User Management — Interview Pack

User Management — Solution Overview (Interview Pack)

Executive Summary

This solution is a clean, layered ASP.NET Core MVC application for managing users with basic CRUD and action logging. It uses EF Core InMemory for a frictionless local run and includes unit tests for the Data, Services, and Web layers.

Architecture

- Web UI (MVC): Controllers and Razor views
- Services (Domain): Business logic via `IUserService` and `ILogService`
- Data Access: EF Core InMemory `DataContext` behind `IDataContext`
- DI: Extension methods register all services with scoped lifetimes

Projects

- `UserManagement.Web` (ASP.NET Core MVC)
  - Controllers: `HomeController`, `UsersController`, `LogsController`
  - Views: Razor pages for Users and Logs
  - Startup: `Program.cs` registers Data + Domain services and MVC, sets middleware
- `UserManagement.Services`
  - Interfaces: `IUserService`, `ILogService`
  - Implementations: `UserService`, `LogService`
  - DI: `AddDomainServices()`
- `UserManagement.Data`
  - Entities: `User`, `LogEntry`
  - Data abstraction: `IDataContext`
  - EF Core: `DataContext` (InMemory) + seeding
  - DI: `AddDataAccess()`

Data Model

- `User`
  - `Id: long`, `Forename: string`, `Surname: string`, `Email: string`, `IsActive: bool`, `DateOfBirth: DateTime?`
- `LogEntry`
  - `Id: long`, `UserId: long?`, `Action: string`, `Description: string?`, `CreatedAtUtc: DateTime`

Services (Key Functions)

- `IUserService`
  - Sync: `FilterByActive`, `GetAll`, `GetById`, `Add`, `Update`, `Delete`
  - Async: `GetAllAsync`, `GetByIdAsync`, `AddAsync`, `UpdateAsync`, `DeleteAsync`
- `ILogService`
  - Sync: `GetAll(skip,take)`, `GetByUser(userId,skip,take)`, `GetById`, `Log`
  - Async: `GetAllAsync`, `GetByUserAsync`, `GetByIdAsync`, `LogAsync`

Web Controllers (Key Actions)

- `HomeController`
  - `Index()` — Home page
- `UsersController`
  - `List()` — All users (async)
  - `ListActive()` / `ListInactive()` — Filtered
  - `Add()` GET / `Add(model)` POST — Create user + log "Created"
  - `View(id)` — Details + log "Viewed"
  - `Edit(id)` GET / `Edit(model)` POST — Update + log "Updated"
  - `Delete(id)` GET / `DeleteConfirmed(id)` POST — Delete + log "Deleted"
- `LogsController`
  - `Index(page,pageSize)` — Paged log list
  - `Details(id)` — Log detail
  - `ForUser(userId,page,pageSize)` — Logs for a user

Endpoints

- `/` — Home
- `/users` — List

- `/users/active` — Active
- `/users/inactive` — Inactive
- `/users/add` (GET/POST) — Create
- `/users/{id}/view` — Details
- `/users/{id}/edit` (GET/POST) — Edit
- `/users/{id}/delete` (GET/POST) — Delete
- `/logs` — Logs list
- `/logs/{id}` — Log details
- `/logs/user/{userId}` — Logs for a user

How to Build and Run

```
dotnet restore
dotnet build
dotnet test

# Run Web app
dotnet run --project UserManagement.Web/UserManagement.Web.csproj --urls http://localhost:50
# Open http://localhost:5010/
```

Testing

- Unit tests cover Data, Services, and Web layers.
- Run with: `dotnet test`

Design Decisions

- EF Core InMemory for speed and zero setup
- Scoped DI lifetimes to align with web request scope
- Async APIs provided alongside sync for easy upgrade to a real DB
- Simple Bootstrap-based UI

Limitations / Next Steps

- InMemory storage; consider SQL + EF migrations
- Minimal validation/auth; add authentication/authorization
- Add server-side paging/sorting/filtering for users
- Introduce DTOs and mapping (e.g., AutoMapper)

Appendix: Function Reference

See `docs/FunctionReference.md` for the full list of classes and methods.