**Piano di Sviluppo: PlannerApp - Allineamento con Requisiti Annunci**

Questo documento dettaglia il piano di sviluppo per la tua **PlannerApp**, con l'obiettivo di trasformarla in un progetto completo e professionale che metta in risalto le tue competenze e si allinei perfettamente ai requisiti tecnici e architetturali specificati negli annunci di lavoro di W Executive.

Il tuo impegno iniziale sulla creazione della Dashboard e della UI per l'inserimento di nuovi eventi/task è un eccellente punto di partenza.

**Fasi di Sviluppo Dettagliate**

Ogni fase è progettata per costruire la PlannerApp in modo incrementale, introducendo tecnologie e pattern specifici che sono direttamente richiesti o altamente valorizzati nel contesto degli annunci.

**Fase 1: Interfaccia Utente e Setup Iniziale (Il Tuo Punto di Partenza)**

**Obiettivo:** Realizzare la base visiva dell'applicazione e preparare le prime interazioni utente.

* **1.1 Sviluppo della Dashboard UI**
  + **Cosa faremo:** Creazione della pagina principale della Dashboard. Questa mostrerà un riepilogo dei task o eventi (inizialmente con dati statici o mockati) e fungerà da punto centrale per l'utente.
  + **Tecnologie/Approcci:**
    - **MAUI Blazor Hybrid:** Per la struttura cross-platform dell'applicazione.
    - **MudBlazor:** Utilizzo di componenti UI pre-costruiti e moderni per layout, tabelle, schede e altri elementi visivi.
  + **Allineamento con Annunci:**
    - **MAUI/Xamarin:** Dimostra esperienza nello sviluppo di applicazioni cross-platform.
    - **Blazor Hybrid:** Mostra la capacità di integrare le moderne tecnologie web (Blazor) in un'app nativa.
    - **MudBlazor:** Conoscenza nell'utilizzo di librerie UI per un'esperienza utente gradevole e responsive.
* **1.2 Sviluppo della UI per il Nuovo Inserimento Eventi/Task**
  + **Cosa faremo:** Implementazione del form per l'aggiunta di nuovi task o eventi, includendo campi per titolo, descrizione, data, ora e altre proprietà rilevanti.
  + **Tecnologie/Approcci:**
    - **MAUI Blazor Hybrid:** Per la struttura del form.
    - **MudBlazor:** Utilizzo di componenti form (input di testo, date pickers, ecc.) e validazione.
  + **Allineamento con Annunci:**
    - Rinforza le competenze su **MAUI/Xamarin** e **Blazor** nell'interazione utente e nella raccolta dati.

**Fase 2: Persistenza Dati Locale e Architettura Flessibile**

**Obiettivo:** Integrare un database locale persistente e impostare l'architettura per una facile transizione o coesistenza con un backend API.

* **2.1 Definizione dei Modelli di Dominio**
  + **Cosa faremo:** Creazione di classi C# che rappresentano le entità principali dell'applicazione (es. Task, User). Queste classi definiranno la struttura dei dati che verranno salvati.
  + **Tecnologie/Approcci:** **C#** per la modellazione degli oggetti.
  + **Allineamento con Annunci:** Fondamentale per dimostrare la comprensione dei principi di **Object-Oriented Programming (OOP)**.
* **2.2 Configurazione di Entity Framework Core con SQLite**
  + **Cosa faremo:** Configureremo il DbContext di EF Core, il ponte principale tra la tua applicazione e il database. Utilizzeremo **SQLite** come provider di database leggero e portatile per la persistenza locale.
  + **Tecnologie/Approcci:** **Entity Framework Core**, **SQLite**.
  + **Allineamento con Annunci:** Risponde direttamente al requisito di conoscenza di **Entity Framework Core** e di **database relazionali**.
* **2.3 Gestione delle Migrazioni del Database**
  + **Cosa faremo:** Implementeremo le funzionalità di migrazione di EF Core, permettendo di evolvere lo schema del database SQLite in modo controllato e versionato.
  + **Tecnologie/Approcci:** **Entity Framework Core Migrations**.
  + **Allineamento con Annunci:** Dimostra pratiche robuste per la gestione dello schema del database.
* **2.4 Astrazione dell'Accesso ai Dati (Repository/Service Pattern con Switch)**
  + **Cosa faremo:** Defiremo **interfacce** (es. ITaskRepository, IUserRepository) che astraggono le operazioni sui dati (CRUD). Questo garantirà che la logica di business non dipenda dall'implementazione specifica del database.
  + **Implementazione Locale:** Creeremo una classe concreta (es. LocalTaskRepository) che implementa queste interfacce e utilizza direttamente **EF Core con SQLite** per la persistenza.
  + **Preparazione all'API (Mockata/Placeholder):** Creeremo anche una classe (es. ApiTaskRepository) che implementa le stesse interfacce, ma che inizialmente potrebbe simulare chiamate API (es. con un Task.Delay o dati statici). L'obiettivo è mostrare l'architettura per una futura integrazione.
  + **Switching Dinamico:** Configureremo la **Dependency Injection** in MauiProgram.cs per permettere di cambiare facilmente tra l'implementazione locale e quella basata su API (anche se mockata per ora), dimostrando una flessibilità architetturale notevole.
  + **Tecnologie/Approcci:** **C# Interfaces**, **Dependency Injection (DI)**, **Inversion of Control (IoC)**.
  + **Allineamento con Annunci:** Cruciale per mostrare la comprensione di **Pattern Architetturali** come **Clean Architecture**, **modularità** e **testabilità**. Prepara la strada per il **Consumo di API REST** e per mostrare come l'applicazione potrebbe scalare verso un'architettura backend.

