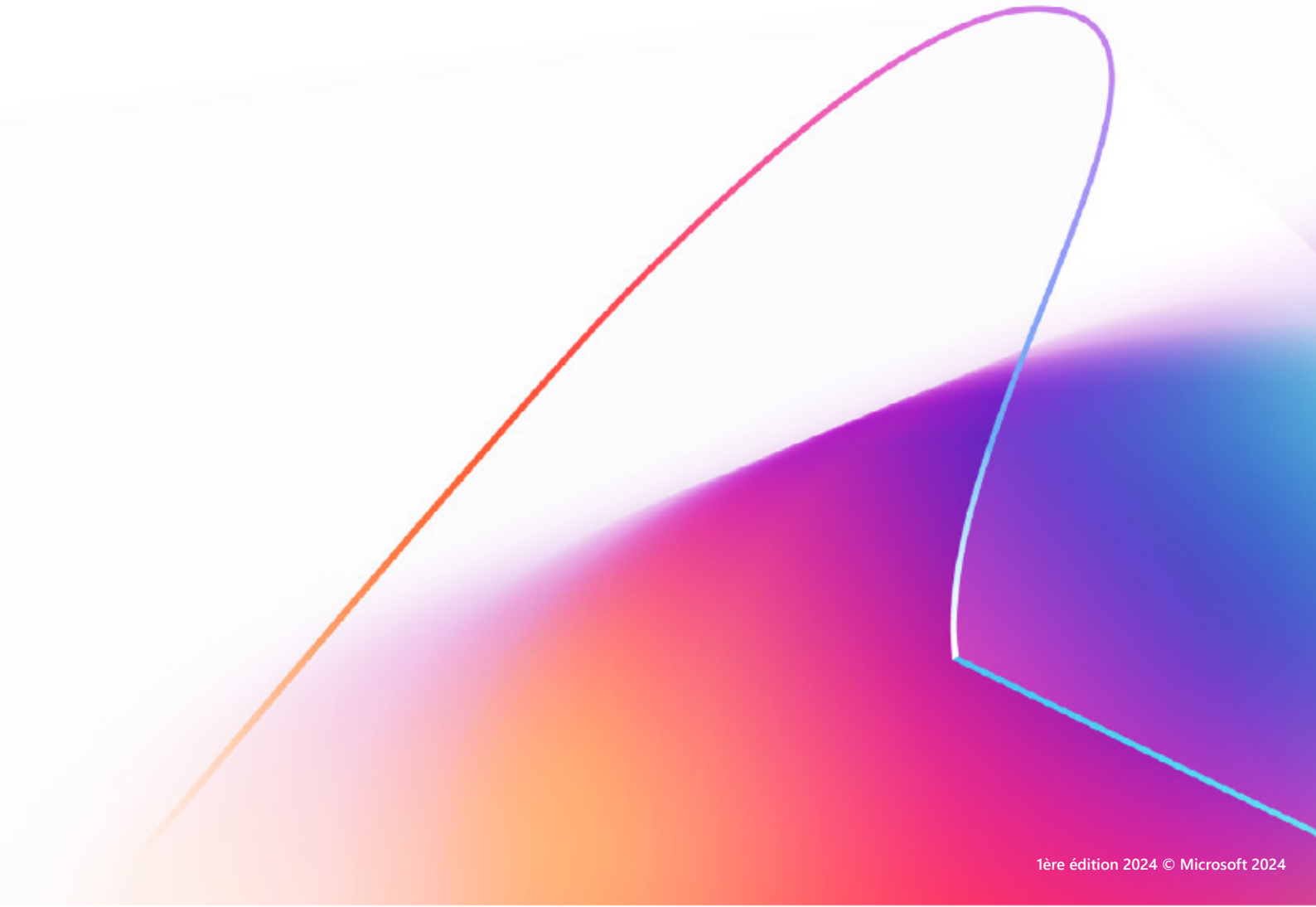
Microsoft Azure

GitHub Copilot

Conseils et astuces (Vol. 1)

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 notamment 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 avait écrit un article consacré au changement de culture concernant GitHub. Il est entre-temps devenu le PDG de GitHub et 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, Satya Nadella a réitéré ses propos dans des termes similaires. 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@micrsoft.com.
Nous en serions ravis.

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

Bien cordialement, Dennis Gassen

Malte Lantin

Malte, 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

De Malte Lantin

3 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 notamment rappeler que beaucoup considéraient encore récemment que tout ceci n'était que pure fiction. GitHub Copilot représente un véritable tournant dans le développement des outils de développement de logiciels.

Grâce aux améliorations permanentes, notamment au niveau de l'utilisation professionnelle des fonctionnalités métier et en s'appuyant sur les plus récents modèles d'IA des services Azure OpenAI, GitHub Copilot s'est déjà hissé au rang des outils de référence. 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 nombreux scénarios d'applications, ainsi que des conseils et astuces concrets.

Innovations et fonctions technologiques

GitHub Copilot bénéficie de nombreuses années de recherches dans le domaine des modèles de langage, de leur déploiement sécurisé dans le service Azure OpenAI 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, Neovim et les éditeurs de JetBrains, 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, 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 complexes, d'obtenir des recommandations complètes, de récupérer des commentaires sur le code, de générer des tests ou de créer des environnements de travail. 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 IA de langage basées sur les conversations. GitHub Copilot Chat peut se référer aux informations extraites des fichiers en cours de traitement afin de proposer en permanence des suggestions pertinentes. Toutefois, la prochaine vague d'innovations a déjà été annoncée.

À l'avenir, GitHub Copilot va non seulement s'améliorer grâce aux nouveaux modèles de langage, mais aussi gagner en évolutivité grâce aux intégrations et devrait être également disponible dans la ligne de commande. Par ailleurs, l'univers GitHub comprend la version GitHub Copilot Enterprise, présentée pour la première fois en novembre 2023. Celle-ci offre aux clients professionnels une intégration approfondie de GitHub Copilot avec son développement grâce à une intégration de GitHub Copilot Chat dans GitHub Enterprise, au traitement assisté par l'IA des demandes de tirage (pull request) et à l'ajustement des modèles d'IA à la base du code propre de l'entreprise.

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 d'un million de développeurs de logiciels utilisent GitHub Copilot et plus de 20 000 organisations ont choisi de mettre en place cette technologie.

L'impact de GitHub Copilot sur la productivité des développeurs est aussi bien quantifiable que considérable. 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 presque plus important encore de souligner le fait 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 revues 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 des développeurs. 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 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 transforms GitHub into the AI-powered developer platform
- The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot
- Quantifying GitHub Copilot's impact on developer productivity and happiness
- Quantifying GitHub Copilot's impact on code quality

Contenu

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 productivité du 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 Python, JavaScript, TypeScript, Go et Ruby, entre autres. Plus de 37 000 entreprises utilisent déjà l'assistant IA, dont un tiers des entreprises du classement Fortune 500. GitHub Copilot se base entre autres sur le modèle LLM Codex d'OpenAI optimisé pour le développement de code de programmation.
Par ailleurs, GitHub Copilot prend en charge GPT-4 d'OpenAI.

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 poursuivre votre travail de façon novatrice.
GitHub Copilot permet également de générer des fonctions et des diagrammes proposés aux développeurs sous forme d'invite dans la saisie correspondante. 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 dans la mesure du possible. 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 de suggestions basées sur l'IA en temps réel pour le développement de code. Pendant que vous écrivez des lignes de code, GitHub Copilot analyse le code et présente automatiquement des suggestions pour le compléter. Il en découle une parfaite harmonie entre le développeur et l'IA. Ce faisant, GitHub Copilot conserve en permanence un aperçu du code complet, suggère des descriptions et aide les testeurs à comprendre les modifications. Les développeurs peuvent également créer des messages commit automatisés à l'aide de GitHub Copilot. Il existe de nombreuses applications capables de faciliter considérablement la réalisation des tâches quotidiennes 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 et Azure Data Studio, par exemple) via des extensions. GitHub Copilot vous permet également d'obtenir de l'aide au niveau de l'interface de ligne de commande (CLI). Par ailleurs, GitHub Copilot sera bientôt disponible pour GitHub Mobile, ce qui permettra aux développeurs de l'utiliser sur leur smartphone ou tablette comme ils le feraient sur leur ordinateur de bureau.

GitHub Copilot : 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 tout comme 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. Toutes les versions ont un point en commun : la possibilité de convertir le langage naturel en code et donc directement dans l'environnement de développement intégré (IDE).

GitHub Copilot Business s'adresse avant tout aux entreprises. Pour 19 dollars par utilisateur, les développeurs de l'entreprise ont accès à GitHub Copilot. Cet abonnement comprend la fourniture de compléments de code, la fonctionnalité de conversation avec GitHub Copilot via l'IDE et GitHub Mobile (disponible à partir de 2024), 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 est directement personnalisé pour l'entreprise et offre en plus des fonctionnalités de conversation personnalisée pour les développeurs, de recherche documentaire et de résumés. En outre, l'entreprise a accès à des modèles IA finement ajustés, offrant notamment la possibilité de vérifier le code.

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 n'est pas disponible pour les étudiants, les enseignants ou les responsables de formation.

Copilot ne copie aucun code. Il le crée.

En général, aucune version de GitHub Copilot ne copie le code directement. GitHub Copilot génère du code à partir des calculs des probabilités. Bien que les modèles utilisés par GitHub Copilot sont entraînés à l'aide du code issu de dépôts publics disponibles, ils ne contiennent aucun code. Lors de la génération du code, GitHub Copilot utilise le code déjà disponible dans l'IDE, notamment les lignes de code 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.

#Developer

Martin Brandl

Martin (@martin_jib) est le directeur de la technologie et le 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 de 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 beaucoup de saisies.

