株式会社ウエルストーン様向け

ハンズオンガイド
GitHub CopilotによるAI駆動開発

スキルシートアプリにGitHub Copilotで機能を足し、受入基準とテストで確かめる手順を、Session03から05までの3つの演習で通します。最後にできあがるのは、案件ごとに条件を満たす技術者を出す画面と、その画面のもとになった3つの仕様書です。

2026年11月17日（火） オンライン 個人演習 VS CodeとGitHub Copilot

目次

本日のゴール
本日の学び
作るもの

当日の流れ
進め方

始める前の準備
skillsheetフォルダの開き方
プロンプトの送り方

Session03 GitHub Copilotとバイブコーディング
ねらい
できあがりの目安
使うファイルと作るファイル
STEP 1 コード補完で1つのメソッドを書く
STEP 2 チャットで既存のコードを読む
STEP 3 1行の依頼からアプリを作る
STEP 4 仕様のメモを渡して直す
うまくいかないとき
早く終わった方へ

Session04 Issue駆動開発
ねらい
できあがりの目安
使うファイルと作るファイル
STEP 1 Issueを選び、受入基準を3つ選ぶ
STEP 2 Issueから実装を頼む
STEP 3 受入基準をテストにする
STEP 4 AIのレビューを読み、マージを判断する
STEP 5 PRの説明文と2つ目のIssue
うまくいかないとき
早く終わった方へ

Session05 Kiro式の仕様駆動開発
ねらい
できあがりの目安
使うファイルと作るファイル
STEP 1 依頼文から要件を書かせる
STEP 2 設計とタスクを書かせる
STEP 3 タスクを1つずつ実装させる
STEP 4 仕様を1つ変える
STEP 5 役割を分けたエージェントでつなぐ
うまくいかないとき
早く終わった方へ

よくある質問

01

本日のゴール

Copilotに頼む範囲を自分で決め、出てきたコードを受入基準とテストと画面で確かめられるようになります。作るものは、スキルシートアプリの追加機能と、そのもとになったIssueと仕様書です。

本日の学び

- コード補完とチャットとエージェントを、頼む仕事の大きさで使い分ける
- 受入基準を選んでからIssueをCopilotに渡し、受入基準ごとのテストで結果を確かめる
- AIのレビューの結果を読み、マージしてよいかを自分で判断する
- 要件と設計とタスクを先に書かせ、タスクを1つずつ実装させる

作るもの

skillsheetフォルダのアプリに、Session04でIssue1件分の機能、Session05で案件ごとの候補の画面を足します。Issueのレビューの結果はskillsheet/reviews/に、仕様書はskillsheet/specs/matching/に保存されます。



図1：Session05を終えたときの画面です。枠で囲んだ「候補4名 必要2名」と、表の4名を見れば、仕様どおりに動いていると分かります。

02

当日の流れ

全体で[240min]です。講義のあとに、必ず各自のPCでの演習が続きます。

枠	中身	形式	時間
Session01	オープニング、Copilotと演習フォルダの確認	説明	[10min]
Session02	生成AIの最新動向とガバナンス	説明	[45min]
Session03	GitHub Copilotとバイブコーディング	個人演習	[60min]
Session04	Issue駆動開発	個人演習	[50min]
Session05	Kiro式の仕様駆動開発	個人演習	[65min]
閉会	質疑応答	説明	[10min]

進め方

- 講師が画面を共有し、各演習の最初に同じ操作を見せてから、受講者が自分のPCで進めます。
- 質問は、Zoomのチャットに「Session04 STEP2」のように番号を書き、画面に出ている文言を貼ってください。
- 各STEPの[Nmin]は目安です。時間になったら講師が次のSTEPへ進めます。生成が終わらないときは、講師が完成例のフォルダを個別にお渡しします。

03

始める前の準備

事前セットアップのSTEP 5までが済んでいれば、skillsheetフォルダを開いてテストを1回実行するだけで準備が整います。

skillsheetフォルダの開き方

1. VS Codeのメニューで **File** から **Open Folder...** を選びます。
2. デスクトップの1117_handsonの中にある `skillsheet` を選んで開きます。1117_handsonそのものは開きません。
3. メニューの **Terminal** から **New Terminal** を選びます。
4. ターミナルで `java tools/Build.java test` を実行します。

ターミナル（skillsheet）

```
$ java tools/Build.java test
[Build] コンパイル中: skillsheet（9ファイル）
[Build] テストをコンパイル中: 2ファイル
[Build] テストを実行します
└─ JUnit Jupiter ✔
   ├─ EngineerServiceTest ✔
   │  ├─ 初期データの技術者12名を、スキルと一緒に取得できる ✔
   │  └─ 存在しないIDを指定すると、空の結果になる ✔
   └─ EngineerControllerTest ✔
      ├─ 氏名が空のときは、登録画面にエラーが表示される ✔
      ├─ 存在しない技術者の詳細画面は404になる ✔
      ├─ 一覧画面に技術者の氏名が表示される ✔
      └─ 必要な項目を入力して登録すると、詳細画面へ移る ✔
[         6 tests successful      ]
[         0 tests failed          ]
[Build] すべてのテストが成功しました。
```

図2：最後の行に「すべてのテストが成功しました」と出ていれば、演習を始められます。

プロンプトの送り方

演習では、チャットの入力欄に「/」に続けてプロンプトの名前を入れ、そのあとに対象のファイルを書いて送ります。プロンプトの本文はskillsheet/.github/prompts/にあり、Copilotはその本文と一緒に依頼を受け取ります。

1. Ctrl+Alt+I（macOSはControl+Command+I）でチャットのビューを開きます。
2. 入力欄に `/issue-implement issues/ISSUE-101.md` のように入力して、Enterで送ります。

