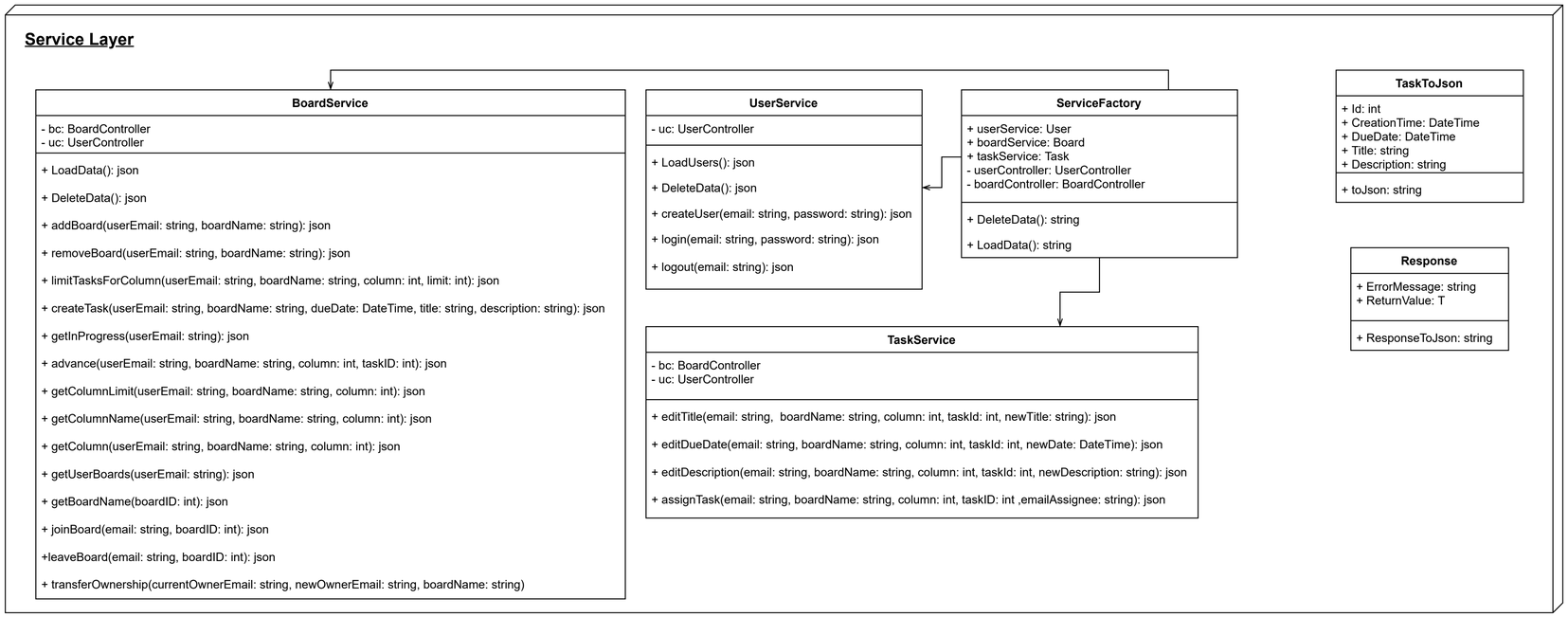
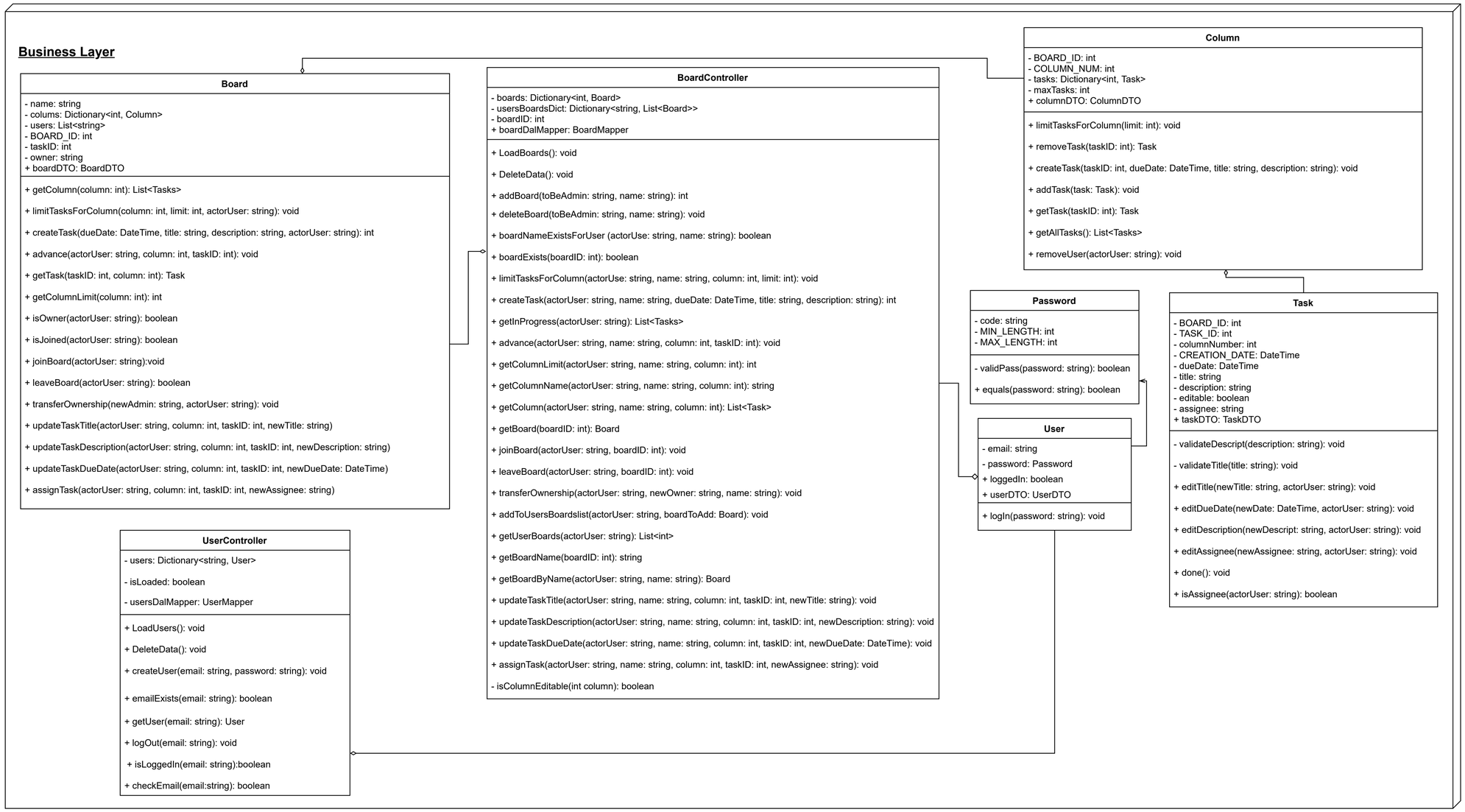
Service Layer



