Part 6: Final Report

CSCI 5448
Elizabeth Boese

Name - Project Management System
Team members - Ankit Srivastava, Nikhil Sulegaon, Rohit Mehra

Summary :-

Prorg is a web-based collaboration tool that organizes projects into boards. In one glance, Prorg tells you what's being worked on, who's working on what, and where something is in a process.

Imagine a white board, filled with lists of sticky notes, with each note as a task for you and your team. Now imagine that each of those sticky notes has photo-attachments. Now imagine that you can take that whiteboard anywhere you go, and can access it from any computer through the web. That's Prorg!

1. Project Features:

Features Implemented	
ID	**Title**
UR-1	User Registration
UR-2	User Login
UR-3	User should be able to Add Storyboard
UR-4	User should be able to Remove Storyboard
UR-5	User should be able to Update Storyboard
UR-6	User should be able to Add Swimlane to Storyboard
UR-7	User should be able to Remove Swimlane
UR-8	User should be able to Update Swimlane
UR-9	User should be able to Add Card to Swimlane
UR-10	User should be able to Remove Card from SwimLane
UR-11	User should be able to Update Card
UR-12	User should be able to Assign User to Card
UR-13	User should be able to Move Card from one Swimlane to another
UR-14	User should be able to Approve Card
UR-15	User should be able to Assign User to StoryBoard
NFR-01	Application can be accessed from all the browsers(Laptop/Desktop/Mobile/Tablet)

NFR-02	Application can only be accessed by authorized users after successful login
NFR-03	Adding a storyboard , swimlane or card should not take more than 2 seconds
NFR-04	High Speed Connectivity will be needed to use the application
NFR-05	Application should be able to scale itself depending on the number of users

2. Features listed while starting the project

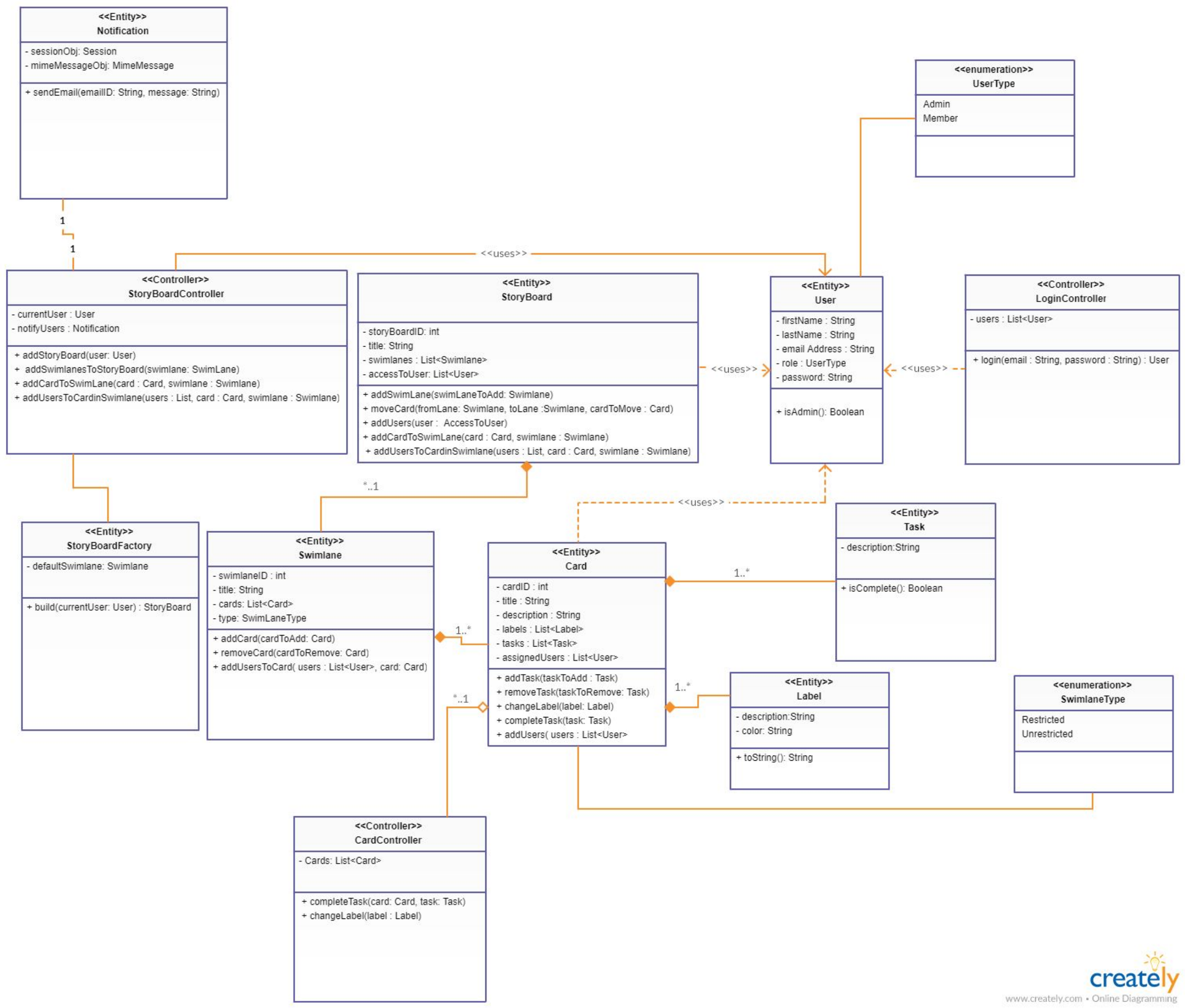
Features Mentioned in the start	
ID	Title
UR-1	User Registration
UR-2	User Login
UR-3	User should be able to Add Storyboard
UR-4	User should be able to Remove Storyboard
UR-5	User should be able to Update Storyboard
UR-6	User should be able to Add Swimlane to Storyboard
UR-7	User should be able to Remove Swimlane
UR-8	User should be able to Update Swimlane
UR-9	User should be able to Add Card to Swimlane
UR-10	User should be able to Remove Card from SwimLane
UR-11	User should be able to Update Card
UR-12	User should be able to Assign User to Card
UR-13	User should be able to Move Card from one Swimlane to another
UR-14	User should be able to Approve Card
UR-15	User should be able to Assign User to StoryBoard
UR-16	Comment functionality

