention à Copilot et de lui donner des précisions sur le résultat attendu.

Voici un exemple d'invite pour Copilot :

```
I am going to build a website for trainers that can be used during IT Workshops. This website should have a couple of smaller tools that you can pull up in between the sessions.
```

The website is being built in React with JavaScript.

The first component that I want to build is a timer for a coffee break. Please help me building that timer. Here are the features:

- Customizable duration for a break
- Some buttons for a default break duration (5 min, 10 min, 15 min, 45 min, 1 hour)
- A Button «Timer Start»
- Upon clicking «Timer Start» the timer should start and count down.
- When the timer is up an alarm should go off.

Can you build that component for me?

Dans ce cas, nous indiquons à Copilot ce que nous voulons faire (« Construire un site Web que les formateurs pourront utiliser pendant les ateliers informatiques ») et la technologie utilisée (« Ce site web est construit en React avec JavaScript »). Après cela, nous indiquons nos exigences à Copilot et nous obtenons un composant entièrement fonctionnel en React, que nous pouvons ajouter à notre page.

Cependant, il peut arriver qu'une fonctionnalité manque dans le composant : le bouton de démarrage qui devient un bouton d'arrêt lorsque la minuterie fonctionne et vice versa.

Avec une invite supplémentaire, nous pouvons demander à Copilot d'effectuer cette mise en œuvre :

Please toggle the button to «Timer Stop» after the timer has started.
Also change the JavaScript code to stop the timer once the «timer Stop»
Button has been pushed.

Un autre moyen efficace d'utiliser GitHub Copilot consiste à décrire la fonctionnalité souhaitée directement dans les commentaires de votre code. Cette approche permet d'intégrer des instructions détaillées dans la base de code elle-même, ce qui facilite la génération d'extraits de code pertinents par Copilot.

Voici un exemple :

```
1 // Create a Timer component in React
2 // Features:
3 // - Customizable duration for a break
4 // - Default break durations: 5 min, 10 min, 15 min, 45 min, 1 hour
5 // - "Timer Start" button to initiate countdown
6 // - Countdown triggers an alarm when time is up
7 // - Toggle button to "Timer Stop" when countdown starts and vice versa
8
9 import React, { useState, useEffect } from 'react';
10
11 const TimerComponent = () => {
  const [duration, setDuration] = useState(0);
  const [timer, setTimer] = useState(null);
  const [isRunning, setIsRunning] = useState(false);

  const startTimer = () => {
    if (duration > 0) {
      setTimer(setInterval(() => {
        setDuration(prevDuration => prevDuration - 1);
      }, 1000));
      setIsRunning(true);
    }
  };

  const stopTimer = () => {
    clearInterval(timer);
    setIsRunning(false);
  };

  useEffect(() => {
    if (duration === 0) {
      clearInterval(timer);
      setIsRunning(false);
      alert('Time is up!');
    }
  }, [duration]);

  return (
    <div>
```

Dans cet exemple, les commentaires sont utilisés pour décrire l'intégralité de la fonctionnalité du composant Minuterie. Lorsque GitHub Copilot lit ces commentaires, il peut générer une implémentation complète qui correspond aux caractéristiques décrites.

Au-delà de la génération de code, GitHub Copilot peut également être utilisé pour générer des commentaires pour la documentation et fournir des explications sur l'utilisation des bibliothèques. Cette fonctionnalité peut s'avérer particulièrement utile pour comprendre un code complexe ou des bibliothèques mal connues. Dans l'exemple ci-dessous, pour lancer la description de la fonction par Copilot, nous commençons la description et nous laissons travailler l'outil.

```
8 import React, { useState, useEffect } from 'react';
9 // here is a description of how the TimerComponent works:
10 // - The TimerComponent is a functional component that uses the useState and useEffect hooks to manage the state of the timer.
11 const TimerComponent = () => {
12   const [duration, setDuration] = useState(0);
13   const [timer, setTimer] = useState(null);
14   const [isRunning, setIsRunning] = useState(false);
15
16   const startTimer = () => {
17     if (duration > 0) {
```

À partir des commentaires, vous pouvez guider Copilot non seulement pour qu'il génère du code, mais aussi pour qu'il crée une documentation et des explications complètes. En plus d'améliorer la lisibilité et la maintenabilité du code, cela permet de comprendre et d'utiliser plus efficacement les différentes bibliothèques.