Business Layer



Data Access Layer

DTO
- id: int
- controller: DalController
- isPersistent: bool

BoardDTO
- name: string
- owner: string
- users: List<BoardUserDTO>
- columns: List<ColumnDTO>

+ Users: ReadOnlyCollection<string>

+ Columns: ReadOnlyCollection<ColumnDTO>

+ addBoardToDAL(): void

+ transferOwnershipInDAL(newOwner: string): void

+ advanceInDAL(column: int, taskID: int): void

+ taskMaxID(int boardID): int

+ joinBoardUserToDAL(email: string): void

+ deleteBoardFromDAL(): void

ColumnDTO
- columnName: string
- taskLimit: int
- columnNum: int
- BoardId: int
- tasks: List<TaskDTO>

+ Tasks: ReadOnlyCollection<string>

+ addToTasksList(taskToAdd: TaskDTO): void

+ removeFromTasksList(taskID: int): TaskDTO

+ getTaskDTO(taskID: int): TaskDTO

+ limitTasksForColumnInDAL(limit: int): void

+ addTaskToDAL(newTask: TaskDTO): void

+ removeUserFromTasks(email: string): void

BoardUserDTO
- boardId: string
- userEmail: string

TaskDTO
- BOARD_ID: int
- columnNum: int
- CREATION_DATE: string
- dueDate: string
- title: string
- description: string
- editable: boolean
- assignee: string