3 Copilotがコマンドを実行しようとすると、実行してよいかの確認が出ます。コマンドが`java tools/Build.java`で始まることを確かめてから許可します。

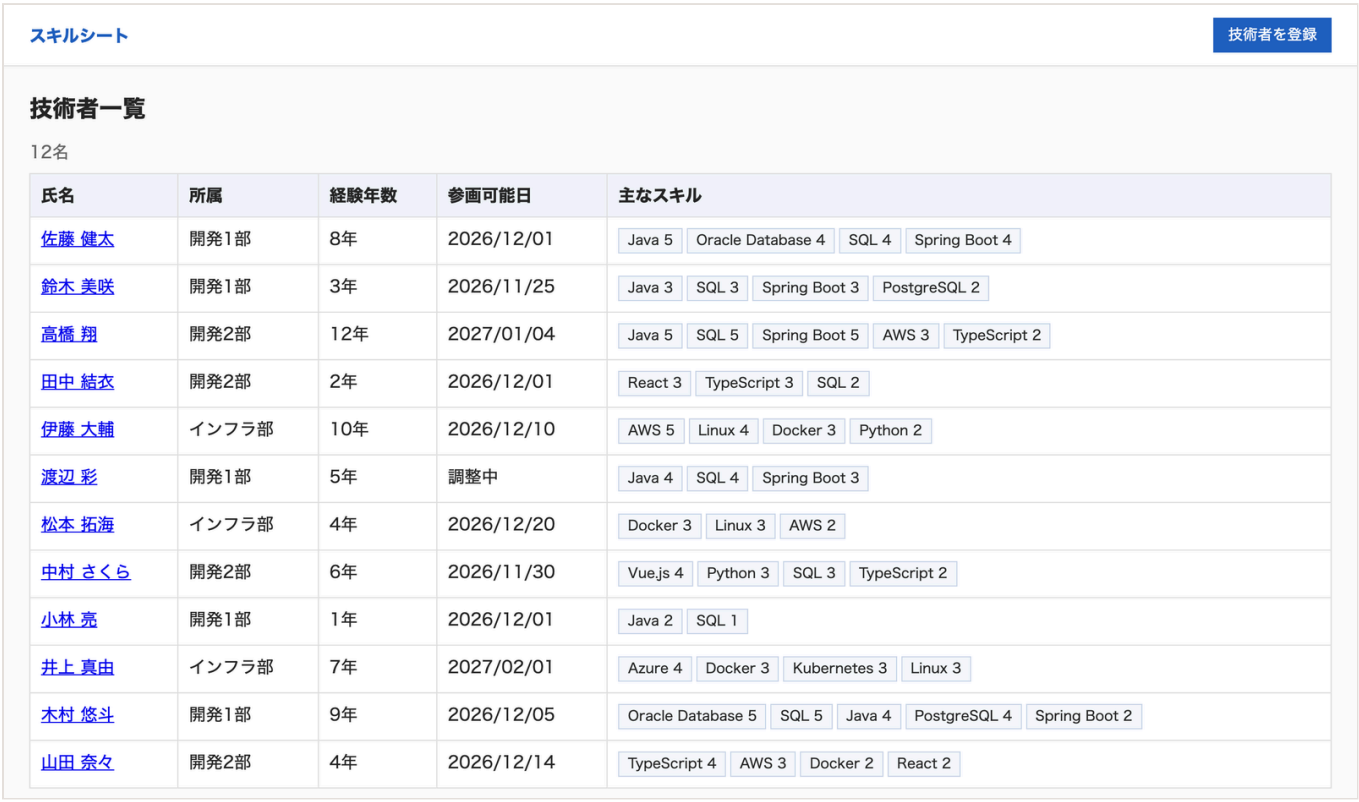


図3：演習の前のアプリの画面です。`java tools/Build.java run`で起動し、ブラウザでhttp://localhost:8080/を開くと、12名の一覧が出ます。

04

Session03 GitHub Copilotとバイブコーディング

[60min] 個人演習

ねらい

Copilotには、カーソルの位置で続きを出すコード補完、質問に答えるチャット、ファイルを作ってコマンドまで動かすエージェントの3つの使い方があります。1行を書くときと、アプリを丸ごと作るときで、どれに頼むのが早いかを自分の手で比べます。後半は、同じアプリに仕様のメモを渡して直し、1行の依頼のときより思った動きに近づくかを見る構成です。

できあがりの目安

- ☐ vibeフォルダで作ったアプリが起動し、ブラウザで一覧と登録の画面が見える
- ☐ spec.mdで選んだ3つの仕様が、画面の操作で確かめられる



図4：「日報アプリを作って」と1行で頼んだときにできた画面の例です。上に登録欄、下に一覧があれば、この演習の前半は終わりです。

使うファイルと作るファイル

区分	ファイル	使い方
使う	`skillsheet/src/main/java/com/example/skillsheet/util/YearsFormatter.java`	STEP 1のコード補完で、メソッドの中身を書く
使う	`skillsheet/src/main/java/com/example/skillsheet/engineer/EngineerService.java`	STEP 2のチャットで、何をしているクラスかを聞く
使う	`vibe/README.md`、`vibe/spec.md`	STEP 3の依頼の候補と、STEP 4の仕様のメモ
作る	`vibe/src/main/java/com/example/vibe/`の下	Copilotが作るアプリのクラスと画面とSQL

補完で1行を書く → チャットで読む → エージェントで作る → 画面で確かめる

STEP 1 コード補完で1つのメソッドを書く [8min]

コード補完は、コメントとメソッド名から次の数行を予想して、薄い文字で出します。1つのメソッドを書くくらいの大きさなら、チャットを開くより早く終わります。

1. skillsheetフォルダを開いたVS Codeで、`YearsFormatter.java`を開きます。
2. `throw new UnsupportedOperationException`の行を消します。
3. 空いた行に`if (years == null) {`と入力し、薄い文字の続きが出るのを待ちます。
4. 続きがJavadocのとおりであればTabで受け入れ、違っていればEscで消して自分で書きます。
5. メソッドの最後まで、同じように受け入れるか書くかを選びながら進めます。