Maxim Salnikov

Basé à Oslo, Maxim Salnikov est un passionné de technologie et un membre enthousiaste de la communauté cloud. Avec plus de vingt ans d'expérience en tant que développeur web, il partage ses vastes connaissances de la plateforme web, du cloud computing et de l'IA en donnant des conférences et en dispensant des formations dans le monde entier lors d'événements réunissant des développeurs. Le jour, Maxim contribue de façon importante au développement de solutions cloud et IA au sein des entreprises européennes, en tant que responsable de l'unité productivité des développeurs chez Microsoft. Le soir, il organise parfois des événements pour les plus grandes communautés de développement web et cloud de Norvège. Maxim a pour passion de découvrir et expérimenter les possibilités de l'IA générative, y compris le développement assisté par l'IA.

Conseil n° 16 :

Tirer parti de la syntaxe de conversation : participants au chat, variables de chat et commandes slash

par Maxim Salnikov

GitHub Copilot Chat est une fonctionnalité incroyablement puissante et utile qui permet d'échanger avec ou sur votre code. Même s'il est 100 % compatible avec le langage naturel (autrement dit, vous pouvez envoyer vos messages sans employer une syntaxe spécifique), l'exploitation de certaines fonctionnalités de chat spéciales peut débloquer de nouveaux scénarios de développement assistés par l'IA et accroître considérablement votre productivité.

Ces fonctionnalités puissantes, utilisables en appliquant une syntaxe spéciale, incluent les participants au chat, les commandes slash et les variables contextuelles. Notez que les fonctionnalités décrites sont disponibles dans VS Code mais peuvent ne pas être entièrement prises en charge dans d'autres IDE où GitHub Copilot Chat est disponible.

Ciblez votre question ou votre demande en envoyant votre message à l'un des participants au chat disponibles

Dans GitHub Copilot Chat, vous pouvez vous référer à l'un des « experts du domaine » alimentés par l'IA en utilisant la syntaxe de conversation conventionnelle, autrement dit en faisant précéder le nom du participant du préfixe @. Les participants au chat actuellement disponibles sont les suivants :

- **@workspace** : connaît parfaitement le code présent dans l'espace de travail actuellement ouvert. C'est probablement le participant au chat avec lequel vous communiquerez le plus fréquemment.

- **@terminal** : sait tout sur l'interpréteur de commandes intégré, son contenu et sa mémoire tampon.
- **@vscode** : connaît l'éditeur VS Code, ses commandes et ses fonctionnalités.

Exemple : demandons des informations sur la partie back-end du projet auquel nous venons d'être affectés en interrogeant le participant au chat @workspace immédiatement après l'ouverture du dossier du projet dans VS Code.

webmaxru
@workspace what kind of backend is in this project? and how to run it?

Workspace
Used 3 references >

This project uses Azure Functions for the backend, which can be run either locally or from an external server.

To run the backend locally, follow these steps as described in the [README.md](#) :

1. Install Python 3+, Azure CLI, and Azure Functions Core Tools.
2. Create a new resource group in your Azure account with the required services.
3. Prepare your data as described in the README.
4. Rename `local.settings.template.json` to `local.settings.json` in the `app/api/` folder and fill in the values.
5. Run `func start` in the `app/api/` folder to start Azure Functions locally.