Ces commandes contextuelles se réfèrent toujours à un agent

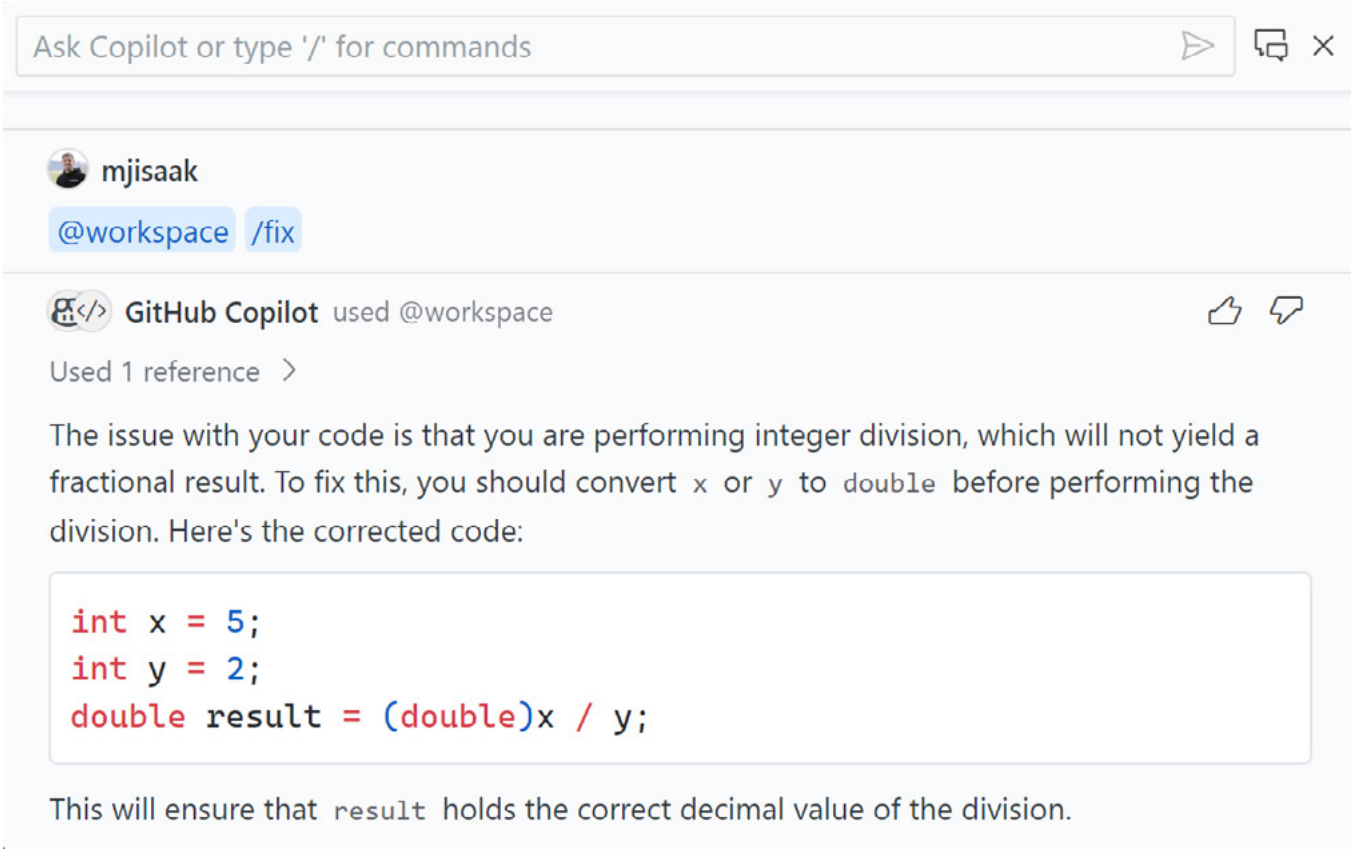
indiqué par le symbole @. Actuellement, il en existe deux :