できあがりの例

```
if (years == null) {
    return "未登録";
}
if (years < 1) {
    return "1年未満";
}
if (years >= 10) {
    return "10年以上";
```

```
}
return years + "年";
```

ここまでできたら

- ☐ Javadocに書かれた「1年未満」「10年以上」「未登録」の3つの場合を、メソッドが返す
- ☐ 薄い文字の候補を、受け入れたか消したかを自分で選んだ

手が止まったら開く −

候補が出ないときは、行の最後で少し待つか、Alt+\（macOSはOption+\）で候補を出します。候補がJavadocと違うときは、受け入れずに次の1行を自分で書くと、そこから先の候補が変わります。

STEP 2 チャットで既存のコードを読む [7min]

チャットのAskは、ファイルを書き換えずに質問へ答えます。初めて触るコードの役割を、ファイルを指定して聞くと、読む順番の見当が付きます。

1. チャットのビューを開き、入力欄の下のエージェントの選択で `Ask` を選びます。
2. 下の文面をコピーして入力欄に貼り、送ります。
3. 返事を読んだら、`EngineerRepository.java` を開き、`findAllSkills` のSQLを選択します。
4. 入力欄に「#selection このSQLは何を取得していますか。結合している2つのテーブルの役割も書いてください」と入れて送ります。

チャットに送る文面

```
#file:src/main/java/com/example/skillsheet/engineer/EngineerService.java
このクラスが何をしているかを3行で説明してください。
findAllの中で、技術者とスキルをどうまとめているかも書いてください。
```

ここまでできたら

- ☐ EngineerServiceが、技術者の一覧とスキルの一覧を技術者のIDでまとめていると説明できる
- ☐ チャットのAskでは、ファイルが書き換わらないことを確かめた

STEP 3 1行の依頼からアプリを作る [20min]

エージェントは、ファイルを作り、ビルドとテストのコマンドまで実行します。1行の依頼から、Copilotがどこまで自分で決めるかを見ます。

1. メニューの **File** から **New Window** を選び、新しいウィンドウで `vibe` フォルダを開きます。
2. チャットのビューを開き、エージェントの選択で **Agent** を選びます。
3. vibe/README.mdの「依頼の候補」の3つから1つを選び、そのまま入力欄に貼って送ります。
4. コマンドの実行の確認が出たら、`java tools/Build.java` で始まることを確かめてから許可します。
5. Copilotの報告が終わったら、チャットの上に並んだ変更のファイルを1つずつ開き、変わった箇所を見ます。
6. 残す変更は **Keep** 、戻す変更は **Undo** を押します。
7. ターミナルで `java tools/Build.java run` を実行し、ブラウザでhttp://localhost:8080/を開きます。

依頼の候補（1つを選んで送る）

日報アプリを作って。日付と作業内容と作業時間を登録して、一覧で見られるようにして
備品の貸出アプリを作って。備品の一覧と、誰がいつ借りたかを記録できるようにして
社内勉強会の参加登録アプリを作って。勉強会の一覧と、参加したい人の登録ができるようにして


README.md

vibe フォルダ（Session03）

自然言語の依頼から、Spring Boot の小さな Web アプリを作るためのフォルダです。 最初は、設定ファイルとビルド補助だけが入っています。

依頼の候補

Copilot のチャットでエージェントを「Agent」にして、次の3つから1つを選んで送ります。

1. 日報アプリを作って。日付と作業内容と作業時間を登録して、一覧で見られるようにして
2. 備品の貸出アプリを作って。備品の一覧と、誰がいつ借りたかを記録できるようにして
3. 社内勉強会の参加登録アプリを作って。勉強会の一覧と、参加したい人の登録ができるようにして

起動のしかた

ターミナルで次のコマンドを実行し、ブラウザで http://localhost:8080/ を開きます。

```
java tools/Build.java run
```

止めるときは、ターミナルで Ctrl+C を押します。

図5：vibe/README.mdの「依頼の候補」です。3つのうち1つを、そのままチャットの入力欄に貼って送ります。


日報

登録

日付 年 /月/日

作業内容

作業時間（分）

登録

一覧

日付	作業内容	作業時間（分）
2026-11-05	テストの作成と修正	90
2026-11-04	登録フォームの実装	150
2026-11-03	一覧画面の実装	180
2026-11-02	要件の確認と画面の設計	120

図6：日報アプリを頼んだときの結果の例です。作業時間が「分」になっているように、1行の依頼に無いことはCopilotが決めています。

ここまでできたら

- □ ブラウザで一覧が表示され、登録した内容が一覧に増える
- □ Copilotが自分で決めたことを、画面から2つ挙げられる（項目の単位、並び順など）

手が止まったら開く −

CopilotがmvnやgradleをCopilotを実行しようとしたら、許可せずに「tools/Build.javaを使ってください」と送ります。8080番ポートのエラーが出たら、skillsheetのウィンドウで起動したアプリをCtrl+Cで止めます。

STEP 4 仕様のメモを渡して直す [15min]

同じアプリに、仕様を3行書いたメモを渡して直してもらいます。メモに書いたことと、画面で起きたことを1つずつ見比べます。

1. `vibe/spec.md` を開き、コメントの候補AからHのうち3つを選んで、「- 」の行に書き写します。
2. アプリを起動している場合は、ターミナルでCtrl+Cを押して止めます。
3. 下の文面を、チャットの入力欄に貼って送ります。

4 Copilotの報告が終わったら、アプリを起動し、選んだ3つの仕様をブラウザで1つずつ操作して確かめます。

チャットに送る文面

`#file:spec.md` の仕様どおりに、作ったアプリを直してください。
直したら、仕様の3つを確かめるテストを書いて、`java tools/Build.java test` で実行してください。



