System Requirements Documentation

Gavin Power

Table of Contents

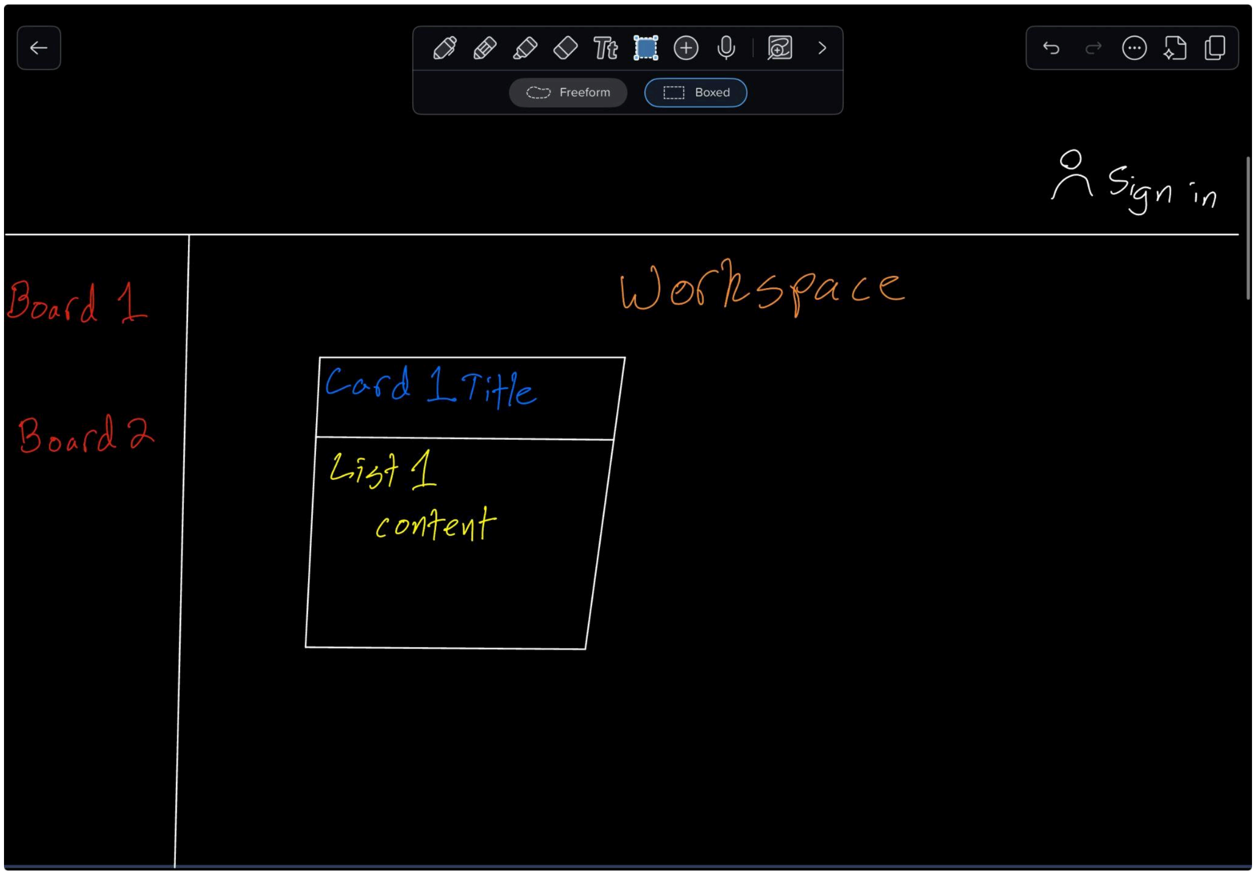
Customer Problem Statements and System Requirements

Gavin Power

Problem Statement: A task monitoring/management system is used to list, prioritize, and check off tasks. These systems have many professional and non-professional uses. Strategy games are often very deep and complex, and a player may have multiple tasks they need to complete. Sometimes, these many tasks can get "lost in the shuffle", resulting in a player forgetting to complete them. If a system were designed that could help the user manage these tasks and give recommendations for tasks to complete, it would be worthwhile to use. A system that can ease the learning curve in these deeply complex strategy games would lead to players enjoying their games and the task management system.

Glossary:

- **Board** - The board is a full workspace. It includes lists and cards.
- **Card** - The card is an item in the workspace that has a title and contains the content of the list.
- **List** - The list is the core content. It is what the user will use in order to write their tasks.
- **Task** - A task is anything a user wishes to accomplish.
- **Workspace** - The workspace is the largest area on the webpage and contains cards and lists. The user is able to drag and drop cards throughout the workspace.



System Requirements:

- **Functional, Non-functional, and UI Requirements**

No.	Priority Weight	Description
REQ-1	High	The system should be able to list tasks
REQ-2	High	Users should be able to have multiple, different task lists
REQ-3	High	Users must be able to prioritize their tasks, either being numbered or color-coded
REQ-4	High	Tasks should be sortable
REQ-5	High	The system should be able to

		provide specific, relevant recommendations for strategy game tasks when appropriate
REQ-6	High	The system should not just be for games, but should be able to be used for other task monitoring/management purposes as well.
REQ-7	Medium	The system should be able to ask the user if they are playing a game that has recommendation support
REQ-8	High	The system should be persistent, so the website will remember boards even after the user closes the site.
REQ-9	Low	When deleting a task for the first time, the user should be warned.
REQ-10	Medium	The system should be modern and look professional.
REQ-11	Medium	The system should allow for the user to create as many boards as they want.

Functional Requirements Specification

Stakeholders

The stakeholders for this application are users (both strategy game players and not), strategy game developers/publishers, and community managers for these games. Potential sponsors are also a stakeholder, as this system is designed for a specific audience that certain companies (such as computer accessories companies) can capitalize on.

Actors and Goals

- Primary Actors:
    - Players: Players will be able to use this application to plan their games, manage their tasks within the game, and keep track of the various objectives they wish to complete.
    - Game content creators: Content creators could use this application to plan their content and manage their schedule, but could also use this application for "game strategy optimization" content (e.g. making a video explaining how task-tracking can help players be better at the game)
- Secondary Actors:
    - Admin: Can see and analyze accounts, monitor system performance, and configure integrations with games.
    - Video game data analysts: Can work with admins to gather data about what users often make lists about. For example, if multiple users are making lists reminding themselves to build new roads in *Civilization*, this could be useful data

for the *Civilization* developers, possibly indicating that this is not communicated well enough in-game.

- Modders: Can extend the task management system to integrate with other games than those pre-programmed in, and can provide compatibility and niche features to other games.
- System: Must store and manage tasks effectively, must be durable and persistent, and provide recommendations about certain games when applicable.

Function

General System Requirements:

- Add user
- Verify user existence (can’t add a new user “123” if a user “123” is already in the database)
- Log in/log out
- Reset Password
- Retrieve user boards
- If a board/card/list has a certain keyword, the system may create a small pop-up in the corner asking the user if they are playing a selected game.

Admin Requirements:

- Special login (system knows it is an admin)
- Able to access SQL database
- Able to update accounts

Board Requirements

- Create new board
- Assign name to board
- Assign visibility to board (public or private)
- Save edited board
- Delete board graphically and in the database
    - If board is deleted, all boards below must graphically move up to “fill” the empty space left
- Board must have an options menu
    - Board options menu should at least include options to:
        - Edit board name
        - Edit board visibility
        - Add or remove color code for sorting.

