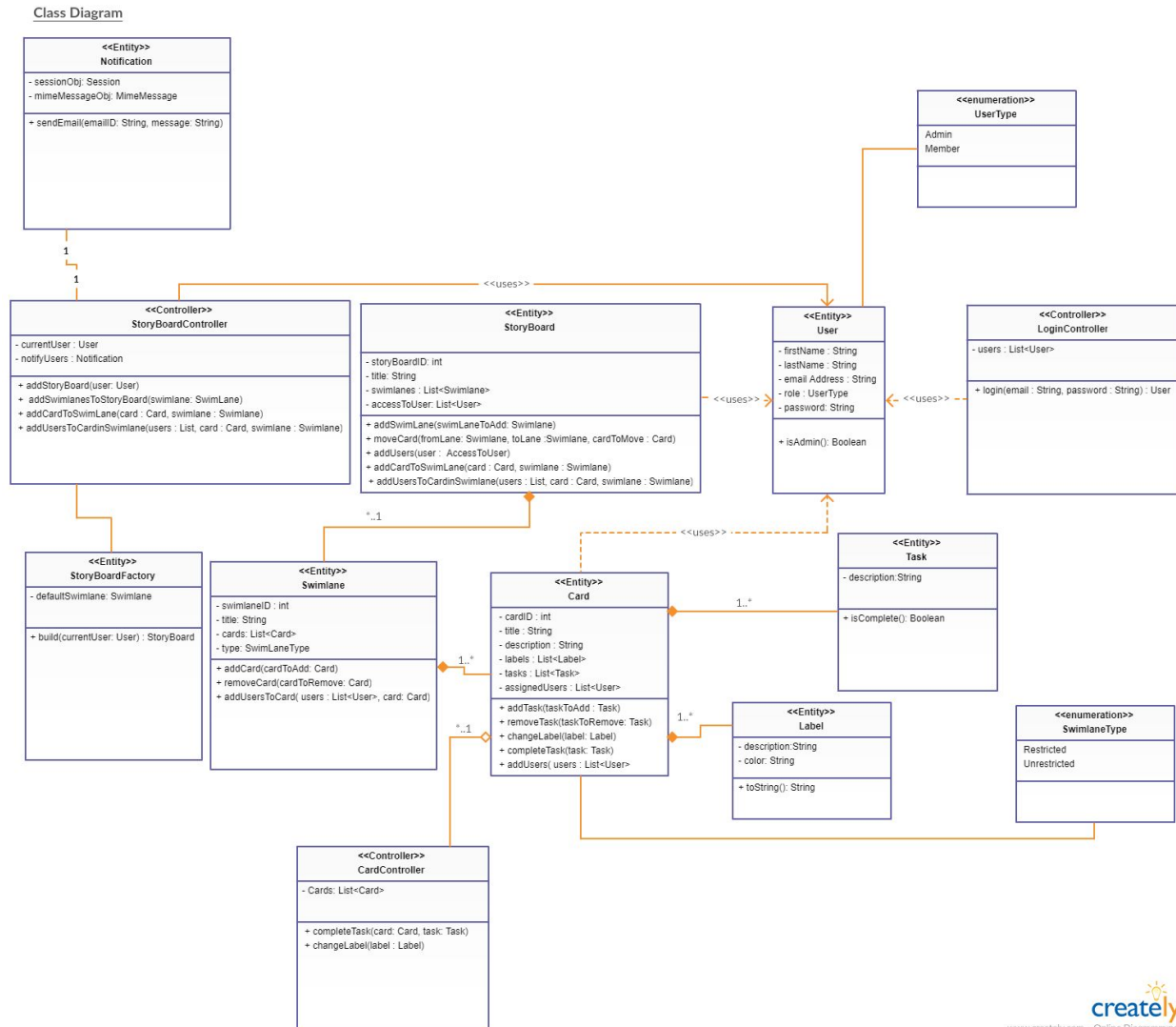


Name - Project Management System

Team members - Ankit Srivastava, Nikhil Sulegaon, Rohit Mehra

Old Class Diagram



```

classDiagram
    class StoryBoardController {
        <<Controller>>
        -currentUser: User
        +addStoryboard(user: User):Void
        +addSwimlanesToStoryboard(swimlane: Swimlane):Void
        +addCardToSwimlane(card: Card, swimlane: Swimlane):Void
        +addUsersToCardInSwimlane(users: List, card: Card, swimlane: Swimlane):Void
    }
    class StoryBoard {
        <<Entity>>
        -storyBoardID: int
        -title: String
        -swimlanes: List<SwimlaneProxy>
        -accessToUser: List<User>
        +addSwimLane(swimLaneToAdd: Swimlane):Void
        +moveCard(fromLane: Swimlane, toLane: Swimlane, cardToMove: Card):Void
        +addUsers(user: AccessToUser):Void
        +addCardToSwimlane(card: Card, swimlane: Swimlane):Void
        +addUsersToCardInSwimlane(users: List, card: Card, swimlane: Swimlane):Void
    }
    class User {
        <<Entity>>
        -firstName: String
        -lastName: String
        -email Address: String
        -role: UserType
        -password: String
        +isAdmin(): Boolean
    }
    class LoginController {
        <<Controller>>
        users: List<User>
        login(email: String, password: String):Void
    }
    class StoryBoardFactory {
        <<Entity>>
        -defaultSwimlane: Swimlane
        +build(currentUser: User):Void
    }
    class SwimlaneProxy {
        <<Entity>>
        -swimlane: Swimlane
        +addCard(cardToAdd: Card):Void
        +removeCard(cardToRemove: Card):Void
        +addUsersToCard(users: List<User>, card: Card):Void
    }
    class Card {
        <<Entity>>
        -cardID: int
        -title: String
        -description: String
        -labels: List<Label>
        -tasks: List<Task>
        -assignedUsers: List<User>
        -observers: EmailNotificationObserver
        +addTask(taskToAdd: Task):Void
        +removeTask(taskToRemove: Task):Void
        +changeLabel(label: Label):Void
        +completeTask(task: Task):Void
        +addUsers(users: List<User>):Void
    }
    class Task {
        <<Entity>>
        -description: String
        +isComplete(): Boolean
    }
    class Label {
        <<Entity>>
        -description: String
        -color: String
        +toString(): String
    }
    class CardController {
        <<Controller>>
        -cards: List<Card>
        +completeTask(card: Card, task: Task):Void
        +changeLabel(label: Label):Void
    }
    class Swimlane {
        <<Entity>>
        -swimlaneID: int
        -title: String
        -cards: List<Card>
        -type: SwimlaneType
        +addCard(cardToAdd: Card):Void
        +removeCard(cardToRemove: Card):Void
        +addUsersToCard(users: List<User>, card: Card):Void
    }
    class EmailNotificationObserver {
        <<Entity>>
        -emailAddress: String
        +notify():Void
    }
    class IObserver {
        <<Interface>>
        +notify():Void
    }
    class UserType {
        <<enumeration>>
        Admin
        Member
    }
    class SwimlaneType {
        <<enumeration>>
        Restricted
        Unrestricted
    }

    StoryBoardController --> StoryBoard : 1..*
    StoryBoardController --> User : 1..*
    StoryBoardController --> LoginController : 1..*
    StoryBoardFactory --> StoryBoard : 0..1
    StoryBoardFactory --> SwimlaneProxy : 0..1
    SwimlaneProxy --> Card : 1..*
    Card --> Task : 1..*
    Card --> Label : 1..*
    Card --> EmailNotificationObserver : 1..*
    CardController --> Card : *..1
    CardController --> Swimlane : *..1
    Swimlane --> Card : 1..*
    Swimlane --> EmailNotificationObserver : 1..*
    EmailNotificationObserver --> IObserver : 1..*
    User --> UserType : 1..*
