# 1) MVP features (what to build first)

These are the things you **must** have for a usable Task Management System.

### 1.1 Projects (MVP)

**User story:** As a user I can create and manage projects so tasks are organized.  
**Acceptance:** Create, Read, Update, Delete projects; get list of projects; project has owner and optional members.

### 1.2 Tasks (MVP)

**User story:** As a user I can create tasks inside projects, assign people, set due date, priority and status.  
**Acceptance:** CRUD tasks; assign/unassign; change status (ToDo → InProgress → Done); filter by project/assignee/status/due date.

### 1.3 Authentication & Authorization (MVP)

**User story:** Users must sign up/log in; actions respect roles (Admin vs User).  
**Acceptance:** JWT login/register; role-based endpoints; protected endpoints require token.

### 1.4 Task comments & activity (MVP)

**User story:** Users can comment on tasks and see history of actions.  
**Acceptance:** Add comments, store activity log entries for create/update/status-change.

### 1.5 Basic search & filters (MVP)

**User story:** Find tasks by title, label, or assignee.  
**Acceptance:** Search endpoint; filters for status, priority, due date.

### 1.6 File attachments (optional but common)

**User story:** Attach files to tasks.  
**Acceptance:** Upload + associate to task, download, and remove.

### 1.7 Notifications (basic)

**User story:** Notify assignee when task is assigned or due soon.  
**Acceptance:** Simple email or in-app notifications queue.  
  
  
  
**2) “Nice-to-have” (v1+)**

Kanban board (drag & drop).

Subtasks / checklists.

Recurring tasks.

Calendar sync / iCal export.

Bulk import/export (CSV)

Real-time updates (SignalR).

Analytics / reports.

# 3) Roles & Permissions

**Admin** — full access (manage users/projects/global settings).

**ProjectOwner** — create projects, assign members, delete project resources.

**Member/User** — create/manage own tasks, comment, upload attachments.

**Guest** (read-only) — optional.

# 4) Folder-by-folder: what to implement (concrete files/classes)

src/

├─ Domain/TaskManagementSystem.Domain/

├─ Application/TaskManagementSystem.Application/

├─ Infrastructure/TaskManagementSystem.Infrastructure/

├─ Infrastructure/TaskManagementSystem.Persistence/

└─ Presentation/TaskManagementSystem.API/

### Domain (TaskManagementSystem.Domain)

**What to put:**

Entities/TaskItem.cs, Entities/Project.cs, Entities/UserProfile.cs, Entities/Comment.cs, Entities/Attachment.cs, Entities/Label.cs, Entities/ActivityLog.cs, Entities/SubTask.cs

ValueObjects/Email.cs, etc.

Enums/TaskStatus.cs, Priority.cs

Events/TaskAssignedEvent.cs (domain events)

Domain validations/business rules (methods inside entities or domain services).

**Keep domain pure** — no EF attributes or framework code. Use simple POCOs.

### **Application (TaskManagementSystem.Application)**

**What to put:**

Interfaces/ → ITaskRepository.cs, IProjectRepository.cs, IEmailService.cs, IFileStorageService.cs, IApplicationDbContext.cs (Db access abstraction).

DTOs/ → TaskDto, ProjectDto, CreateTaskRequest, UpdateTaskRequest.

Commands/ and Queries/ (or Services/ if not using MediatR):

Commands/Tasks/CreateTaskCommand.cs, Handlers/CreateTaskHandler.cs

Queries/Tasks/GetTasksQuery.cs, Handlers/GetTasksHandler.cs

Mappings/AutoMapperProfile.cs

Validators/ (FluentValidation).

Application-level exceptions.

**Rule:** Application references Domain only.

### **Persistence (TaskManagementSystem.Persistence)**

**What to put:**

Context/AppDbContext.cs : DbContext, IApplicationDbContext

Configurations/ → EF Fluent API entity configs

Migrations/ → EF Migrations

Optionally repository implementations if you use repository pattern (TaskRepository implementing ITaskRepository)

**Important:** This project references Application (for interfaces) and Domain (for entity types).

### Infrastructure (TaskManagementSystem.Infrastructure)

**What to put:**

Services/EmailService.cs, FileStorage/AwsS3FileStorage.cs or local disk, NotificationService.cs

Integrations/ → 3rd party integrations

Logging wrappers, caching implementations (Redis), background job workers (Hangfire/Quartz)

Implementation of cross-cutting services referenced by Application.

References: Application, Domain.

### Presentation (TaskManagementSystem.API)

**What to put:**

Controllers/ProjectsController.cs, TasksController.cs, AuthController.cs, UploadController.cs

Program.cs or Startup.cs (DI registrations)

DTO mapping usage

Middlewares/ExceptionMiddleware.cs, Middlewares/RequestLoggingMiddleware.cs

Swagger, CORS, JWT configuration