Card Requirements

- Create new card
- Assign name to card
- Assign description to card
- Save edited card
- Delete card graphically and in the database
    - If card is deleted, all cards to the right must graphically move left to “fill” the empty space left
- Card should have an options menu
    - Options menu should at least have the options:
        - Edit name

        - Edit description
        - Add list
        - Delete list
        - Delete card
        - Add/remove color to card

List requirements

- Create new list
- Assign name to list
- Assign optional description to list
- Save edited list
- Delete list graphically and in the database
    - If list is deleted, all lists below must graphically move up to “fill” the empty space left
- List should have an options list
    - List options should at least include:
        - Edit name
        - Edit description
        - Add/remove color

Use Cases:

User:

- Log in/out
- Reset password

- Create/delete board
- Assign name to board
- Assign visibility to board
- Color code board
- Create/delete card
- Assign name to card
- Assign description to card
- Color code card
- Create/delete list
- Assign name to list
- Assign description to list
- Color code list

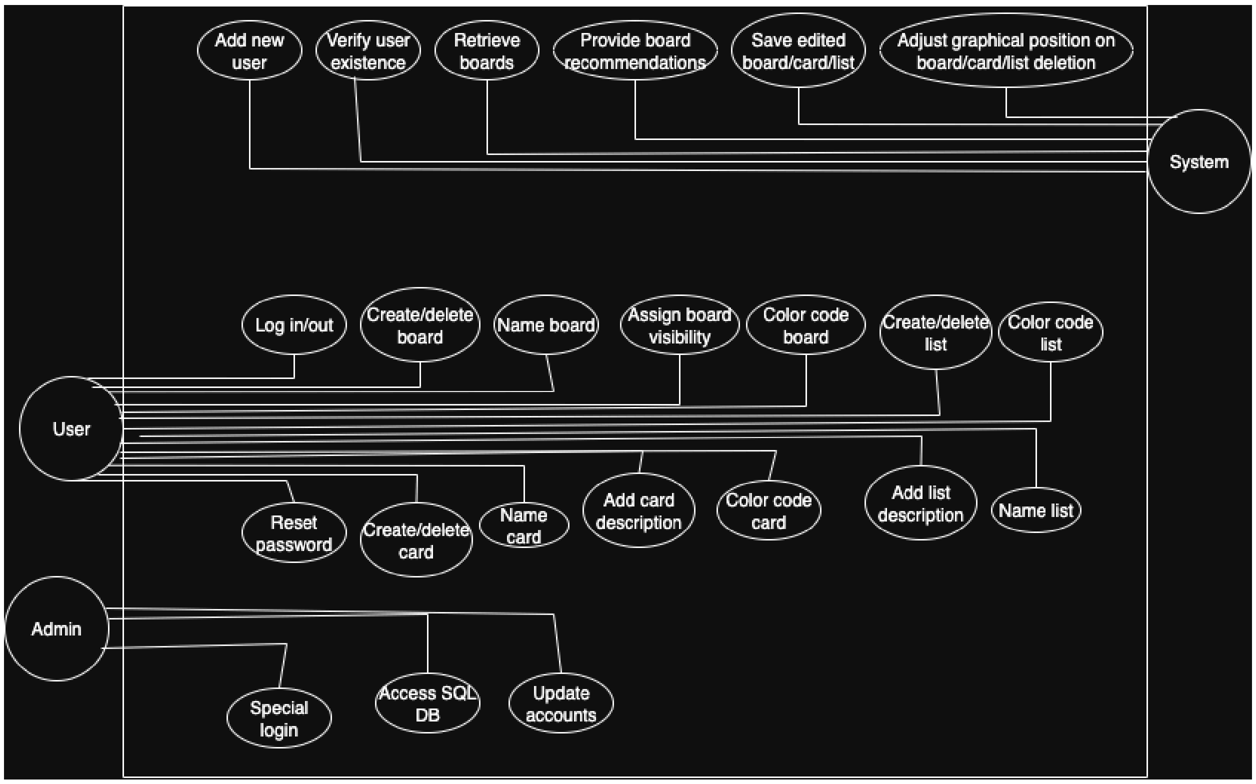
System:

- Add new user
- Verify user existence
- Retrieve boards (and by extension, cards and lists)
- Provide board recommendations
- Save edited board
- Save edited card
- Save edited list
- Adjust the graphical position for a deleted board
- Adjust the graphical position for a deleted card
- Adjust the graphical position for a deleted list

Admin:

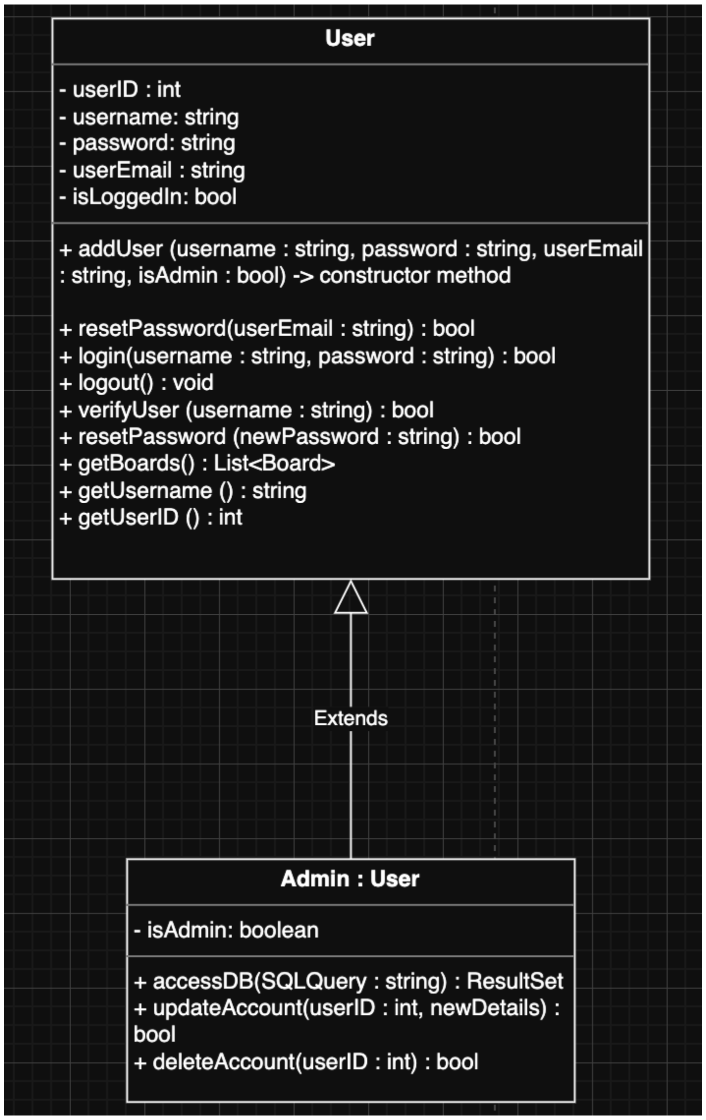
- Special login (system knows it is an admin)
- Able to access SQL database
- Able to update accounts

Use Case Diagram:



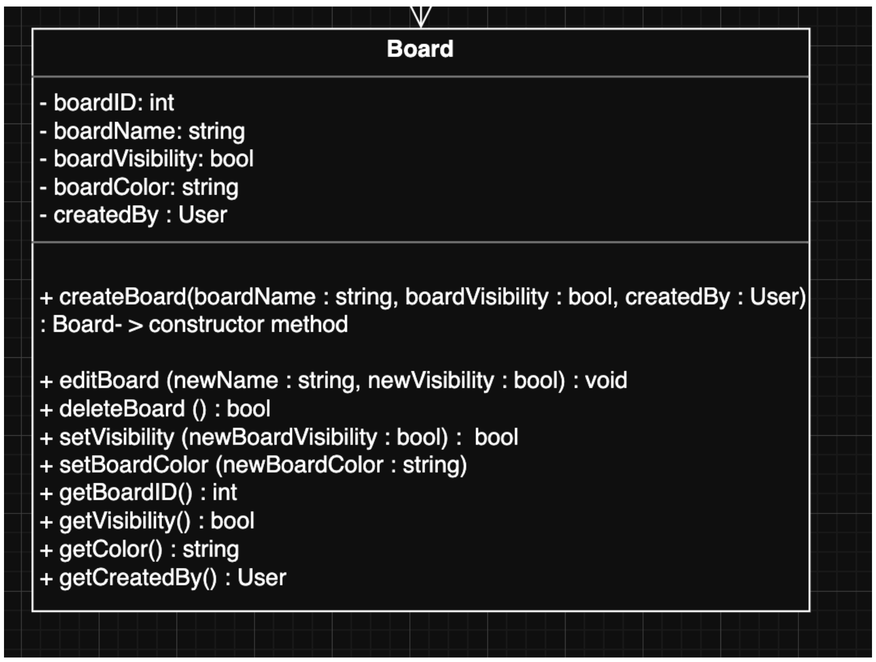
Class Diagram:

A user is someone who works with the system in some way. A standard user has a username, password, and associated email, and is assigned an ID in the database system. They can log in, log out, and reset their password. The system will verify a user's existence when signing up or logging in, and will retrieve the user's boards. They can either be an admin or not. An admin has full functionality of the task management system as well as some admin-specific features, such as querying the SQL database, updating user accounts, and deleting user accounts.


User
- userID : int
- username: string
- password: string
- userEmail : string
- isLoggedIn: bool
+ addUser (username : string, password : string, userEmail : string, isAdmin : bool) -> constructor method
+ resetPassword(userEmail : string) : bool
+ login(username : string, password : string) : bool
+ logout() : void
+ verifyUser (username : string) : bool
+ resetPassword (newPassword : string) : bool
+ getBoards() : List<Board>
+ getUsername () : string
+ getUserID () : int
Extends
Admin : User
- isAdmin: boolean
+ accessDB(SQLQuery : string) : ResultSet
+ updateAccount(userID : int, newDetails) : bool
+ deleteAccount(userID : int) : bool

A board has an ID (assigned by the system), name, visibility, the person it was created by, and optionally a color. Board names, visibility, and colors can be edited, and boards can be deleted.

Boards contain cards and lists.



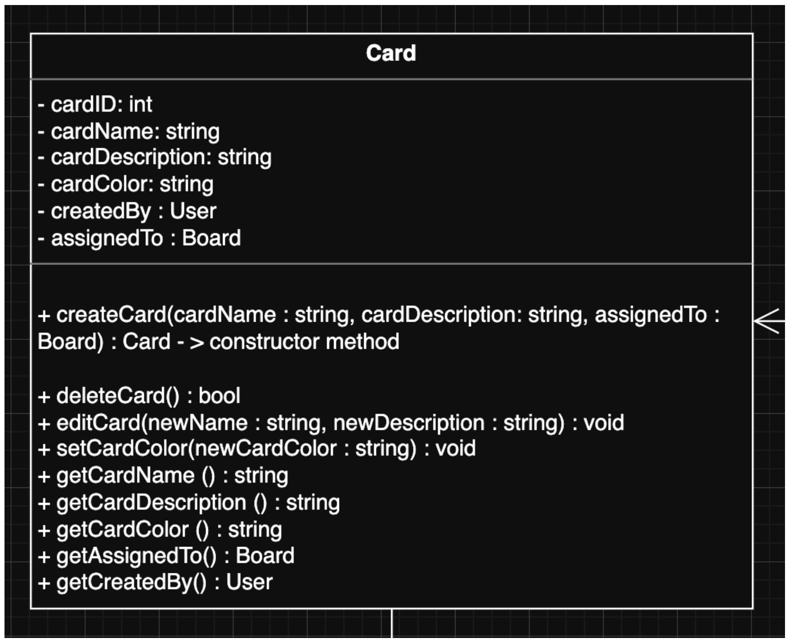
Board

- boardID: int
- boardName: string
- boardVisibility: bool
- boardColor: string
- createdBy : User

+ createBoard(boardName : string, boardVisibility : bool, createdBy : User) : Board- > constructor method

+ editBoard (newName : string, newVisibility : bool) : void
+ deleteBoard () : bool
+ setVisibility (newBoardVisibility : bool) : bool
+ setBoardColor (newBoardColor : string)
+ getBoardID() : int
+ getVisibility() : bool
+ getColor() : string
+ getCreatedBy() : User

A card is an item in a board. It has an ID (assigned by the system), name, optional description, color, a person who created it, and the board it is assigned to. Similar to boards, cards can be edited or deleted.



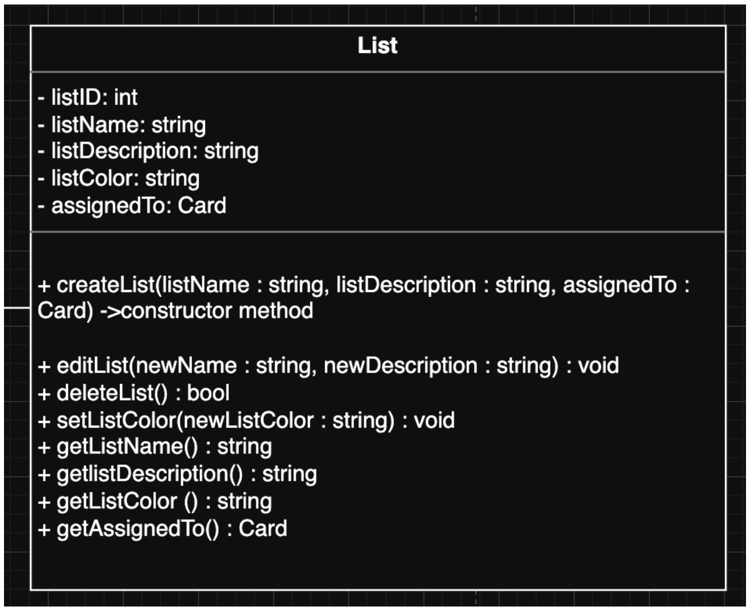
Card

- cardID: int
- cardName: string
- cardDescription: string
- cardColor: string
- createdBy : User
- assignedTo : Board

+ createCard(cardName : string, cardDescription: string, assignedTo : Board) : Card - > constructor method

+ deleteCard() : bool
+ editCard(newName : string, newDescription : string) : void
+ setCardColor(newCardColor : string) : void
+ getCardName () : string
+ getCardDescription () : string
+ getCardColor () : string
+ getAssignedTo() : Board
+ getCreatedBy() : User

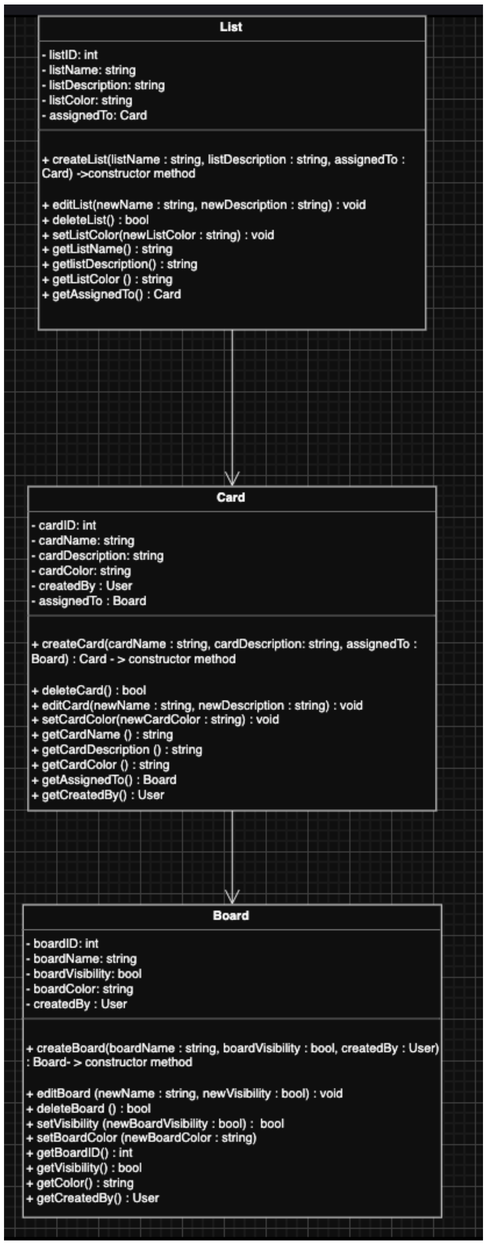
A list is an item in a card. It has an ID (assigned by the system), name, optional description, color, a person who created it, and the card it is assigned to. They can be edited or deleted.