UR-17	Insert Label to Card
UR-18	Email functionality
NFR-01	Application can be accessed from all the browsers(Laptop/Desktop/Mobile/Tablet)
NFR-02	Application can only be accessed by authorized users after successful login
NFR-03	Adding a storyboard , swimlane or card should not take more than 2 seconds
NFR-04	High Speed Connectivity will be needed to use the application
NFR-05	Application should be able to scale itself depending on the number of users

3. Class Diagram

Part 2 Class Diagram

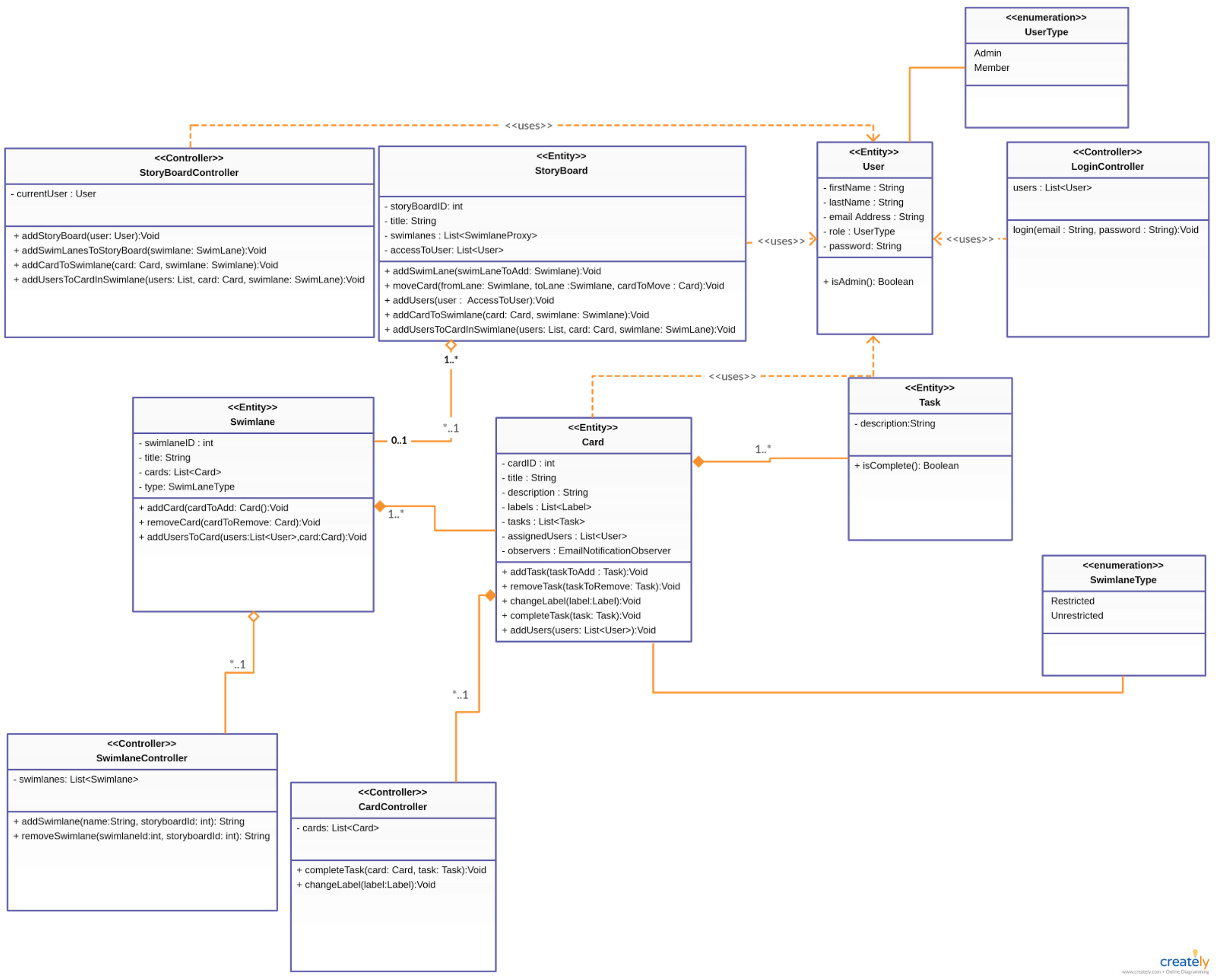
Class Diagram



<<Entity>>
Notification
- sessionObj: Session
- mimeMessageObj: MimeMessage
+ sendEmail(emailID: String, message: String)
1
1
<<uses>>
<<Controller>>
StoryBoardController
- currentUser : User
- notifyUsers : Notification
+ addStoryBoard(user: User)
+ addSwimlanesToStoryBoard(swimlane: SwimLane)
+ addCardToSwimLane(card : Card, swimlane : Swimlane)
+ addUsersToCardinSwimlane(users : List, card : Card, swimlane : Swimlane)
<<Entity>>
StoryBoard
- storyBoardID: int
- title: String
- swimlanes : List<Swimlane>
- accessToUser: List<User>
+ addSwimLane(swimLaneToAdd: Swimlane)
+ moveCard(fromLane: Swimlane, toLane :Swimlane, cardToMove : Card)
+ addUsers(user : AccessToUser)
+ addCardToSwimLane(card : Card, swimlane : Swimlane)
