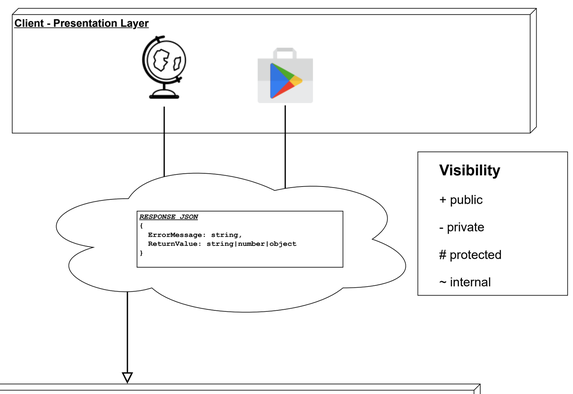
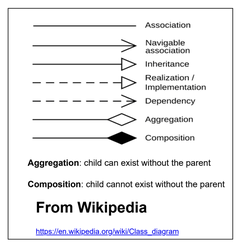




Visibility

+ public

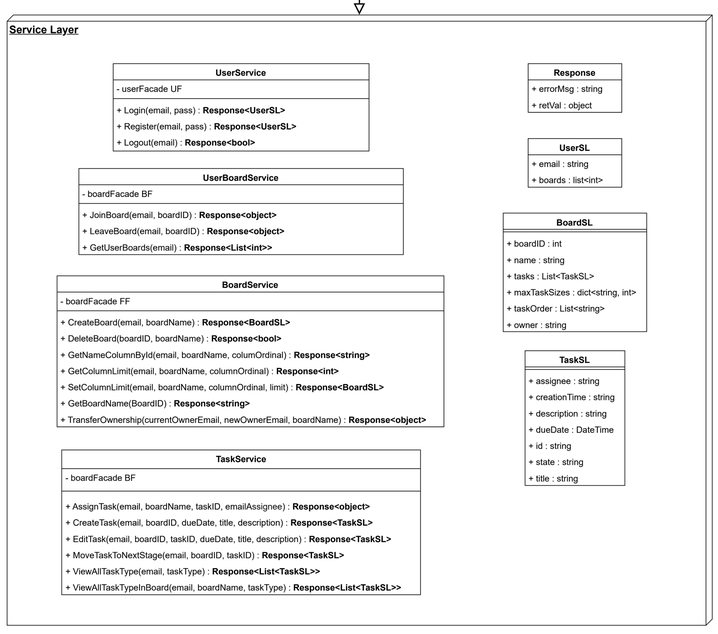
- private

protected

~ internal

BS - SL

```
createBoard(email,boardName){
	BoardBL B = BoardFacade.createBoard(email,boardName)
	(if No exceptions)
	BoardSL BSL = new Board(email,boarname)
	return BSL;
}

deleteboard(uuid,name)
	if( currentUser.boards.exists(uuid){
		return BoardFacade.removeboard(uuid)
	}
}
```