    Swimlane --> SwimlaneType : 1..*
    
```

The diagram illustrates a Factory Pattern for a project management application. It includes classes for controllers, entities, and interfaces, along with enumerations for user types and swimlane types. The relationships are defined by associations, generalizations, and enumerations.

Classes and Interfaces:

- StoryboardController** (Controller): Manages the storyboard, including adding users, swimlanes, cards, and tasks.
- Storyboard** (Entity): Represents a storyboard with attributes like ID, title, swimlanes, and access to users.
- User** (Entity): Represents a user with attributes like first name, last name, email address, role, and password.
- LoginController** (Controller): Handles user login.
- StoryboardFactory** (Entity): A factory for creating storyboards and swimlanes.
- SwimlaneProxy** (Entity): A proxy for a swimlane, used by the factory.
- Card** (Entity): Represents a card with attributes like ID, title, description, labels, tasks, assigned users, and observers.
- Task** (Entity): Represents a task with a description and a completion status.
- Label** (Entity): Represents a label with a description and color.
- CardController** (Controller): Manages a list of cards, including completing tasks and changing labels.
- Swimlane** (Entity): Represents a swimlane with attributes like ID, title, cards, and type.
- EmailNotificationObserver** (Entity): An observer that receives email notifications.
- IObserver** (Interface): An interface for observers.

Enumerations:

- UserType**: Admin, Member.
- SwimlaneType**: Restricted, Unrestricted.

Relationships:

- StoryboardController** has a 1..* association with **Storyboard**, **User**, and **LoginController**.
- StoryboardFactory** has a 0..1 association with **Storyboard** and **SwimlaneProxy**.
- SwimlaneProxy** has a 1..* association with **Card**.
- Card** has a 1..* association with **Task**, **Label**, and **EmailNotificationObserver**.
- CardController** has a *..1 association with **Card** and a *..1 association with **Swimlane**.
- Swimlane** has a 1..* association with **Card** and a 1..* association with **EmailNotificationObserver**.
- EmailNotificationObserver** has a 1..* association with **IObserver**.
- User** has a 1..* association with **UserType**.
- Swimlane** has a 1..* association with **SwimlaneType**.

Notes:

- A note on the **EmailNotificationObserver** class states: "Send email to all observers after every card change".

Design Pattern Used

- Factory Design Pattern

Factory Design Pattern is one of the most important creational design pattern. In our project we are using factory design pattern to create storyboard object using storyboardFactory.

- Factory Design Pattern

Factory Design Pattern is one of the most important creational design pattern. In our project we are using factory design pattern to create storyboard object using storyboardFactory.

Design Pattern Added

- Observer Design Pattern
- Observer is one of the behavioral design pattern. It defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. In our project we are using it to send email notifications to users(assigned to)

whenever any card is changed. EmailNotificationObserver sends email to all the users associated to a card whenever any change is made to that card.

- Proxy Design Pattern

Proxy pattern is one of the structural design pattern. It provides a surrogate or placeholder for another object to control access to it. In our project we are using it to restrict user access to specific card. Users should be only able to change card if the card is not in restricted swimlane. Proxy pattern helps us to restrict access to card if the card is in restricted swimlane. SwimlaneProxy class helps to restrict user access to restricted swimlanes.