Check

Async Await Dashboard

☐ M1 — Simulare la latenza di rete implementando `NetworkLatencySimulator.WaitAsync` con `Task.Delay` casuale (150-400 ms) e propagazione del `CancellationToken`.

☐ M2 — Completare i provider asincroni (`GetCurrentAsync`, `GetQuoteAsync`, `GetStatusAsync`) invocando `SimulatedApiClient`, mappando i payload nei record di dominio e propagando il token di cancellazione.

☐ M3 — Aggregare le chiamate ai provider avviandole in parallelo e utilizzare `Task.WhenAll` per costruire un `DashboardSnapshot`.

☐ M4 — Esporre gli eventi diagnostici come `IAsyncEnumerable<DiagnosticEvent>` con `await foreach`, `yield return` e supporto alla cancellazione.

☐ M5 — In `Program.cs` attendere `BuildAsync`, stampare `ToConsoleString()` e iterare `StreamDiagnosticsAsync` limitando l'output a cinque eventi.

GestoreMagazzino.TDD

☐ M1 — Scrivere il test per l'inventario vuoto (TODO `1.T1`) e implementare `Inventario` / `ElencoProdotti()` per restituire lo stato iniziale.

☐ M2 — Aggiungere il test di inserimento prodotti (TODO `2.T1`) e completare `AggiungiProdotto` nei TODO `2.1` e `2.2`.

☐ M3 — Gestire rimozioni e duplicati aggiornando i test `3.T1` / `3.T2` e completando `RimuoviProdotto` (TODO `3.1` – `3.3`) con eccezioni mirate.

☐ M4 — Introdurre quantità aggiornando/aggiungendo test (`4.T1`, `4.T2`), progettando `Prodotto` e gestendo le scorte in `Inventario`.

☐ M5 — Integrare le notifiche: definire `INotificatoreMagazzino`, usare Moq nei test (`5.T1`, `5.T2`) e collegare gli avvisi alla riduzione scorte.

Gestione Banca (OOP)

☐ M1 — Implementare `Cliente` con proprietà in sola lettura e costruttore validante.

☐ M2 — Definire l'astrazione `ContoBancario` con proprietà chiave, collezione di transazioni e metodi per deposito/prelievo/estratto conto.

☐ M3 — Creare `ContoCorrente` con gestione del fido e override di `Preleva` per consentire saldi negativi entro la soglia.

☐ M4 — Implementare `ContoRisparmio` con tasso annuale e metodo `ApplicaInteressi()` che registra la transazione.

☐ M5 — Introdurre `Transazione` (data, importo, tipo, descrizione) e registrare ogni operazione.

☐ M6 — In `Program.cs` orchestrare cliente, conti e operazioni, stampando gli estratti conto in modo polimorfico.

Gestione Studenti (OOP)

☐ M1 — Completare la classe `Studente` con proprietà, costruttore e `AggiungiVoto`, seguendo i TODO indicati.

☐ M2 — Implementare `CalcolaMedia()` gestendo l'assenza di voti e stampare nome completo più media in `Program.cs`.

☐ M3 — Abilitare i test xUnit rimuovendo gli `Skip`, verificando `AggiungiVoto` e `CalcolaMedia`, quindi eseguire `dotnet test`.

☐ M4 *(opzionale)* — Creare `StudenteUniversitario` con `CorsoDiLaurea`, override di `CalcolaMedia()` (+1 oltre 5 voti) e aggiornare `Program.cs`.

☐ M5 *(opzionale "wow")* — Completare `CsvRepository` per salvare `Nome;Cognome;Media` e richiamarlo dopo il calcolo in `Program.cs`.