+ editTitleInDAL(newTitle: string): void

+ editDescriptionInDAL(newDescript: string): void

+ editDueDateInDAL(newDate: DateTime): void

+ editAssigneeInDAL(newAssignee: string): void

UserDTO
- email: string
- password: Password

+ addUserToDAL(): void

Mapper
- tableName: string
- connectionString: string

+ update(id: int, attributeName: string, attributeValue: string): boolean

+ update(id: int, attributeName: string, attributeValue: int): boolean

select(): List<DTO>

 # ConvertReaderToObject(reader: SQLiteDataReader): DTO

+ deleteAll(): boolean

+ delete(DTO): boolean

BoardMapper
- boardTableName: string
- columnMapper: ColumnMapper
- membersMapper: BoardUserMapper

+ insert(boardDTO: BoardDTO): boolean

+ selectAllBoards(): List<BoardDTO>

+ deleteAll(boardID: int): boolean

+ BoardMaxId(): int

+ loadBoards(): Dictionary<int,BoardDTO>

BoardUserMapper
- boardUserTableName: string

+ loadBoardUsers(boardId: int): List<BoardUserDTO>

+ Insert(member: BoardUserDTO): boolean

+ DeleteBoardUsers(boardID: int): void

ColumnMapper
- columnTableName: string
- taskMapper: TaskMapper

+ insert(columnDTO: ColumnDTO): boolean

+ loadColumns(boardId: int): List<ColumnDTO>

+ DeleteBoardColumns(boardID: int): void

UserMapper
- userTableName: string

+ insert(userDTO: UserDTO): boolean

+ selectAllUsers(): List<UserDTO>

+ loadUsers(): Dictionary<string,UserDTO>

TaskMapper
- taskTableName: string

+ insert(taskDTO: TaskDTO): boolean

+ loadTasks(boardID: int, columnNum: int): List<TaskDTO>

+ TaskMaxId(boardID: int): int

+ update(id: int, boardID: int, attributeName: string, attributeValue: string): boolean

+ update(id: int, boardID: int, attributeName: string, attributeValue: int): boolean

+ delete(id: int, boardID: int, attributeName: string, attributeValue: int): boolean

DB

Design changes
Milestone 2 – 2nd submission – Group 18

Service Layer:

- ***ServiceFactory:***
  - Adding this new class for creating the different service objects, and controlling the methods of load and delete data.
  - Fields - create new *UserService, BoardService* and *TaskService* objects.
  - Methods – Activate the *Load* and *Delete* data methods in the Service layer, that cause the activation of all loading and deleting methods in the system.
- ***BoardService*:**
  - *LoadData* method **–** creation of this new method, to activate loading all boards and their columns and tasks from DB, by calling the matching method from Business Layer, that activate the other related ones.
  - *DeleteData* method **-** creation of this new method, to activate deleting all boards and their columns and tasks , by calling the matching method from Business Layer, that activate the other related ones.
  - *leaveBoard* method **-** creation of this new method, for the user to be able to leave a board if desired.
- ***UserService:***
  - *LoadUsers* method - creation of this new method, to activate loading all the Users from DB, by calling the matching method from Business Layer, that activate the other related ones.
  - *DeleteData* method - creation of this new method, to activate deleting all Users, by calling the matching method from Business Layer, that activate the other related ones.

Business Layer:

- **General changes:**
  - *getX methods* – generating getters and setters in all classes, thus removing all methods of "*getX"* that are simple "get" methods that return a fields value.
- ***UserController:***
  - *isLoaded* field - adding this new field to verify the loading of all the Users to the system before loading all the Boards to the system.
  - *DeleteData* method – Adding this new method, that calls the Mapper matching method to delete all Users data from the system**.**
- ***BoardController:***
  - *usersBoardsDict* field - adding this new field, that contains a dictionary of a *UserEmail* and a list of all of the User's *boards*. This field makes many actions more efficient, especially the actions that have consequences on all of one user's boards, such as *leaveBoard*.
  - *limitTasksForColumn, createTask, advance, getColumnLimit, getColumnName, getColumn, joinBoard, leaveBoard, tranferOwnership, updateTaskTiltle, updateTaskDescription, updateTaskDueDate, assignTask, isColumnEditable* methods – Adding these methods that exist in other classes in *BusinessLayer* to the *BoardController*. This addition enables preforming these actions from the *ServiceLayer* by calling them straight from *BoardController*, without the need of using "getters" to other classes of *BusinessLayer* in *ServiceLayer*. Thus, it improves the cohesion.
  - BoardNameExistsForUser method – creation this new method to check weather a User is a member of a board with a specific name, to answer the request of a board name being unique per user. This method is being used when a user desires to create a new board or join one.
  - addToUsersBoardsList method – Adding this new method to enable the proper usage of the new *usersBoardsDict* field.
  - *getBoardByName* method – Adding this new method to enable a user preform actions on a board by it's name, although it is not unique for the whole system. This method returns the board's ID.
- ***Board:***
  - Removal of *Mapper* field - removing this field because communication with the mapper goes through *BoardDTO* class.
  - *getColumnLimit, updateTaskTiltle, updateTaskDescription, updateTaskDueDate, assignTask, isColumnEditable* methods – Adding these methods that exist in *Column* and *Task* classes to *Board,* and improve the system's cohesion.
- ***Column:***
  - Removal of *Mapper* field - removing this field because communication with the mapper goes through *ColumnDTO* class.
  - *BOARD_ID* field – Adding this new field because the way the data is arranged in DB.

