TaskManager

1. Documento Básico del Proyecto

El proyecto **TaskManager** es una aplicación web API, diseñada para la gestión colaborativa de tareas, proyectos y equipos de trabajo. Su propósito central es permitir a los usuarios registrar, organizar y dar seguimiento a las actividades dentro de proyectos definidos. El diseño se adhiere a una **Arquitectura en Capas** y principios de **Clean Architecture**, asegurando la separación de responsabilidades, la mantenibilidad y la escalabilidad del sistema. La interacción se realiza a través de una API RESTful, que utiliza DTOs y validaciones robustas para garantizar la integridad de los datos.

2 Los casos de uso identificados

El proyecto TaskManager fue desarrollado en un orden de prioridades: Tareas (TaskEntity), Usuarios (User) y Asignaciones (TaskAssignment).

2.1 Casos de Uso del Controlador de Tareas (TaskEntityController)

Esta es la funcionalidad central de la aplicación, cubriendo las operaciones CRUD y consultas avanzadas para la entidad TaskEntity.

- **Listar Todas las Tareas** GetTasksDtoMapper()
  Recuperar una lista completa de todas las tareas registradas en el sistema.
- **Obtener Tarea por ID** GetTaskEntityDtoMapperById(int id)
  Consultar y obtener los detalles específicos de una sola tarea usando su identificador único.
- **Crear Nueva Tarea** InsertTaskEntityDtoMapper(TaskEntityDto taskEntityDto)
  Registrar una nueva tarea en el sistema, aplicando validaciones de datos (Título, Proyecto, etc.) antes de la persistencia.
- **Actualizar Tarea** UpdateTaskEntityDtoMapper(int id, [FromBody] TaskEntityDto TaskEntityDto)
  Modificar los datos de una tarea existente. Incluye la regla de negocio de prevenir el cambio de proyecto.
- **Eliminar Tarea** DeleteTaskEntityDtoMapper(int id)
  Suprimir permanentemente un registro de tarea del sistema.
- **Buscar Tareas por Texto** SearchTasks([FromQuery] string text)
  Realizar una búsqueda rápida de tareas que contengan una palabra clave en su título o descripción.
- **Obtener Tareas por Proyecto (y/o Estado)** GetAllProjectTasks(int projectId, [FromQuery] int? statusId)
  Listar todas las tareas asociadas a un projectId específico, permitiendo un filtro opcional por statusId.

2.2 Casos de Uso del Controlador de Usuarios (UserController)

Se enfoca en la gestión de la entidad User, esencial para la colaboración y el acceso al sistema.

- **Listar Todos los Usuarios** GetAllUsers()

Recuperar una lista completa de todos los usuarios registrados en el sistema.