+ addUsersToCardinSwimlane(users : List, card : Card, swimlane : Swimlane)
<<uses>>
<<Entity>>
User
- firstName : String
- lastName : String
- email Address : String
- role : UserType
- password: String
+ isAdmin(): Boolean
<<uses>>
<<Controller>>
LoginController
- users : List<User>
+ login(email : String, password : String) : User
<<enumeration>>
UserType
Admin
Member
*..1
<<uses>>
<<Entity>>
StoryBoardFactory
- defaultSwimlane: Swimlane
+ build(currentUser: User) : StoryBoard
<<Entity>>
Swimlane
- swimlaneID : int
- title: String
- cards: List<Card>
- type: SwimLaneType
+ addCard(cardToAdd: Card)
+ removeCard(cardToRemove: Card)
+ addUsersToCard(users : List<User>, card: Card)
1..*
*..1
<<Entity>>
Card
- cardID : int
- title : String
- description : String
- labels : List<Label>
- tasks : List<Task>
- assignedUsers : List<User>
+ addTask(taskToAdd : Task)
+ removeTask(taskToRemove: Task)
+ changeLabel(label: Label)
+ completeTask(task: Task)
+ addUsers(users : List<User>
1..*
<<Entity>>
Task
- description:String
+ isComplete(): Boolean
1..*
<<Entity>>
Label
- description:String
- color: String
+ toString(): String
<<enumeration>>
SwimlaneType
Restricted
Unrestricted
<<Controller>>
CardController
- Cards: List<Card>
+ completeTask(card: Card, task: Task)
+ changeLabel(label : Label)