List

- listID: int
- listName: string
- listDescription: string
- listColor: string
- assignedTo: Card

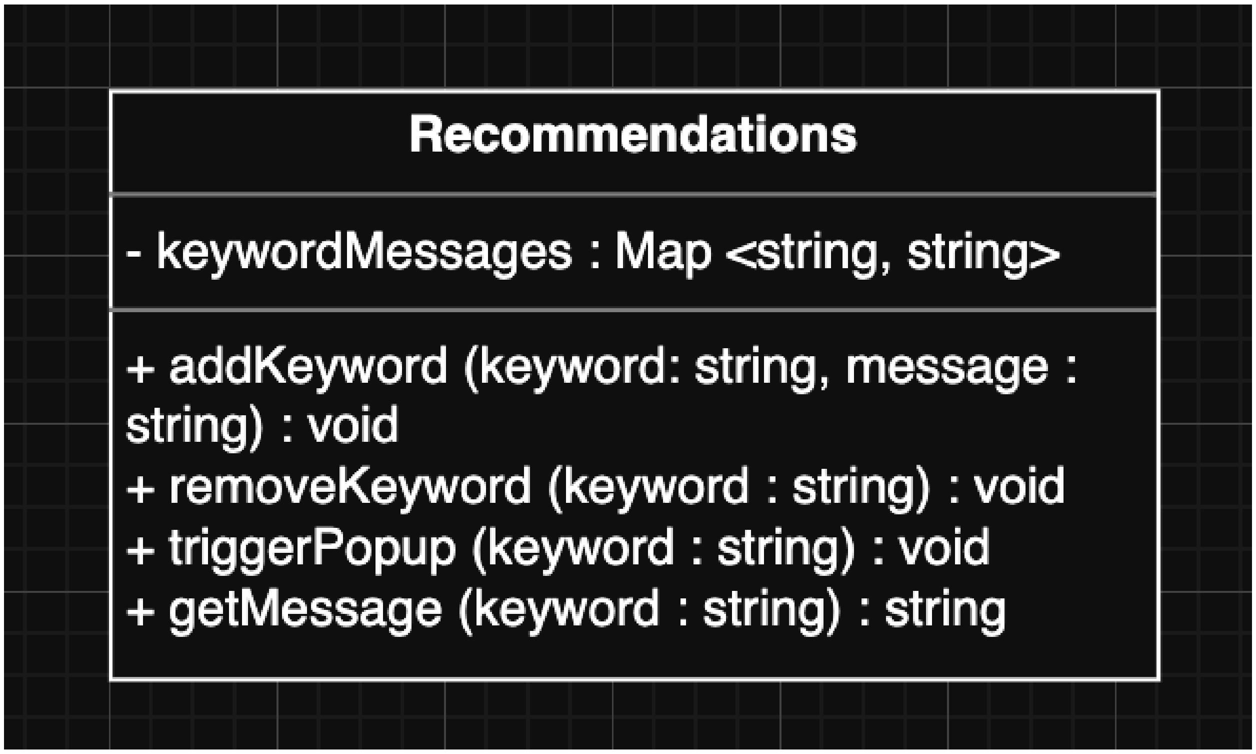
+ createList(listName : string, listDescription : string, assignedTo : Card) ->constructor method

+ editList(newName : string, newDescription : string) : void
+ deleteList() : bool
+ setListColor(newListColor : string) : void
+ getListName() : string
+ getlistDescription() : string
+ getListColor () : string
+ getAssignedTo() : Card


List
- listID: int
- listName: string
- listDescription: string
- listColor: string
- assignedTo: Card
+ createList(listName : string, listDescription : string, assignedTo : Card) ->constructor method
+ editList(newName : string, newDescription : string) : void
+ deleteList() : bool
+ setListColor(newListColor : string) : void
+ getListName() : string
+ getlistDescription() : string
+ getListColor () : string
+ getAssignedTo() : Card
Card
- cardID: int
- cardName: string
- cardDescription: string
- cardColor: string
- createdBy : User
- assignedTo : Board
+ createCard(cardName : string, cardDescription: string, assignedTo : Board) : Card - > constructor method
+ deleteCard() : bool
+ editCard(newName : string, newDescription : string) : void
+ setCardColor(newCardColor : string) : void
+ getCardName () : string
+ getCardDescription () : string
+ getCardColor () : string
+ getAssignedTo() : Board
+ getCreatedBy() : User
Board
- boardID: int
- boardName: string
- boardVisibility: bool
- boardColor: string
- createdBy : User
+ createBoard(boardName : string, boardVisibility : bool, createdBy : User) : Board- > constructor method
+ editBoard (newName : string, newVisibility : bool) : void
+ deleteBoard () : bool
+ setVisibility (newBoardVisibility : bool) : bool
+ setBoardColor (newBoardColor : string)
+ getBoardID() : int
+ getVisibility() : bool
+ getColor() : string
+ getCreatedBy() : User

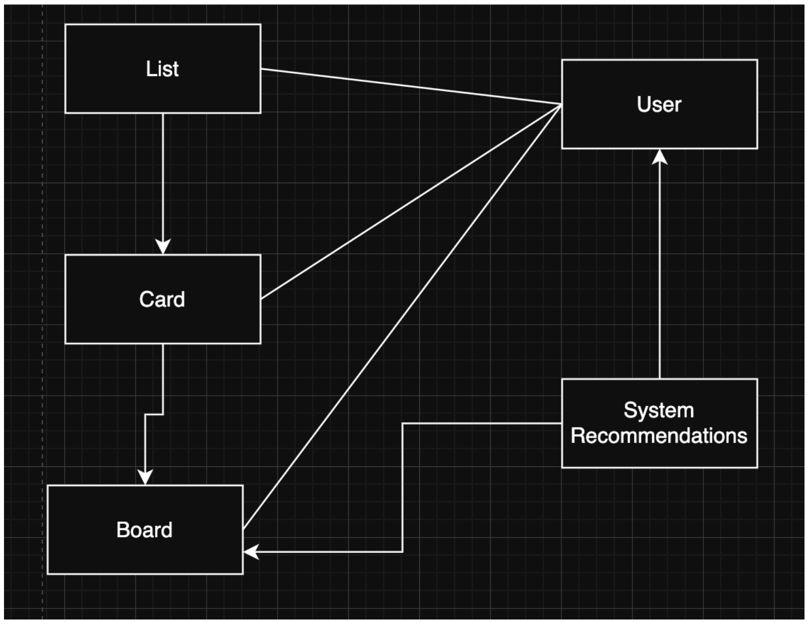
Boards contain cards, and cards contain lists.