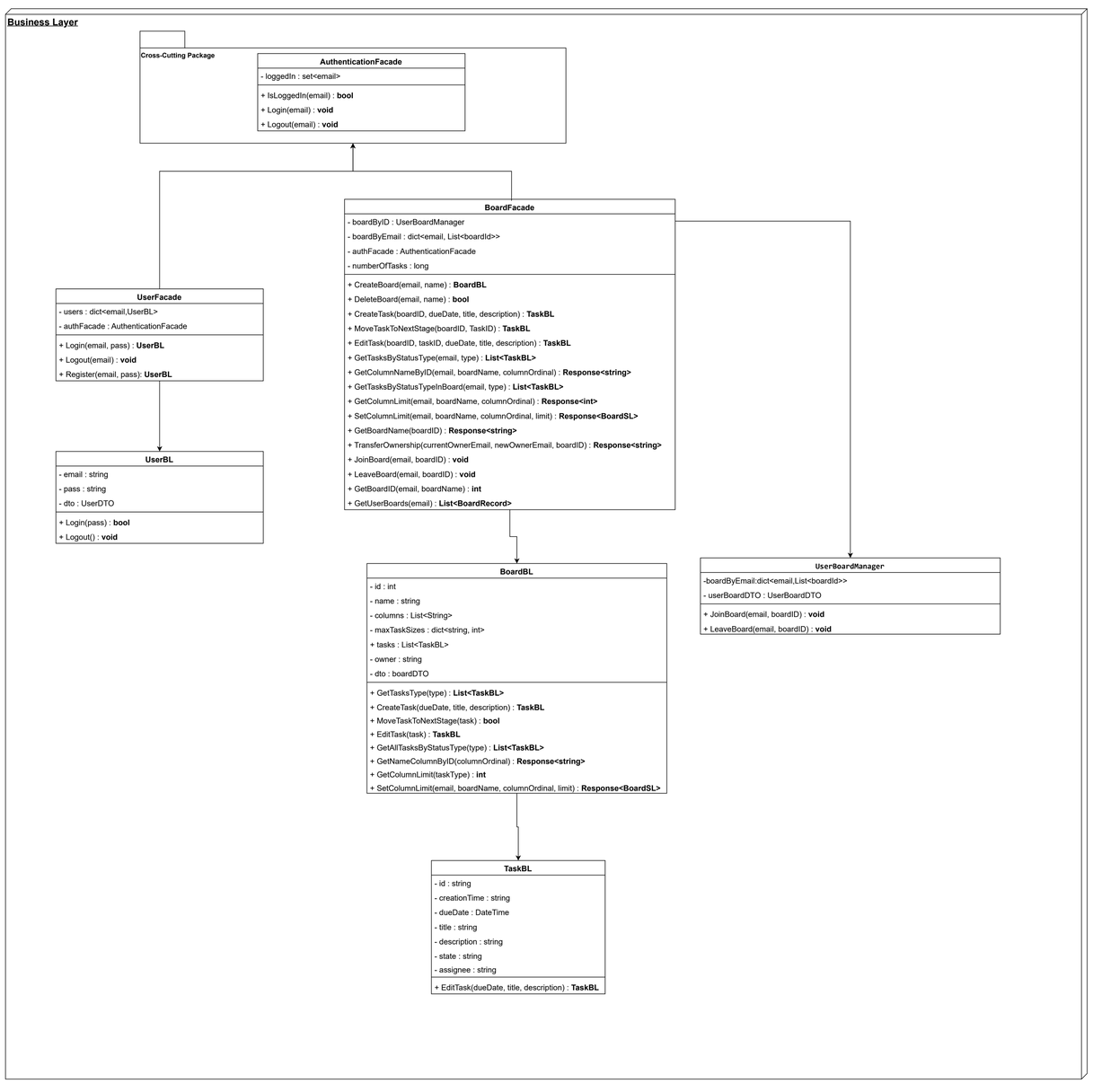
BL - Facade

```
public BoardBL createBoard(email,boardName){

string uuid=email,+,boardname;
if(boards.contiankey(uuid)) throw  "cant have same board name"
BoardBL br= new BoardBL(uuid,name,[],new List["backlog","inprogress","done"],
email);
boards[uuid]=br;
return br;

};
public boolean deleteBoard(uuid)
 board.removeKey(uuid);
}
public BoardBL getBoard(uuid){
BoardBL br=boards.get(uuid)
if(br==null) throw "board not exsits"
}

public List<Task> getBoardTasksType(uuid,type:string){
BoardBL br=getBoard(uuid);
return br.getTasksType(type);
}

public TaskBL createTask(uuid,dueDate,title,description){

BoardBL br=getBoard(uuid);
return br.createTask(dueDate,title,description);
}
public TaskBL moveTaskToNextStage(uuid,id){
BoardBL br=getBoard(uuid);
return br.moveTaskToNextStage(id);
}
```