creately
www.creately.com • Online Diagramming

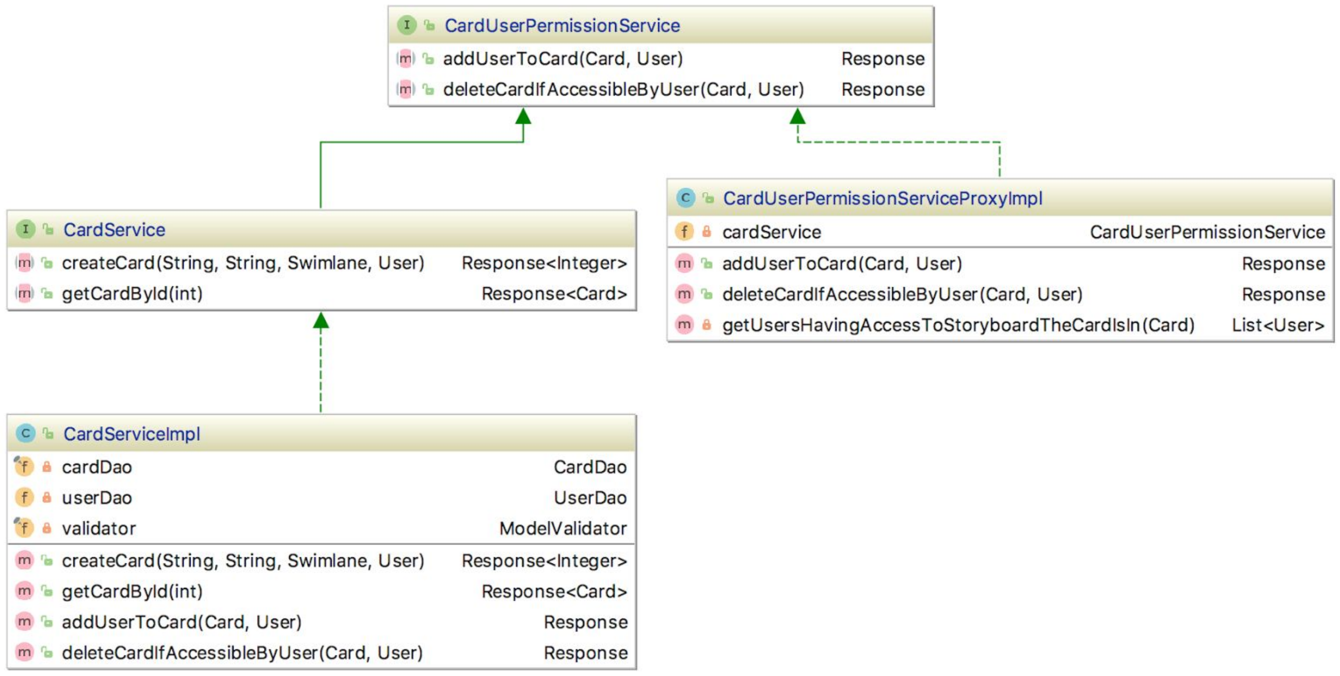
Latest Class Diagram



<<enumeration>> UserType
Admin
Member
<<uses>>
<<Controller>> StoryBoardController
- currentUser : User
+ addStoryBoard(user: User):Void
+ addSwimLanesToStoryBoard(swimlane: SwimLane):Void
+ addCardToSwimlane(card: Card, swimlane: Swimlane):Void
+ addUsersToCardInSwimlane(users: List, card: Card, swimlane: SwimLane):Void
<<Entity>> StoryBoard
- 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
<<uses>>
<<Entity>> User
- firstName : String
- lastName : String
- email Address : String
- role : UserType
- password: String
+ isAdmin(): Boolean
<<uses>>
<<Controller>> LoginController
users : List<User>
login(email : String, password : String):Void
1..*
<<uses>>
<<Entity>> Swimlane
- swimlaneID : int
- title: String
- cards: List<Card>
- type: SwimLaneType
+ addCard(cardToAdd: Card():Void
+ removeCard(cardToRemove: Card):Void
+ addUsersToCard(users:List<User>,card:Card):Void
0..1
*..1
<<Entity>> Card
- 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
1..*
<<Entity>> Task
- description:String
+ isComplete(): Boolean
1..*
<<enumeration>> SwimlaneType
Restricted
Unrestricted
*..1
*..1
<<Controller>> SwimlaneController
- swimlanes: List<Swimlane>
+ addSwimlane(name:String, storyboardId: int): String
+ removeSwimlane(swimlaneId:int, storyboardId: int): String
<<Controller>> CardController
- cards: List<Card>
+ completeTask(card: Card, task: Task):Void
+ changeLabel(label:Label):Void
creately