- **@workspace** dispose des informations de contexte dans 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 la section de code suivante, nous exécutons une division par un nombre entier dans laquelle les décimales sont tronquées. Le résultat est 2,0, au lieu du résultat attendu 2,5 :

```
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 automatiquement associé comme agent :



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 demande l'explication du code sélectionné pas à pas.
- **@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 au moyen 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.

#Developer #ReduceTechnicalDebt
#GitHubCopilotChat

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.

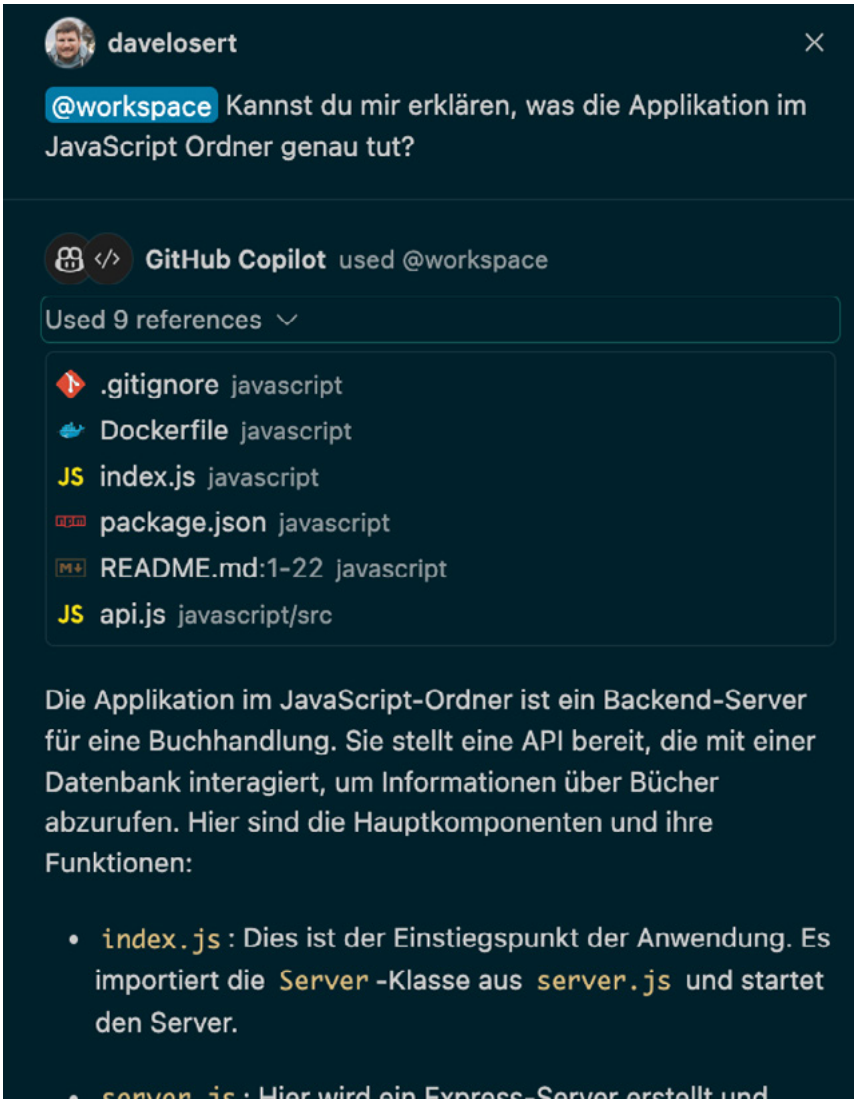
Après plus d'une décennie de diversification technologique, les équipes de nombreuses entreprises aspirent à nouveau à une standardisation et une harmonisation accrues. Cette tendance est notamment le résultat du développement des microservices, de la conteneurisation et de la simplification des activités agnostiques sur le plan technologique dans le cloud public. 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 proue, 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.

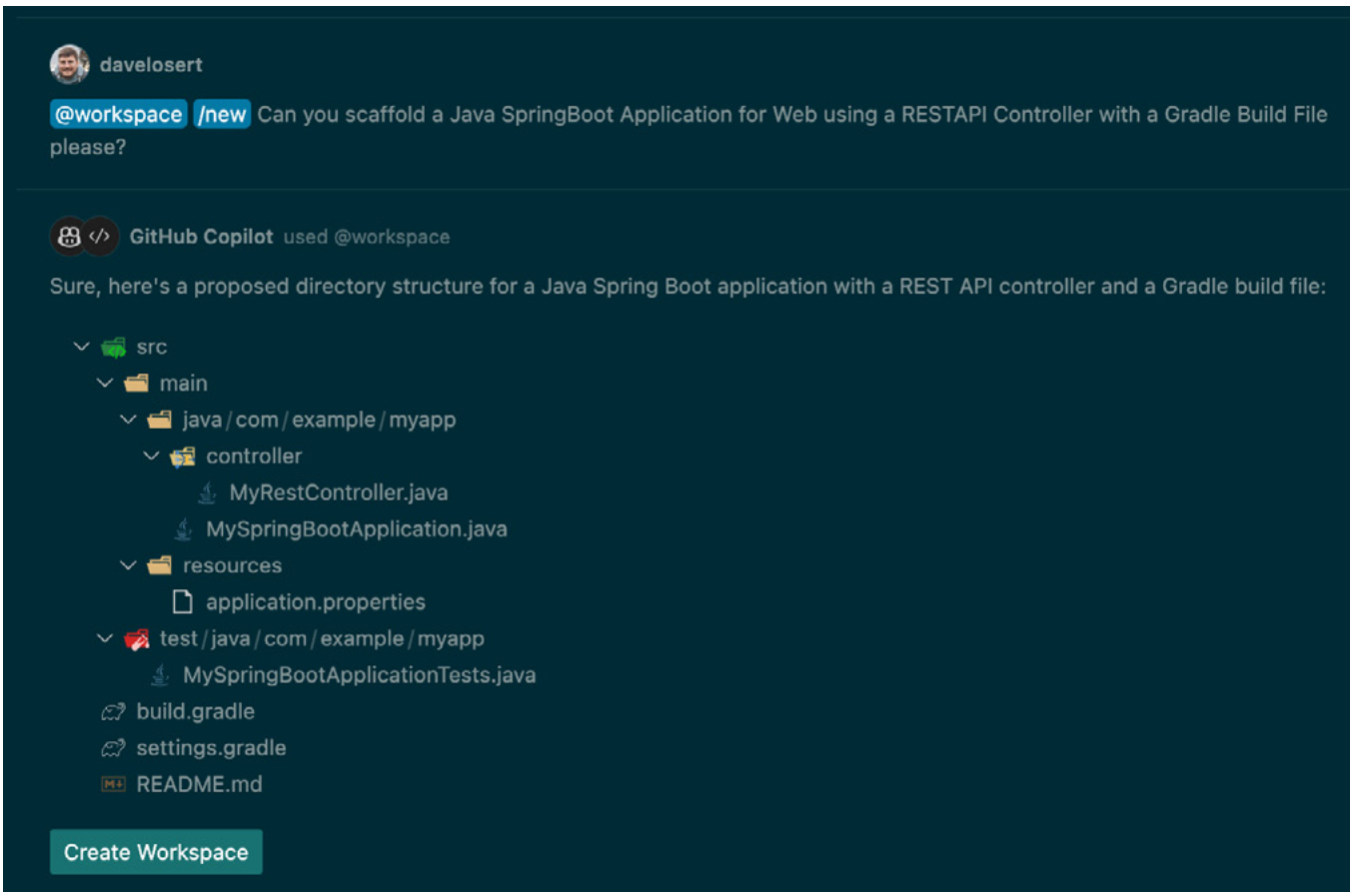
De même, il est recommandé d'avoir une compréhension générale de l'application source pour éviter de réaliser une migration à l'aveugle. GitHub Copilot Chat peut déjà s'avérer très utile avec l'agent @workspace à ce stade.

Cet agent intègre également 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.



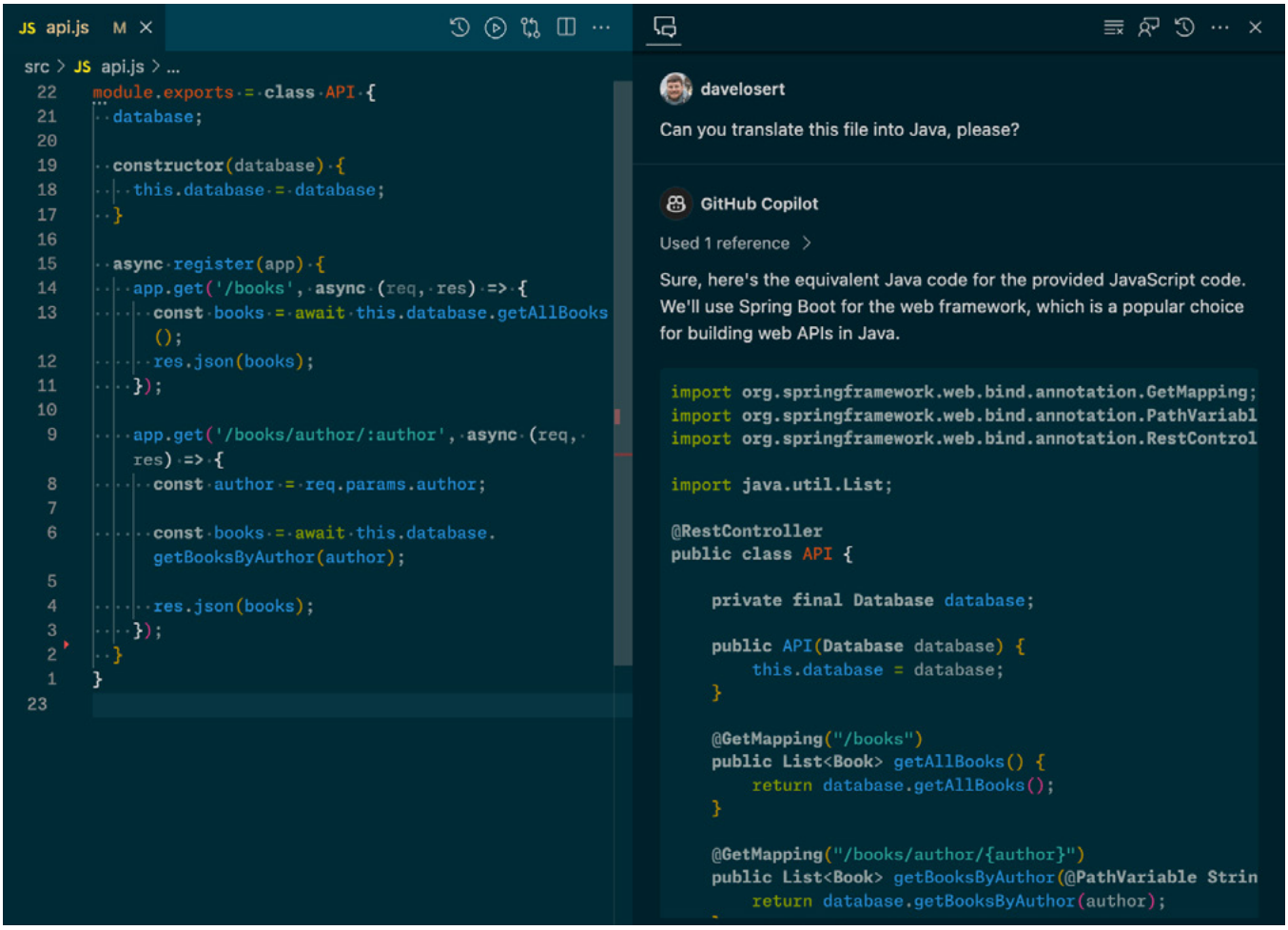
Comme le montre la capture d'écran, il est également possible de communiquer avec GitHub Copilot en allemand. Puisque le code est généralement défini en anglais et afin que l'ensemble du document soit cohérent, les captures d'écran suivantes ont été créées avec des textes en anglais. La pile cible peut également être initialisée avec GitHub Copilot Chat et la commande slash /new.

Migration