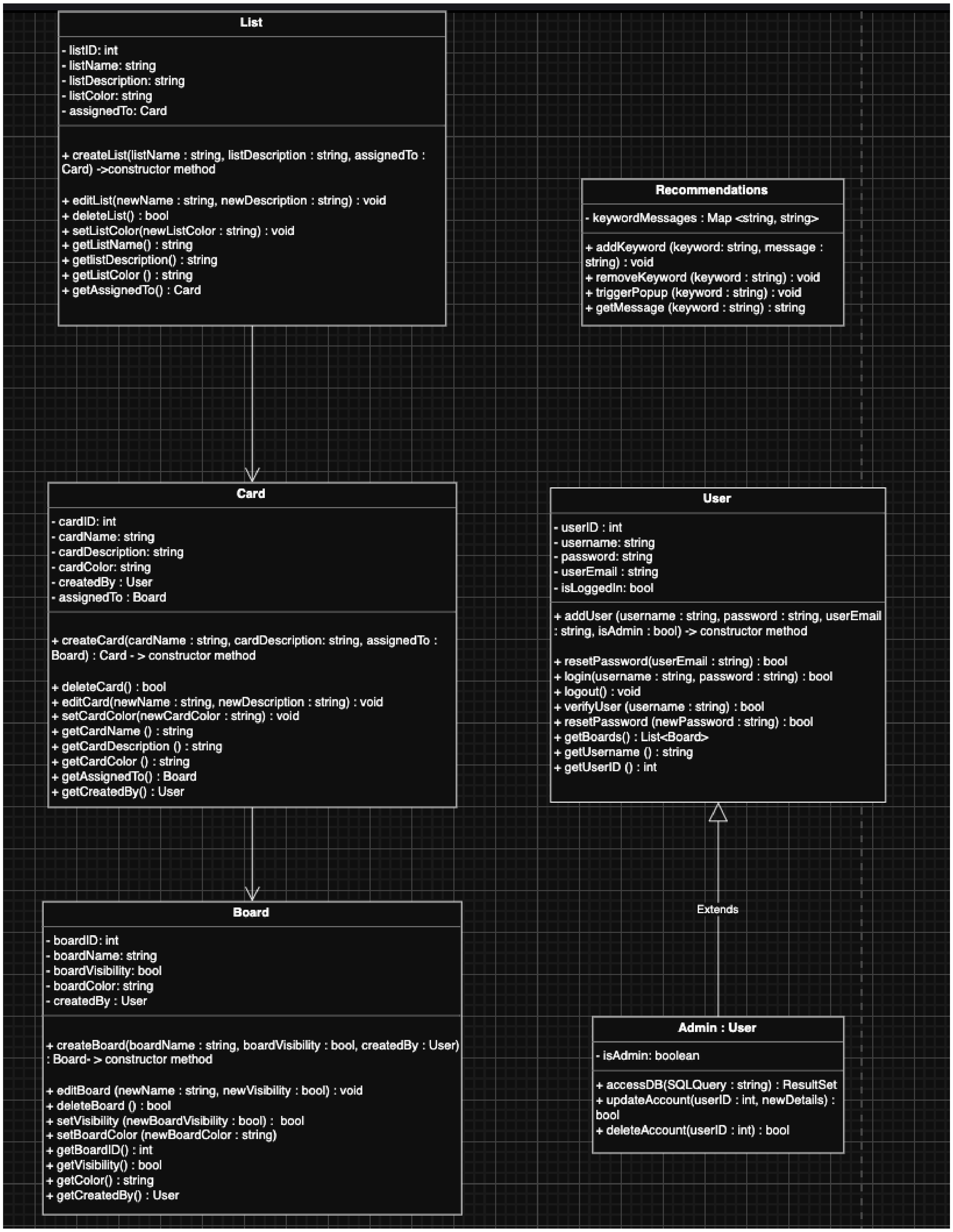
The system can provide recommendations to the user based on specific keywords and phrases in their lists or cards.



```
Recommendations
- keywordMessages : Map <string, string>
+ addKeyword (keyword: string, message : string) : void
+ removeKeyword (keyword : string) : void
+ triggerPopup (keyword : string) : void
+ getMessage (keyword : string) : string
```

Lists are tied to cards and boards. The user can create all three of them. The system can provide recommendations to the user and this can affect the boards, cards, and lists.

