

GitHub Copilot Vision in IntelliJ

Overview

GitHub Copilot Vision enables developers to turn images and diagrams directly into working code using GitHub Copilot's AI capabilities, integrated into IntelliJ IDEA.

Table of Contents

- [GitHub Copilot Vision in IntelliJ](#)
 - [Overview](#)
 - [Table of Contents](#)
 - [Introduction to Copilot Vision](#)
 - [Setup Instructions](#)
 - [Requirements:](#)
 - [How to Use:](#)
 - [Demo Breakdown](#)
 - [Demo 1: Analyze Architecture Diagrams](#)
 - [High-level System Architecture Diagram](#)
 - [Demo 2: Generate Class Files from Class Diagrams](#)
 - [UML Class Diagram](#)
 - [Demo 3: Generate APIs and Models from Use Case Diagrams](#)
 - [Use case Diagram](#)
 - [Demo 4: Data Modeling from ER Diagrams](#)
 - [Entity Relationship Diagram](#)
 - [Demo 5: Generate Test Cases from Visual Flows](#)
 - [Flowchart for Process Logic](#)
 - [Demo 6: Parse Network Topology to Generate Config Scripts](#)
 - [Network Topology Diagram](#)
 - [Demo 7: Screen Design to Frontend Code](#)
 - [Wireframe/Mockup of a UI Screen](#)
 - [Demo 8: Modify Existing Code Based on a Diagram](#)
 - [Updated Component Diagram](#)
 - [Demo 9: Hand-Drawn Diagram to Code](#)
 - [Code Scaffolding and Suggestions](#)
 - [Conclusion](#)

Introduction to Copilot Vision

GitHub Copilot Vision is an AI tool that allows developers to interact with visual inputs (images/diagrams) inside the IntelliJ IDE. Instead of writing boilerplate code manually, developers can upload or paste a diagram and prompt Copilot to generate relevant code.

Setup Instructions

Requirements:

- IntelliJ IDEA (latest version recommended)
- GitHub Copilot plugin installed
- GitHub Copilot subscription
- Enable Vision capabilities under Copilot Labs (Beta)

How to Use:

1. Open IntelliJ and install the GitHub Copilot plugin.
 2. Log in with your GitHub credentials.
 3. Drag and drop an image or diagram into the editor.
 4. Use `Ask Copilot` or `/` to trigger Vision analysis.
-

Demo Breakdown

Demo 1: Analyze Architecture Diagrams

High-level System Architecture Diagram

Identified Key Components

- The system design layout will highlight all the key components that have been identified through the analysis of the diagram.

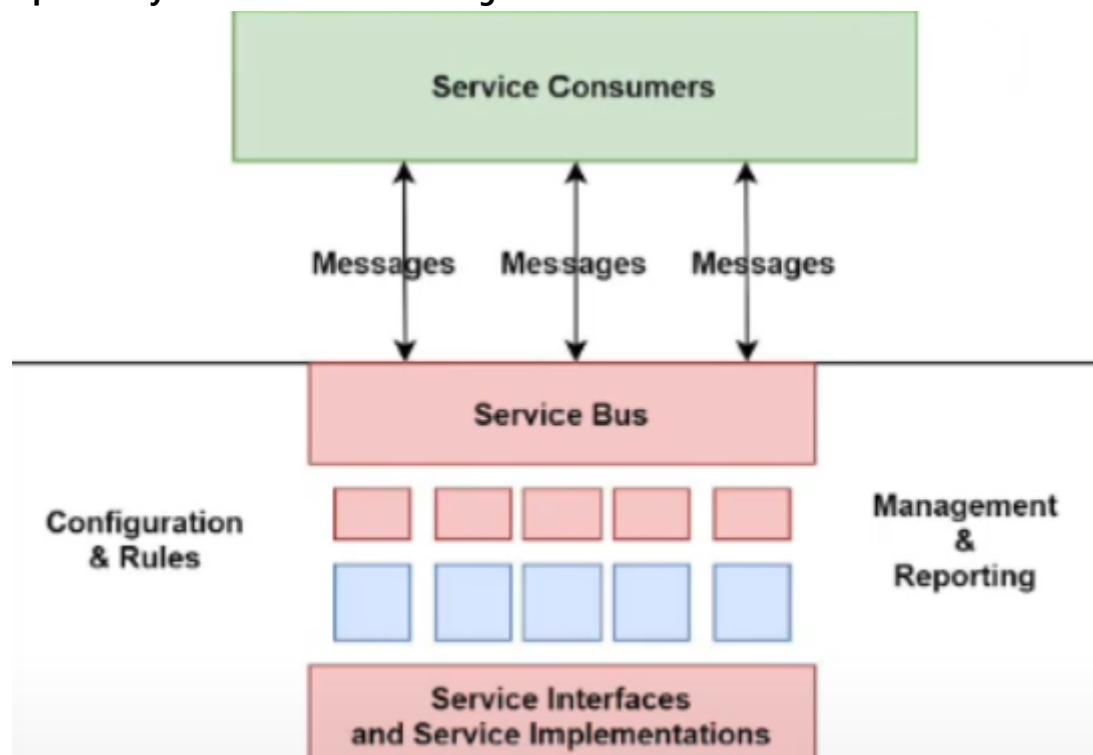
Proposed System Design Layout

- A coherent system design layout will be proposed, showcasing how the identified components interact with each other.

Analytical Capabilities

- The output demonstrates the analytical capabilities of GitHub Copilot Vision in processing and interpreting complex diagrams.

- Upload a system or architecture diagram



Example Prompt:

- Ask Copilot in your IDE

Explain this Architecture

soa-architecture.png ×

+ GPT-4o ▼ ➤

Copilot Output:

- Copilot generates a summary explaining components, flow, and interactions.

Ask Edit : -

Understanding Service-Oriented Architecture

Understanding Service-Oriented Architecture

This diagram represents a **Service-Oriented Architecture (SOA)** with a **Service Bus** as the central component. Here's a breakdown of the architecture:

1. Service Consumers (Top Layer):

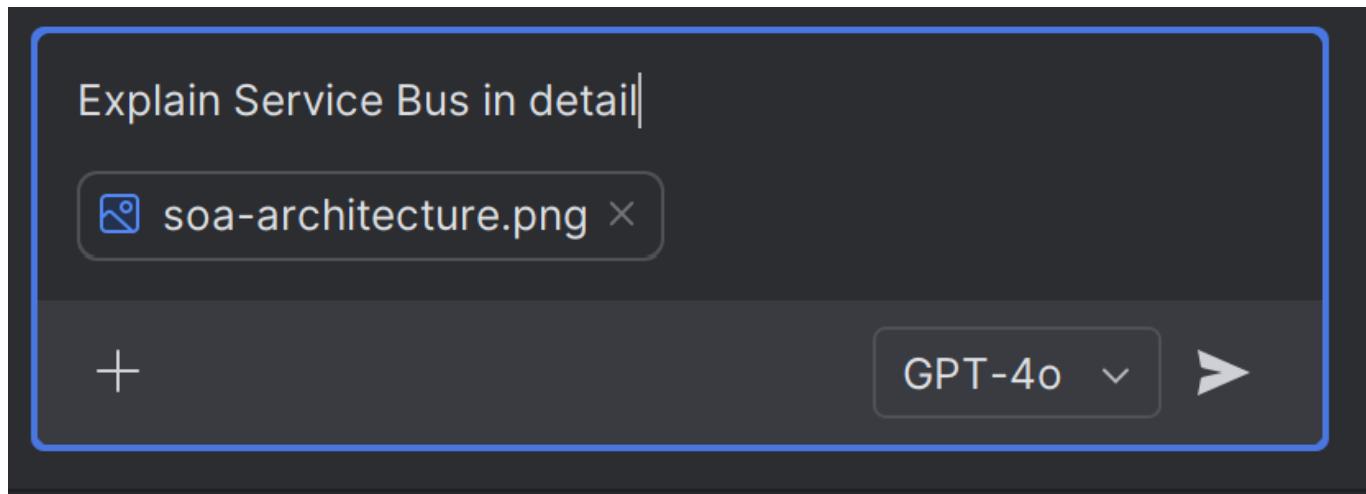
- These are the clients or applications that consume the services.
- They interact with the system by sending and receiving messages through the Service Bus.