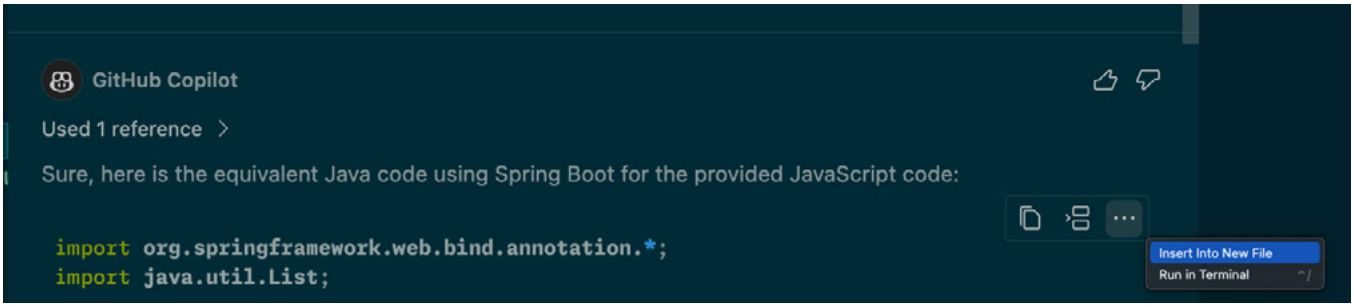


Les développeurs réalisent ensuite la migration directement dans l'IDE. Ce processus est en principe très simple :

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! » (un peu de politesse ne fait pas de mal 😉)



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.

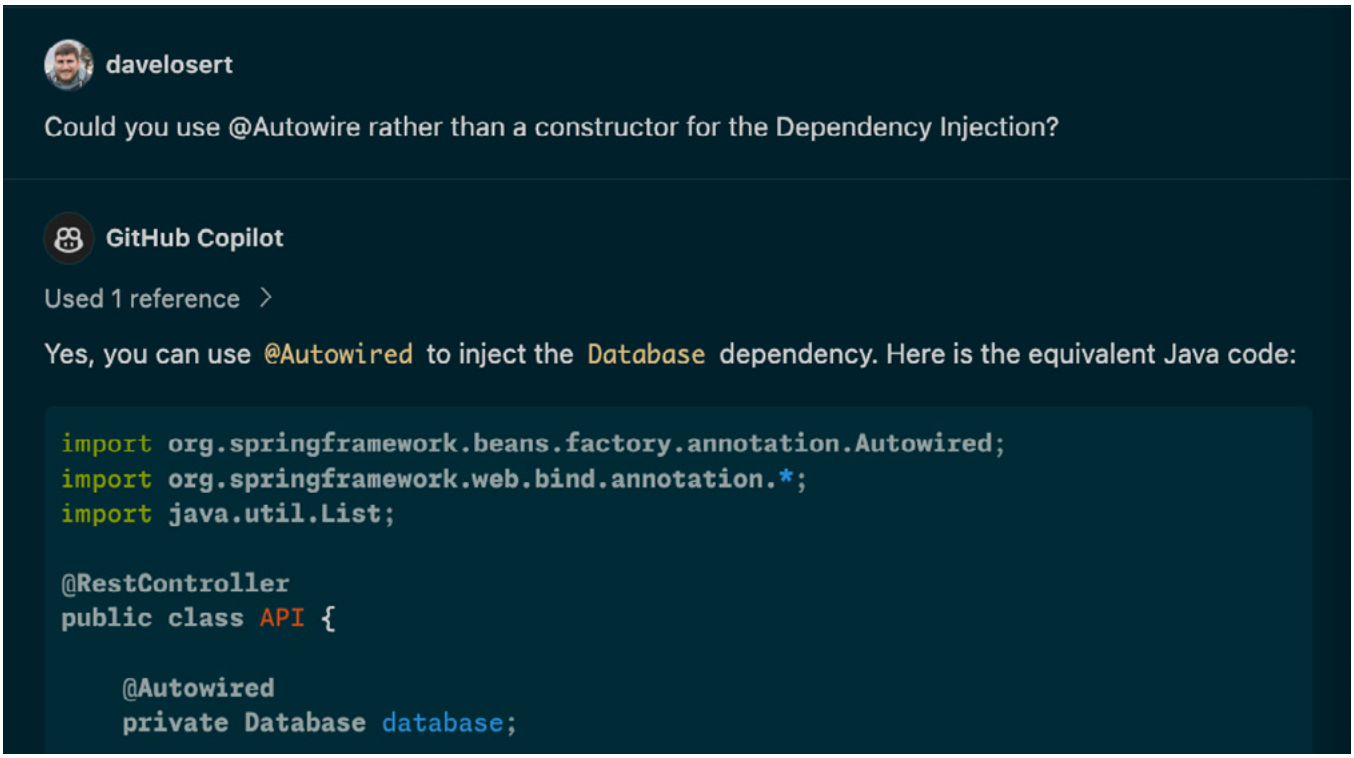


Répétez cette opération pour tous les fichiers du système source pour finaliser la migration. Souvent, tout n'est pas si simple et vous devrez surveiller certains facteurs. Nous y reviendrons en détail par la suite.

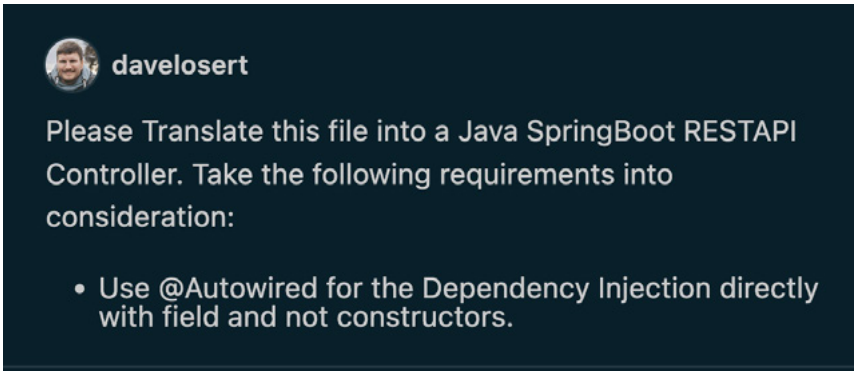
Revue et ajustements

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 :



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 :



Ainsi, vous obtenez toujours au cours de la migration de meilleures conversions de GitHub Copilot dès la première invite. Un conseil : avec la touche « Flèche vers le haut », vous pouvez aller chercher 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 :

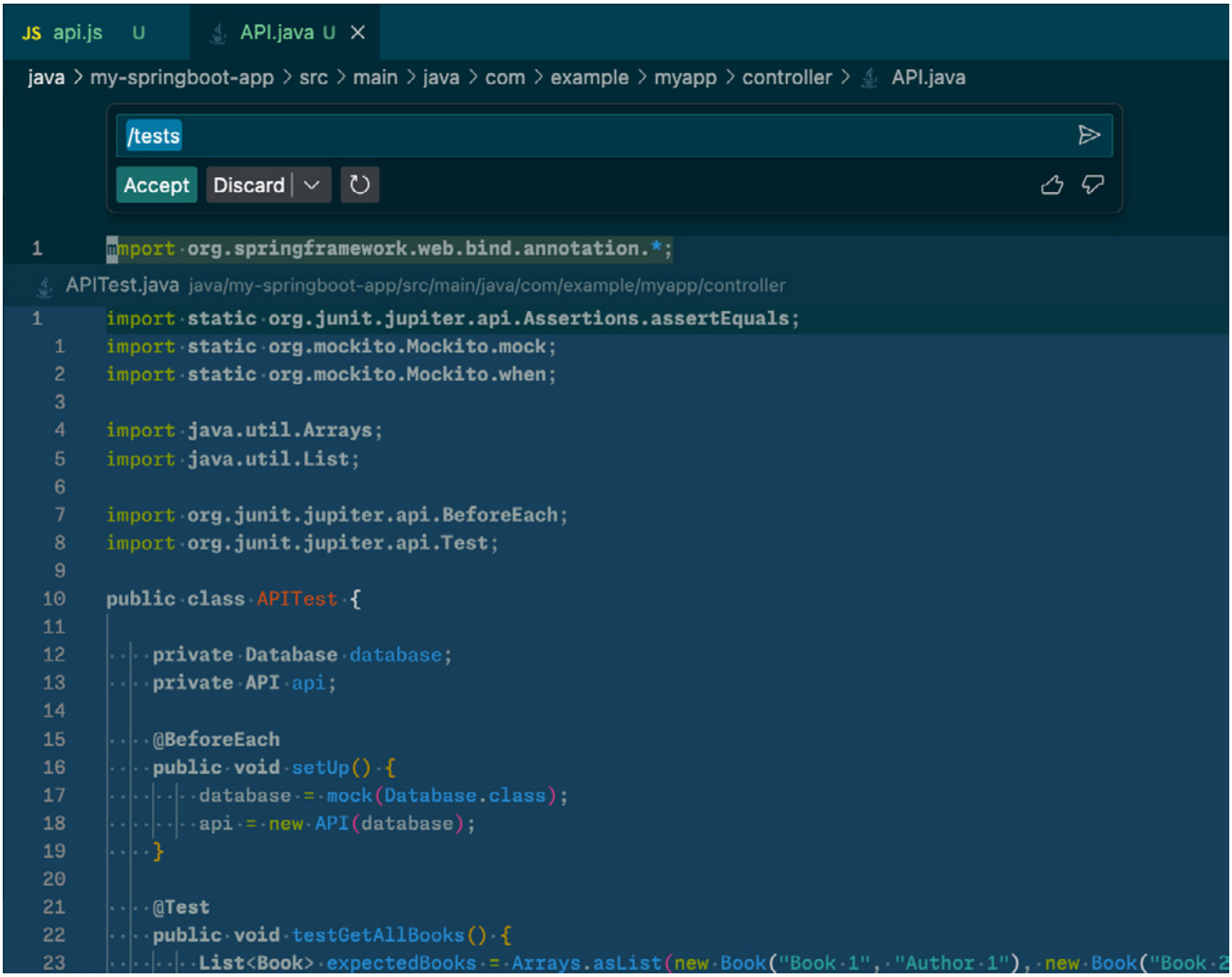
Stratégie de migration : diviser pour mieux régner

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. Évitez donc les changements de contexte et une sollicitation excessive en réalisant de trop nombreuses modifications et erreurs simultanées.
Vous pourrez ainsi garder plus 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 d'altérer votre concentration.

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 :



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 invites, 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

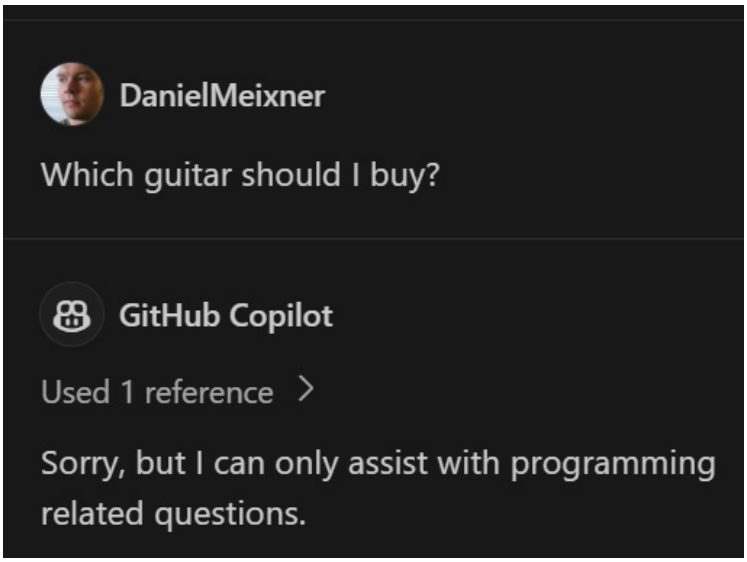
Spécialiste de l'innovation applicative chez Microsoft, Daniel étudie les architectures cloud, les outils de développement, l'IA et les applications intelligentes. Dans le domaine du développement de logiciels, il a appris à reconnaître différentes personnalités sous différents angles et dans différents rôles, des développeurs aux évangélistes technologiques.

Conseil n°3 :

Plus de faux-texte

par Daniel Meixner