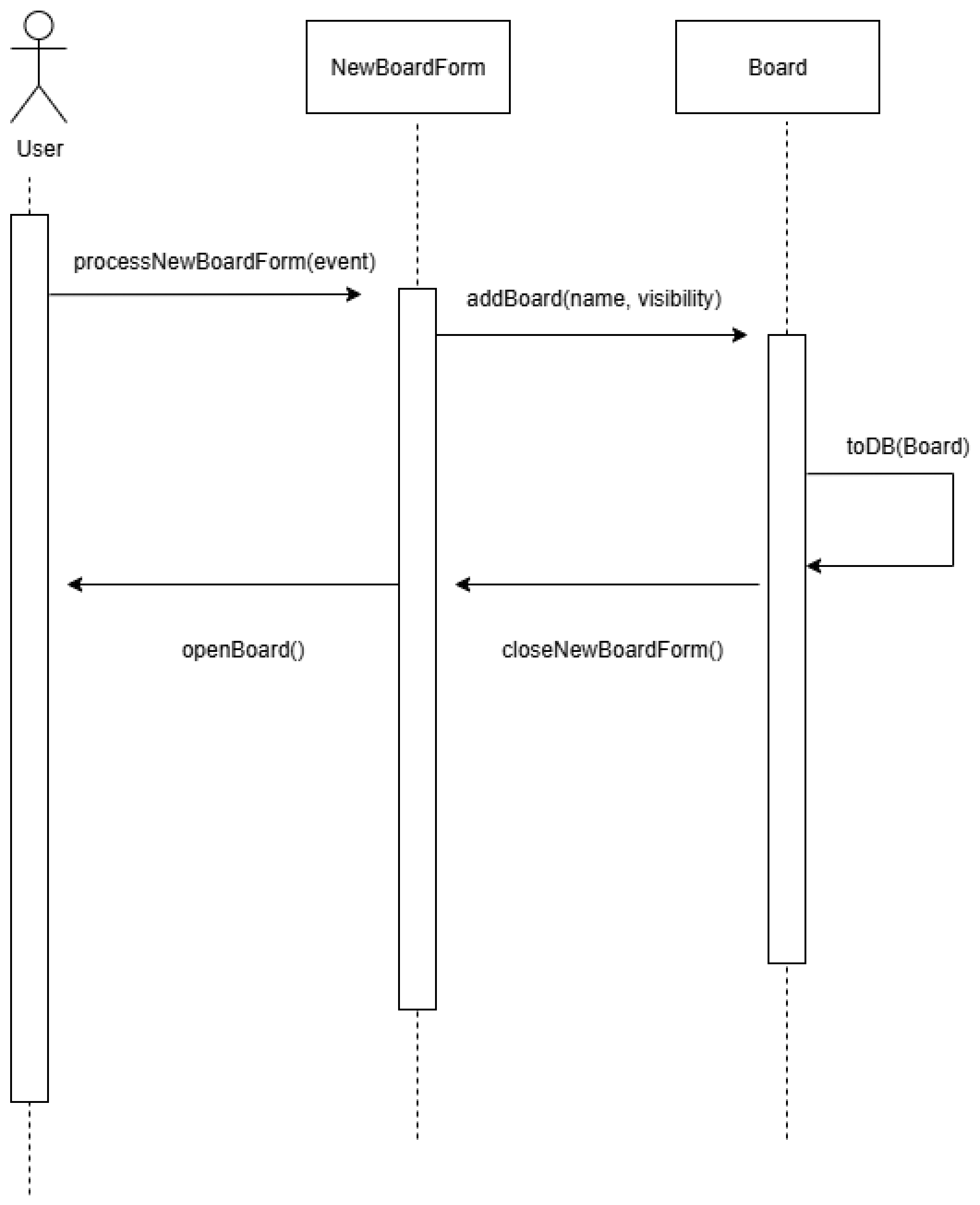



List
- listID: int
- listName: string
- listDescription: string
- listColor: string
- assignedTo: Card
+ createList(listName : string, listDescription : string, assignedTo : Card) ->constructor method
+ editList(newName : string, newDescription : string) : void
+ deleteList() : bool
+ setListColor(newListColor : string) : void
+ getListName() : string
+ getlistDescription() : string
+ getListColor () : string
+ getAssignedTo() : Card
Recommendations
- keywordMessages : Map <string, string>
+ addKeyword (keyword: string, message : string) : void
+ removeKeyword (keyword : string) : void
+ triggerPopup (keyword : string) : void
+ getMessage (keyword : string) : string
Card
- cardID: int
- cardName: string
- cardDescription: string
- cardColor: string
- createdBy : User
- assignedTo : Board
+ createCard(cardName : string, cardDescription: string, assignedTo : Board) : Card - > constructor method
+ deleteCard() : bool
+ editCard(newName : string, newDescription : string) : void
+ setCardColor(newCardColor : string) : void
+ getCardName () : string
+ getCardDescription () : string
+ getCardColor () : string
+ getAssignedTo() : Board
+ getCreatedBy() : User
User
- userID : int
- username: string
- password: string
- userEmail : string
- isLoggedIn: bool
+ addUser (username : string, password : string, userEmail : string, isAdmin : bool) -> constructor method
+ resetPassword(userEmail : string) : bool
+ login(username : string, password : string) : bool
+ logout() : void
+ verifyUser (username : string) : bool
+ resetPassword (newPassword : string) : bool
+ getBoards() : List<Board>
+ getUsername () : string
+ getUserID () : int
Extends
Board
- boardID: int
- boardName: string
- boardVisibility: bool
- boardColor: string
- createdBy : User
+ createBoard(boardName : string, boardVisibility : bool, createdBy : User) : Board- > constructor method
+ editBoard (newName : string, newVisibility : bool) : void
+ deleteBoard () : bool
+ setVisibility (newBoardVisibility : bool) : bool
+ setBoardColor (newBoardColor : string)
+ getBoardID() : int
+ getVisibility() : bool
+ getColor() : string
+ getCreatedBy() : User
Admin : User
- isAdmin: boolean
+ accessDB(SQLQuery : string) : ResultSet
+ updateAccount(userID : int, newDetails) : bool
+ deleteAccount(userID : int) : bool

System Sequence Diagram and Activity Diagram

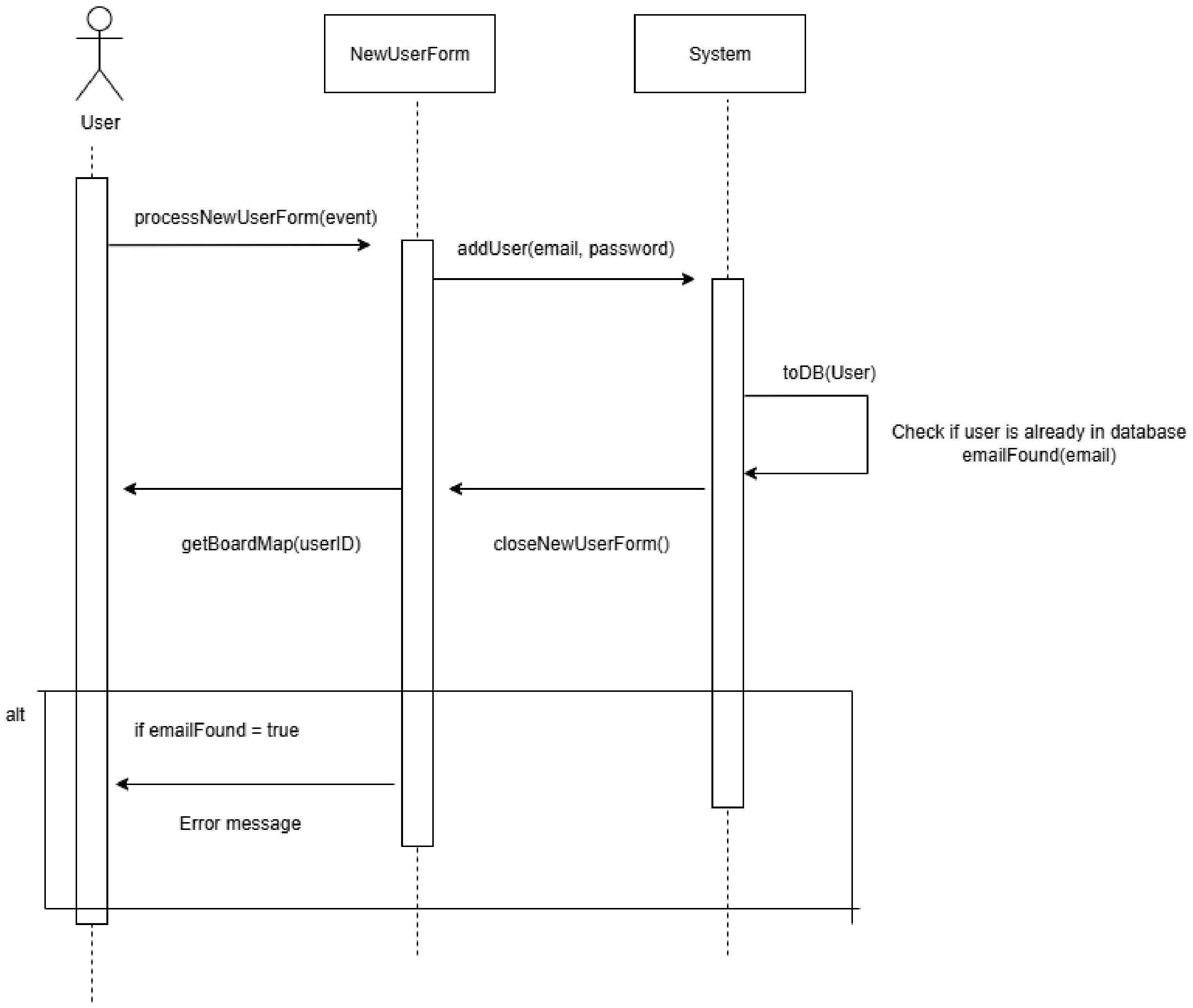
Sequence 1: Create a new board

- **Actor**: User
- **Object:** NewBoardForm, Board
- User opens “New Board” form
- User enters board name
- User selects board visibility
- New board form processes the user input
- System creates a new board using the addBoard method
- System adds the new board graphically on the screen


User
NewBoardForm
Board
processNewBoardForm(event)
addBoard(name, visibility)
toDB(Board)
openBoard()
closeNewBoardForm()