2. Service Bus (Middle Layer):

- Acts as a communication layer between service consumers and service implementations.
- Handles message routing, transformation, and protocol mediation.
- Ensures loose coupling between consumers and services.

Example Prompt:

- Ask Copilot in your IDE

**Copilot Output:**

Ask Edit : - ⚡ 🔔

Understanding Service Bus in SOA ⏱ + ⋮ 🔍 📁 🌐 🗃

Key Features of a Service Bus:

1. Message Routing:

- Directs messages from service consumers to the appropriate service providers.
- Supports dynamic routing based on rules or configurations.

2. Protocol Mediation:

- Bridges communication between systems using different protocols (e.g., HTTP, JMS, AMQP).
- Ensures compatibility between heterogeneous systems.

3. Message Transformation:

- Converts message formats to ensure compatibility

Demo 2: Generate Class Files from Class Diagrams

UML Class Diagram

Structure Representation

- The diagram illustrates how classes are structured, including attributes, methods, and associations between them.

Input for Code Generation

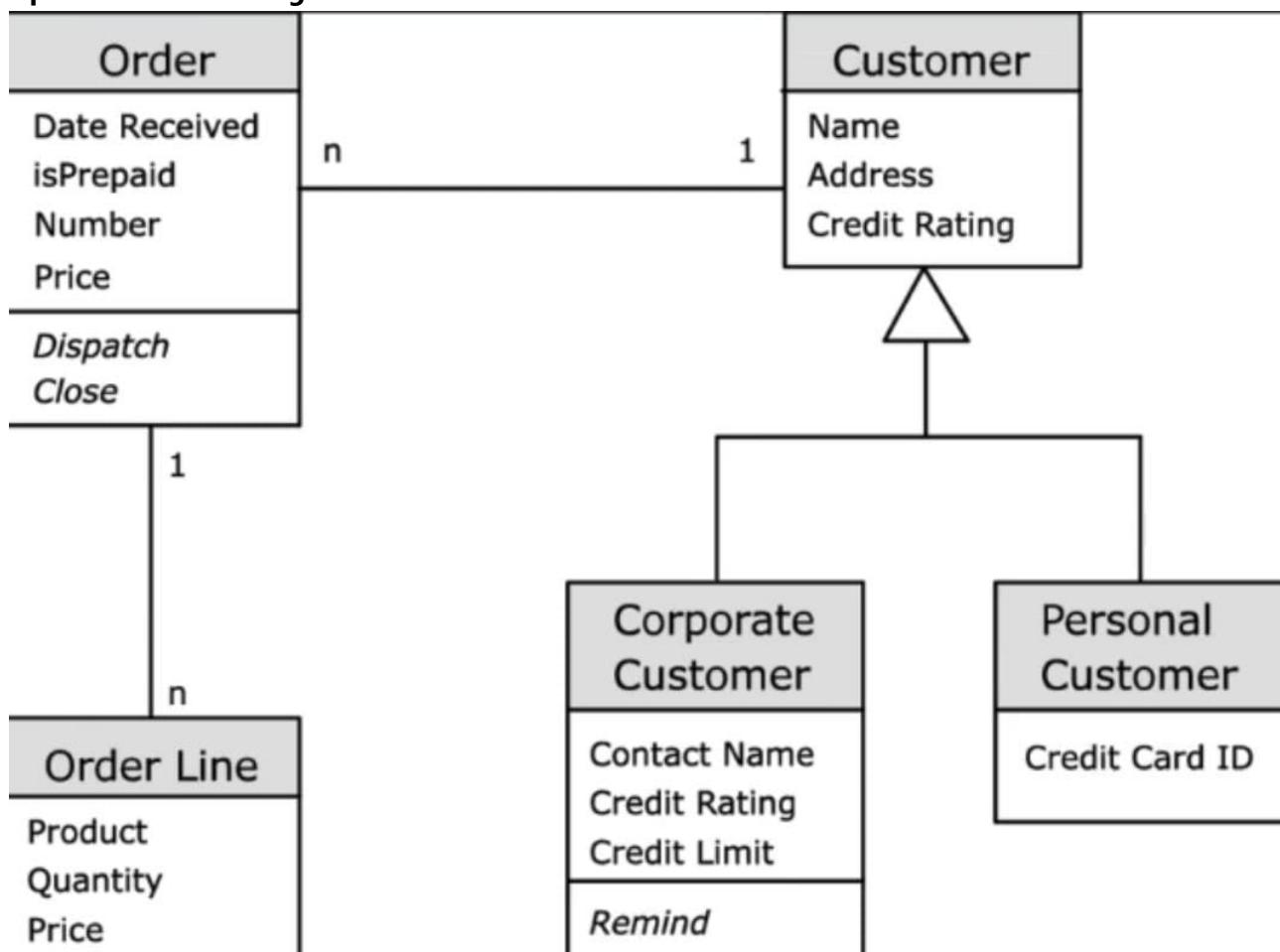
- This UML class diagram serves as a crucial input for automated code generation processes in software development.

Automatic Class Generation

- The tool automatically generates class stubs in Java or Kotlin from a provided UML diagram, saving developers time.

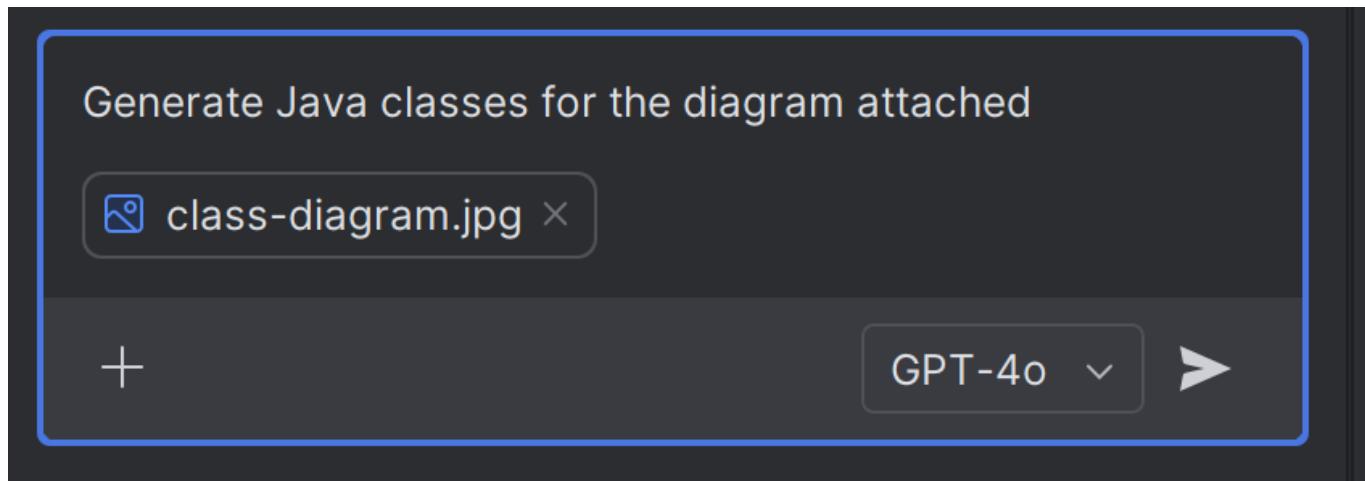
Efficiency and Accuracy

- The generated class stubs maintain high accuracy and efficiency, reflecting the structure defined in the UML diagram.
- Upload UML class diagram**



Example Prompt:

- Ask Copilot in your IDE



Copilot Output:

- Output: Java/Python classes with attributes and methods.