US - SL

```
UserSL login(email, pass) {
	try{
	UserBL newUserBL = UserFacade.login(email, pass)
	if(newUserBL != null)
	{
	UserSL USL= new UserSL(email)
	return USL
	}
	}
	catch(e) {
		return new Response("user or password are invalid",
		null)
	}
}

bool register(email, pass) {
	try{
	UserBL newUserBL=UserFacade.register(email,pass)
	if(UserBL!=null)
	{
	UserSL USL=newUserSL(email)
	return newUserSL;
	}

	}
	catch(e) {
		return new Response("user or password are invalid",
		null)
	}
}

UserSL(email){
email=email
}


viewAllTaskType(type)
{
return UserFacade.viewAllTasksType(this.email, type)
}
```

Changes
UserSL - • Added field 'boards' UserBL - • Added field 'boards' BoardService - • Added method 'GetBoardName' BoardSL - • Changed field 'uuid' to 'id' BoardBL - • Changed field 'uuid' to 'id' BoardFacade - • Changed field 'boards' • Added method 'GetBoardName' • Added method 'TransferOwnership' TaskSL - • Added field 'assignee' TaskBL - • Added field 'assignee' Added 'UserBoardsFacade' - • Added field 'boards' • Added method 'JoinBoard' • Added method 'LeaveBoard' • Added method 'TransferOwnership' • Added method 'GetUserBoards' • Added method 'AddBoard' • Added method 'DeleteBoard' adding board cordinator to mange bewen the board facade and list of user boards to reduce depedancy bewtwen 2 diffrent class adding UserBoardService for action that are more in the responablity of the conaction betwen user namd boards (that responablity dont fall in user service or boardservice)