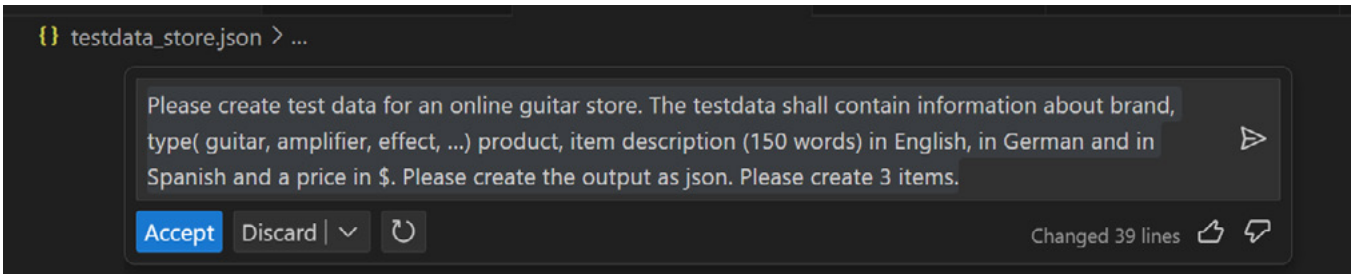
GitHub Copilot étant un outil de développement, il est donc logique que l'assistant IA ne se concentre, de préférence, que sur les réponses relatives au développement. Par exemple, si vous interrogez GitHub Copilot pour obtenir une recommandation d'achat concernant un produit spécifique, l'assistant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



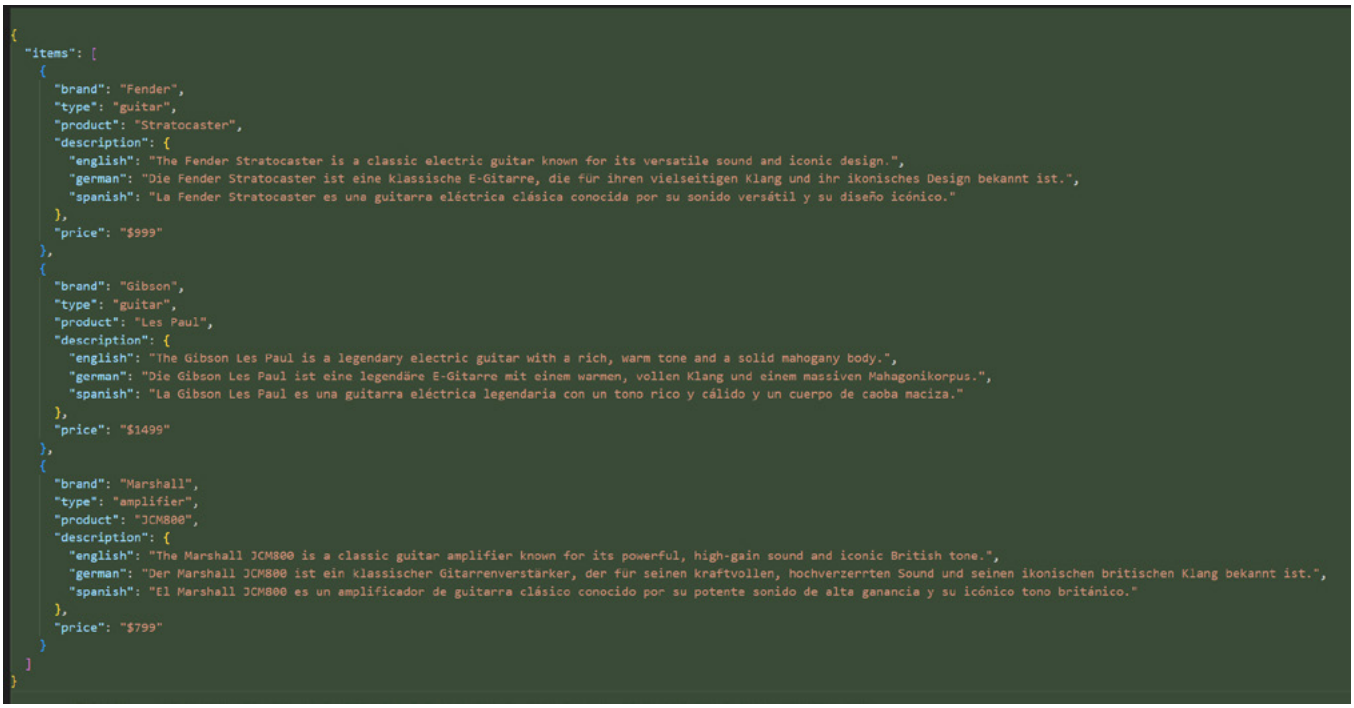
Mais cela ne signifie pas que GitHub Copilot ne peut pas utiliser de connaissances générales de différents domaines en dehors du développement de logiciels. Vous pouvez par exemple utiliser ces connaissances pour générer des données de test pertinentes et appropriées à une application au niveau technique qui nécessitent d'autres 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 d'obtenir souvent des données très utiles. Toutefois, cette approche chronophage exige pas mal de travail. GitHub Copilot vous permet d'utiliser l'IA pour générer très facilement ces données, à la demande et au format JSON, afin de les traiter de façon pertinente.

Dans cet exemple, vous créez un fichier testdata.json et demandez en ligne les données de test pour votre application dans ce fichier à l'aide du raccourci Strg + I. Prenons à présent l'exemple d'une boutique en ligne qui vend des guitares et des accessoires :



Comme vous pouvez le voir sur cette version, cette boutique 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 ou un texte d'exemple inutile. 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.



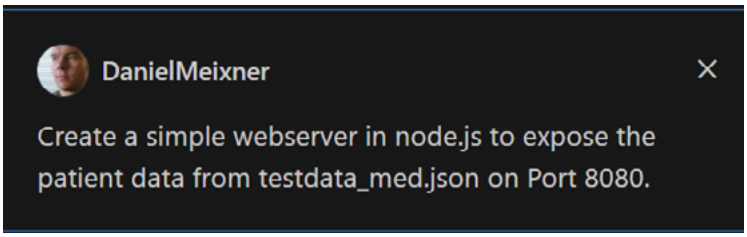
Loin de se limiter à un seul domaine, les connaissances de GitHub Copilot peuvent être également utilisées pour d'autres cas d'utilisation. Supposons que vous travailliez à une application médicale et que vous ayez besoin de données de test de patients. GitHub Copilot peut vous aider à créer des données 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 phsyscian and patient
address (an address in Germany). Please create 5 records.
```