- ***Task*:**
  - *columnNumber* field - Adding this new field because the way the data is arranged in DB.

Data Access Layer:

- ***DTO*:**
  - This abstract class represents a line in a table in the DB.
  - *Id* field – the ID of the object (if it doesn't has one, ID will be -1)
  - *controller* field - a mapper field that enables editing this line, by calling methods from a *xMapper* class
  - *isPersistent* field – a boolean field that indicates the synchronize between the data in the DB to the data in RAM.
- ***xDTO :***
  - *ReadOnlyCollections* – getters that help us connect between plural to plural in between tables in DB
  - Methods – all methods in these classes are called from Business layer by the matching methods there. Their job is to call the matching Mappers, that create the proper SQL query to update / insert / delete the desired information in the DB, in order to preserve the persistence of the system.
- ***Mapper:***
  - This abstract class contains all information that enables the communication between the DB and our system through SQL.
  - *tableName field – contains the table's name.*
  - *connectionString field– contains the string that gives the order of connection to the DB.*
  - *update* methods – templates of different update SQL queries that require different kinds of arguments.
  - *delete* methods – templates of different update SQL queries that require different kinds of arguments.
  - *insert* methods – templates of different update SQL queries that require different kinds of arguments.
  - *ConvertReaderToObject* method – Gets a line in DB and converts it into a DTO object. Enables the "translation" of DB information to objects in our system.
- ***xMapper:***
  - Methods - these classes contains specific methods for each DTO class. The specific methods are methods that require specific SQL queries for the matching objects.
  - Fields – Some of the classes hold other Mapper fields. these classes' matching *BussinesLayer* classes contain collections of objects from other classes. Thus, any action preformed on these classes in DB also require preforming calling other mappers to preform the actions on them as well.

Design changes
Milestone 1 – 2nd submission – Group 18

Service Layer:

- **New classes:**
  - *TaskToJson* – execute the convertion of the matching service layer class to Json, in order to return response as required.
  - *Response* – build the object that is eventually returned to the user (as a Json). Composed of a string message and a returned object (frequently a Json).
- **General changes:**
  - Change of names – change of fields / methods names to be consistence with the decided names format.
- ***Board*:**
  - *Column*:*int* argument – change from string type to int (*columnName* to *column*). Performed to match the change in Column class in business layer - altering field *title*(string) to *columnNum*(int).
    This change is reflected in methods: *limitTasksForColumn*, *advance*, *getColumnLimit*.
  - *createTask* method – change of the signature – *addTask* to *createTask*, in order to be coherent with the difference between the actions of adding and creating a new class, as reflected in *Column* class in business layer.
  - *getColumnName* method – creation of this new method, for the user to be able to get a column title as described in the assignment's requests, despite the implementation of classifying the columns as numbers.
  - *getColumn* method - creation of this new method, for the user to be able to get a list of a specific column's tasks in a specified board as desired. Also enables the implementation of the equivalent method in *GradingService* class, by calling this method.
- ***Task*:**
  - *bc:BoardController* field – setting of a *BoardController* field to enable the communication of the service layer and the business layer, for the user to be able to perform actions on boards' tasks.
  - *uc:UserController* field - setting of a *UserController* field to enable the communication of the service layer and the business layer. Enables the user to access his boards and tasks and preform changes and actions as desired.
  - *taskId:int* argument - change from string type to int . Performed to match the change in *Task* class in business layer - altering field *id*(string) to *TASK_ID*(int).
  - *email:string, boardName:string* arguments - demand of this arguments in order to specify the task the user desires to edit (task ID is unique per board, and a board is

unique per user). In addition, this arguments are required for accessing the user (in *UserController*), and his Boards (in *BoardController*) to access the task that the user desires to edit.

