**Holly Trace Application Timeline**

|  |  |
| --- | --- |
| **Week** | **Focus & Deliverables** |
| **1** | **Core Setup & Data Layer** • Install .NET 9 SDK + MAUI workload • Scaffold solution (Domain, Data, UI) • Define EF Core entities (Boards, Skids, Users) and DbContext • Create & apply initial code-first migration to LocalDB • Seed an Admin user in startup logic |
| **2** | **Authentication & DI + Submit Prototype** • Implement simple username/password + role check (BCrypt.NET) • Configure built-in DI for DbContext, services, ViewModels • Build LoginPage.xaml + LoginViewModel • Scaffold SubmitPage.xaml with stubbed ViewModel (dropdowns, entries, date picker, switch) |
| **3** | **Finish Submit Flow** • Hook SubmitViewModel to real EF Core “create board” service • Implement skid-cycler logic and validation • Add “mark shipped” toggle behavior • Optional: basic barcode-scan support via a focused Entry control |
| **4** | **Extract & CSV Export** • Prototype ExtractPage.xaml + ExtractViewModel stub (filters + CollectionView stub) • Wire up EF Core queries with simple paging/sorting • Inline edit on a single row + bulk-update of IsShipped or dates • “Download CSV” button writing JSON → CSV file client-side |
| **5** | **Polish, Packaging & Testing** • Basic styling: ResourceDictionary for colors/fonts, consistent margins • Manual end-to-end test of Login → Submit → Extract → CSV → Login • Write a simple PowerShell “bootstrap.ps1” that ensures LocalDB is installed (or prompts the user) and launches the EXE • Zip the published folder via dotnet publish --self-contained -r win-x64 --self-contained true /p:PublishSingleFile=true and distribute |

**Week 1: Core Setup & Data Layer**

1. **Install & Verify Tooling**
   * Install .NET 9 SDK & MAUI workload.
   * Confirm dotnet --list-sdks shows 9.0.x.
   * Create a blank MAUI solution in VS or via CLI (dotnet new sln -n PCBTracker).
2. **Scaffold Projects**
   * dotnet new classlib -n PCBTracker.Domain → add to solution.
   * dotnet new classlib -n PCBTracker.Data → add to solution.
   * dotnet new maui -n PCBTracker.UI → add to solution.
   * Wire up project references: UI → Services → Data → Domain (even if Services is just a placeholder this week).
3. **Define Your Entities** (in **PCBTracker.Domain**)
   * Board with properties: int BoardID, string SerialNumber, string PartNumber, string BoardType, int SkidID, DateTime PrepDate, DateTime? ShipDate, bool IsShipped.
   * Skid with int SkidID and string SkidName.
   * User with int UserID, string Username, string PasswordHash, string Role.
4. **Configure your DbContext** (in **PCBTracker.Data**)
   * Install Microsoft.EntityFrameworkCore.SqlServer & Design packages.
   * Create AppDbContext : DbContext, add DbSet<Board>, DbSet<Skid>, DbSet<User>.
   * In OnConfiguring, point to LocalDB:

options.UseSqlServer(

@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\PCBTracking.mdf;Integrated Security=True;");

1. **Initial Migration & Database Creation**
   * In CLI:

dotnet ef migrations add InitialCreate -p PCBTracker.Data -s PCBTracker.UI

dotnet ef database update -p PCBTracker.Data -s PCBTracker.UI

* + Verify that LocalDB created PCBTracking.mdf with the three tables.

1. **Seed Admin User**
   * In your MAUI app startup (e.g. MauiProgram.cs), after dbContext.Database.Migrate(), check if Users.Any() → if none, create one User { Username="admin", PasswordHash=Hash("password"), Role="Admin" }.

**Week 2: Authentication, DI & Submit Prototype**

1. **Implement Authentication Service**
   * In **PCBTracker.Services**, create IUserService with Authenticate(username, password) → User and CreateUser(...).
   * Implement UserService using BCrypt.NET to hash and verify passwords via PasswordHash.
2. **Configure Dependency Injection**
   * In MauiProgram.cs, register:

builder.Services.AddDbContext<AppDbContext>();

builder.Services.AddScoped<IUserService, UserService>();

builder.Services.AddScoped<IBoardService, BoardService>(); // stub for now

builder.Services.AddTransient<LoginViewModel>();

builder.Services.AddTransient<SubmitViewModel>();

1. **Build LoginPage**
   * Create LoginPage.xaml with two Entry fields (Username, Password) and a Button bound to LoginCommand.
   * In LoginViewModel:
     + Username, Password props;
     + ICommand LoginCommand that calls IUserService.Authenticate(...);
     + On success, navigate to SubmitPage, passing the authenticated user/role.
2. **Scaffold SubmitPage UI**
   * Add SubmitPage.xaml with placeholder controls:
     + Picker for BoardType;
     + Entry for SerialNumber & PartNumber;
     + DatePicker for PrepDate;
     + Switch for “Mark as Shipped”;
     + Picker for Skid;
     + Button for Submit.
   * Hook up SubmitViewModel in the XAML’s BindingContext.
   * In SubmitViewModel, define the properties (BoardTypes, SerialNumber, etc.) and stub out SubmitCommand without logic.