図7：試走で選んだspec.mdです。候補のA、C、Fの3つを、下の「- 」の行に書き写しています。

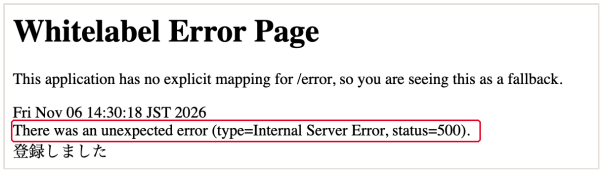


図8：11月6日の試走で、登録したあとに出たエラーです。Copilotは既存のテスト2件の成功を報告していましたが、足した機能のテストを書いていませんでした。

ここまでできたら

- ☐ spec.mdの3つの仕様を、ブラウザの操作で1つずつ確かめた
- ☐ 1行の依頼とメモを渡す依頼で、直す回数が少なかった方を1文で言える

手が止まったら開く −

画面でエラーが出たら、ブラウザの文言と、アプリを起動したターミナルの赤い行をコピーし、「この画面のエラーを直して、同じ操作を確かめるテストを足してください」と送ります。

うまくいかないとき

Copilotがpom.xmlを作ろうとします −

vibe/.github/copilot-instructions.mdを読み落としています。作られた変更は **Undo** で戻し、「.github/copilot-instructions.md のきまりを読んでから、もう一度作ってください」と送ります。

「@SpringBootApplication を付けたクラスが複数あります」と出ます −

Copilotが起動用のクラスを2つ作っています。チャットに「起動用のクラスを1つにまとめてください」と送ります。

早く終わった方へ

追加と考察

spec.mdで選ばなかった候補のうち、H「いい感じのデザインにする」だけを送ってみてください。確かめ方の無い仕様を渡すと、Copilotが何を変えるかを見比べられます。

発展課題

ご自身の業務で、Excelや紙で管理している一覧を1つ思い浮かべ、その一覧の登録と検索をする依頼を1行で書いて送ってください。そのあと、確かめられる仕様を3行書いて、違いを比べます。

05

Session04 Issue駆動開発

[50min] 個人演習

ねらい

Issueは、Copilotに渡す依頼書です。受入基準のないIssueを渡すと、Copilotは決まっていない点を自分で埋めて実装します。この演習では、受入基準を選んだIssueを渡し、受入基準ごとのテストとAIのレビューで、マージしてよいかを自分で判断するところまで通します。

できあがりの目安

- ☐ 選んだIssueの受入基準3つに対応するテストが、すべて成功している
- ☐ reviews/にレビューの結果があり、「人の判断」の欄を自分で書いた



図9：ISSUE-102を選んだときのできあがりの例です。枠で囲んだ所に日本語のエラーが出ていれば、受入基準の1つを満たしています。

使うファイルと作るファイル

区分	ファイル	使い方
使う	`skillsheet/issues/ISSUE-100.md`	あいまいなIssueの例。STEP 1でCopilotに読ませる
使う	`skillsheet/issues/ISSUE-101.md` から `ISSUE-103.md`	3つから1つを選び、受入基準を選んで書く
使う	`skillsheet/.github/prompts/` のissue-implement、issue-test、review	STEP 2から4で「/」に続けて呼び出す
作る	`skillsheet/reviews/ISSUE-10x-review.md`	Copilotのレビューの結果と、自分の判断

受入基準を選ぶ → 実装を頼む → テストを頼む → レビューを頼む → マージを判断する

STEP 1 Issueを選び、受入基準を3つ選ぶ [10min]

最初に、あいまいなIssueをCopilotに読ませて、決まっていない点がどれだけあるかを見ます。そのあと、取り組むIssueを1つ選び、確かめられる受入基準を候補から3つ選びます。

1. チャットのエージェントの選択で `Ask` を選び、下の文面を送ります。
2. 返事に挙がった「決まっていない点」を読みます。
3. ISSUE-101、ISSUE-102、ISSUE-103のうち、1つを選んで開きます。
4. 「受入基準」のコメントにある候補AからHを読み、確かめられるものを3つ選びます。
5. 選んだ3つを、コメントの下の「- 」の行に書き写して保存します。

チャットに送る文面

```
#file:issues/ISSUE-100.md
このIssueだけで実装を始めるとき、決まっていない点を5つ挙げてください。
実装はしないでください。
```



図10：あいまいなIssueの例です。何で検索するか、結果をどう出すかが書かれていません。



図11：ISSUE-101で受入基準を3つ選んだ例です。下の方の「受入基準」の欄に、候補の文をそのまま3行書き写しています。

ここまでできたら

- □ 選んだIssueの受入基準の欄に、「- 」で始まる行が3つある
- □ 候補のGとHを選ばなかった理由を、1文で言える

手が止まったら開く −

迷ったら、「その基準をテストで確かめるとき、何を入力して、画面に何が出れば合格か」を言えるかで選びます。言えない基準は、Copilotにも確かめ方が分かりません。

STEP 2 Issueから実装を頼む [12min]

受入基準を書いたIssueのファイルを、プロンプトと一緒にCopilotへ渡します。Copilotは方針を先に書いてから実装し、既存のテストを実行して報告します。

1. チャットのビューで新しいチャットを始めます。
2. 入力欄に下の文面を入れ、ISSUE-10xの部分を選んだIssueの番号に直して送ります。
3. コマンドの実行の確認が出たら、`java tools/Build.java` で始まることを確かめてから許可します。

4 変更のファイルを1つずつ開き、Issueの「やらないこと」に手が入っていないかを見て、`Keep` か `Undo` を選びます。

5 アプリを起動し、受入基準の操作をブラウザで1つ行います。

チャットに送る文面 ISSUE-10xを選んだ番号に直す

```
/issue-implement issues/ISSUE-10x.md
```



図12：ISSUE-102の実装の前です。経験年数に「abc」を入れると、枠の英語のエラーがそのまま出ます。



図13：実装のあとです。同じ入力で、受入基準に書いた日本語のエラーが出ています。

ここまでできたら

- [ ] Copilotの報告に、受入基準ごとに満たした場所の表がある
- [ ] 受入基準の操作の1つを、ブラウザで確かめた

手が止まったら開く −

「受入基準を3つ選んでから、もう一度送ってください」と返ってきたら、STEP 1の「- 」の行が空のまま保存されています。ファイルを保存し直してから、同じ文面を送ります。ISSUE-101を選んだ場合は、一覧の上に検索欄ができ、「Java」で6名が表示されれば合格です。