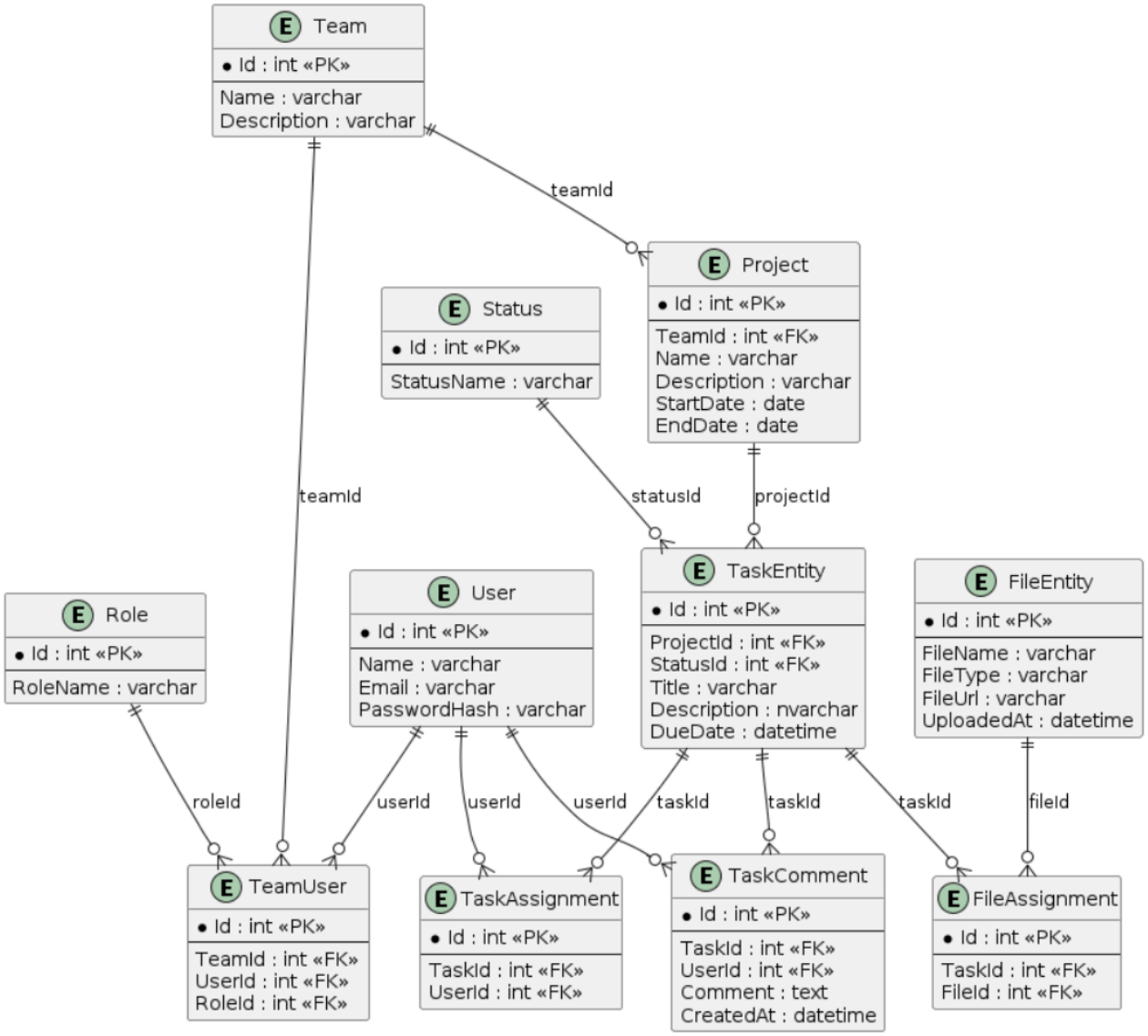
- **Obtener Usuario por ID** GetUser(int id)
  Consultar los detalles de un usuario específico usando su identificador único.
- **Registrar Nuevo Usuario** InsertUser([FromBody] UserDto userDto)
  Insertar un nuevo usuario, incluyendo la validación de sus datos de entrada y el manejo inicial de la contraseña (PasswordHash).
- **Actualizar Usuario** UpdateUser(int id, [FromBody] UserDto userDto)
  Modificar los datos de un usuario existente, aplicando validaciones y actualizando la información de la contraseña si es necesario.
- **Eliminar Usuario** DeleteUser(int id)
  Eliminar permanentemente un registro de usuario.

2.3 Casos de Uso del Controlador de Asignaciones (TaskAssignmentController)

Este controlador gestiona las relaciones entre Tareas y Usuarios, que es crucial para la funcionalidad colaborativa.

- **Listar Todas las Asignaciones** GetAll()
  Obtener un listado de todos los registros de asignaciones entre tareas y usuarios.
- **Asignar Tarea a Usuario** Assign([FromBody] TaskAssignmentDto dto)
  Crear una nueva asignación, vinculando una tarea a un usuario específico. Esto previene la duplicación de la misma asignación.
- **Eliminar Asignación** Delete(int id)
  Eliminar una asignación específica usando su ID, desvinculando al usuario de la tarea.
- **Obtener Tareas Asignadas a Usuario** GetTasksByUser(int userId)
  Consultar y listar todas las tareas que han sido asignadas a un userId en particular.
- **Obtener Usuarios Asignados a Tarea** GetUsersByTask(int taskId)
  Listar todos los usuarios que han sido asignados a un taskId específico.

3 Entidad Relación de la Base de Datos



4 Priorización de los Procesos y Etapa

La prioridad se centró en establecer las operaciones CRUD básicas y las validaciones esenciales para la entidad TaskEntity que es considerado el núcleo del sistema.

Prioridad Alta (Foco Principal): Se trabajó en la implementación completa de la gestión básica de tareas (Creación, Lectura por ID y Total, Actualización y Eliminación), asegurando que todas las operaciones validen la existencia de entidades relacionadas (Project y Status).