4. Design Pattern

- **Proxy Pattern**

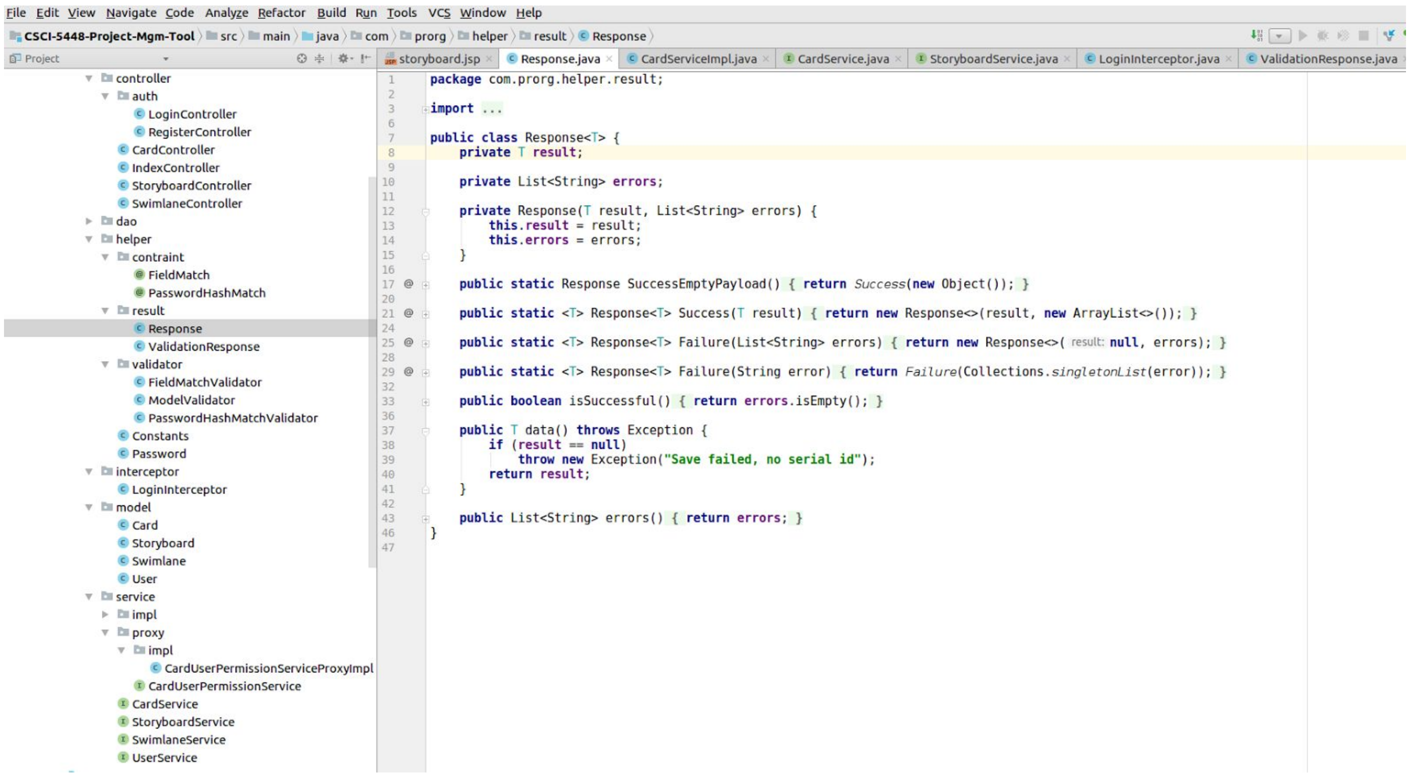


CardUserPermissionService
addUserToCard(Card, User) Response
deleteCardIfAccessibleByUser(Card, User) Response
CardService
createCard(String, String, Swimlane, User) Response<Integer>
getCardById(int) Response<Card>
CardUserPermissionServiceProxyImpl
cardService CardUserPermissionService
addUserToCard(Card, User) Response
deleteCardIfAccessibleByUser(Card, User) Response