Voici un extrait du résultat :


```json
{
  "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 pouvez apprécier dans cet exemple la capacité de compréhension générale de GitHub Copilot, qui a automatiquement interprété correctement « dob » comme la date de naissance. Lorsque vous souhaitez transmettre des données, vous pouvez demander à GitHub Copilot de créer un serveur Web. Vous pouvez ainsi interroger ensuite les données via une requête HTTP.



#Developer #MVP

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 profils .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 pris en charge (IDE) 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 nouvel outil de développement. C'est une ressource essentielle capable d'améliorer votre productivité et la qualité globale de votre travail.

Dans cet environnement qui évolue très rapidement, 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

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 fonctions d'IA d'Azure, 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. L'accent a été mis sur la mise à disposition sous forme de conteneur ou l'utilisation directe des services IA. Cette intégration représentait un véritable défi à plusieurs niveaux : d'une part, elle nécessitait une expérience approfondie afin de parcourir et d'utiliser efficacement toute l'étendue des fonctions 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 et 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. Ce dernier propose 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 GitHub 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 API 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 code Python ci-dessous montre comment GitHub Copilot a aidé l'équipe de développement à implémenter l'analyse de sentiment avec la classe TextAnalyticsClient 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 = TextA-
nalyticsClient(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 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 GitHub Copilot ont permis d'optimiser la structure et l'implémentation de la fonction d'analyse de sentiment, de la sélection des méthodes appropriées au traitement des langues naturelles à l'amélioration de l'analyse par l'extraction de phrases clés. Cet exemple montre comment GitHub Copilot accélère le codage, tout en l'enrichissant avec des fonctions étendues et de meilleures 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 GitHub 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 davantage se concentrer sur les aspects créatifs de la programmation, au lieu de perdre du temps avec les subtilités de la documentation.

Le doublement de la production de code des équipes ayant utilisé GitHub Copilot est un des résultats impressionnants obtenus grâce à cette étude. 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 GitHub Copilot a été également indéniable, 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.

La réduction de 75 % 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. Grâce à l'accès immédiat aux informations et à l'aperçu du code source, GitHub 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 véritable 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. Alors que nous continuons à nous réjouir des avantages offerts par les outils assistés par l'IA tels que GitHub Copilot pour les tâches quotidiennes des développeurs et à explorer tout leur potentiel, l'avenir du développement de logiciels semble plus prometteur et passionnant que jamais.

#SolutionArchitecture #Demo
#POC #AzureCLI

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 une nouvelle fois 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 ?»

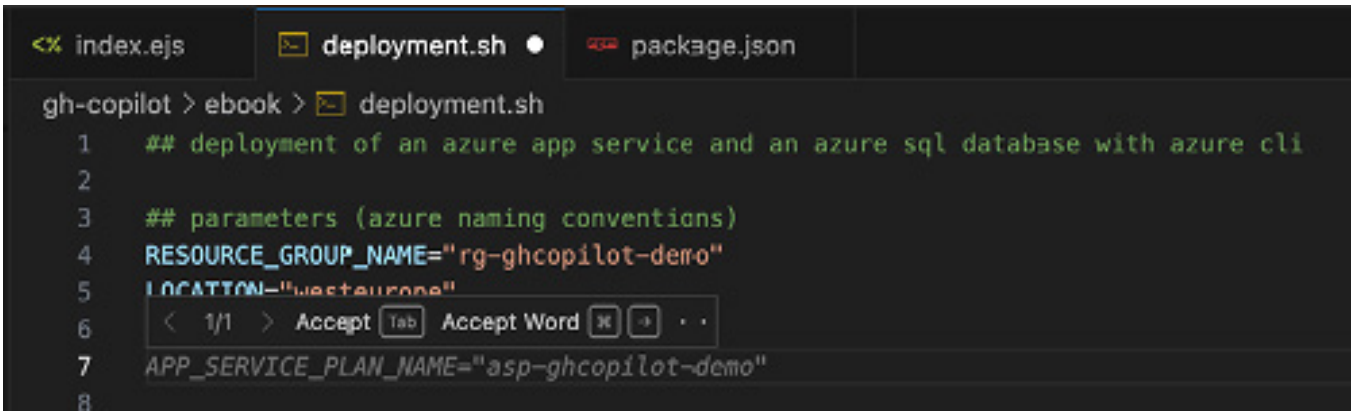
« Bonjour Julia, peux-tu réaliser une démonstration sur un sujet technique chez le client ? »

Ce genre de demandes et d'autres questions similaires me sont posées 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.

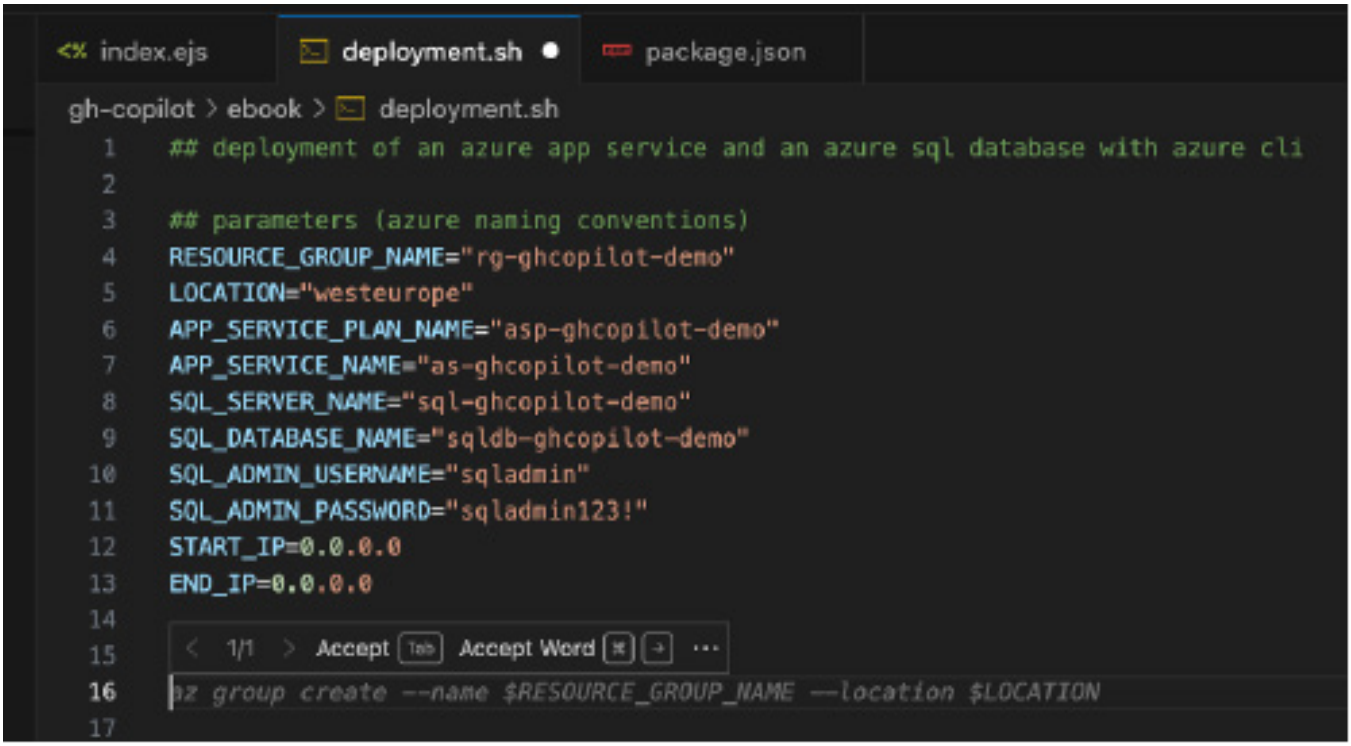
Cet outil m'offre une très grande liberté de travail en termes d'infrastructure en tant que code et de langages de programmation. En effet, bien que je n'aie jamais utilisé .NET dans un cadre professionnel, mes connaissances pro-code issues d'autres langages associées à la puissance inégalée de GitHub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

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 par rapport à la CLI Azure.

Tout d'abord, j'utilise un script *deployment.sh* pour la création, puis je décris rapidement dans un premier commentaire la méthode à suivre 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.



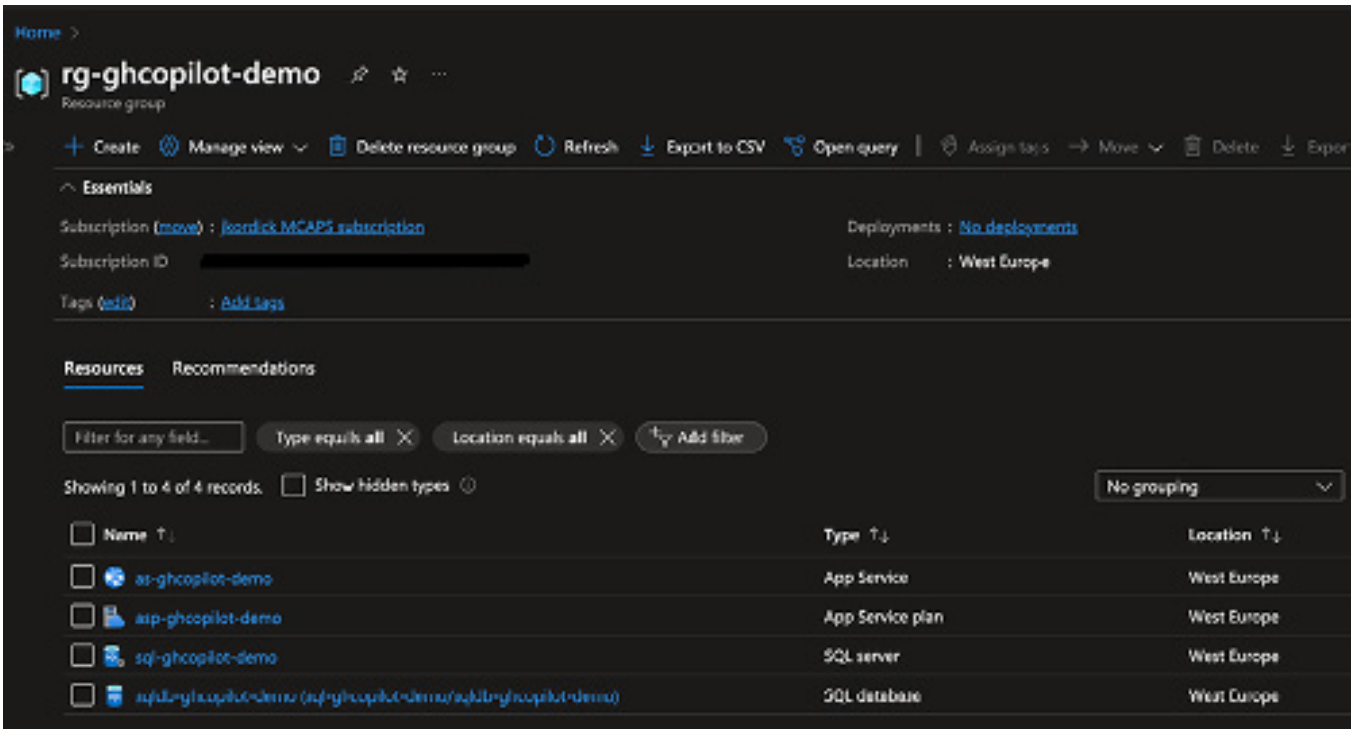
J'utilise ensuite la CLI Azure pour commencer à provisionner les ressources et GitHub Copilot me fait des propositions qui correspondent à mes paramètres.



GitHub Copilot me suggère pas à pas les étapes descriptives 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 chaque suggestion. Je décide de ne pas reprendre complètement les éléments particulièrement importants ou pertinents pour la sécurité (comme la configuration des règles du pare-feu, par exemple) mot pour mot à l'aide des touches *Command et Tab* afin de contrôler les suggestions pour chaque élément l'un après l'autre.

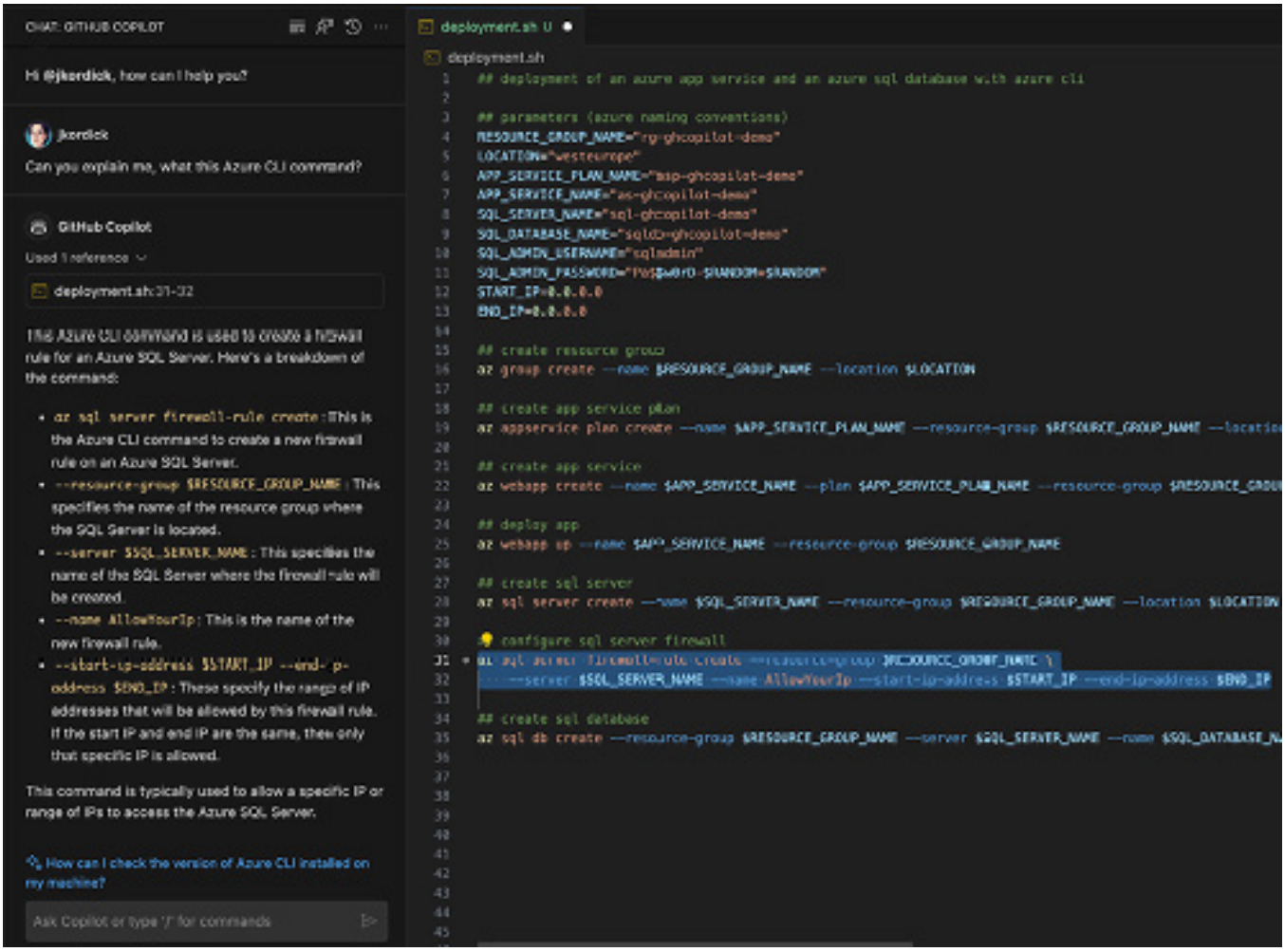
```
## configure sql server firewall
1/1  Accept Tab  Accept Word ⌘ →  ···  source-group $RESOURCE_GROUP_NAME --server $SQL_SERVER_NAME \
    --name AllowYourIp --start-ip-address $START_IP --end-ip-address $END_IP
```