Prioridad Media: Se desarrolló la funcionalidad de consulta avanzada para tareas, incluyendo el filtro por proyecto y la búsqueda textual por palabras clave.

Prioridad Baja (Implementación Inicial): Se implementaron los *endpoints* para la gestión de asignaciones de tareas (TaskAssignment), crucial para el funcionamiento colaborativo del sistema, y la gestión de usuarios.

5 Explicación Detallada de los Casos de Uso

5.1 Casos de Uso del Controlador de Tareas (TaskEntityController)

Los siguientes casos de uso gestionan el ciclo de vida completo de las tareas, la entidad central del proyecto, incluyendo validaciones y búsquedas especializadas.

- Listar Todas las Tareas (GetTasksDtoMapper())

  Este caso de uso se enfoca en la recuperación masiva de datos. La función realiza una llamada al servicio para obtener todas las instancias de TaskEntity y, posteriormente, utiliza AutoMapper para transformar estas entidades a una lista de DTOs (TaskEntityDto) antes de ser enviadas al cliente. Esto garantiza que solo se exponga la información relevante de la tarea.

- Obtener Tarea por ID (GetTaskEntityDtoMapperById(int id))

  Este caso de uso permite la consulta detallada de una tarea específica. El proceso recibe el id de la tarea, la busca en el servicio (_taskService.GetTaskAsync(id)), y si se encuentra, la mapea a un DTO. El Flujo Crítico aquí es la gestión de la inexistencia de la tarea: si el servicio devuelve null, se propaga una excepción que debería ser manejada para retornar un error de negocio claro (por ejemplo, "Tarea no encontrada"), en lugar de un error genérico del servidor.

- Crear Nueva Tarea (InsertTaskEntityDtoMapper(TaskEntityDto taskEntityDto))

  Este es un caso de uso de escritura fundamental.

  Validación de Datos: Antes de cualquier lógica de negocio, se valida el TaskEntityDto de entrada utilizando un IValidationService (como FluentValidation), asegurando que campos como Título y ProjectId sean válidos.

  Mapeo y Persistencia: Si la validación pasa, el DTO se mapea a la entidad TaskEntity.

  Flujo Crítico (Regla de Negocio): Dentro del servicio, se verifica que el ProjectId exista. Si el campo StatusId no viene especificado en el DTO, el servicio aplica la regla de negocio de inicializarlo automáticamente al estado "Pendiente" (ID 1), garantizando la integridad del estado inicial de la tarea.

- Actualizar Tarea (UpdateTaskEntityDtoMapper(int id, [FromBody] TaskEntityDto TaskEntityDto))

  Este caso de uso permite la modificación de una tarea existente.

  Validación Inicial: El controlador primero verifica que el id proporcionado en la URL coincida con el Id dentro del cuerpo del DTO (prevención de manipulaciones).

Validación de Datos DTO: Se ejecuta la validación del DTO de entrada.

Regla de Inmutabilidad de Proyecto: Una vez obtenida la tarea existente, se aplica la regla de negocio clave: se compara el ProjectId del DTO entrante con el ProjectId de la tarea original. Si son diferentes, se retorna un BadRequest con el mensaje: "La tarea no puede cambiar de proyecto".

Actualización de Estado: El servicio es responsable de verificar que el nuevo StatusId (si se modifica) corresponda a un estado existente.

- Eliminar Tarea (DeleteTaskEntityDtoMapper(int id))

  Este caso de uso de eliminación es directo. Primero, busca la tarea por id para asegurar que existe (o lanza una excepción si no la encuentra). Luego, invoca al servicio (_taskService.DeleteTaskAsync(task)) para remover permanentemente el registro. Retorna un estado NoContent (204) si la eliminación es exitosa, indicando que la operación se completó sin necesidad de devolver contenido.

- Buscar Tareas por Texto (SearchTasks([FromQuery] string text))

  Este caso de uso de consulta avanzada permite a los usuarios buscar rápidamente tareas. La palabra clave de búsqueda (text) se recibe como un parámetro de consulta ([FromQuery]). El servicio es el encargado de implementar la lógica de búsqueda, que típicamente incluye la normalización del texto (a minúsculas) y la búsqueda por coincidencia parcial en campos como el título y la descripción de la tarea.