getUsersHavingAccessToStoryboardTheCardIsIn(Card) List<User>
CardServiceImpl
cardDao CardDao
userDao UserDao
validator ModelValidator
createCard(String, String, Swimlane, User) Response<Integer>
getCardById(int) Response<Card>
addUserToCard(Card, User) Response
deleteCardIfAccessibleByUser(Card, User) Response

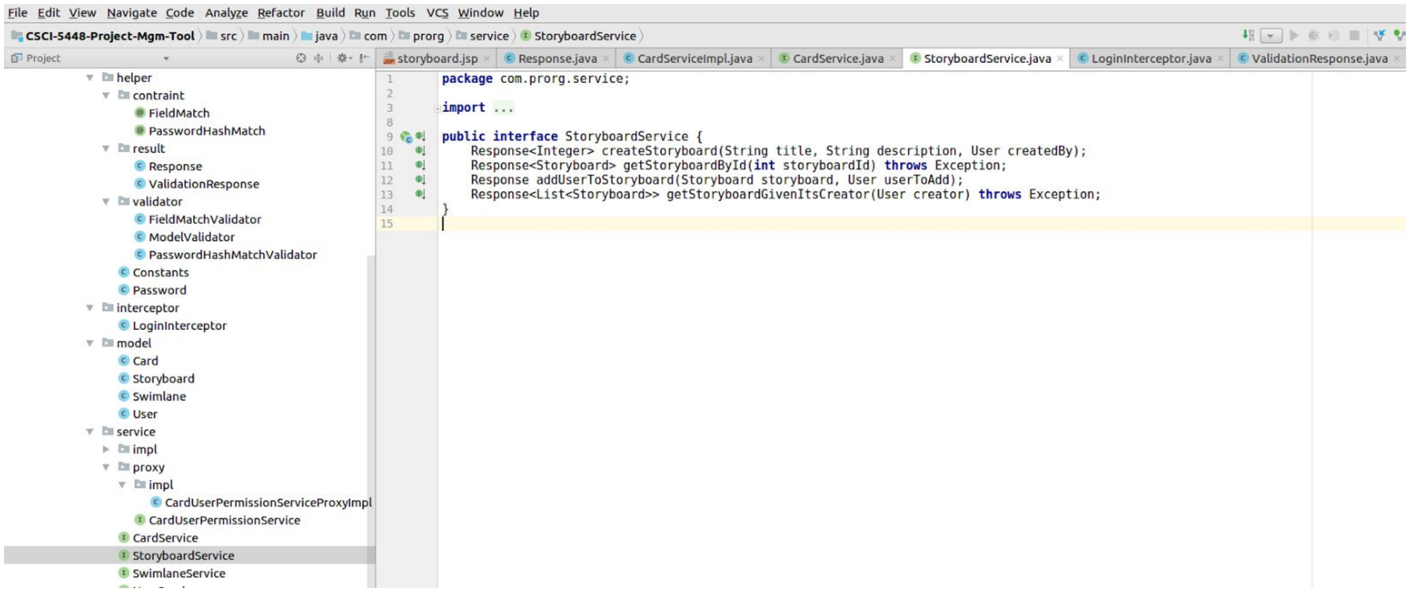
We are using proxy pattern to restrict access to a card. Error will be thrown, if unauthorized user tries to access a card.