The screenshot shows the Microsoft Copilot interface with a UML class diagram on the left and generated Java code on the right.

UML Class Diagram:

```

classDiagram
    class Order {
        Date Received
        isPrepaid
        Number
        Price
        Dispatch
        Close
    }
    class Customer {
        Name
        Address
        Credit Rating
    }
    class OrderLine {
        Product
        Quantity
        Price
    }
    class CorporateCustomer {
        Contact Name
        Credit Rating
        Credit Limit
        Remind
    }
    class PersonalCustomer {
        Credit Card ID
    }

    Order "n" --> "1" Customer
    Order "n" --> "1" OrderLine
    Customer --> "1" CorporateCustomer
    Customer --> "1" PersonalCustomer
  
```

Generated Java Code (Order Class):

```

package main.java;

import java.util.Date;
import java.util.List;

public class Order {
    private Date dateReceived;
    private boolean isPrepaid;
    private String number;
    private double price;
    private List<OrderLine> orderLines;
}
  
```

Copilot Interaction:

Ask GPT-4o for Java classes based on the provided diagram.

Here are the Java classes based on the provided diagram:

Order Class

Can you provide an example of how to implement the Service Bus in a Java application?

Ask Copilot a question or type '/' for commands

Demo 3: Generate APIs and Models from Use Case Diagrams

Use case Diagram

System Interactions

- The diagram outlines key interactions within the system, providing insights into user roles and functionalities.

Code Generation Process

- Once defined, these interactions can be processed to generate the corresponding code outputs for development.

Automatically Generated API Endpoints

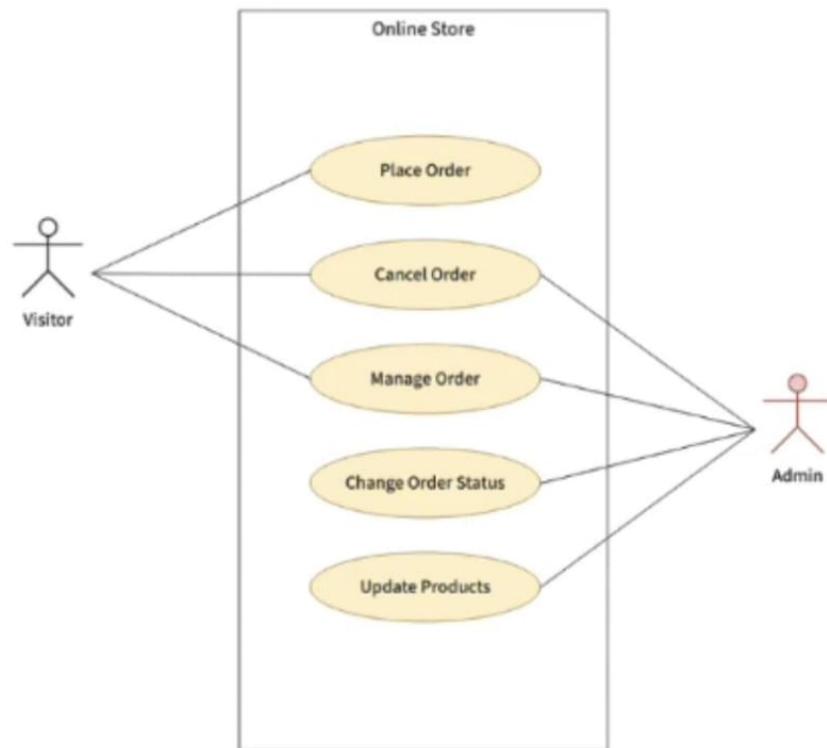
- The tool can generate API endpoints automatically, streamlining the development process and ensuring consistency.

Service Classes Overview

- Service classes define business logic and facilitate communication within the application.

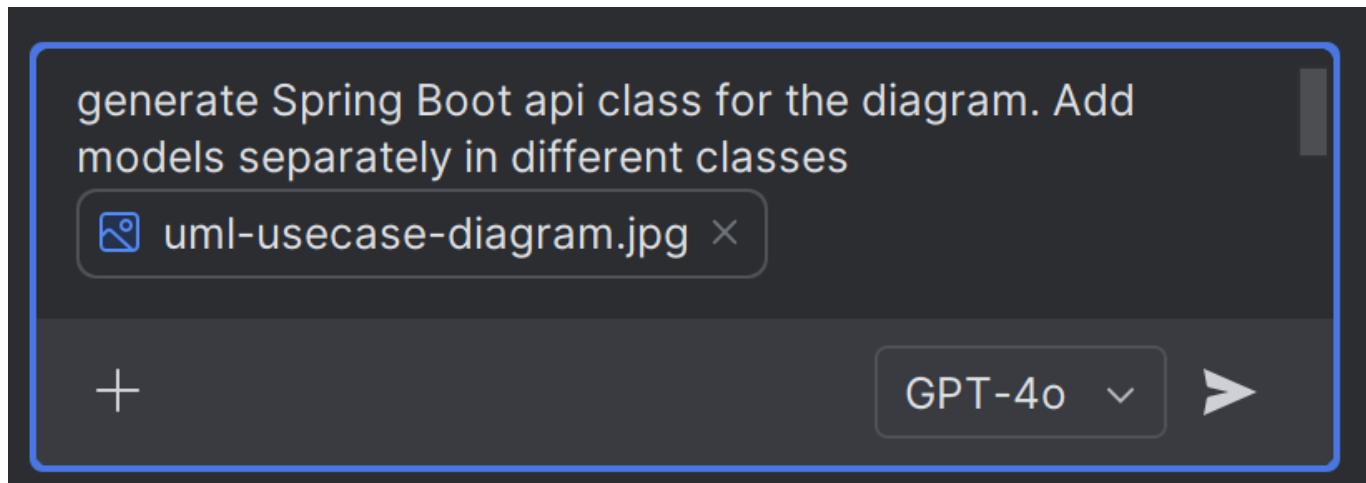
Data Transfer Objects (DTOs)

- DTOs efficiently transfer data between layers, reducing redundancy and improving clarity.
- Upload use case diagram showing actors and use cases**



Example Prompt:

- Ask Copilot in your IDE



Copilot Output:

- Copilot generates controller methods and DTOs.
- Example: `registerUser()`, `loginUser()` API handlers.

The screenshot shows a browser window with a UML use-case diagram on the left and generated API code on the right. The browser tab is titled "uml-usecase-diagram.jpg". The code is annotated with red boxes highlighting specific sections:

```

package com.example.onlinestore.controller;

import com.example.onlinestore.model.Order;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/orders")
public class OrderController {

    @PostMapping("/place")
    public String placeOrder(@RequestBody Order order) {
        // Logic to place an order
    }
}

```

At the bottom of the code editor, there is a question: "Can you help me implement error handling in the API?". A text input field at the bottom says "Ask Copilot a question or type '/' for commands".

Demo 4: Data Modeling from ER Diagrams

Entity Relationship Diagram

Visual Representation of Data

- An ER diagram visually represents data models, showcasing entities and their relationships within the system.

