AB-400T00 - Extend Microsoft Power Platform Solutions with code and AI

Overview

Course Duration: 3 Days

About This Course

Extend Microsoft Power Platform solutions with custom code and AI to build modern business applications. Explore advanced extensibility and customization, Power Apps code apps, and the integration of Microsoft Foundry and Copilot Studio agents. Apply core Power Platform development skills alongside AI-powered application development patterns to create connected, intelligent solutions.

Audience Profile

Microsoft Power Platform developers who extend and customize business solutions with code and integrate AI capabilities into applications. This course is intended for pro-developers building custom application experiences and agent integrations, including those preparing for Exam AB-400 toward the Microsoft Certified: Power Platform Developer Associate certification. Prerequisites for this course are that you have working experience with:

- Microsoft Power Platform services
- JavaScript, TypeScript, JSON, and Power Fx
- C#, Python, HTML, and YAML
- RESTful web APIs
- Microsoft Azure
- Microsoft Foundry
- Microsoft Copilot Studio

Course Outline

Module 1: Extend Dataverse with code and Azure compute

Extend Microsoft Dataverse with secure, supportable business logic and data operations. You choose where code runs, harden server-side extensions, diagnose API failures, and move suitable scheduled, event-driven, or long-running workloads to Azure Functions.

- Get started with Microsoft Power Platform development
- Implement business logic with Microsoft Dataverse plug-ins
- Harden Dataverse extensions for production workloads
- Perform reliable operations with Microsoft Dataverse APIs
- Diagnose advanced Microsoft Dataverse API failures
- Process Power Platform workloads with Azure Functions

Module 2: Build professional Power Apps experiences

Deliver connected, secure Power Apps experiences that remain supportable in production. You combine Dataverse-backed code apps and platform services with supported model-driven app scripting and reusable Power Apps component framework controls, then apply deployment, monitoring, and troubleshooting practices.

- Build Dataverse experiences with Power Apps code apps
- Connect Power Apps code apps to platform services
- Deploy and operate secure Power Apps code apps
- Apply client-side logic in model-driven Power Apps
- Build reusable controls with Power Apps component framework

Module 3: Build automation and agent integrations

Build reliable automation and agent integrations for Microsoft Power Platform solutions. You bring together reusable Power Automate logic that handles failures, governed Dataverse Model Context Protocol (MCP) consumption, and supported connections between custom agents, enterprise data, Copilot Studio agents, and applications.

- Implement resilient Power Automate cloud-flow logic
- Consume Dataverse MCP servers from MCP clients
- Integrate custom agents with Microsoft Power Platform

Module 4: Integrate Dataverse with services and external data

Connect Microsoft Dataverse with external systems through reliable data exchanges. You combine event-driven notifications, record synchronization, and secure custom connectors to keep data aligned and make external services available to Microsoft Power Platform apps and flows.

- Publish Microsoft Dataverse events to external services
- Synchronize external data reliably with Microsoft Dataverse
- Connect external APIs with Power Platform custom connectors

Module 5: Design and deliver secure Power Platform solutions

Turn Microsoft Power Platform requirements into secure, supportable components and repeatable releases. You evaluate implementation choices within approved identity and security models, define component contracts, and govern solution delivery through application lifecycle management (ALM).

- Evaluate Microsoft Power Platform solution implementation options
- Design extensible Microsoft Power Platform solution components
- Deliver Power Platform solutions through governed ALM

Prerequisites

- Ability to create Dataverse tables and a model-driven app in a Power Platform developer environment, identify table, column, and relationship logical names, and create and update rows in the app
- Ability to write and build a C# class library using classes, interfaces, and exception handling
- Ability to implement and register a Dataverse plug-in in C#
- Ability to send authenticated HTTP requests and interpret Dataverse API requests, responses (including JSON), and SDK exceptions
- Ability to write and run a basic C# or Python Azure Function
- Ability to create, build, and run React applications with TypeScript, including writing classes and using npm scripts
- Ability to query and update Dataverse rows
- Experience building a Power Apps code app with the Power Apps CLI and a code-first IDE
- Ability to build and run a production-mode web application
- Experience writing and debugging JavaScript functions
- Familiarity with inspecting browser network requests and console output using browser developer tools
- Ability to use a code-app-enabled developer environment with maker access
- Ability to create and test an automated Power Automate cloud flow
- Ability to run a Node.js or Python command-line client
- Ability to modify and run a supplied Python or C# agent project
- Ability to call an authenticated API and inspect JSON tool definitions and structured responses
- Familiarity with Microsoft Foundry, Microsoft Power Platform, Work IQ, and Microsoft Copilot Studio concepts
- Access to an approved, preconfigured Dataverse MCP server
- Ability to read C# plug-in code
- Ability to query Microsoft Dataverse through the Web API or an SDK, read an OpenAPI (Swagger) definition, and call a REST API
- Ability to translate stakeholder needs into functional and nonfunctional requirements for a Microsoft Power Platform solution
- Ability to specify a solution component's API inputs, outputs, errors, identity, and telemetry requirements
- Ability to export a Power Platform solution, unpack it into source files, commit the files to Git, pack the solution, and import it into a nonproduction environment