- **Custom Pattern (for communication between the Controller, Service and DAO layer)**

While implementing the different layers of the project we observed that communication among these layers introduced a lot of duplicate "if-checks" with magic numbers, as we had to know if an operation, say, called by a controller on a service executed successfully or not. We thus extracted a class that was solely responsible for communication between controller class & repository class. This also eliminated magic numbers and nested "if-checks".


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
CSCI-5448-Project-Mgm-Tool › src › main › java › com › prorg › helper › result › Response
storyboard.jsp
Response.java
CardServiceImpl.java
CardService.java
StoryboardService.java
LoginInterceptor.java
ValidationResponse.java
Project

```
package com.prorg.helper.result;

import ...

public class Response<T> {
    private T result;

    private List<String> errors;

    private Response(T result, List<String> errors) {
        this.result = result;
        this.errors = errors;
    }

    public static Response SuccessEmptyPayload() { return Success(new Object()); }

    public static <T> Response<T> Success(T result) { return new Response<>(result, new ArrayList<>()); }

    public static <T> Response<T> Failure(List<String> errors) { return new Response<>( result: null, errors); }

    public static <T> Response<T> Failure(String error) { return Failure(Collections.singletonList(error)); }

    public boolean isSuccessful() { return errors.isEmpty(); }

    public T data() throws Exception {
        if (result == null)
            throw new Exception("Save failed, no serial id");
        return result;
    }

    public List<String> errors() { return errors; }
}
```


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
CSCI-5448-Project-Mgm-Tool › src › main › java › com › prorg › service › StoryboardService
storyboard.jsp
Response.java
CardServiceImpl.java
CardService.java
StoryboardService.java
LoginInterceptor.java
ValidationResponse.java
Project

```
package com.prorg.service;

import ...

public interface StoryboardService {
    Response<Integer> createStoryboard(String title, String description, User createdBy);
    Response<Storyboard> getStoryboardById(int storyboardId) throws Exception;
    Response addUserToStoryboard(Storyboard storyboard, User userToAdd);
    Response<List<Storyboard>> getStoryboardGivenItsCreator(User creator) throws Exception;
}
```

We have skipped the email functionality where we could have implemented the observer design pattern.

5. Learning

- Database Migrations

    We are using flyway tool for database migration. Flyway updates a database from one version to a next using migrations. We can write migrations either in SQL with database specific syntax or in Java for advanced database transformations. We do not need to do db base setup manually, everything is done automatically with flyway. We can easily deploy our project on various environments without worrying about db setup.