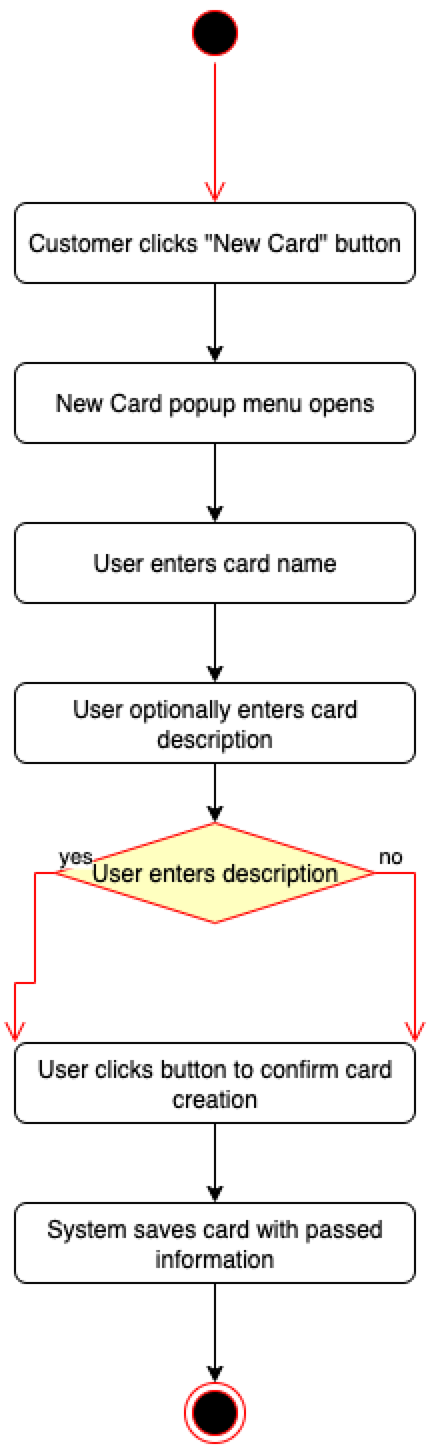
Sequence 2: Create a new user

- Actor: User
- User opens “Sign Up” form
- User enters email and password
- System checks if email is already present in database
- If the email is already present in the database, return an error message
- If the email is not already present in the database, create the new user



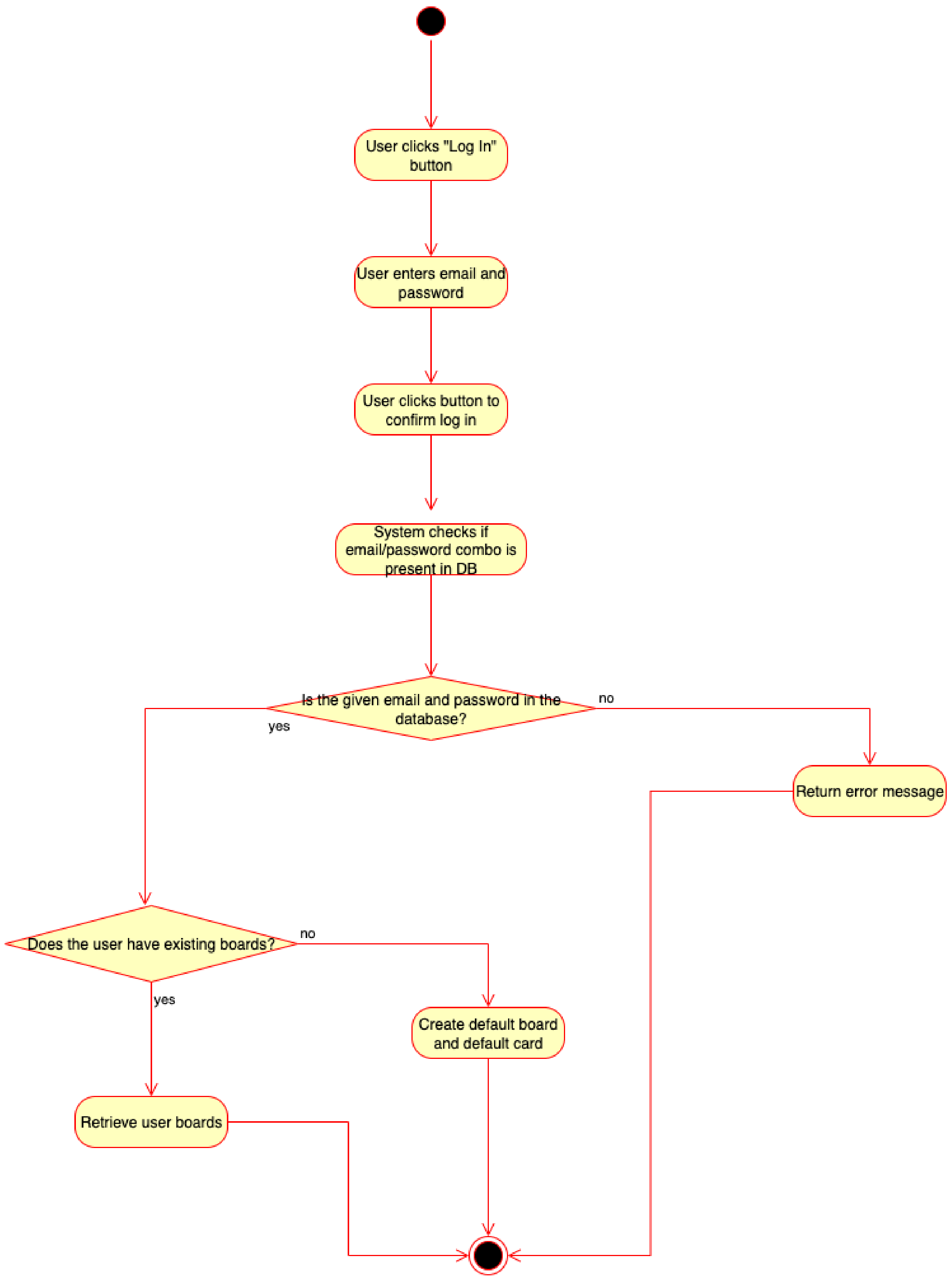
Activity 1: Create a new card

- Initial state: User clicks “New Card” button
- Final state: A new card is created
- Actions: The user clicks the “New Card” button, and enters the card name and optionally, a card description. The user will click a button to confirm card creation. The system will create the card with the passed information and save this information to the database.


Customer clicks "New Card" button
New Card popup menu opens
User enters card name
User optionally enters card description
yes
User enters description
no
User clicks button to confirm card creation
System saves card with passed information

Activity 2: Retrieve boards

- Initial state: User logs into site
- Final states:
    - The system retrieves the user’s boards
    - The system finds no boards and returns a single default board
- Actions: The user clicks the “Log In” button, enters their login credentials, and then clicks another button to log in. The system will verify that the username and password are a match in the database. If they are not a match, the system will return an error message. If they are a match, the system will retrieve the user’s boards. If the user has no boards, a default board with a default card will be created.


User clicks "Log In" button
User enters email and password
User clicks button to confirm log in
System checks if email/password combo is present in DB
Is the given email and password in the database?
no
yes
Return error message
Does the user have existing boards?
no
yes
Create default board and default card
Retrieve user boards

UI Specification

User Interface Specification
Alt link:
https://docs.google.com/presentation/d/1Rqph3MpXU9_SXSnDY4NbcoZqgcU44I_RR9e98R5a6qA/edit?slide=id.p#slide=id.p

Traceability Matrix

Traceability System Requirements

No.	Priority Weight 1-5	Description
REQ 1	5	The system should be able to list tasks
REQ 2	5	Users should be able to have multiple, different task lists
REQ 3	4	Users must be able to prioritize their tasks through color coding
REQ 4	3	Tasks should be sortable
REQ 5	4	The system should be able to provide specific, relevant recommendations for strategy game tasks when appropriate
REQ 6	5	The system should not just be for games, but should be able to be used for other task monitoring/management purposes as well.
REQ 7	4	The system should be able to ask the user if they are playing a game that has

		recommendation support
REQ 8	5	The system should be persistent
REQ 9	2	The user should be warned when deleting a task for the first time
REQ 10	5	The system should allow the user to create as many boards as they want