ER Diagram to Schema

- The ER diagram serves as the foundation for generating a complete database schema automatically, improving accuracy.

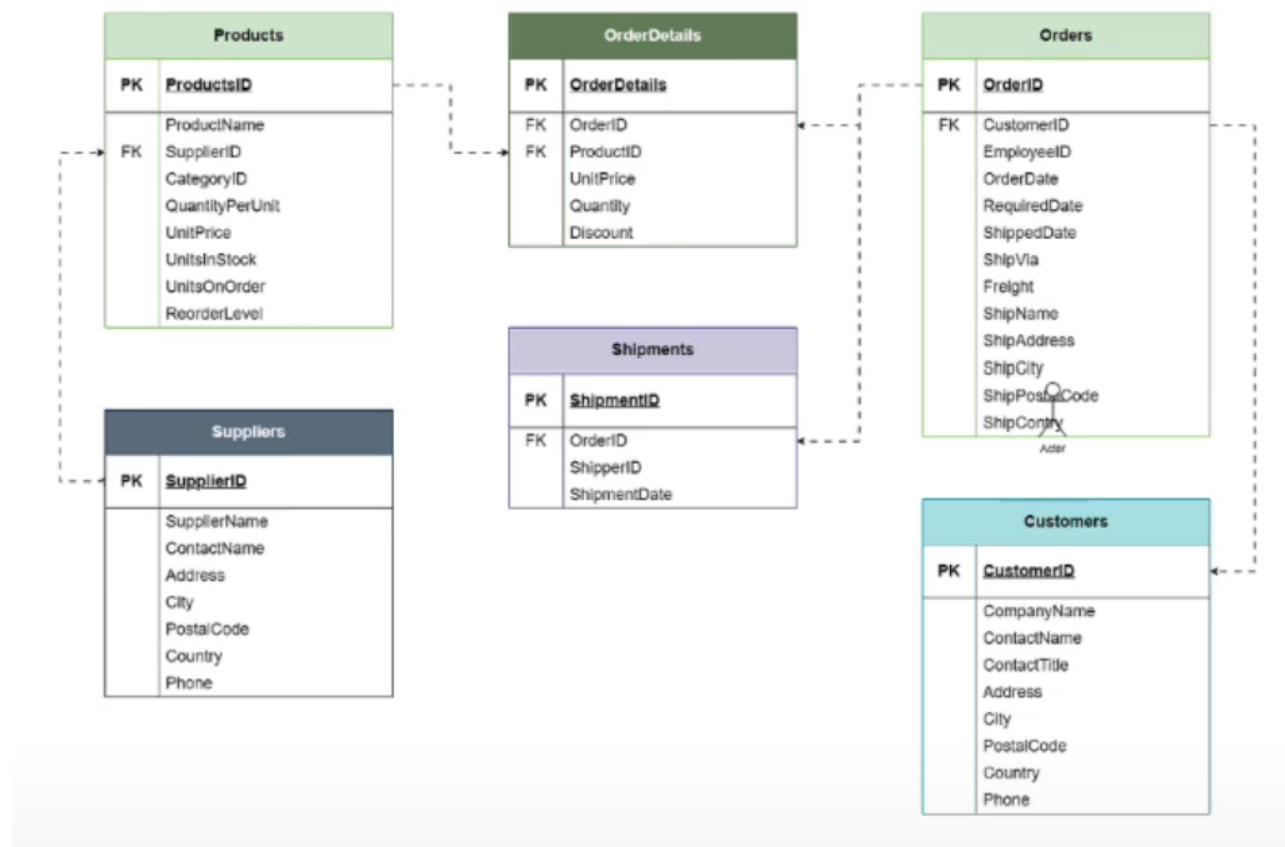
Model Classes Generation

- Copilot generates corresponding model classes from the schema, enabling faster and more efficient development of applications.

Streamlined Development Process

- This automated generation process greatly reduces manual effort, allowing developers to focus on building features.
- Upload ERD with entities and relationships**

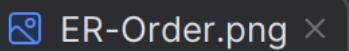
ENTITY RELATIONSHIP DIAGRAM



Example Prompt:

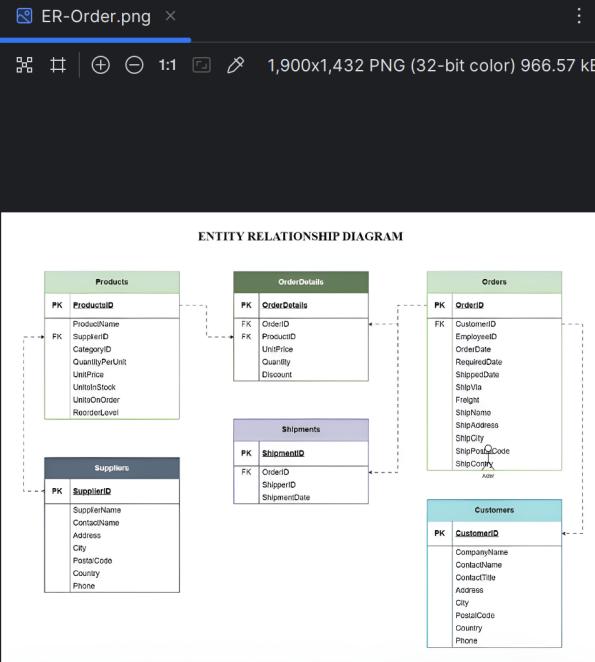
- Ask Copilot in your IDE

generate tsql create schema script for the er diagram|

 GPT-4o ▶

Copilot Output:

- Copilot generates SQL:



Ask Edit

Spring Boot API Development

> 3 steps completed successfully

CREATE SCHEMA Sales;

CREATE TABLE Sales.Products (

ProductID INT PRIMARY KEY,
ProductName NVARCHAR(255) NOT NULL,
SupplierID INT,
CategoryID INT,
QuantityPerUnit NVARCHAR(255),
UnitPrice DECIMAL(10, 2),
UnitsInStock INT,
UnitsOnOrder INT,
ReorderLevel INT,
FOREIGN KEY (SupplierID) REFERENCES Sales.Suppliers(SupplierID)

Can you help me with unit tests for the API controller?

Ask Copilot a question or type '/' for commands

GPT-4o ▶

```
CREATE SCHEMA Sales;

CREATE TABLE Sales.Products (
    ProductID INT PRIMARY KEY,
    ProductName NVARCHAR(255) NOT NULL,
    SupplierID INT,
```

Demo 5: Generate Test Cases from Visual Flows

Flowchart for Process Logic

Visual Outline of Logic

- The flowchart serves as a clear visual representation of the business logic, making it easy to understand complex processes.

Deriving Test Scenarios

- By using the flowchart, Copilot can efficiently derive necessary test scenarios to ensure effective process validation.

Well-Structured Test Cases

- Unit and integration test cases are well-structured to cover various scenarios and edge cases for effective testing.

Thorough Logic Testing

- The application's logic is thoroughly tested through comprehensive unit and integration tests to ensure reliability.

Integration Testing

- Integration tests ensure that different components of the application work together seamlessly.
- Upload logic or decision flowchart.
- Copilot generates unit tests covering decision paths.
- Example: Login validation tests.

Demo 6: Parse Network Topology to Generate Config Scripts

Network Topology Diagram

Visual Representation

- The network topology diagram provides a visual representation of the network setup, helping to understand connections and layout.

Importance for Configuration

- This diagram is crucial for generating configuration scripts, ensuring proper network setup and functionality.

Tailored Configuration Scripts

- Configuration scripts are customised to meet specific networking needs efficiently.