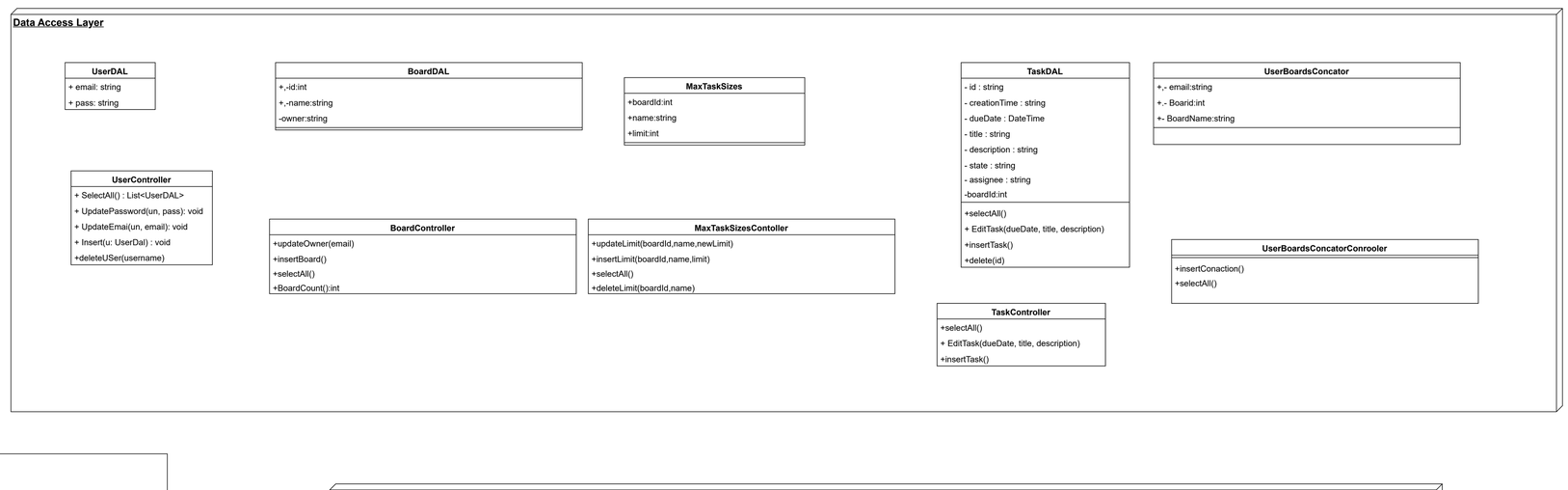


BL - board

```
{
	private long numberOfTasks = 0;

	private List<TaskBL> tasks = []:

	private dict<string, int> maxTaskSizes = new dict{"backlog" : -1, "in progress" : -1, "done" : -1};

	private List<String> columns = ["backlog", "in progress", "done"];


	BoardBL(uuid, name, email)

	{
		this.uuid = uuid;
		this.name = name;
		this.email = email;
	}


	public List<TaskSL> getTaskType(string type)

	{
		return tasks.find(x => x.type == type);
	}


	public TaskBL createTask(date dueDate, string title, string description)

	{
		string type = "backlog";
		if(maxTaskSize[type] <= getTaskType(type).length)
			throw "Max size reached";

		maxTaskSize[type] += 1;

		TaskBL = new TaskBL(numberOfTasks++, getTime(), dueDate, title, description, type);

		return(TaskBL);
	}


	public moveTaskToNextStage(TaskBL task)

	{
		if(task.status == column.getlast())
			return(false);

		task.status = column[column.getIndex(task.status) + 1];

		return(true);
	}
}
```

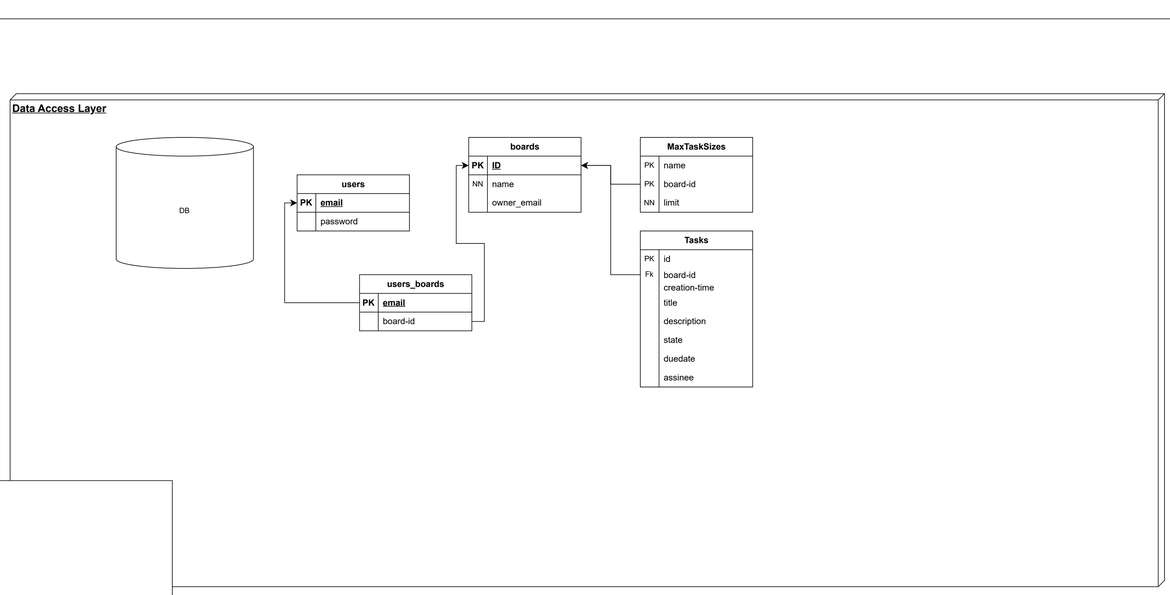
TaskService - ServiceLayer

```
public Response<TaskSL> CreateTask(email, boardID, dueDate, title, description)
{
   try
   {
      TaskBL task = boardFacade.CreateTask(email,  boardID,  dueDate,  title,  description);
      return new Response<TaskSL>(new TaskSL(task));
   }
   catch (Exception e)
   {
      return new Response<TaskSL>("An error occurred while processing your request.");
   }
}


public Response<TaskSL> MoveTaskToNextStage(email, boardID, taskID)
{
   try
   {
      TaskBL task = boardFacade.MoveTaskToNextStage(email, boardID, taskID);
      return new Response<TaskSL>(new TaskSL(task));
   }
   catch (Exception e)
   {
      return new Response<TaskSL>("An error occurred while processing your request.");
   }
}


public Response<TaskSL> EditTask(email, boardID, taskID, dueDate, title, description)
{
   try
   {
      TaskBL task = boardFacade.EditTask(email, boardID, taskID, dueDate, title, description);
      return new Response<TaskSL>(new TaskSL(task));
   }
   catch (Exception e)
   {
      return new Response<TaskSL>("An error occurred while processing your request.");
   }
}


public Response<List<TaskSL>> ViewAllTaskType(email, TaskType)
{
   try
   {
      List<TaskBL> tasks = boardFacade.GetTasksByStatusType(email, TaskType);
      List<TaskSL> result = new();
      foreach (TaskBL task in tasks)
         result.Add(new TaskSL(task));
      return new Response<List<TaskSL>>(result);
   }
   catch (Exception e)
   {
      return new Response<List<TaskSL>>("An error occurred while processing your request.");
   }
}


public Response<List<TaskSL>> ViewAllTaskTypeInBoard(email, boardName, TaskType)
{
   try
   {
      List<TaskBL> tasks = boardFacade.GetTasksByStatusTypeInBoard(email, boardName, TaskType);
      List<TaskSL> result = new();
      foreach (TaskBL task in tasks)
         result.Add(new TaskSL(task));
      return new Response<List<TaskSL>>(result);
   }
   catch (Exception e)
   {
      return new Response<List<TaskSL>>("An error occurred while processing your request.");
   }
}
```