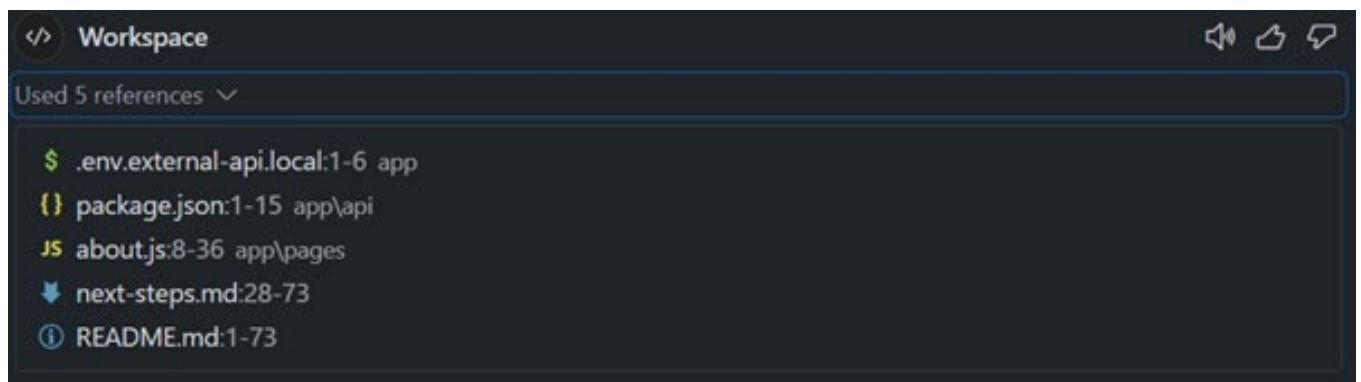
To run the backend from an external server, rename `.env.external-api.local` to `.env.local`. This will use <https://enterprise.promptengineering.rocks/api> as a backend proxied via Rapid API to deal with CORS.

Dans ce cas particulier, vous n'avez même pas besoin d'ouvrir les fichiers dans votre éditeur. Comparez cette réponse à la réponse obtenue sans taguer @workspace :

webmaxru
what kind of backend is in this project? and how to run it?

Github Copilot
As an AI, I don't have direct access to your files or project structure. However, you can typically determine the backend technology used in a project by looking at the project's file structure and contents.

Le participant au chat @workspace joue un rôle déterminant pour toutes les requêtes qui portent sur l'ensemble de la solution et pour lesquelles vous souhaitez que l'ensemble du code soit pris en compte pour la réponse du chat. Toutefois, cela ne signifie pas que tout le code sera utilisé et envoyé dans le cadre de l'invite. L'extension GitHub Copilot Chat dans VS Code fait son possible pour identifier les fichiers et parties de fichiers pertinents en utilisant d'abord les connaissances et les renseignements locaux. Vous pouvez vérifier quels fichiers et lignes de code ont été utilisés pour l'invite en développant la ligne « Références utilisées » :



The screenshot shows the GitHub Copilot Chat interface with a dark theme. At the top, there's a header bar with icons for workspace, volume, and message. Below it, a dropdown menu says "Used 5 references". The main area displays a list of files and their line ranges used in the current message:

- \$.env.external-api.local:1-6 app
- { } package.json:1-15 app\api
- JS about.js:8-36 app\pages
- next-steps.md:28-73
- README.md:1-73

Conseil pour gagner en productivité : Utilisez Ctrl-Entrée (Cmd-Entrée) et pas simplement Entrée une fois que vous avez saisi votre message, et la chaîne @workspace sera automatiquement insérée dans votre message avant envoi.

Définissez précisément le contexte à l'aide des variables de conversation

Dans de nombreux cas, prendre en compte l'intégralité de la solution comme contexte de votre question ou de votre demande (en utilisant @workspace) est exagéré. Vous souhaiterez peut-être pointer vers des fichiers spécifiques ou même des parties de fichiers dans votre message. Les variables de chat peuvent vous aider ! Utilisez # pour appeler un élément de cette liste :

- **#file** : pointe vers un fichier spécifique de votre espace de travail.
- **#codebase** : tout le contenu de l'espace de travail ouvert. Son utilisation est similaire à celle de @workspace et peut être utile lorsque vous discutez avec un autre agent (comme @terminal) mais que vous souhaitez toujours faire référence à la solution complète.
- **#editor** : code source dans la fenêtre de l'éditeur (partie visible).
- **#git** : dépôt git actuel : branche, dépôts à distance, chemin, etc.
- **#selection** : code actuellement sélectionné.
- **#terminalLastCommand** : dernière commande exécutée dans le terminal de l'éditeur.
- **#terminalSelection** : sélection dans le terminal de l'éditeur.