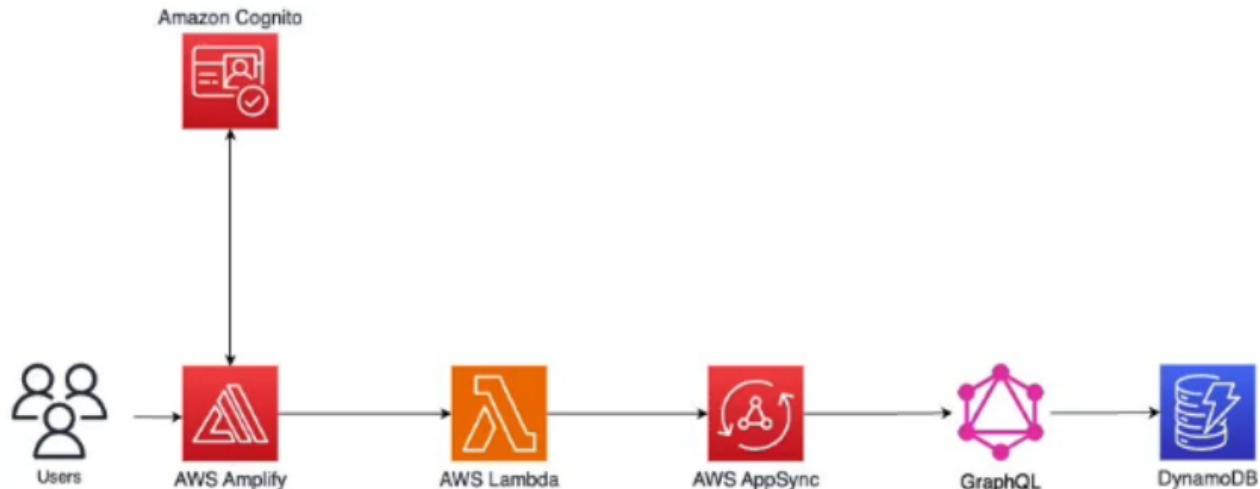
Streamlined Network Setup

- These scripts simplify and accelerate the network setup process, saving time and reducing errors.

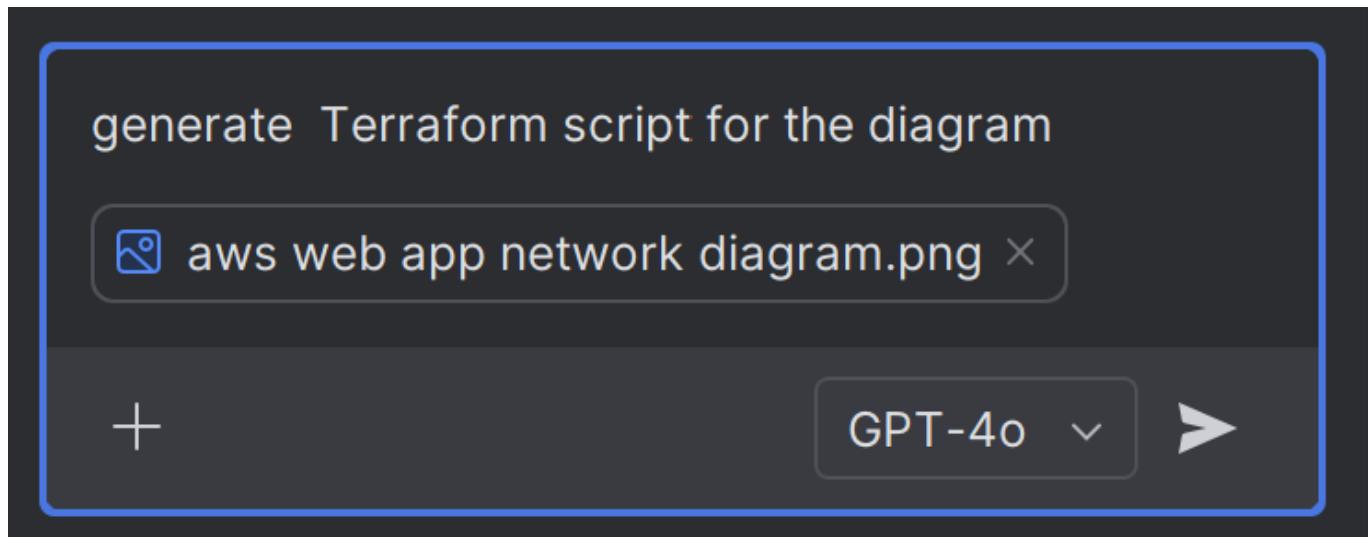
Environment-Specific Solutions

- By focusing on specific environments, scripts ensure optimal performance and compatibility across networks.

- Upload AWS/Azure network diagram

**Example Prompt:**

- Ask Copilot in your IDE

**Copilot Output:**

- Copilot generates Terraform, Bicep, or CLI scripts:

The screenshot shows the AI interface displaying the generated Terraform script. The script provisions resources for a service-oriented architecture, including Amazon Cognito, AWS Amplify, AWS Lambda, AWS AppSync, and DynamoDB. The script uses the AWS provider and defines a Cognito User Pool and its client.

```
provider "aws" {  
    region = "us-east-1"  
}  
  
# Cognito User Pool  
resource "aws_cognito_user_pool" "user_pool" {  
    name = "user-pool"  
}  
  
resource "aws_cognito_user_pool_client" "user_pool_client" {  
    name          = "user-pool-client"  
    user_pool_id = aws_cognito_user_pool.user_pool.id  
}
```

```
resource "aws_lambda_function" "api_backend" {
    function_name = "backend"
    ...
}
```

Demo 7: Screen Design to Frontend Code

Wireframe/Mockup of a UI Screen

Foundation for Development

- Wireframes and mockups are essential for outlining the user interface, guiding developers in the coding process.

Design Elements

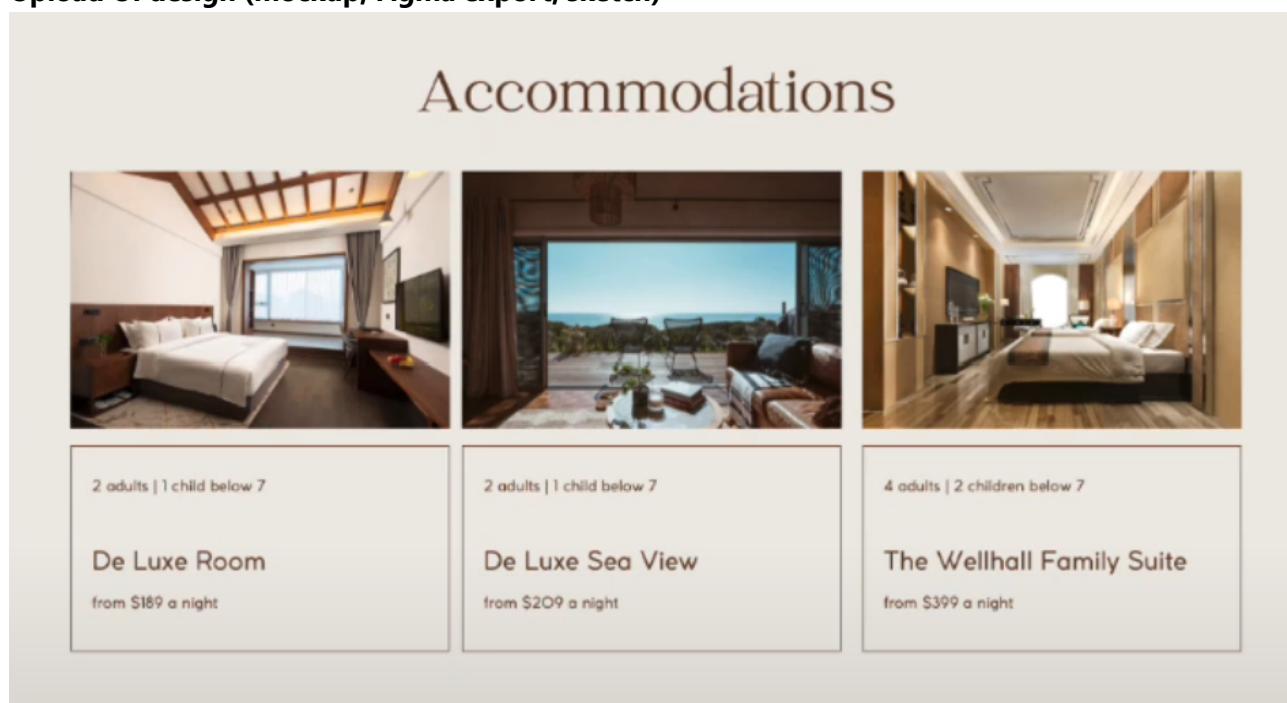
- Mockups incorporate key design elements that illustrate layout, navigation, and user interaction for developers.