BL - UserFacade

```
UserBL login(email,pass){
	if(!users.containsKey(email)) throw "user doesn't exist"
	userBL u = users[email]
	if (u.login(pass)) return u;
	return null;
}

UserBL register(pass, email) {
	if(users.containsKey(email)){
	 throw "user already exists";
	return null
	}
	userBL u = new userBL(email,pass);
	users .add(email, u);
	return u;
}

ViewAllTasksType(email, type){
	userBL u = users[email]
	return u.ViewAllTasksType(email, type);
}
```



BL - TaskBL

```
editTask(description,title,dueDate){
	(all the requirements with the title and description)
	(if some data was not given , "" was the input , then the original data will stay)
	this. title=title
	this.description=description
	this.dueDate=dueDate
}
```

BL - UserBL

```
private string email,pass;

private list<board> boards

bool login(pass) {

	return this.pass == pass

}


private string password;
private string Password { get => password
	set {
		if(pass.length < 6 || pass.length >20 || !pass.contains(uppercase charchter) || !pass.contains(number) || !pass.contains(lowercase charchter) ) throw "invalid pass"
		password = value;
	}
}

UserBL( email, pass) {
	if(pass.length < 6 || pass.length >20 || !pass.contains(uppercase charchter) || !pass.contains(number) || !pass.contains(lowercase charchter) ) throw
	"invalid pass"
	password =pass;
	email=email;
	list <string> boards = new List<string>();
}


bool logout(){
	.....
}

ViewAllTasksType(type)
{
	List AllType<Task> = new List<Task>();
	for board in : this.boards
	{
		Alltype.concat( board.getTasksType(type))
(always conenct to the end of the list)
	}
	return AllType;
}
```