STEP 3 受入基準をテストにする [8min]

受入基準の1行ごとに、JUnitのテストを書かせます。テストの名前には受入基準の文がそのまま入るので、どの基準を確かめたテストかが一目で分かります。

1. STEP 2と同じチャットに、下の文面を、Issueの番号を直して送ります。
2. コマンドの実行を許可し、テストの結果が出るのを待ちます。
3. ターミナルの結果で、受入基準の文が付いたテストに成功の印が付いていることを確かめます。

チャットに送る文面 ISSUE-10xを選んだ番号に直す

```
/issue-test issues/ISSUE-10x.md
```

ターミナル（skillsheet）

```
$ java tools/Build.java test
[Build] コンパイル中: skillsheet（9ファイル）
[Build] テストをコンパイル中: 2ファイル
[Build] テストを実行します
└─ JUnit Jupiter ✔
   ├─ EngineerServiceTest ✔
   │  ├─ 初期データの技術者12名を、スキルと一緒に取得できる ✔
   │  └─ 存在しないIDを指定すると、空の結果になる ✔
   └─ EngineerControllerTest ✔
      ├─ 経験年数を空のまま登録すると、「経験年数を入力してください」と表示され、登録されない ✔
      ├─ 経験年数に「abc」を入れて登録すると、「経験年数は数字で入力してください」と表示される ✔
      ├─ 氏名が空のときは、登録画面にエラーが表示される ✔
      ├─ 経験年数に 51 を入れて登録すると、「経験年数は0〜50の範囲で入力してください」と表示される ✔
      ├─ 存在しない技術者の詳細画面は404になる ✔
      ├─ 一覧画面に技術者の氏名が表示される ✔
      ├─ 経験年数に -1 を入れて登録すると、「経験年数は0〜50の範囲で入力してください」と表示される ✔
      └─ 必要な項目を入力して登録すると、詳細画面へ移る ✔
[        10 tests successful      ]
[         0 tests failed          ]
[Build] すべてのテストが成功しました。
```

図14：ISSUE-102で実行した結果です。「経験年数を空のまま登録すると」のように、受入基準の文がテストの名前に入っています。枠の10 tests successfulと0 tests failedを確かめます。

ここまでできたら

- □ 受入基準の3つそれぞれに、名前の対応するテストがある
- □ 失敗したテストが0件になっている

手が止まったら開く −

ISSUE-103を選ぶと、Copilotが既存のテストの「先頭は佐藤 健太」を書き換えることがあります。並び順を変える

Issueでは期待どおりの変更なので、書き換えた理由が報告にあるかを確かめてください。

STEP 4 AIのレビューを読み、マージを判断する [8min]

Copilotに変更をレビューさせ、結果をファイルに残します。AIの所見は参考にとどめ、マージしてよいかは受講者が決めて、理由を1行で書きます。

1. 同じチャットに、下の文面を、Issueの番号を直して送ります。
2. reviews/にできたファイルを開き、「受入基準ごとの判定」と「気になる点」を読みます。
3. 最後の「人の判断」の見出しの下に、「マージしてよい」「直してからマージ」「マージしない」のどれかと、理由を1行書いて保存します。

チャットに送る文面 ISSUE-10xを選んだ番号に直す

```
/review issues/ISSUE-10x.md
```

reviews/ISSUE-102-review.md

ISSUE-102 レビュー

対象: EngineerForm.java、messages.properties（新規）、EngineerControllerTest.java

1. 受入基準ごとの判定

受入基準	判定	根拠
経験年数を空のまま登録すると、「経験年数を入力してください」と表示され、登録されない	満たす	EngineerForm.java:23（`@NotNull`）。エラー時は EngineerController.java の `create` が `service.register` を呼ばずに form を返す。テスト EngineerControllerTest.java:66
経験年数に「abc」を入れて登録すると、「経験年数は数字で入力してください」と表示される	満たす	messages.properties:1（`typeMismatch.yearsOfExperience`）。テスト EngineerControllerTest.java:77
経験年数に -1 または 51 を入れて登録すると、「経験年数は0〜50の範囲で入力してください」と表示される	満たす	EngineerForm.java:24-25（`@Min(0)`、`@Max(50)`）。テスト EngineerControllerTest.java:88（-1）、:99（51）

実際のブラウザ画面での確認（アプリの起動）は行っていない。MockMvc のテストで確かめた範囲の判定である。

2. テスト

- 3つの受入基準すべてに対応するテストがある（基準3は -1 と 51 で別メソッドに分けた）。
- `java tools/Build.java test` は10件すべて成功した。
- 足りないテスト
  - 空のまま登録したときに、件数が増えていないことを確かめるテストはない（「登録されない」は、エラー画面が返る

図15：ISSUE-102でCopilotが書いたレビューの冒頭です。受入基準ごとに「満たす」と根拠のファイルの行が書かれているかを見ます。

ここまでできたら

- □ 「人の判断」の欄に、判断と理由を自分で書いた
- □ AIの所見と自分の判断が違う場合、その理由を言える

> **ここまでが必須です**
>
> STEP 1〜4が終わると、この演習の必須はすべて済みます。STEP 4のあとは、講師がGitHub上でIssueからPRを作り、AIのレビューを受けるまでを実演します。STEP 5は発展で、時間が残った方と、研修のあとに続ける方が進めてください。

STEP 5 PRの説明文と2つ目のIssue

発展 [10min]