Code Generation Basis

- The initial wireframe serves as a baseline for code generation, ensuring alignment between design and functionality.

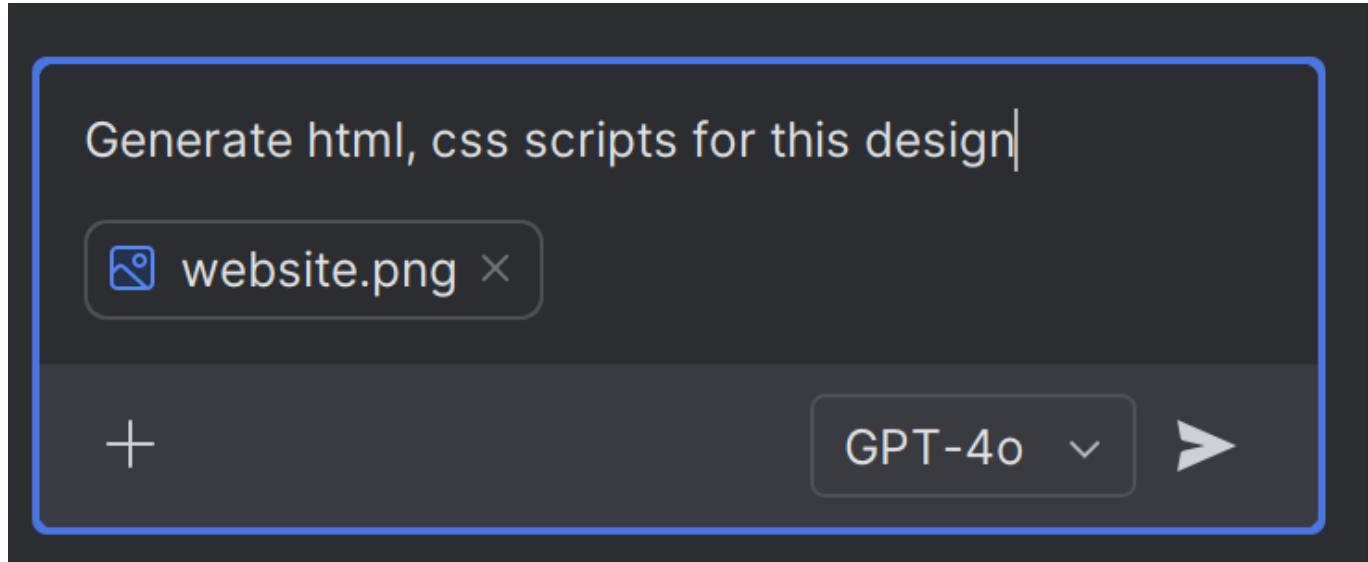
Design to Code Alignment

- The process ensures alignment between design vision and code execution, making sure visual elements function correctly.
- **Upload UI design (mockup, Figma export, sketch)**



Example Prompt:

- Ask Copilot in your IDE

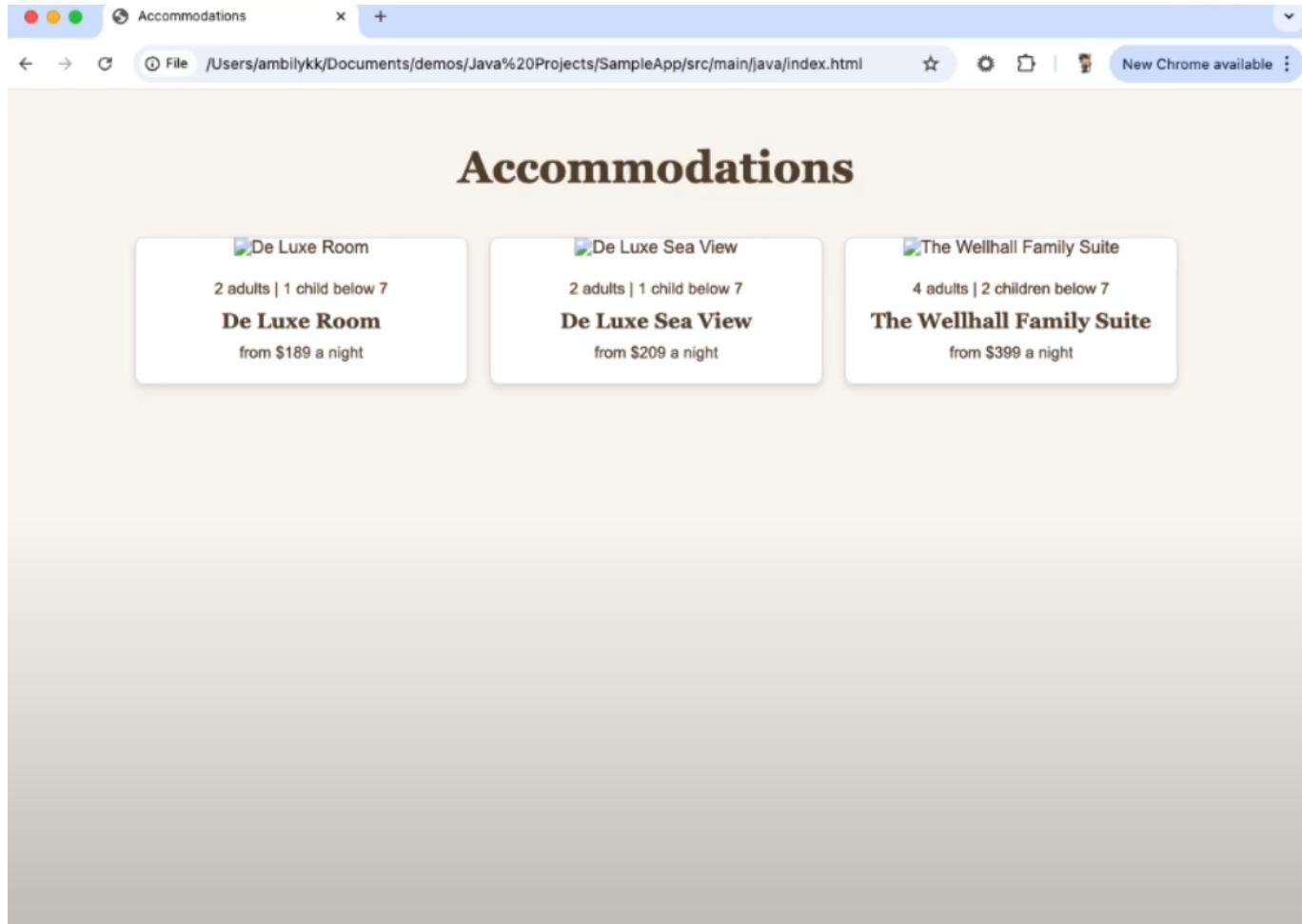


Copilot Output:

- Copilot outputs HTML/CSS code:

A screenshot of the Microsoft Copilot interface showing the generated HTML and CSS code for the "Accommodations" page. On the left, there is a preview of the website showing three room types: "De Luxe Room", "De Luxe Sea View", and "The Wellhall Family Suite". The right side shows the generated code in a code editor window. The code includes the DOCTYPE declaration, HTML and head tags, meta tags for charset and viewport, a title, and a link to a stylesheet. The body contains a container div with a title and a cards div containing three card divs, each with an image and a caption.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Accommodations</title>
  <link rel="stylesheet" href="styles.css">
</head>
```



Demo 8: Modify Existing Code Based on a Diagram

Updated Component Diagram

Component Diagram

- The updated component diagram illustrates the changes made to the system architecture, ensuring clarity in design.

Refactored Code

- The refactored code improves readability and maintainability while aligning with the new structure of the component diagram.

Updated Code Structure

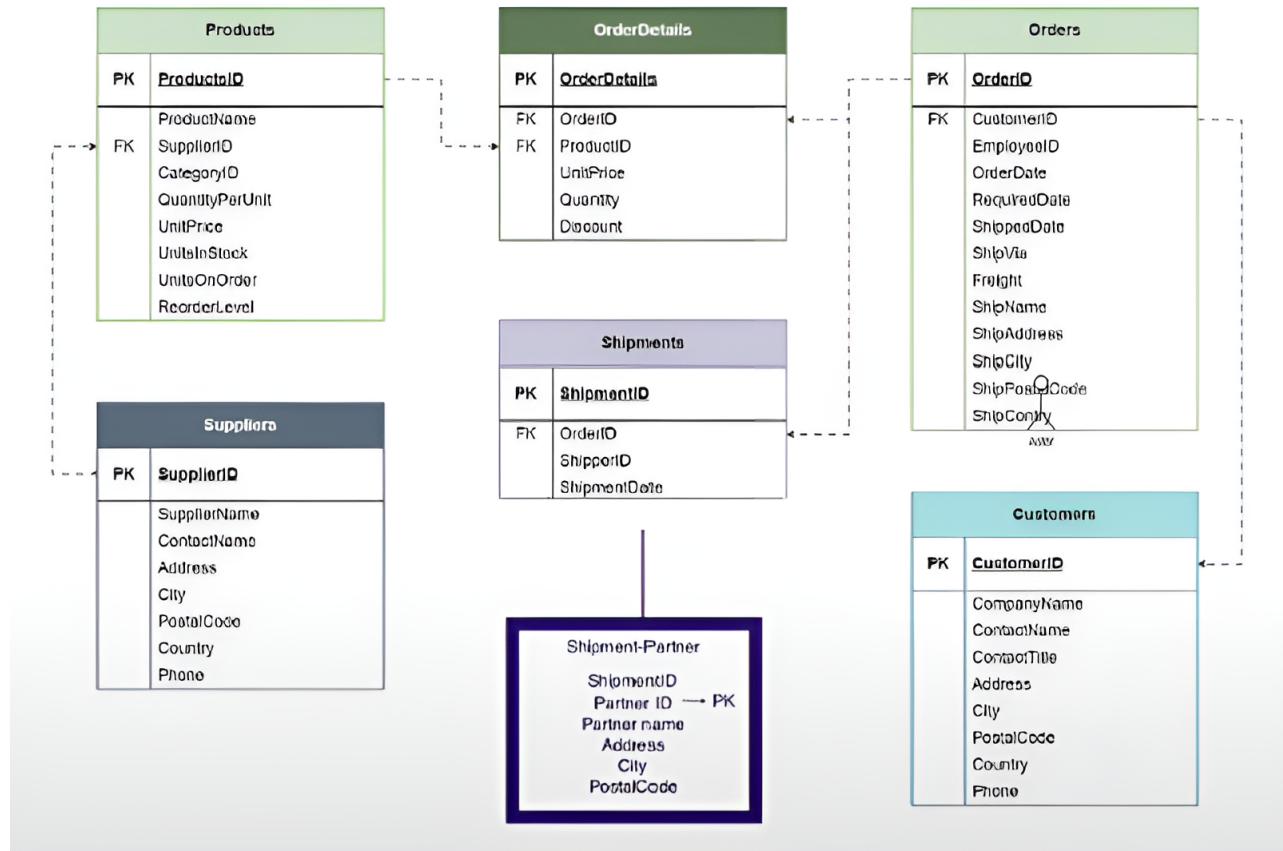
- The updated code structure ensures that all components are consistently implemented across the system, enhancing overall quality.

Ensuring Consistency

- Consistency in code structure is crucial for collaboration and future development, making the system easier to navigate.

- Provide current code + updated diagram

ENTITY RELATIONSHIP DIAGRAM



Example Prompt:

- Ask Copilot in your IDE

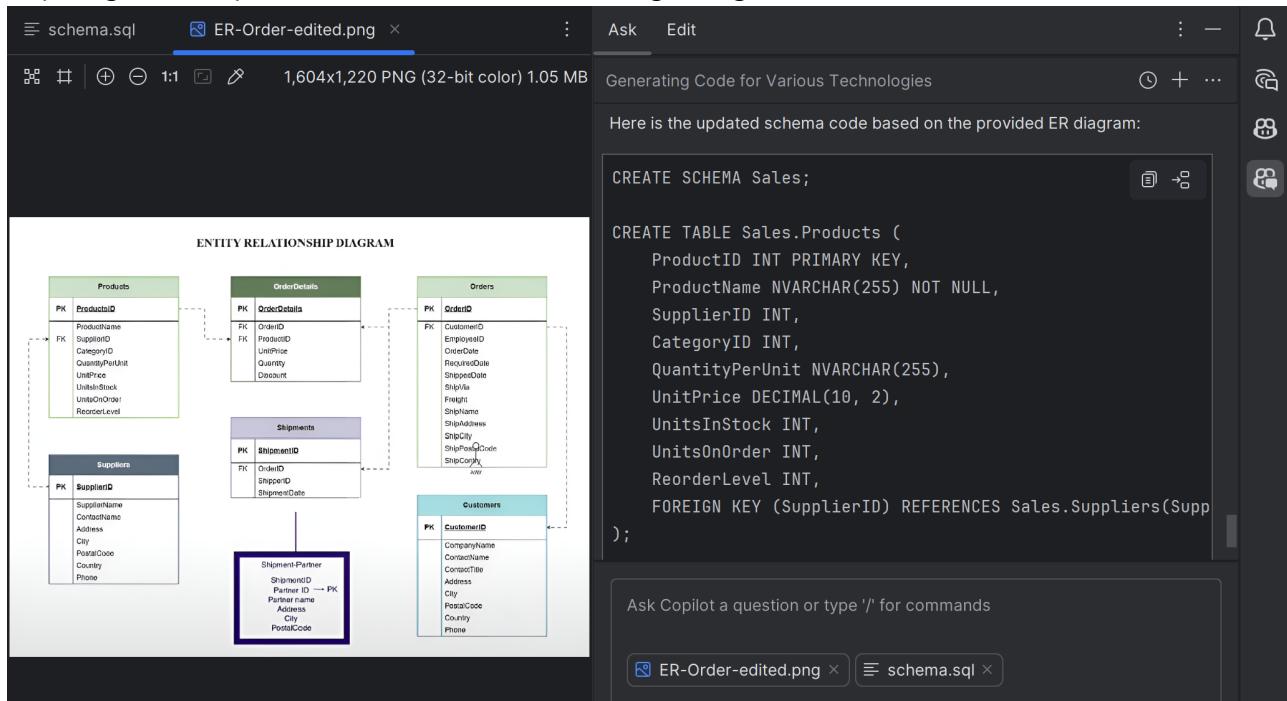
Modify the schema code based on the updaed ER diagram

schema.sql
ER-Order-edited.png

+
GPT-4o
>

Copilot Output:

- Copilot generates patch/diff or new version reflecting changes.



- Useful for updating schemas or API structure.

Demo 9: Hand-Drawn Diagram to Code

Code Scaffolding and Suggestions

Code Scaffolding

- Copilot provides structured code scaffolding, which helps developers build applications efficiently and effectively.

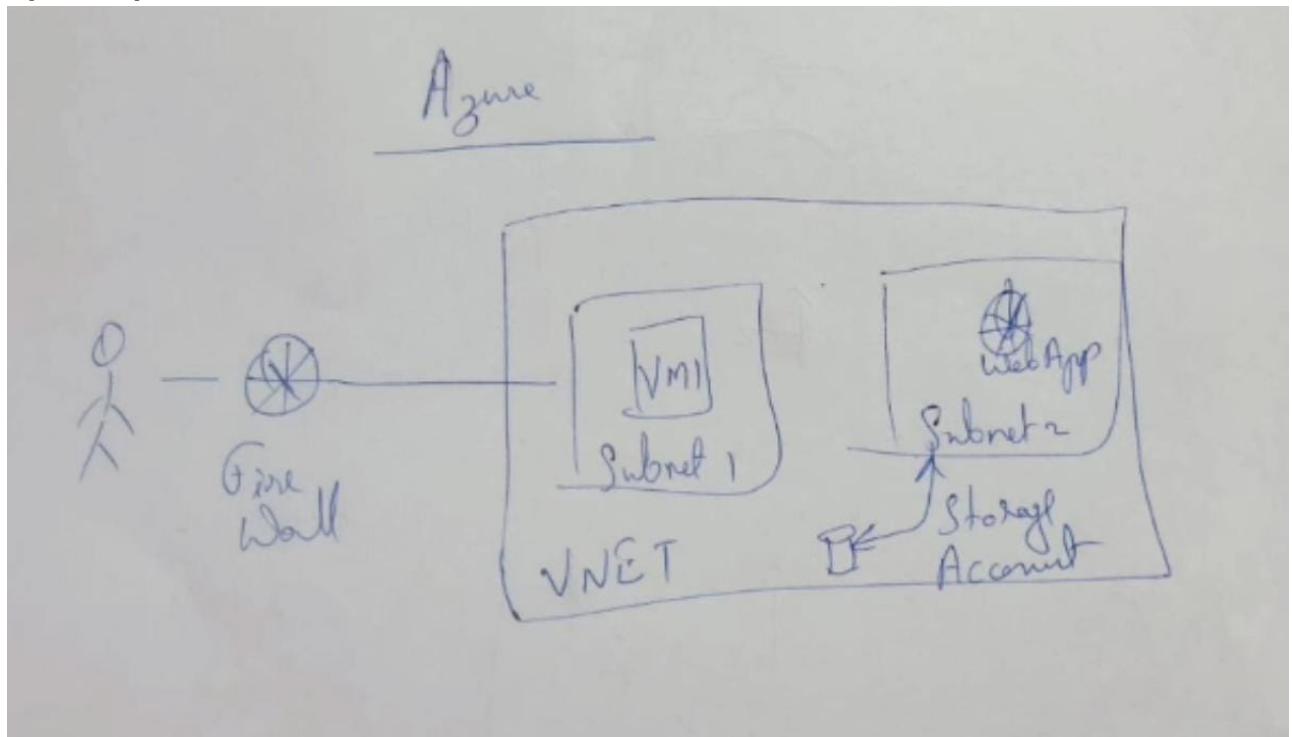
Implementation Suggestions

- The suggestions generated by Copilot guide developers in implementing features, ensuring high-quality code output.