**Week 3: Finish Submit Flow**

1. **Implement BoardService**
   * In **PCBTracker.Services**, define IBoardService.CreateBoard(BoardDto dto) and implement it to map DTO → entity, Add(), SaveChanges().
2. **Hook SubmitViewModel to Real Service**
   * Inject IBoardService into SubmitViewModel.
   * In SubmitCommand’s Execute, build a BoardDto from bound props and call CreateBoard.
   * On success, clear form or show confirmation alert.
3. **Skid-Cycler Logic**
   * In BoardService (or a dedicated ISkidService), implement GetNextSkidName() by querying Skids.OrderByDescending(ID).First() and incrementing (or looping through a predefined list).
   * Expose a CycleSkidCommand in SubmitViewModel that calls that service and updates SelectedSkid.
4. **Validation & “Mark as Shipped” Behavior**
   * Add simple checks: SN/PN not empty, a Skid selected.
   * If IsShipped is toggled on, set ShipDate = PrepDate (or allow user override).
   * Disable Submit until CanSubmit is true.
5. **Basic Barcode-Scan Support** (optional)
   * Focus the SN Entry, and in its TextChanged handler, detect full barcode (e.g. ends with newline) → auto-fire SubmitCommand.
6. **Manual Testing**
   * Run the app, login as “admin”, and exercise the Submit flow end-to-end, verifying records in the DB via SSMS or Visual Studio’s SQL Object Explorer.

**Week 4: Extract & CSV Export**

1. **Prototype ExtractPage UI**
   * Create ExtractPage.xaml with:
     + Picker for BoardType filter;
     + Two DatePicker controls for PrepDate range;
     + Two DatePicker controls for ShipDate range;
     + Entry for Serial search;
     + A CollectionView or simple ListView stub for results;
     + A Button for “Download CSV.”
2. **Define IBoardQueryService**
   * In **PCBTracker.Services**, add IEnumerable<BoardDto> QueryBoards(filters…).
   * Implement it to accept optional type, date ranges, serial substring, and return a projected DTO.
3. **Hook ExtractViewModel to Real Queries**
   * Inject IBoardQueryService into ExtractViewModel.
   * On “Search” command, call QueryBoards and assign to an ObservableCollection<BoardDto>.
   * Bind that collection to the CollectionView.ItemsSource.
4. **Inline Edit & Bulk Update**
   * Enable CollectionView selection; allow editing of one or more fields inline (e.g. toggle IsShipped).
   * Provide an UpdateSelectedCommand that collects SelectedItems, calls IBoardService.UpdateBoard(...) in a loop or batch, then refreshes the list.
5. **CSV Export Implementation**
   * In ExtractViewModel, implement ExportCsvCommand that:
     + Iterates your current ObservableCollection<BoardDto>,
     + Builds a CSV string (e.g. StringBuilder + string.Join(",")),
     + Uses FilePicker (MAUI Essentials) to let user choose save path,
     + Writes file via File.WriteAllText(path, csv).
6. **Validation & Edge Cases**
   * Handle “no results” gracefully.
   * Confirm overwrite when CSV file exists.
   * Show error toast if write fails.

**Week 5: Polish, Packaging & Testing**

1. **Basic Styling & ResourceDictionary**
   * Create Styles.xaml in Resources: define global styles for Label, Entry, Button, Picker (margins, font sizes).
   * Apply a consistent color palette and spacing (e.g. StaticResource PrimaryColor).
2. **End-to-End Manual QA**
   * Login (Admin + Standard user) → Submit several boards → Extract with various filters → Inline edit → CSV export.
   * Test “Mark as Shipped” toggles and skid cycling.
   * Verify data in SSMS or by re-running queries.
3. **Bootstrap Script**
   * Write bootstrap.ps1 to:
     1. Check for sqllocaldb existence (Get-Command sqllocaldb).
     2. If missing, silently install SqlLocalDB.msi shipped alongside.
     3. Ensure your PCBTracking.mdf is in the same folder or attached.
     4. Launch your MAUI .exe.
4. **Publish & ZIP**
   * Run:

dotnet publish PCBTracker.UI -c Release -r win-x64 \

--self-contained true /p:PublishSingleFile=true -o publish

* + Copy bootstrap.ps1 and SqlLocalDB.msi into publish/.
  + From a clean directory, Compress-Archive -Path publish\\* -DestinationPath PCBTracker-Local.zip.

1. **Final Sanity Check on Clean VM**
   * Copy the ZIP to a fresh Windows VM (no .NET or LocalDB installed).
   * Unzip, run bootstrap.ps1, and walk through login → submit → extract → export.
   * Fix any missing prerequisites or path issues.