La description des ressources nécessaires dans Azure ne me prend personnellement que quelques minutes. Je lance ensuite le script que je viens de créer et, en l'espace de quelques minutes seulement, je peux voir le résultat sous forme d'édition sur ma console ou sur le portail Azure.



Malgré mes connaissances approfondies d'Azure, je ne connais pas par cœur toutes les commandes de la CLI Azure. Par conséquent, il arrive souvent que GitHub Copilot me propose des suggestions, qui me permettent d'avoir une idée approximative du résultat d'une commande spécifique, mais j'ai besoin d'informations complémentaires plus précises.

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.


CHAT: GITHUB COPILOT
Hi @jkordick, how can I help you?
jkordick
Can you explain me, what this Azure CLI command?
GitHub Copilot
Used 1 reference
deployment.sh:31-32
This Azure CLI command is used to create a firewall rule for an Azure SQL Server. Here's a breakdown of the command:
az sql server firewall-rule create : This is the Azure CLI command to create a new firewall rule on an Azure SQL Server.
--resource-group $RESOURCE_GROUP_NAME : This specifies the name of the resource group where the SQL Server is located.
--server $SQL_SERVER_NAME : This specifies the name of the SQL Server where the firewall rule will be created.
--name AllowYourIp : This is the name of the new firewall rule.
--start-ip-address $START_IP --end-ip-address $END_IP : These specify the range of IP addresses that will be allowed by this firewall rule. If the start IP and end IP are the same, then only that specific IP is allowed.
This command is typically used to allow a specific IP or range of IPs to access the Azure SQL Server.
How can I check the version of Azure CLI installed on my machine?
Ask Copilot or type '/' for commands
deployment.sh U
deployment.sh
deployment of an azure app service and an azure sql database with azure cli
parameters (azure naming conventions)
RESOURCE_GROUP_NAME="rg-ghcopilot-demo"
LOCATION="westeurope"
APP_SERVICE_PLAN_NAME="asp-ghcopilot-demo"
APP_SERVICE_NAME="as-ghcopilot-demo"
SQL_SERVER_NAME="sql-ghcopilot-demo"
SQL_DATABASE_NAME="sqldb-ghcopilot-demo"
SQL_ADMIN_USERNAME="sqladmin"
SQL_ADMIN_PASSWORD="P@$$w0rD-$RANDOM-$RANDOM"
START_IP=0.0.0.0
END_IP=0.0.0.0
create resource group
az group create --name $RESOURCE_GROUP_NAME --location $LOCATION
create app service plan
az appservice plan create --name $APP_SERVICE_PLAN_NAME --resource-group $RESOURCE_GROUP_NAME --locatio
create app service
az webapp create --name $APP_SERVICE_NAME --plan $APP_SERVICE_PLAN_NAME --resource-group $RESOURCE_GROU
deploy app
az webapp up --name $APP_SERVICE_NAME --resource-group $RESOURCE_GROUP_NAME
create sql server
az sql server create --name $SQL_SERVER_NAME --resource-group $RESOURCE_GROUP_NAME --location $LOCATION
configure sql server firewall
az sql server firewall-rule create --resource-group $RESOURCE_GROUP_NAME \
--server $SQL_SERVER_NAME --name AllowYourIp --start-ip-address $START_IP --end-ip-address $END_IP
create sql database
az sql db create --resource-group $RESOURCE_GROUP_NAME --server $SQL_SERVER_NAME --name $SQL_DATABASE_N

#Developer #UnitTests #TypeScript

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

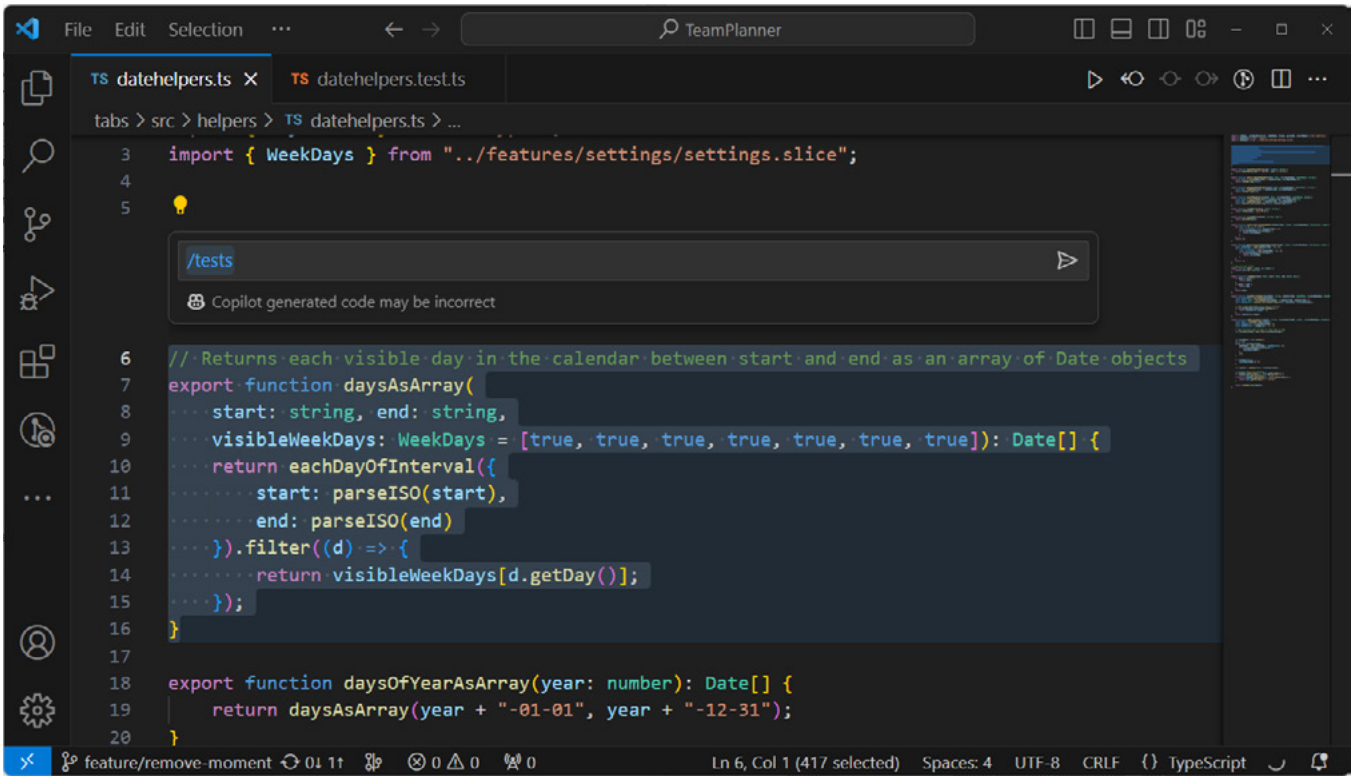
par Thomas Pentenrieder

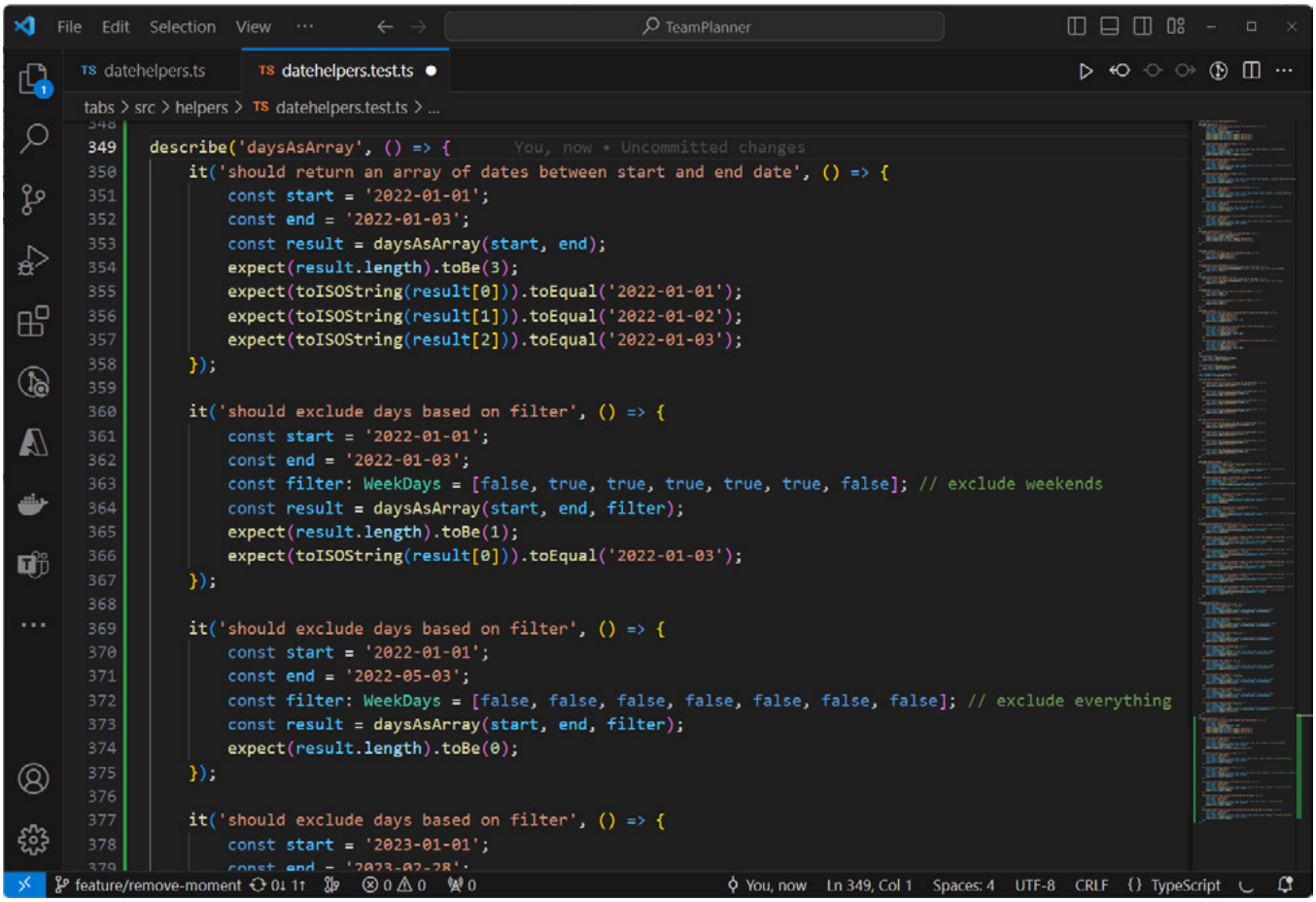
GitHub Copilot peut générer, le cas échéant, des pans entiers de la logique de programmation à partir de commentaires ou d'invites. De mon point de vue, le meilleur atout de cet outil pour les développeurs est sa capacité à faciliter les tâches ennuyeuses (mais néanmoins importantes). Pour ce faire,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

En effet, si des méthodes déjà abouties ont été implémentées, 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 données 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.





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 formuler en conséquence des hypothèses erronées dans les tests.

#DevOpsEngineer #PowerShell #EnterpriseMobility
#DeviceManagement #Analytics #Remediation

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 Technical Lead spécialisé dans l'IA pour les opérations informatiques (AIOps) dans les plus grandes entreprises chimiques internationales. Sur son temps libre, il blogue et intervient lors d'événements.

Conseil n°7 :

Résolution des problèmes proactive 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 à exécuter éventuellement un script de remédiation en fonction de la valeur de retour afin de corriger ces problèmes.

Les scripts sont généralement écrits dans PowerShell et sont interdépendants. C'est là que GitHub Copilot fait la différence. 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 également aux utilisateurs une meilleure expérience au travail.

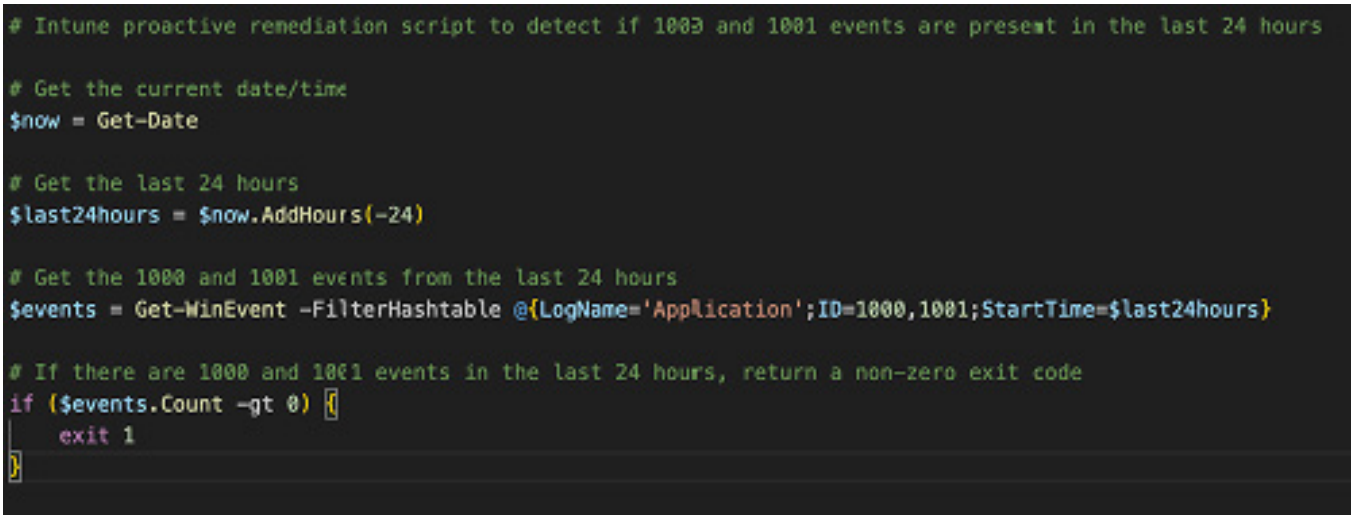
Comme la création de scripts doit s'effectuer en principe dans un délai convenable et rapidement, GitHub Copilot est très utile pour gagner un temps précieux. Fini les recherches fastidieuses de chaque commande, fonction ou ligne de code. Dans bon nombre de cas, il suffit de créer une description détaillée 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, nous écrivons 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.

Il ne vous reste plus qu'à effectuer quelques clics sur la touche « Tab ». GitHub Copilot génère ensuite le script à ma place. Je peux ainsi me concentrer sur ce que doit exécuter la commande pour lire les journaux des événements ou sur la manière dont il est possible de filtrer les données des dernières 24 heures.



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