Use Cases

No.	Description
UC1	Standard login: to log in to a standard user account
UC2	Admin login: to log in to an admin account
UC3	Validate credentials: to verify that username and password are valid
UC4	Validate login: to ensure that the user is logged in
UC5	Create Board: to create a new board
UC6	Edit Board: to color-code and edit board details.
UC7	Delete Board: to delete the board
UC8	Create Card: to create a new card
UC9	Edit Card: to color-code card, edit card name, or edit card description
UC10	Delete Card: to delete the card
UC11	Create List: to create a new list
UC12	Edit List: to color-code a list, edit list name, or edit list description
UC13	Delete List: to delete the list

UC14	Provide Recommendation: to ask the user if they are playing a supported game
UC15	Check User Content: to search user content that can help provide game recommendations
UC16	Accept System Recommendation: to accept the system's recommendation for a new card or list item
UC17	Logout: to log out the user

Traceability Matrix

RQ	PW	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6	UC 7	UC 8	UC 9	UC 10	UC 11	UC 12	UC 13	UC 14	UC 15	UC 16	UC 17
RQ 1	5				X	X	X		X	X		X	X				X	
RQ 2	5					X			X			X						
RQ 3	4						X			X			X					
RQ 4	3						X			X			X					
RQ 5	4														X	X	X	
RQ 6	5					X	X	X	X	X	X	X	X	X				
RQ 7	4														X	X		
RQ 8	5	X	X	X	X													X
RQ 9	2							X			X			X				
RQ 10	5				X	X												
Max PW		5	5	5	5	5	5	5	5	5	5	5	5	5	4	4	5	5

Total PW	5	5	5	15	20	17	7	15	17	7	15	17	7	8	8	9	5

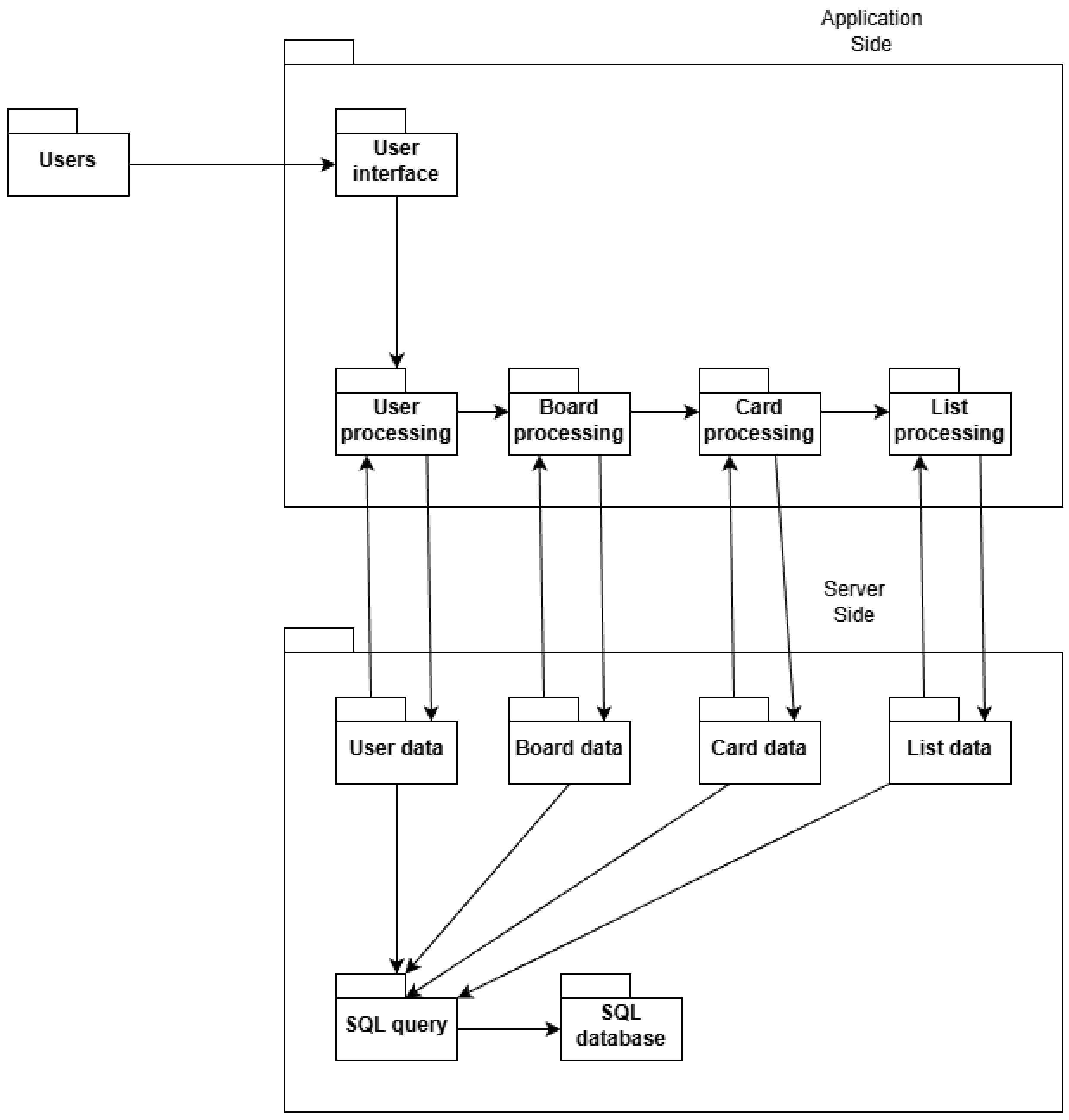
System Architecture and System Design

Architectural Styles

This task management system is a client-server architectural style. The client sends requests to the server and receives a response. In this case, the client is sending requests to do things like log in, retrieve boards and lists, and read and write functions to these lists. A client in this instance is a computer, and the server is a local server through XAMPP. The client side is built through HTML, CSS, and JavaScript. The server side uses Apache, MySQL, and PHP. The user will interact with the client side, and methods in JavaScript will call PHP files that will, in turn, be able to create, read, update, and delete items from the database. The MySQL is used to store information about users, and the information about the users is then used to connect their boards, cards, and lists to their account. There is one client/machine, my computer.

Identifying Subsystems

The package diagram below shows the subsystems of the task management website. Since it is a client-server style website, the diagram includes a client (application) side and server side. The user begins the procedure by interacting with the user interface. This is through clicking HTML/CSS/JavaScript buttons in order to do something. Through the user interface, the user can do something to their user account, their boards, their cards, or their lists. This information is then passed to the server through the system’s PHP files, which are able to create, read/write, update, and delete from the SQL database. The database is then updated, and the results of this are displayed graphically on the application, so that the user can further interact with the system.



Persistent Data Storage

This system requires the preservation of data beyond a single execution of the system. This is done through storing information in a relational database, which in this instance is the MySQL database. This database stores usernames and passwords, as well as information about boards (board ID, board name, board privacy, board color, and the associated user), cards (card ID, card name, card description, card color, associated user, and associated board), and lists (list ID, list name, list description, list color, associated card, and associated user).

Global Control Flow

Execution Orders

The task management system is event-driven that waits in a loop, so users can generate actions in whichever order they want. The user can click whichever buttons they want, and these buttons all have different actions assigned to them, with many that are reflected graphically on the website as well as in the database.

Time Dependency

The task management system does not have any timers, and there is no concern for real-time. There are sequential operations, but no time constraints for these. For example, a user needs to have a board created in order to use the “edit board” features, but there is no timer for this.

Hardware Requirements

The website relies on the user having a monitor, storage (to hold the source files), and XAMPP. Currently, no mobile support is implemented. The user’s hardware should consist of any computer monitor, 1 GB RAM, and a minimum of 100 mb of storage space.

Project Plan

Were this project to continue, I would want to implement the recommendation system as an object on the website that could be interacted with. For example, a user could click on a certain recommendation to create a new card for that certain recommendation. I would want to further iron out bugs and continue developing the UI with more modern methods of web development, such as using React or Angular instead of basic CSS.