Exemple : Faisons-nous aider pour améliorer les noms de méthodes dans un fichier spécifique (nous voulons aussi nous assurer que l'ensemble du contenu du fichier est pris en compte).



webmaxru

suggest consistent naming of the methods in `#file:index.js`

GitHub Copilot

Used 2 references >

Here are some suggestions for consistent naming of the methods in your `index.js` file:

1. `parseStringTemplate` to `parseTemplateString`: This makes it clear that the function is parsing a template string, not a string template.
2. `getChatHistoryAsText` to `convertChatHistoryToText`: This makes it clear that the function is converting the chat history into text format.
3. `removeNewLines` to `removeNewlineCharacters`: This makes it clear that the function is removing newline characters from a string.

Remember, the goal of naming functions is to make it clear what they do. The names should be descriptive and follow a consistent pattern.

Conseil pour gagner en productivité : Après avoir saisi #, utilisez les flèches haut et bas du clavier pour sélectionner la variable de chat dont vous avez besoin. Dans le cas de #file, utilisez à nouveau la navigation au clavier pour choisir l'un des fichiers proposés.

Appelez rapidement les actions les plus fréquemment utilisées à l'aide de commandes slash

Il est certes amusant de discuter avec votre code en langage naturel, mais la possibilité d'appeler des actions fréquemment utilisées à l'aide de raccourcis est encore plus pratique. Entre saisir l'intégralité du message « Expliquez le fonctionnement du code sélectionné » et taper « / » puis sélectionner « /explain » dans la fenêtre contextuelle à l'aide des flèches du clavier, le choix est vite fait ! La certitude que GitHub Copilot comprend parfaitement notre intention est un autre avantage de l'utilisation de la syntaxe pré définie pour les commandes (le langage naturel peut présenter une certaine ambiguïté). De nombreuses commandes slash sont disponibles. Vous pouvez les utiliser conjointement avec le référencement du participant au chat pour définir le périmètre souhaité. Voici quelques-unes des commandes disponibles :

- **/help** : fournir une aide sur les commandes slash disponibles, les participants au chat, les variables de chat, etc.
- **/doc** : générer de la documentation pour le code.
- **/explain** : expliquer comment fonctionne le code (ou obtenir de l'aide pour les commandes de terminal en ajoutant @terminal).
- **/fix** : optimiser le code et/ou corriger les problèmes dans le code.
- **/tests** : créer des tests unitaires pour le code.
- **/new** : échafauder un nouvel espace de travail.

Exemple : obtenons une explication pour l'une des expressions régulières de notre code. Sélectionnez la ligne de code et utilisez la commande « /explain ».

The screenshot shows a GitHub Copilot Chat interface. At the top, it says "webmaxru" and has a message "@workspace /explain". Below this, it says "Workspace" and "Used 1 reference >". A detailed explanation follows:

The code you've selected is a regular expression in JavaScript. Regular expressions, or regex, are a powerful tool for manipulating strings. They are used to find, replace, and match patterns within strings.

This particular regular expression `/{{\s?([^\{\}\s]*)\s?}}/g` can be broken down as follows:

- `\{\{` and `\}}`: These are the literal characters that the regex will match. In this case, it's looking for strings that have `\{\{` and `\}}` in them.
- `\s?`: This matches any whitespace character (spaces, tabs, line breaks). The `?` makes it optional, meaning it will match zero or one of the preceding element.
- `([^{\}\s]*):` This is a capture group, which means the part of the string it matches will be 'captured' for later use. The `[^{\}\s]` part is a character set that matches any character that is not a `{`, `}`, or whitespace. The `*` means it will match zero or more of the preceding element.
- `/g`: This is a flag that makes the regex global, which means it will find all matches in the string, not just the first one.