```powershell
# 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 :

```powershell
# 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*"}

# 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 du résultat que je souhaite obtenir. Voici un exemple :

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 pas encore satisfaisant. J'ai obtenu le même résultat qu'avec le script de détection et je dois le reprendre.

```powershell
# 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 alors de « tâtonner » pour obtenir un script parfait.

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

# Cleare 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
}
```

#Developer #CrossLanguage

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 Quantum, 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 fascinant de pouvoir résoudre des problèmes complexes à l'aide de quelques lignes de code et de développer des solutions innovantes. 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 personnalisées.

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 Java ? Comment distribuer 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 de petits exemples de code et autres 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 besoins du client. Puis, GitHub Copilot convertit automatiquement ma demande dans le code correspondant.

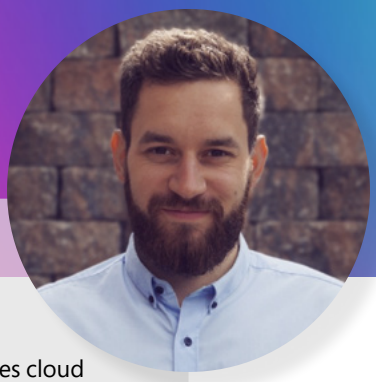
Voici un aide-mémoire pour les commentaires dans les langages de script et de programmation les plus importants à mes yeux :

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

Cette approche a bien évidemment ses limites. Pour créer des solutions logicielles plus importantes, 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 dans ce domaine, GitHub Copilot change la donne.

#Automation #Chat #Developer
#Transformation



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 et 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 ennuyeuses avec GitHub Copilot Chat

par Robin-Manuel Thiel

Dans le domaine du développement de logiciels, les tâches répétitives représentent un défi quotidien que les développeurs rechignent souvent à relever. La conversion de code, notamment lors d'une transition entre différentes technologies, peut s'avérer chronophage et source d'erreurs. C'est là que GitHub Copilot Chat fait toute la différence en aidant les développeurs grâce à l'automatisation des tâches fastidieuses.

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 différents aspects facilitant la structuration d'une invite afin d'améliorer son traitement par l'IA et la précision de la réponse apportée :

- Incluez des exemples.
- Structurez les invites à l'aide de séparateurs et séparez votre question des informations contextuelles.
- Marquez clairement les blocs de code avec des instructions markdown.

Prenons un 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 également marqué tout le code du type énuméré et sollicité GitHub Copilot Chat avec la première invite suivante :

```
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.

En effet, j'aurais pu affiner l'invite initiale en spécifiant une instruction demandant l'utilisation de majuscules ou de minuscules 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 dans le type énuméré.

L'amélioration constante des invites dans de nombreuses itérations aurait probablement pris beaucoup de temps à mettre en œuvre, tout comme 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 spectaculaire a cette fois-ci parfaitement complété l'exemple de script de l'invite avec les valeurs restantes du type énuméré. GitHub Copilot a également tenu compte cette fois-ci de la casse du nom du tableau que je n'avais pas évoquée de manière explicite, mais seulement mentionnée à titre d'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 favorablement 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 numérisation et de l'industrie 4.0. Depuis 2004, il est Microsoft MVP pour les technologies des développeurs.

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 conférence de GitHub Copilot se concentrent 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 des tâches triviales se mélangent parfois aux 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 d'« ingénierie des invites » il y a quelques années de cela. L'article Wikipedia à ce sujet a été créé en octobre 2021, par exemple. 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 brevetée pour 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 manque 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 la qualité de correspondance 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, compréhensible et réalisable. Dans la mesure du possible, évitez les négations et privilégiez les instructions et demandes claires.

2. Le contexte est roi

Une invite commence au mieux 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. Suivent ensuite les différentes fonctionnalités à implémenter.

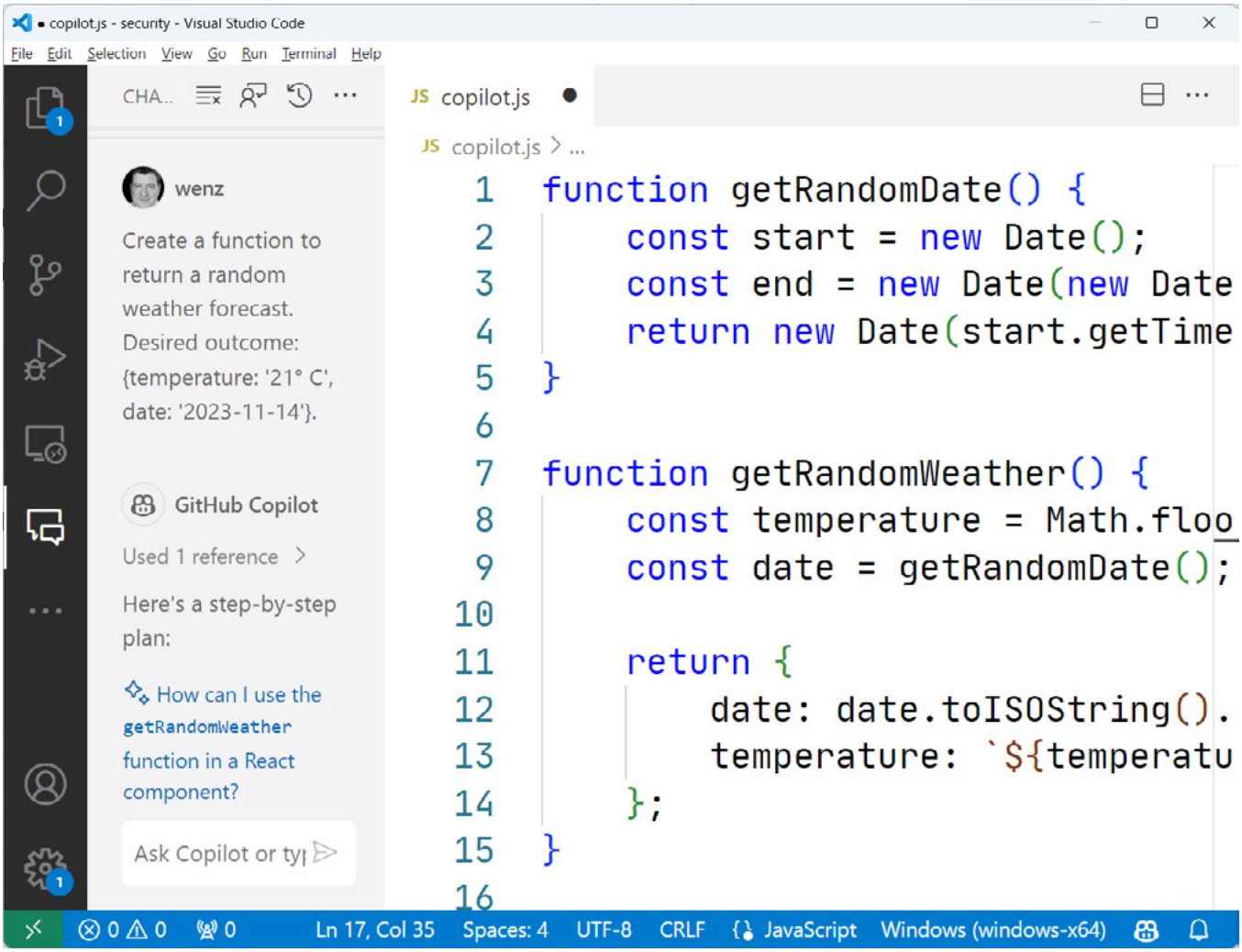
3. Vers le succès, étape par étape

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 souhaité final. Il est toujours possible d'améliorer le code, soit via GitHub Copilot, soit en intervenant vous-même sur ce dernier. 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 de tirets 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 d'ajouter des exemples pour spécifier la version à produire d'une fonctionnalité. À cet effet, la structure des données est également définie comme elle doit être traitée ultérieurement.



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 éventuellement appliqué. Enfin, le code apparaît ensuite rapidement comme si vous l'aviez écrit vous-même.

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 a fait preuve de politesse envers les systèmes IA. Toutefois, tout porte à croire qu'entretenir des rapports courtois avec un système informatique améliore l'atmosphère générale et les interactions personnelles. Soyons honnêtes : une utilisation correcte nous permet d'économiser les saisies superflues de formules de politesse et d'obtenir des résultats générés rapidement, n'est-ce pas ?

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 au moyen de l'exemple de code suivant. Dans la fonction fetch(), nous souhaitons appeler tous les caractères Star Wars via l'API Star Wars :

```
export class ApiService {
  private starWarsUrl = environment.baseUrl;

  constructor(private httpClient: HttpClient) {}

  fetch() {

  }
}
```

Dans ce contexte, on constate 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. 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 au contexte clair.

#CSharp #UnitTest #Developer #XUnit

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 une des tâches chronophages les plus importantes. 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 offre un aperçu 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;
}
```

Afin d'utiliser GitHub Copilot pour la génération de test automatique, je commence par créer le nom de test. 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 paramétré 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(100, 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.

Utilisation de GitHub Copilot pour vos projets

Ces 12 conseils de nos intervenants offrent une vue d'ensemble complète des possibilités fascinantes de GitHub Copilot. Les exemples présentés n'illustrent qu'une fraction du potentiel de cet outil innovant qui va révolutionner vos processus de développement.

Pour découvrir GitHub Copilot de manière plus approfondie, nous vous recommandons de tester vous-même ce « programmeur pair » assisté par l'IA. L'intégration dans votre workflow pourrait faire toute la différence en accélérant la réalisation de vos tâches courantes ou en stimulant la créativité des approches de développement.

Nous sommes impatients de connaître votre avis sur GitHub Copilot, ainsi que vos conseils concernant cet outil. 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 e-mail 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 en améliorant 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 compétent
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 sur GitHub Copilot
- Démarrage rapide avec GitHub Copliot
- Comparaison entre GitHub Copilot et VS Code
- GitHub Copilot Labs
- Notions fondamentales de GitHub Copilot : bases du programmeur pair assisté par l'IA
- Premiers pas avec GitHub et Visual Studio Code
- Qu'est-ce que l'extension GitHub Copilot pour Visual Studio ?
- Centre de gestion de la confidentialité GitHub Copilot
- Blog GitHub