/pr-descriptionで、レビューの結果とテストの結果からPRの説明文を書かせます。続けて、選ばなかったIssueを1つ、STEP 1から通します。

チャットに送る文面

```
/pr-description issues/ISSUE-10x.md
```

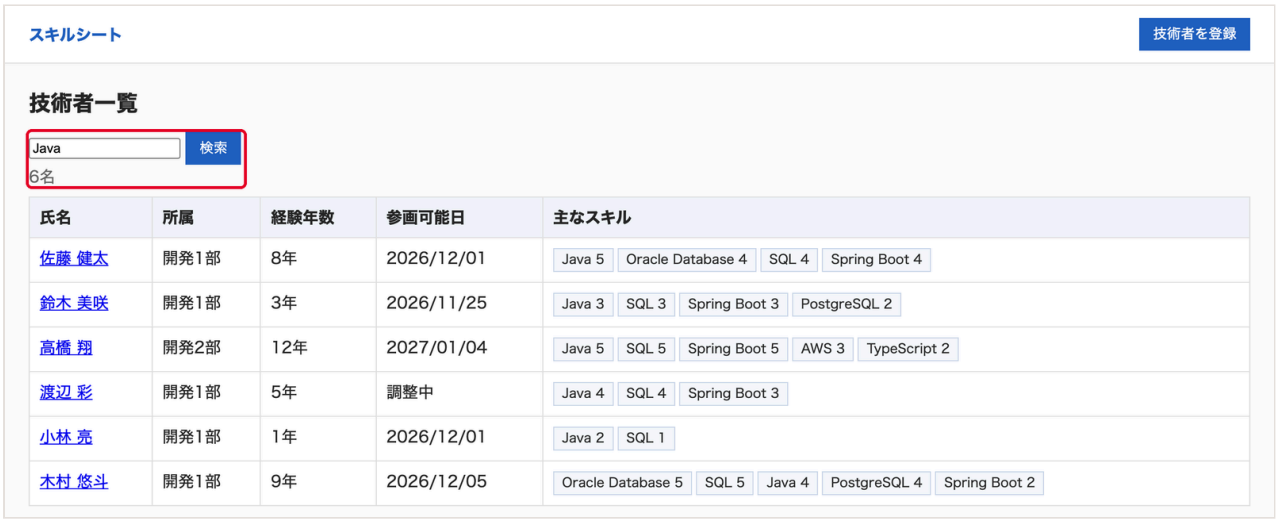


図16：2つ目にISSUE-101を選んだときのできあがりの例です。枠の検索欄に「Java」と入れると、6名だけが表に残ります。

うまくいかないとき

/issue-implementを入力しても候補に出てきません

VS Codeで開いているフォルダが、skillsheetではない可能性があります。エクスプローラーの一番上の名前がSKILLSHEETになっているかを確かめ、違えばskillsheetフォルダを開き直します。

テストが失敗したまま、Copilotの報告が終わりました

同じチャットに「失敗したテストの原因を説明してから直してください」と送ります。原因の説明を先に求めると、テストを消して成功させるような直し方を避けられます。

早く終わった方へ

追加と考察

Issueの「やらないこと」の欄を空にして、同じIssueを新しいチャットで実装させてみてください。Copilotが足した機能のうち、頼んでいないものを数えると、「やらないこと」を書く理由が分かります。

発展課題

ご自身のチームで最近あった修正を1つ選び、.github/ISSUE_TEMPLATE/feature.mdの形でIssueを書いてください。受入基準を3行書けるかどうかで、Copilotに渡せる粒度かを判断できます。

06

Session05 Kiro式の仕様駆動開発

[65min] 個人演習

ねらい

Kiro式の仕様駆動開発では、要件、設計、タスクの3つの仕様書を先にCopilotに書かせ、人が確かめてから実装に進みます。Issue1件より大きい機能でも、タスクを1つずつ実装させるので、どこまで終わったかがtasks.mdのチェックで分かります。最後に仕様を1つ変え、3つの仕様書とコードがそろって変わるかを確かめる流れです。

できあがりの目安

- ☐ specs/matching/に、requirements.md、design.md、tasks.mdの3つがある
- ☐ tasks.mdのタスクにチェックが付き、/projects/1に候補の表が出る
- ☐ 仕様を変えたあと、候補の人数が変わった



図17：STEP 4で仕様を変えたあとの画面の例です。枠の人数が「候補4名」から「候補3名」に変われば、仕様の変更がコードまで届いています。

使うファイルと作るファイル

区分	ファイル	使い方
使う	`skillsheet/specs/matching/request.md`	営業担当からの機能の依頼文。追加する要件を候補から2つ選ぶ

区分	ファイル	使い方
使う	`skillsheet/.github/prompts/` のspec-で始まる5本	STEP 1から4で「/」に続けて呼び出す
作る	`skillsheet/specs/matching/requirements.md`	要件と、EARSの形で書いた受入基準
作る	`skillsheet/specs/matching/design.md` 、`tasks.md`	設計と、実装の手順のチェックリスト
作る	`skillsheet/src/main/java/com/example/skillsheet/project/` の下	案件の一覧と候補の画面のクラス

追加の要件を選ぶ → 要件を書かせる → 設計とタスク → タスクごとに実装 → 仕様を変える

STEP 1 依頼文から要件を書かせる [12min]

営業担当の依頼文に、追加する要件を2つ足してから、requirements.mdを書かせます。受入基準は「〜のとき、システムは〜する」というEARSの形で書かれるので、1行ずつテストに置き換えられます。

1. `specs/matching/request.md` を開き、「追加する要件」の候補AからIのうち2つを選んで、「- 」の行に書き写して保存します。
2. チャットで新しいチャットを始め、下の文面を送ります。
3. できたrequirements.mdを開き、選んだ2つが要件に入っているかと、最後の「確認したいこと」を読みます。
4. 「確認したいこと」のうち、決め方を変えたい項目があれば、requirements.mdを自分で直して保存します。

チャットに送る文面

```
/spec-requirements specs/matching
```



specs/matching/requirements.md

要件 案件ごとの候補者の表示

概要

営業担当が、案件の必要スキルを満たす技術者を探すのに時間がかかっている問題を解決する。 案件の一覧から案件を選ぶと、その案件の必要スキルをすべて満たす技術者が候補として表示される画面を作る。 案件と技術者の登録や編集は、この機能の対象にしない。