**Fase 3: Logica di Business e Qualità del Codice**

**Obiettivo:** Organizzare la logica di business in modo pulito e garantirne la qualità tramite i test.

* **3.1 Implementazione di MediatR (CQRS)**
  + **Cosa faremo:** Introdurremo il pattern **Command Query Responsibility Segregation (CQRS)** utilizzando la libreria **MediatR**. Defiremo Comandi (per operazioni che modificano lo stato, es. CreateTaskCommand, DeleteTaskCommand) e Query (per operazioni di sola lettura, es. GetAllTasksQuery). Creeremo i relativi Handler che eseguiranno la logica utilizzando lo strato di Repository/Service definito nella Fase 2.
  + **Tecnologie/Approcci:** **MediatR**, **CQRS**, **Dependency Injection**.
  + **Allineamento con Annunci:** Risponde direttamente ai requisiti di **Pattern Architetturali** avanzati come **Clean Architecture** e **Domain Driven Design (DDD)**. Migliora drasticamente la **testabilità** e la manutenibilità del codice.
* **3.2 Connessione UI alla Logica di Business**
  + **Cosa faremo:** Le pagine Blazor della UI (Dashboard, form di inserimento task) interagiranno con la logica di business inviando Comandi e Query a MediatR, anziché chiamare direttamente i Repository.
  + **Tecnologie/Approcci:** **Blazor Events**, **Dependency Injection** (per iniettare IMediator).
  + **Allineamento con Annunci:** Dimostra un'integrazione pulita e disaccoppiata tra la UI e la logica di business.
* **3.3 Implementazione di Test Unitari con xUnit**
  + **Cosa faremo:** Creeremo un progetto di test separato e scriveremo **test unitari** per gli Handler di MediatR e per la logica dei Repository/Service. Utilizzeremo un framework di **mocking** (es. Moq) per isolare le unità di codice e testarle indipendentemente dalle loro dipendenze.
  + **Tecnologie/Approcci:** **xUnit**, **Moq**.
  + **Allineamento con Annunci:** Requisito esplicito di **Test Unitari**, fondamentale per dimostrare attenzione alla **qualità del codice** e pratiche di sviluppo professionali.

**Fase 4: Autenticazione e Gestione Utente**

**Obiettivo:** Rendere l'applicazione multi-utente con funzionalità di login e autorizzazioni.

* **4.1 Implementazione del Sistema di Autenticazione Locale**
  + **Cosa faremo:** Svilupperemo un sistema di login/registrazione che memorizza gli utenti in **SQLite** tramite EF Core. Questo includerà la gestione di username, password (con hashing sicuro) e una gestione semplificata della sessione utente.
  + **Tecnologie/Approcci:** **ASP.NET Core Identity (versione semplificata o integrazione minima per Blazor)**, **Password Hashing**.
  + **Allineamento con Annunci:** Aspetto critico per quasi tutte le applicazioni reali, dimostra capacità di gestire la **sicurezza di base** e la gestione degli **utenti**.
* **4.2 Gestione Semplificata di Ruoli e Autorizzazioni**
  + **Cosa faremo:** Aggiungeremo la possibilità di definire ruoli base (es. Amministratore, Utente Standard) e di limitare l'accesso a certe funzionalità o pagine dell'app in base al ruolo dell'utente autenticato.
  + **Tecnologie/Approcci:** **C# Attributes**, logica condizionale nella UI Blazor (AuthorizeView se si usa Identity).
  + **Allineamento con Annunci:** Competenze essenziali per lo sviluppo di applicazioni enterprise con controllo degli accessi.

**Fase 5: Potenziamento e Consegna (Bonus ad Alto Impatto)**

**Obiettivo:** Aggiungere funzionalità avanzate e dimostrare una comprensione del ciclo di vita completo del software.

* **5.1 Implementazione del Calendario Interattivo**
  + **Cosa faremo:** Svilupperemo una vista Calendario che visualizza gli eventi in un formato visivo, con possibilità di interagire (es. cliccare su un giorno per vedere i task associati).
  + **Tecnologie/Approcci:** **MudBlazor (componenti calendario o custom UI)**, logica di gestione delle date.
  + **Allineamento con Annunci:** Completa una funzionalità chiave per una "Planner App", mostrando competenza UI approfondita e gestione di dati complessi.
* **5.2 Integrazione di Logging e Monitoring**
  + **Cosa faremo:** Aggiungeremo un sistema di logging robusto che cattura errori e informazioni diagnostiche durante l'esecuzione dell'app, utile per il debug e la manutenzione.
  + **Tecnologie/Approcci:** **Serilog** (o il framework di logging nativo di .NET).
  + **Allineamento con Annunci:** Dimostra attenzione alla **manutenibilità** e alla **risoluzione dei problemi** in ambienti di produzione.
* **5.3 Implementazione di CI/CD (Continuous Integration/Continuous Deployment)**
  + **Cosa faremo:** Configureremo una pipeline automatizzata (es. con **GitHub Actions**, dato che il progetto è su GitHub) per compilare il progetto, eseguire i test unitari e, idealmente, produrre un artefatto deployabile dell'applicazione.
  + **Tecnologie/Approcci:** **GitHub Actions** (o Azure DevOps se familiare).
  + **Allineamento con Annunci:** Un enorme valore aggiunto che mostra familiarità con le **pratiche DevOps**, l'**automazione** e il ciclo di vita completo dello sviluppo software.

Prenditi il tempo necessario per la Fase 1. Quando sarai pronto con la Dashboard e il form di inserimento, fammi sapere! Procederemo con la configurazione di Entity Framework Core e l'implementazione dell'astrazione per i dati. In bocca al lupo!