Presentation Layer

Complete WPF Frontend Application. It allows users to log in, register, view a dashboard of their boards, create/delete boards, and interact with tasks on a specific board view.
Purpose: To provide a rich, interactive client for the Kanban system, moving it from a backend-only project to a full-stack application.

All code within the Frontend namespace, including Views, ViewModels, Models, and Frontend Controllers.
Board Membership Persistence (BoardMembers Table)
The Data Layer was extended with a new BoardMembersController and BoardMemberDTO. This creates and manages a new BoardMembers database table to handle the many-to-many relationship between users and boards.
Purpose: This is the database foundation for all collaboration features. It persistently stores which users are members of which boards.
Location: BoardMembersController.cs and BoardMemberDTO.cs in the DataAccessLayer.
Standardized API Response Wrapper (Response<T>)

A generic Response<T> class was created to standardize the format of all data returned by the Service Layer. It encapsulates either a return value or an error message.
Purpose: To create a consistent, predictable, and robust communication protocol between the backend and frontend, simplifying error handling on the client side.
Location: IntroSE.Kanban.Backend.ServiceLayer.Entities.Response<T>.
Formalized Column Entity

The concept of a board column was formalized into its own set of classes: Column (Business), ColumnDTO (Data), and ColumnController (Data). This allows columns to have their own properties, like a task limit, which are persisted in the database.
Purpose: To make board columns first-class citizens in the system, allowing for more complex features like setting individual column limits.
Location: Column.cs, ColumnDTO.cs, ColumnController.cs.

Service Layer Changes

The BoardService was expanded with new methods to support multi-user collaboration.
Purpose: To expose the new business logic for board membership and ownership to the client application.

Location:
BoardService.JoinBoard(string email, int boardID): Allows a user to become a member of an existing board.
BoardService.LeaveBoard(string email, int boardID): Allows a member to leave a board.
BoardService.TransferOwnership(...): Allows the current owner to transfer ownership to another member.
BoardService.GetUserBoards(string email): Retrieves all boards a user is a member of.
BoardService.GetBoardName(int boardId): A utility method to get a board's name by its ID.

Task Assignment API
Description: The TaskService received a new method to assign tasks.
Purpose: To allow users to assign a task to another member of the same board.
Location: TaskService.AssignTask(string email, ..., string emailAssignee).

Business Layer Changes

BoardFacade Collaboration Logic
The BoardFacade now contains the core logic for managing board membership and ownership.
Purpose: To orchestrate the interactions between users and boards for all collaboration-related actions.
Location:
BoardFacade.JoinBoard() / LeaveBoard(): Manages adding/removing users from a board's member list.
BoardFacade.TransferOwnership(): Handles the logic of changing a board's owner.
Board Ownership

The Board class now has an Owner property. Logic to enforce that only the owner can perform critical actions (like deleting a board or transferring ownership) has been added.
Purpose: To establish a clear hierarchy and control over board management.

Location: Board.Owner property and inline ownership checks within BoardFacade.DeleteBoard() and BoardFacade.TransferOwnership().
Task Assignment and Validation
Description: The Task class now includes an Assignee property. It also contains new methods (AssignTask, Unassign) to manage this state.
New validation constants for title and description length were also added.
To enable tasks to be formally assigned to a user and to enforce stricter data validation.
Location: Task.Assignee property, Task.AssignTask(), Task.Unassign(), and constants TITLE_MAX_LENGTH and DESCRIPTION_MAX_LENGTH.
Significant Unlisted Additions
These are major features and architectural improvements that were implemented but not mentioned in the list.

Project changes

1. Singleton Logger

The logging mechanism was refactored into a static Logger class that provides a single, globally accessible log4net instance.
Purpose: To enforce a Singleton pattern for the logger.

2. Abstract DAL Controller and DTO

The base Controller class in the Data Access Layer was made abstract. Similarly, a new abstract base class named DTO was introduced, from which all specific DTOs

3.Separation of Concerns: Callback Mechanism
The direct dependency of UserFacade on BoardFacade was removed. Instead, UserFacade now exposes a public callback property (OnUserAddedCallback). The ServiceFactory is responsible for "wiring" these two components together by assigning BoardFacade.AddNewUser to this callback.

Data Layer Changes

New DTO Classes Introduced:
- BoardDTO
- UserDTO
- TaskDTO

New Controller Classes Introduced:
- UserController
- BoardController
- TaskController

Each controller supports:
- Insert()
- Update()
- Delete()
- ConvertReaderToObject(SQLiteDataReader)

User and Board Controller supports SelectAll() method.