要件1 案件の一覧

ユーザーストーリー：営業担当として、案件の一覧を見たい。探したい案件をすぐに選べるようにするため。

受入基準

- 1.1 営業担当が /projects を開いたとき、システムは projects テーブルにあるすべての案件を、案件名、開始日、必要人数とともに一覧で表示する
- 1.2 営業担当が一覧の案件名を選んだとき、システムはその案件の /projects/{id} の画面を表示する

要件2 案件ごとの候補の表示

ユーザーストーリー：営業担当として、案件の必要スキルを満たす技術者を一覧で見たい。技術者を1人ずつ見比べる時間をなくすため。

受入基準

- 2.1 営業担当が /projects/{id} を開いたとき、システムは案件名、概要、開始日、必要人数、必要スキルとその min_level を表示する
- 2.2 技術者が案件の必要スキルをすべて持ち、どのスキルも min_level 以上の場合、システムはその技術者を候補に入れる
- 2.3 技術者が必要スキルを1つでも持たない場合、システムはその技術者を候補に入れない
- 2.4 技術者のスキルのレベルが、必要スキルの min_level を1つでも下回る場合、システムはその技術者を候補に入れない
- 2.5 候補があるとき、システムは候補ごとに氏名、所属、経験年数、参画可能日、必要スキルのレベルを表示する
- 2.6 候補が複数あるとき、システムは候補を技術者の ID の小さい順に並べる
- 2.7 存在しない案件の ID が指定された場合、システムは 404 を返す

図18：11月6日の試走でできたrequirements.mdの冒頭です。2.2のように番号の付いた受入基準が、「〜のとき、システムは〜する」の形で書かれているかを見ます。

ここまでできたら

- □ request.mdで選んだ2つの要件が、requirements.mdの要件に入っている
- □ 「確認したいこと」の項目を読み、直すかどうかを決めた

手が止まったら開く −

候補のGとHは、この機能の範囲を広げる項目です。Iは確かめ方の無い項目です。迷ったら、AとEのように画面の表示で確かめられるものを選びます。

STEP 2 設計とタスクを書かせる [10min]

要件から設計を書かせ、続けて実装の手順をtasks.mdに分けさせます。タスクは、Copilotに1回頼んで終わる大きさの4〜6個になります。

1. 同じチャットに `/spec-design specs/matching` を送ります。
2. design.mdの「判定のルール」を開き、候補に入る条件が自分の理解と合っているかを確かめます。

3 続けて `/spec-tasks specs/matching` を送ります。

4 tasks.mdを開き、各タスクの「終わったと判断する条件」を読みます。

チャットに送る文面（1行ずつ送る）

```
/spec-design specs/matching
/spec-tasks specs/matching
```



図19：試走でできたtasks.mdです。タスクごとに、対応する受入基準の番号と、どのテストが成功すれば終わりかが書かれているかを見ます。

ここまでできたら

- □ tasks.mdに、チェックの無いタスクが4〜6個ある
- □ 各タスクに、終わったと判断する条件が書かれている

STEP 3 タスクを1つずつ実装させる [20min]

タスクの番号を指定して、1つずつ実装させます。Copilotはタスクの範囲だけを実装し、テストが成功したらtasks.mdにチェックを付けます。

1. 下の文面を送り、タスク1を実装させます。
2. 報告とtasks.mdのチェックを確かめたら、番号を2に変えて送ります。
3. 同じように、最後のタスクまで1つずつ進めます。
4. アプリを起動し、ブラウザでhttp://localhost:8080/projectsを開いて、案件名を押します。

チャットに送る文面 最後の数字をタスクの番号にする

```
/spec-run-task specs/matching 1
```



図20：案件一覧の画面です。案件名を押すと、その案件の候補の画面に移ります。



図21：追加の要件にAとEを選んだときの例です。枠に、Eで選んだ「候補4名 必要2名」が出ています。

ここまでできたら

- □ tasks.mdのすべてのタスクにチェックが付いている
- □ 案件1の候補が、必要スキルのJavaとSpring BootとSQLをすべてレベル3以上で持つ人になっている

手が止まったら開く −

時間内に最後のタスクまで進まないときは、終わったタスクまでで止めて構いません。講師が完成例のフォルダをお渡しするので、STEP 4はそのフォルダで進められます。

STEP 4 仕様を1つ変える [10min]

営業担当から「開始日に間に合わない人は候補から外してほしい」と追加の依頼が来た場面です。コードを直接直さずに、仕様書を先に直させ、足されたタスクを実装させます。

1. 下の文面の1行目を送り、3つの仕様書を直させます。

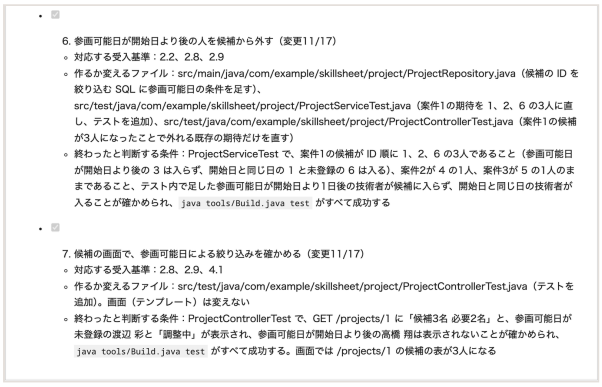
2. tasks.mdの末尾に、チェックの無いタスクが足されたことを確かめます。

3. 足されたタスクの番号で `/spec-run-task` を送り、1つずつ実装させます。

4. ブラウザで案件1の画面を開き直し、候補の人数が変わったことを確かめます。

チャットに送る文面 2行目の番号は足されたタスクの番号にする

```
/spec-update specs/matching 参画可能日が案件の開始日より後の人は、候補から外す。参画可能日が未登録の人は候補に残す
/spec-run-task specs/matching 6
```



