Brief Documentation

Each Layer in the Project

1. **Domain Layer** (TaskManagementSystem.Domain): Core entities (TaskItem) and interfaces (IGenericRepository<T>, ITaskRepository, IUnitOfWork). Defines contracts for data access without implementation details. Updated ITaskRepository to inherit from IGenericRepository<TaskItem>, allowing it to use the generic CRUD methods without redeclaration.
2. **Application Layer** (TaskManagementSystem.Application): Business logic layer. Includes ITaskService and TaskService implementation, which orchestrates operations using the Unit of Work. This layer handles any potential business rules (e.g., validation) and decouples UI from data access. No changes here, but it now indirectly uses the inherited generic methods via ITaskRepository.
3. **Infrastructure Layer** (TaskManagementSystem.Infrastructure): Persistence concerns. Includes TaskDbContext for EF Core setup, GenericRepository<T> as a base implementing common CRUD, TaskRepository inheriting from GenericRepository<TaskItem> and implementing ITaskRepository, and UnitOfWork managing repositories and transactions (SaveAsync). Reintroduced IGenericRepository<T> and GenericRepository<T>; TaskRepository now inherits from GenericRepository<TaskItem> to reuse base functionality while implementing the specific interface.
4. **Presentation Layer** (TaskManagementSystem.Web): MVC UI. TasksController injects ITaskService for operations. Views handle rendering (list, forms for create/edit/delete). Uses standard MVC patterns with async actions. Added _ViewImports.cshtml to enable built-in ASP.NET Core Tag Helpers across views (e.g., asp-for, asp-action), which are used in the Razor views for form handling and validation. No changes here.

How the Repository is Implemented

- IGenericRepository<T>: Defines generic CRUD methods (async where appropriate), including FindAsync for predicate-based queries.
- GenericRepository<T>: Implements the generic interface using EF Core's DbSet<T>. Supports basic operations like add, update, delete, find by predicate, get by ID, and get all.
- ITaskRepository: Inherits from IGenericRepository<TaskItem>, inheriting all generic methods without adding task-specific ones (can be extended if needed).

- TaskRepository: Inherits from GenericRepository<TaskItem> (which provides the implementations) and implements ITaskRepository. This allows reuse of generic code while providing a type-specific repository.

How Unit of Work is Implemented

- IUnitOfWork: Exposes the specific ITaskRepository (Tasks) and SaveAsync for committing changes.
- UnitOfWork: Lazily initializes the TaskRepository, uses shared DbContext for transactions. Calls SaveChangesAsync to persist. Updated to instantiate TaskRepository which now leverages the generic base.

How CRUD Operations Work

- Operations flow: Controller -> Service -> UoW/Repository -> DbContext.
- **Create**: Form post to controller, service adds via repo and saves via UoW.
- **Read**: Service fetches via repo; controller passes to view.
- **Update**: Form post, service updates via repo and saves.
- **Delete**: Confirmation post, service deletes via repo and saves.
- All async, with model validation in MVC. No changes here, but CRUD now uses inherited generic methods.

Bonus Tasks (Optional, Basic Implementation)

To add filter/sort:

- Since ITaskRepository inherits FindAsync, use it for filters (e.g., FindAsync(t => t.IsCompleted == completed)).
- For sort, extend with custom methods in ITaskRepository/TaskRepository if needed (e.g., GetAllAsync with OrderBy via LINQ).
- Update ITaskService/TaskService to expose and call the method.
- In Index action: Pass query params to service.

Data Seeding

- Added a TaskSeeder class in the Infrastructure layer (TaskManagementSystem.Infrastructure/Seeders/TaskSeeder.cs) to populate the database with 10 realistic sample tasks for testing purposes. These include everyday tasks like grocery shopping, work reports, and personal errands, with varied completion statuses, creation dates (simulating past entries), and due dates (some overdue, some upcoming).

- The seeder checks if data already exists to avoid duplicates and uses async operations for efficiency.
- Integrated into the application startup in Program.cs (Web layer) via a service scope to run automatically on app launch, ensuring the database is seeded when running the project (e.g., for development or testing). This can be removed or conditionalized for production.