So, in summary, this regular expression will match and capture all instances in a string where there are characters (excluding `{`, `}`, or whitespace) enclosed in `\{\{` and `\}}`, with optional whitespace after the opening `\{\{` and before the closing `\}}`.

Conseil pour gagner en productivité : Essayez GitHub Copilot Chat en mode inline au lieu de garder le chat ouvert dans le volet latéral. Appuyez sur Ctrl-I (Cmd-I) et tapez votre message dans la petite boîte de dialogue qui s'affiche en superposition juste au-dessus de la ligne où se trouve votre curseur dans la fenêtre de code.

Résumé

Utilisez les participants au chat, les variables de chat et les commandes slash pour garder un contrôle total sur le contexte de la conversation, assurer une compréhension correcte et cohérente de vos intentions et, en fin de compte, discuter et coder plus rapidement !

Utilisation de GitHub Copilot pour vos projets

Ces 16 conseils de nos experts donnent un excellent aperçu des possibilités fascinantes offertes par GitHub Copilot. Toutefois, les exemples présentés n'illustrent qu'une fraction du potentiel de cet outil innovant qui va révolutionner vos processus de développement.

Pour éprouver la puissance de GitHub Copilot, nous vous recommandons de tester vous-même ce « programmeur pair » assisté par l'IA Son intégration dans votre workflow pourrait faire une vraie différence, en accélérant la réalisation de vos tâches courantes ou en suscitant de nouvelles approches de développement.

Nous sommes impatients de connaître votre avis sur GitHub Copilot, ainsi que vos conseils d'utilisation. Nous vous invitons à nous communiquer vos propres découvertes et recommandations. Ces précieuses informations pourraient intégrer la prochaine édition de notre livre blanc et ainsi aider d'autres développeurs désireux d'exploiter tout le potentiel de GitHub Copilot. N'hésitez pas à nous écrire à l'adresse suivante : techwiese@microsoft.com.

L'innovation étant au cœur du développement de logiciels, GitHub Copilot est sans aucun doute un enrichissement révolutionnaire. Saisissez cette occasion d'améliorer votre expérience de codage et contribuons ensemble à façonner l'avenir du développement de logiciels

Lancez-vous dès aujourd'hui

Si vous souhaitez en savoir plus dès aujourd'hui sur la plateforme Azure Cloud, n'attendez plus :



Contactez votre gestionnaire de compte Microsoft

[Contacter le service commercial Azure | Microsoft Azure](#)



Trouvez un partenaire Microsoft expert

[Trouver un partenaire Azure expérimenté | Microsoft Azure](#)



Informations complémentaires sur la productivité des développeurs

[Productivité des développeurs Azure | Microsoft Azure](#)

Ressources complémentaires

- [Fonctionnalités de GitHub Copilot](#)
- [Documentation GitHub Copilot](#)
- [Démarrage rapide pour GitHub Copilot](#)
- [GitHub Copilot dans VS Code](#)
- [GitHub Copilot Labs](#)
- [Notions de base de GitHub Copilot : comprendre le programmeur de pair IA](#)
- [Centre de gestion de la confidentialité GitHub Copilot](#)
- [Blog GitHub](#)

©2024 Microsoft Corporation. Tous droits réservés. Le présent document est fourni « en l'état ».

Les informations et les points de vue exprimés dans ce document et dans les URL ou autres références de sites Web peuvent être modifiés sans préavis. Vous assumez les éventuels risques associés à leur utilisation.

Ce document ne vous fournit aucun droit légal sur une quelconque propriété intellectuelle concernant les produits Microsoft. Vous pouvez le copier et l'utiliser pour votre usage personnel.