- 6. 参画可能日が開始日より後の人を候補から外す（変更11/17）
  - 対応する受入基準：2.2、2.8、2.9
  - 作るか変えるファイル：src/main/java/com/example/skillsheet/project/ProjectRepository.java（候補の ID を絞り込む SQL に参画可能日の条件を足す）、src/test/java/com/example/skillsheet/project/ProjectServiceTest.java（案件1の期待を 1、2、6 の3人に直し、テストを追加）、src/test/java/com/example/skillsheet/project/ProjectControllerTest.java（案件1の候補が3人になったことで外れる既存の期待だけを直す）
  - 終わったと判断する条件：ProjectServiceTest で、案件1の候補が ID 順に 1、2、6 の3人であること（参画可能日が開始日より後の 3 は入らず、開始日と同じ日の 1 と未登録の 6 は入る）、案件2が 4 の1人、案件3が 5 の1人のままであること、テスト内で足した参画可能日が開始日より1日後の技術者が候補に入らず、開始日と同じ日の技術者が入ることが確かめられ、`java tools/Build.java test` がすべて成功する
- 7. 候補の画面で、参画可能日による絞り込みを確かめる（変更11/17）
  - 対応する受入基準：2.8、2.9、4.1
  - 作るか変えるファイル：src/test/java/com/example/skillsheet/project/ProjectControllerTest.java（テストを追加）。画面（テンプレート）は変えない
  - 終わったと判断する条件：ProjectControllerTest で、GET /projects/1 に「候補3名 必要2名」と、参画可能日が未登録の渡辺 彩と「調整中」が表示され、参画可能日が開始日より後の高橋 翔は表示されないことが確かめられ、`java tools/Build.java test` がすべて成功する。画面では /projects/1 の候補の表が3人になる

図22：試走で仕様を変えたあとのtasks.mdの末尾です。「(変更11/17)」の付いたタスクが2つ足されました。



スキルシート

在庫管理システムの画面改修

概要	倉庫ごとの在庫を一覧する画面を、Spring Boot で作り直す。既存の Oracle Database を参照する
開始日	2026/12/01
必要人数	2名
必要スキル	Java 3以上　Spring Boot 3以上　SQL 3以上

候補

候補3名 必要2名

氏名	所属	経験年数	参画可能日	必要スキルのレベル
佐藤 健太	開発1部	8年	2026/12/01	Java 5　Spring Boot 4　SQL 4
前木 美咲	開発1部	3年	2026/11/25	Java 3　Spring Boot 3　SQL 3
渡辺 彩	開発1部	5年	調整中	Java 4　Spring Boot 3　SQL 4

案件一覧に戻る

図23：足されたタスクを実装したあとの画面です。枠の人数が3名に変わり、参画可能日が2027/01/04の高橋 翔が外れています。

ここまでできたら

- [ ] requirements.mdとdesign.mdに、「(変更11/17)」の付いた箇所がある
- [ ] 案件1の候補から、開始日より後に参画できる人が外れた

ここまでが必須です

STEP 1〜4が終わると、この演習の必須はすべて済みます。STEP 4のあとは、講師が3つのカスタムエージェントで仕様から実装とレビューまでをつなぐ流れを実演します。STEP 5は発展で、合計[15min]です。

STEP 5 役割を分けたエージェントでつなぐ

発展 [15min]

skillsheet/.github/agents/には、仕様を書くspec-planner、タスクを実装するtask-runner、レビューするreviewerの3つのエージェントがあります。エージェントを終えるたびに、次のエージェントへ渡すボタンが出ます。

1. `specs/` の下に新しいフォルダを作り、request.mdに「技術者の詳細画面に、その人が候補に入る案件の一覧を出す」と書いて保存します。

2 チャットのエージェントの選択で `spec-planner` を選び、そのフォルダの名前を送ります。

3 3つの仕様書ができたら、返事の下に出るボタンで `task-runner` へ渡します。

4 タスクが終わるたびに、ボタンで `reviewer` へ渡すか、次のタスクへ進むかを選びます。

うまくいかないとき

Copilotが設計の段階でコードまで書き始めました −

停止のボタンで止め、変更を `Undo` で戻します。そのあと「コードは書き換えず、design.mdだけを書いてください」と送ります。

タスクの途中でテストが通らなくなりました −

同じチャットに「設計と食い違っている点があれば、先にdesign.mdの直し方を提案してください」と送ります。spec-run-taskのプロンプトには、設計と食い違うときは実装を止めて提案するきまりが入っています。

早く終わった方へ

追加と考察

request.mdで選ばなかった候補Fの「必要スキルのうち1つだけ足りない人を、あと1つとして別の表に出す」を、/spec-updateで足してみてください。タスクがいくつ足されるかで、仕様の変更の大きさが見えます。

発展課題

ご自身のチームで次に作る機能を1つ選び、request.mdの形で依頼文を書いてください。Issue1件で渡すか、3つの仕様書から作るかを、タスクの数の見込みで決めてみてください。

07

よくある質問

前の演習が途中のままですが、次の演習に進めますか

進めます。Session04とSession05は、skillsheetの別の機能を扱うので、前の演習の結果を使いません。Session04の変更が残っていても、Session05の手順はそのまま通ります。

Copilotの返事が試走の画面と違います

Copilotの返事は、同じ依頼でも毎回少しずつ変わります。ガイドの画面は11月6日の試走の結果で、合っているかは各STEPの「ここまでできたら」の項目で判断してください。

コマンドの実行の確認が毎回出て、手間がかかります

研修では、毎回コマンドの中身を読んでから許可してください。Copilotが実行するコマンドを人が確かめることも、この研修で身に付けたいことの1つです。

演習の途中で、ファイルを元に戻したくなりました

直前の依頼で変わったファイルは、`Undo`で戻せます。演習の最初からやり直すときは、ZIPを別のフォルダに展開し直して、そちらを開きます。

© Givery, Inc.

※本教材の演習に登場する企業・人物・数値はすべて架空です。