Business Layer:

- **General changes:**
  - Change of names – change of fields / methods names to be consistence with the decided names format.
- ***UserController*:**
  - *isLoggedIn* method – creation of this new method, to simplify the verification of the user being logged in to the system. This verification is required in every method, so that only logged in users will be able to access their boards and altering them.
  - *checkEmail* method – creation of this new method, to authenticate the validation of an email when a new user registers to the system (method is used in *createUser* method).
- ***User:***
  - *loggedIn* field – creation of this boolean field to restrict and allow the access of a user to the system, in accordance with him being logged in or not. Enables to simplify the implementation of the essential *isLoggedIn* method.
  - *logIn* method – change of the returned type from boolean to void, consequently to the decision of setting a *loggedIn* field that indicates if a user is logged in or not. This method alters the boolean value of *loggedIn*, thus a boolean return value is no longer required.
- ***Password:***
  - *equals* method – creation of this method, used in *logIn* method to verify that the user's password is correct (equals to the password they inserted in registration).
- ***BoardController:***
  - *actorUser:string* argument – adding the demand for this argument in all the methods of the class. Required to specify a board (board is unique per user), and also to prevent users that a board doesn't belong to them to preform actions on this board.
- ***Board:***
  - *Columns* field - change of type from List<Column> to Dictionary<int, Column>. Key of the dictionary is the *columnNum* (used to specify the column). The change from list to dictionary enables accessing a specific column of the board directly, instead of performing a search through the list (more efficient).
  - *taskID* field – creation of this new field in order to provide the createTask method a unique task ID. This field is basically a counter of all the tasks that are created in a specific board. As a board is unique per user, the combination of a unique task ID per board, board and a user email creates a unique identifier per task.

- *column: int* argument - change from Column type to int. Enables accessing the required column by the column number – the key in the dictionary in *columns* field. Thus, actions can be preformed on a specific column without the need of demanding an entire column object as an argument.
- *removeTask* method – deletion of this method as it is not required in the system requirements document at the moment.
- *getUsers* method – creation of this new method that return the list of users in *users* field. Added in order to enable the update of *users* field when one removes a board. This method can also serve a future optional demand of "adding" additional users to an existing board.

- ***Column:***
  - *columnNum* field - change from string type to int (*title* to *columnNum*). Simplify the management of the 3 columns by referring them in their chronical order.
  - *tasks* field - change of type from List<Task> to Dictionary<int, Task>. Key of the dictionary is the *TASK_ID* (used to specify a task). The change from list to dictionary enables accessing a specific task of the column by its unique ID directly, instead of performing a search through the list by other characteristics (more efficient).
  - *taskID:int* argument - change from string type to int. Performed to match the change in *Task* class - altering field *id*(string) to *TASK_ID*(int).
  - *addTask* method – change of method type from protected to public. Specifying this method for the use of checking the possibility and adding an existing Task to a column, and creating a new *createTask* method that handles the creation of a new Task object.
  - *createTask* method – creation of this new method that create a new Task object that answers all of Task's format requirements. This methods handles the creation of the object, and calls the *addTask* method to insert the created task to the "backlog" board's column if possible.
  - *getMaxTasks* method – creation of this method to enable the implementation of *getColumnLimit* in service layer (allows the user to get the column's tasks limitation if desired).
- ***Task:***
  - *ALL* fields – changing all the class's fields from public to private as accepted.
  - *TASK_ID* field - change from string type to int, and declaration as a "read only" field . Performed to match the requirement of *GradingService* class to receive an int ID argument (and not string). Declaration as "read only" field because the tasks IDs are managed by *Board* class, that maintains the uniqueness of each ID. Changing a task's ID should not be possible in this class, because it might fault the uniqueness .
  - *validateDescript, validateTitle* methods – creation of these new methods to ensure inserting valid values to *title* and *description* fields, according to their restrictions. The validation if required when creating a new Task and when editing one of these fields of an existing Task.

- *getDescription, getTitle, getDueDate, getCreationDate, getID* methods – creation of these new "getters" methods to enable the usage of these fields in *TaskToJson* class in service layer. Required because fields are private.