Bridging Design and Code

- The alignment of code output with hand-drawn diagrams bridges the gap between visual design and practical coding.

- Upload a photo of a whiteboard sketch

**Example Prompt:**

- Ask Copilot in your IDE

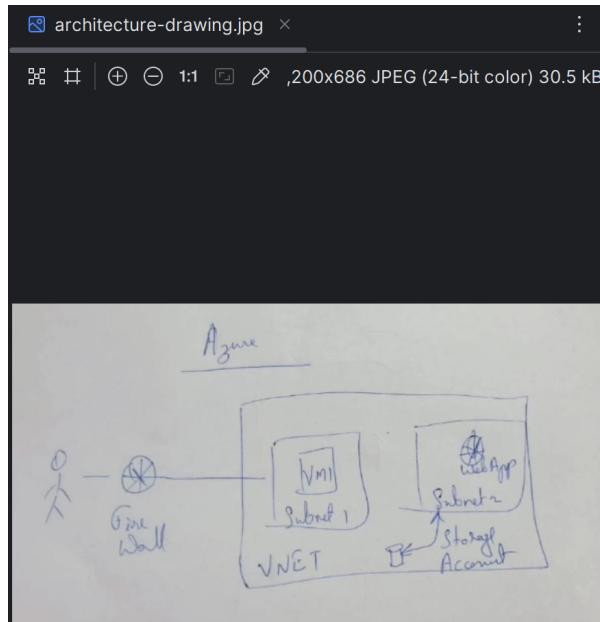
Explain this Diagram

architecture-drawing.jpg X

+ GPT-4o ▾ ▶

Copilot Output:

- Copilot parses drawing → generates usable code or config.



The screenshot shows a hand-drawn sketch of an Azure network architecture. The diagram is titled 'Azure' at the top. On the left, a 'Firewall' is shown with a line connecting it to a 'VNET' (Virtual Network). Inside the 'VNET', there are two subnets: 'Subnet 1' and 'Subnet 2'. 'Subnet 1' contains a 'VM1' icon, and 'Subnet 2' contains a 'WebApp' icon. A 'Storage Account' is also shown connected to 'Subnet 2'. The entire diagram is labeled 'Azure' at the top.

Ask Edit Database Schema and Infrastructure Design

- 1. Firewall:** A security layer that filters incoming traffic from external users before it enters the Azure Virtual Network (VNet). It ensures only authorized traffic is allowed.
- 2. Azure Virtual Network (VNet):** A logically isolated network in Azure that contains two subnets:
 - Subnet 1: Hosts a Virtual Machine (VM1), which could be used for compute or other services.
 - Subnet 2: Hosts a Web App, which is likely a platform for hosting web applications.
- 3. Storage Account:** A storage service in Azure that is connected to Subnet 2. It is used for storing data, such as files, blobs, or application data.
- 4. Traffic Flow:**

Can you help me with a query to retrieve data from the updated schema?

Ask Copilot a question or type '/' for commands

GPT-4o ➔

- Example: Sketch of an API flow becomes Express.js handlers.

Conclusion

Simplifies Coding Process

- GitHub Copilot Vision Revolutionizes Coding by converting visuals into code instantly, making development more efficient.

Integration with IntelliJ

- The integration of GitHub Copilot Vision with IntelliJ enhances productivity and streamlines the coding workflow for developers.

Fostering Collaboration

- GitHub Copilot Vision encourages collaboration between designers and developers, bridging the gap in the development process.