Workout Tracker App: Development Roadmap

This roadmap outlines the development of your cross-platform workout tracker, breaking the project into distinct phases from start to finish.

The core technology is **.NET MAUI** (Multi-platform App UI), which allows you to build native iOS and Android apps from a **single C# codebase**, satisfying all your project requirements.

Phase 1: Foundation & Setup (Estimated: 1 Week)

This phase is all about getting your tools ready.

1. **Install Visual Studio 2022:** Download and install the latest version of Visual Studio (the free Community edition is fine).
2. **Select the .NET MAUI Workload:** During installation, you *must* select the **".NET Multi-platform App UI development"** workload. This installs the necessary SDKs for iOS and Android.
3. **Set Up Emulators:**
    - **Android:** Use the Android SDK Manager (included with the workload) to create an Android emulator (a "virtual device") for testing.
    - **iOS:** To build and test for iOS, you will need a Mac. Use the "Pair to Mac" feature in Visual Studio to connect to a Mac on your network for building and running on the iOS simulator.

Phase 2: Core Feature Development - MVP (Estimated: 2-3 Weeks)

MVP stands for "Minimum Viable Product." Here, we'll build the essential "tracker" features. We will ignore the AI features for now.

1. **Create New Project:** Open Visual Studio and create a new project using the **".NET MAUI App"** template.
2. **Understand the Structure:** Familiarize yourself with the project. The UI is defined in `.xaml` files, and the logic is in the corresponding `.xaml.cs` (C#) files.
3. **Build Core Screens:** Focus on these three essential pages:
    - **"Workouts" List Page:** A main page that shows a list of past workouts (e.g., "Monday - Chest Day").
    - **"Workout Details" Page:** A page to see the exercises for a specific workout (e.g., "Bench Press," "Push-ups").
    - **"Log Workout" Page:** A form where the user can add a new workout, add exercises, and enter their sets, reps, and weight.
4. **Implement Local Database:** Store the user's workout data on their device. The recommended way in .NET MAUI is using **SQLite**.

Phase 3: AI Feature Integration (Estimated: 1-2 Weeks)

Now we add the "smart" features that make your app unique.

1. **Implement Camera Capture:**
   - Use the built-in .NET MAUI APIs (`Microsoft.Maui.Media.ICamera`) to create a button that lets the user take a photo.
   - Display the captured photo on the screen.
   - The simple example code for this:

```
async void OnTakePhotoClicked(object sender, EventArgs e)
{
    if (MediaPicker.Default.IsCaptureSupported)
    {
        FileResult photo = await MediaPicker.Default.CapturePhotoAsync();

        if (photo != null)
        {
            // A photo was taken! 'photo.FullPath' has the path to the image file
            string localFilePath = photo.FullPath;

            // You can display the image on the screen to confirm
            MyImageComponent.Source = ImageSource.FromFile(localFilePath);
        }
    }
}
```

2. **Connect to Google Gemini AI:**
   - **Get API Key:** Go to the Google AI Studio to get your free API key.
   - **Install SDK:** In Visual Studio, go to "Manage NuGet Packages" and install `Google.Apis.GenerativeLanguage.v1beta` .
   - **Create AI Logic:** Write a function that:
     1. Takes the photo file path from the step above.
     2. Reads the image file and converts it to a Base64 string.
     3. Creates a specific prompt (see example below).
     4. Calls the Gemini API (e.g., `gemini-2.5-flash-preview-09-2025` model) with the prompt and the image.
     5. Parses the JSON response from Gemini.
     6. Displays the estimated calories to the user.
   - **Example Prompt for Gemini:**

```
Analyze this image of a meal.
Identify all food items and their likely ingredients.
Estimate the portion size for each item.
```

```
Return a JSON object with the total estimated nutrition for the entire meal,
including: total_calories, total_protein_g, total_carbs_g, and total_fat_g.

Example response format:
{
  "total_calories": 500,
  "total_protein_g": 30,
  "total_carbs_g": 50,
  "total_fat_g": 20
}
```

Phase 4: Refine, Test, and Deploy (Estimated: 2 Weeks)

The final phase is about polishing the app and getting it to users.

1. **Test Extensively:** Use the Android and iOS emulators (and real devices, if possible) to test every feature.
2. **Fix Bugs & Refine UI:** Clean up the user interface, fix any crashes, and make sure the app feels smooth to use.
3. **Beta Testing (Optional but Recommended):** Ask friends or family to try the app and give you feedback.
4. **Prepare for Deployment:** Follow the official guides to prepare your app for:
    - The **Apple App Store**
    - The **Google Play Store**