Design changes
Milestone 2 – 1st submission – Group 18

Data Access Layer:

- ***UserDTO, BoardDTO, BoardUserDTO, ColumnTaskDTO, ColumnDTO, TaskDTO classes:***
  - Consists of the classes' data in the DataBase.
- ***UserMapper, BoardMapper, ColumnMapper, TaskMapper:***
  - Used to reload the data from the DataBase when needed.

Service Layer:

- ***Board*:**
  - *getUserBoards* method – creation of this new method, to enable getting a list of all the boardIDs of the boards a user is a member of.
  - *getBoardName* method – creation of this new method, for the user to be able to get a board's name when given its ID.
  - *joinBoard* method – creation of this new method, for the user to be able to join a board if desired.
  - *leaveBoard* method – creation of this new method, for the user to be able to leave a board he is a member of if desired.
  - *transferOwnership* method – creation of this new method, for the user to be able to transfer ownership of one of its boards to another member of the borad.
- ***Task:***
  - *assignTask* method – creation of this new method, to enable a user to assign an unassigned task to a board's member, or to enable the task's assignee to reassign the task.

Business Layer:

- **General changes:**
  - *xMapper* fields – used to access the *xMapper* classes in DataAccessLayer in order to perform *loadX* methods.
  - *xDTO* fields – used to access the *xDTO* classes in DataAccessLayer in order to change and edit the relevant information in the DataBase.
  - *loadX* methods – used to reload the relevant data for each class from the DataBase.
- ***BoardController:***
  - *boards* field - change of type from Dictionary<string, List<Board>> to Dictionary<int, Board>. Key of the dictionary is the *BoardID* (used to specify the

board). The change is due to the a new functional requirement that defines a unique ID to each board, and enables a user to join several boards, and a board to have several members.
    - *boardID* field – creation of this new field in order to provide the createBoard method a unique board ID. This field is basically a counter of all the boards that are created.
    - *getUserBoards* method – creation of this new method, to enable getting a list of all the boardIDs of the boards a user is a member of.
- ***Board:***
    - *BOARD_ID* field – adding this new field, and declaration as a "read only" field . Performed to match the requirement of setting a unique ID to each board. Declaration as "read only" field because the boards IDs are managed by *BoardController* class, that maintains the uniqueness of each ID. Changing a board's ID should not be possible in this class, because it might fault the uniqueness .
    - *owner* field – creation of this new field to match the new requirement of setting an owner to each board, who can perform exclusive actions that an "ordinary member" can not. Receives a string – email (a unique username) of the owner. As a default the owner is the board's creator.
    - *boardID: int* argument – received in *advance* and *getTask* methods to enable restricting the execution of certain methods in *Column* and *Task* to the task's assignee only. Thus we can verify if the *actorUser* is indeed the assignee and authorized to perform an action.
    - *boardID: int* argument – received in *createTask* method to match the requirement of having a boardID as one of the 5 attributes of each task.
    - *isOwner* method – creation of this new method, to simplify the verification of the user being the owner of a board. This verification is required in certain methods that only the owner is authorized to perform.
    - *joinBoard* method – creation of this new method, to add a user to the board's members list.
    - *leaveBoard* method – creation of this new method, to remove a user from the board's members list, and cause the resulted actions.
    - *transferOwnership* method – creation of this new method, to enable the board's owner to transfer the ownership to another member of the board.
    - *getBoardName* method – creation of this new method that returns the board's name.
- ***Column:***
    - *boardID: int* argument – received in *removeTask, addTask* and *getTask* methods to enable to verify if the *actorUser* is indeed the assignee and authorized to perform certain actions.
    - *boardID: int* argument – received in *createTask* method to match the requirement of having a boardID as one of the 5 attributes of each task.
    - *removeUser* method -creation of this new method to unassign the tasks (that are not 'done') of a user that was removed from a board.

- ***Task:***
  - *BOARD_ID* field – creation of this new field to match the requirement of having a boardID as one of the 5 attributes of each task.
  - *assignee* field – creation of this new field to determine an assignee for each task, who is the only one authorized performing certain actions like editing the task etc**.**
  - *editAssignee* methods – creation of this new method, to assign an unassigned task to a board's member, or to enable the task's assignee to reassign the task.
  - *isAssignee* method – creation of this new method, to simplify the verification of the user being the assignee of a task. This verification is required in certain methods that only the assignee is authorized to perform.
  - *getAssignee, getBoardID* methods - creation of these new "getters" methods to enable the usage of these fields in *TaskToJson* class in service layer. Required because fields are private.