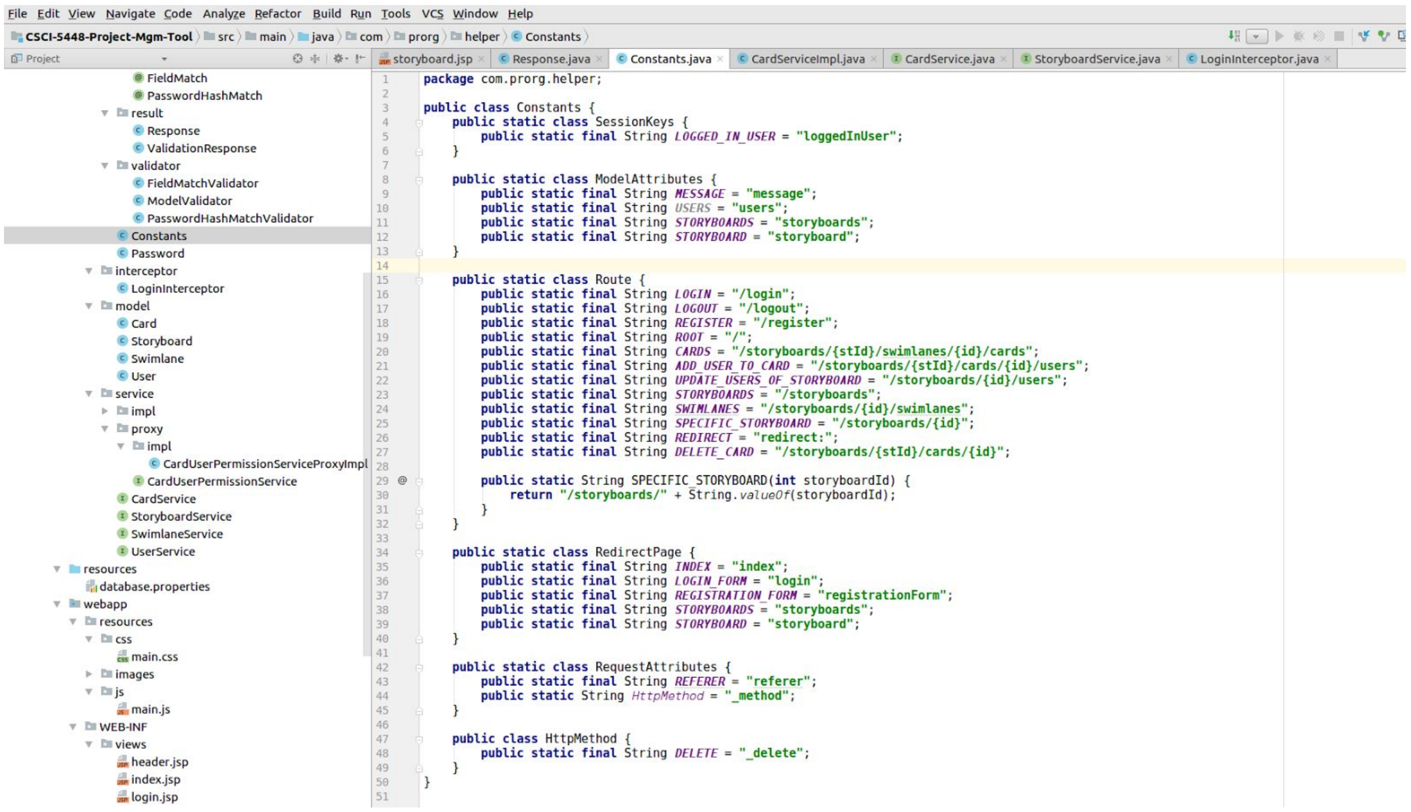
- Interceptors in Spring

  We have Implemented a LoginInterceptor that ensures that a user is logged for all project related tasks. It helped us to remove redundant code (checking user authenticity in all services before any request). Also, we implemented friendly forwarding in our login interceptor.

- Making controller Restful

  We learnt best practice for making controller endpoint restful.

- Namespacing String Constants



```java
package com.prorg.helper;

public class Constants {
    public static class SessionKeys {
        public static final String LOGGED_IN_USER = "loggedInUser";
    }

    public static class ModelAttributes {
        public static final String MESSAGE = "message";
        public static final String USERS = "users";
        public static final String STORYBOARDS = "storyboards";
        public static final String STORYBOARD = "storyboard";
    }

    public static class Route {
        public static final String LOGIN = "/login";
        public static final String LOGOUT = "/logout";
        public static final String REGISTER = "/register";
        public static final String ROOT = "/";
        public static final String CARDS = "/storyboards/{stId}/swimlanes/{id}/cards";
        public static final String ADD_USER_TO_CARD = "/storyboards/{stId}/cards/{id}/users";
        public static final String UPDATE_USERS_OF_STORYBOARD = "/storyboards/{id}/users";
        public static final String STORYBOARDS = "/storyboards";
        public static final String SWIMLANES = "/storyboards/{id}/swimlanes";
        public static final String SPECIFIC_STORYBOARD = "/storyboards/{id}";
        public static final String REDIRECT = "redirect:";
        public static final String DELETE_CARD = "/storyboards/{stId}/cards/{id}";

        public static String SPECIFIC_STORYBOARD(int storyboardId) {
            return "/storyboards/" + String.valueOf(storyboardId);
        }
    }

    public static class RedirectPage {
        public static final String INDEX = "index";
        public static final String LOGIN_FORM = "login";
        public static final String REGISTRATION_FORM = "registrationForm";
        public static final String STORYBOARDS = "storyboards";
        public static final String STORYBOARD = "storyboard";
    }

    public static class RequestAttributes {
        public static final String REFERER = "referer";
        public static String HttpMethod = "_method";
    }

    public class HttpMethod {
        public static final String DELETE = "_delete";
    }
}
```