- Obtener Tareas por Proyecto (y/o Estado) (GetAllProjectTasks(int projectId, [FromQuery] int? statusId))

  Otro caso de uso de consulta avanzada y filtrado. Permite la visualización contextualizada de tareas:

  Filtro Obligatorio: Requiere un projectId para restringir las tareas al alcance de un proyecto.

  Filtro Opcional: El parámetro statusId es opcional (int?). Si se proporciona, el servicio (_taskService.GetAllProjectTasksAsync) debe aplicar un filtro adicional para mostrar solo las tareas de ese proyecto que tienen el estado especificado.

5.2 Casos de Uso del Controlador de Usuarios (UserController)

Estos casos de uso se enfocan en la gestión de la entidad User, la base para la colaboración.

- Listar Todos los Usuarios (GetAllUsers())

  Recupera una lista completa de todos los usuarios registrados, mapeándolos a UserDto para proteger la información sensible, como el hash de la contraseña.

- Obtener Usuario por ID (GetUser(int id))

Consulta los detalles de un usuario específico por su id. Al igual que con las tareas, si el usuario no existe, se gestiona la excepción para evitar errores de servidor. El resultado es mapeado a un UserDto para la presentación.

- Registrar Nuevo Usuario (InsertUser([FromBody] UserDto userDto))

  Validación: Se valida el UserDto de entrada (ej. formato de correo electrónico, longitud de contraseña).

  Mapeo y Hash: El DTO se mapea a la entidad User. El campo userDto.Password se asigna explícitamente a user.PasswordHash. La lógica de hashing real de la contraseña debe residir en el servicio (_userService.InsertUserAsync) para mantener la seguridad y la arquitectura limpia.

  Flujo Crítico (Regla de Negocio): El servicio debe aplicar la regla de negocio para evitar la duplicidad de usuarios (ej. mismo correo electrónico).

- Actualizar Usuario (UpdateUser(int id, [FromBody] UserDto userDto))

  Permite la modificación de los datos de un usuario.

  Validación de Identidad: Se verifica que el id de la URL coincida con el Id del DTO.

  Actualización: Se recupera el usuario existente, se mapean los cambios del DTO y el servicio ejecuta la actualización. Si se recibe un nuevo valor en el campo de contraseña, el servicio debe manejar la lógica de re-hasheo antes de la persistencia.

- Eliminar Usuario (DeleteUser(int id))

  Busca el usuario por id y, si existe, invoca al servicio para suprimir el registro de la base de datos, retornando un estado NoContent (204).

5.3 Casos de Uso del Controlador de Asignaciones (TaskAssignmentController)

Estos casos de uso gestionan la tabla pivote de asignaciones (TaskAssignment), esencial para vincular tareas y responsables.

- Listar Todas las Asignaciones (GetAll())

  Recupera y lista todos los registros existentes en la tabla de asignaciones, mapeándolos a TaskAssignmentDto.

- Asignar Tarea a Usuario (Assign([FromBody] TaskAssignmentDto dto))

  Este es un caso de uso de escritura para la colaboración.

  Validación: Se valida el DTO (TaskId y UserId son requeridos).

  Mapeo: El DTO se mapea a la entidad TaskAssignment.

Flujo Crítico (Regla de Negocio): El servicio (_service.AssignTaskToUserAsync) tiene la responsabilidad de validar la existencia de las entidades referenciadas (TaskId y UserId). Además, aplica la regla de negocio crítica: verifica que la combinación (TaskId, UserId) no exista ya en la tabla. Si existe, lanza la excepción de negocio "Esta tarea ya está asignada a este usuario" para prevenir la duplicidad.

- Eliminar Asignación (Delete(int id))

  Permite eliminar una asignación específica usando su id (el ID del registro de la tabla pivote). El servicio se encarga de buscar y remover el registro, desvinculando al usuario de la tarea sin eliminar la tarea ni el usuario. Retorna NoContent.

- Obtener Tareas Asignadas a Usuario (GetTasksByUser(int userId))

  Este caso de uso de consulta especializada busca todas las tareas donde el userId es responsable. El Flujo Crítico es la lógica de recuperación del servicio (_service.GetTasksByUserAsync), que debe:

  Consultar la tabla TaskAssignment utilizando el userId.

  Recuperar los objetos TaskEntity correspondientes a esos IDs de tarea.

  Mapear y retornar la lista de tareas al cliente.

- Obtener Usuarios Asignados a Tarea (GetUsersByTask(int taskId))

  Inverso al anterior, este caso de uso consulta todos los usuarios asignados a un taskId específico. El servicio utiliza el taskId para encontrar los TaskAssignment correspondientes y luego recupera los objetos User asociados, los cuales son mapeados a UserDto antes de ser devueltos.

6 Criterios de Aceptación y Reglas de Negocio Aplicados

Reglas de Negocio sobre TaskEntity

- **Integridad de Proyecto:** Una Tarea pertenece a un único Proyecto (ProjectId) y el servicio lanzará una excepción si el proyecto referenciado no existe en la base de datos (Ej. TaskEntityService.InsertTaskAsync).
- **Estado Inicial:** Al crear una tarea sin especificar el StatusId (o si es 0), se asigna el estado "Pendiente" (ID 1) por defecto.
- **Inmutabilidad de Proyecto:** Una tarea no puede cambiar de proyecto una vez creada. Esta validación es realizada en el TaskEntityController (UpdateTaskEntityDtoMapper).
- **Integridad de Estado:** El StatusId debe corresponder a un estado existente. Si no existe, se lanza una excepción (Ej. TaskEntityService.UpdateTaskAsync).
- **Búsqueda:** El CU de búsqueda (SearchTasksAsync) normaliza el texto de entrada a minúsculas y requiere que se especifique un texto.
- **Campos Obligatorios:** Los campos Title y Description (para tareas) son obligatorios y su validación se realiza en la capa API utilizando FluentValidation.

- **Fecha Límite (DueDate):** Si se especifica, debe ser una fecha válida y no puede ser anterior a la fecha actual (validación de futuro).
- **Manejo de Tarea No Encontrada:** Si se solicita una tarea inexistente (GetTaskById), se lanza una excepción controlada para devolver un error.

Reglas de Negocio sobre User

- **Correo Electrónico:**
    - **Regla:** El nuevo correo electrónico proporcionado no debe pertenecer a ningún otro usuario existente, excepto al propio usuario que se está actualizando.
    - **Implementación:** El servicio (UserService.UpdateUserAsync) busca duplicados.
    - **Consecuencia Crítica:** Si se encuentra un correo electrónico duplicado asociado a otro ID, se lanza una excepción de negocio: "Ya existe otro usuario con ese correo electrónico."
- **Manejo de Contraseña (Hashing):**
    - **Regla:** La contraseña de un usuario nunca debe almacenarse en texto plano. Debe ser transformada utilizando una función de *hashing* seguro (en este caso, SHA256).
    - **Implementación:** La función privada HashPassword(string password) se invoca en InsertUserAsync y UpdateUserAsync.
    - **Flujo:** En la **creación**, la contraseña del DTO es *hasheada* antes de guardarse. En la **actualización**, si se proporciona una cadena no vacía en la contraseña (!string.IsNullOrWhiteSpace(user.PasswordHash)), esta se *hashea*; de lo contrario, se mantiene el hash existente para no borrar la contraseña.
- **Existencia del Usuario (Consulta/Modificación/Eliminación):**
    - **Regla:** Toda operación que dependa de un usuario existente (consulta por ID, actualización, eliminación) debe confirmar la presencia de dicho usuario en la base de datos.
    - **Implementación:** Los métodos GetUserAsync y UpdateUserAsync inician consultando el repositorio.
    - **Consecuencia Crítica:** Si el usuario no se encuentra (existingUser == null), se lanza una excepción de negocio, como "No se encontró el usuario con Id {id}" (en GetUserAsync) o "El usuario no existe." (en UpdateUserAsync).

Reglas de Negocio sobre TaskAssignment

- **Existencia de Entidades:** Antes de asignar, se debe validar que tanto el **Usuario (UserId)** como la **Tarea (TaskId)** existan en el sistema.
- **No Duplicidad de Asignación:** Un usuario **no puede ser asignado dos veces** a la misma tarea. El servicio (TaskAssignmentService.AssignTaskToUserAsync) verifica la existencia de la tupla (TaskId, UserId) antes de la inserción.