Software Engineering Software Requirements Specification (SRS) Document

DyVert

05/1/2024

Final Version

By: Samuel George, Jayden Cathcart, Sergio Guerra

I HAVE ABIDED BY THE UNCG ACADEMIC POLICY ON THIS ASSIGNMENT.
Student's Signature: Sergio Guerra, Jayden Cathcart, Samuel George Date: 02/19/2024

Table of Contents

1. Introduction

1.1. Purpose (Sergio Guerra)

The goal of the DyVert web application is to connect users to new media content and to provide an interactive method of bookmarking interesting content by swiping right and discarding content by swiping left. The DyVert app will also allow content creators to publish their original content to the app as well to expose it to the whole DyVert userbase.

1.2. Document Conventions (Sergio Guerra

The purpose of this Software Requirements Document (SRD) is to provide a detailed description of the application and describe the requirements that the developers must meet as well as what the clients should expect. These client-side requirements will include a description of different types of users that the app will support. The developer-side requirements will describe the app with more emphasis placed on functional, data, performance, and other software requirements.

1.3. Definitions, Acronyms, and Abbreviations (Sergio Guerra)

Java	A programming language originally developed by James Gosling at Sun Microsystems. We will be using this language to build the Restaurant Manager.
HTML	Hypertext Markup Language. This is the code that will be used to structure and design the web application and its content.
SpringBoot	An open-source Java-based framework used to create a micro Service. This will be used to create and run our application.
MVC	Model-View-Controller. This is the architectural pattern that will be used to implement our system.
Spring Web	Will be used to build our web application by using Spring MVC. This is one of the dependencies of our system.
Thymeleaf	A modern server-side Java template engine for our web environment. This is one of the dependencies of our system.
NetBeans	An integrated development environment (IDE) for Java. This is where our system will be created.
TMDB API	Application Programming Interface. This stands for the "The Movie Database" which is where our app will retrieve movie recommendations based on genres.
JavaScript	A programming language used primarily for the web and alongside HTML and CSS.

1.4. Intended Audience (Jayden Cathcart)

This project is intended for all parties involved or interested in this app. Refer to the following list for relevant topics based on your role:

- Developers:
    - Section 1 – To understand purpose of application and this document.

    - Section 2 – To understand the features provided by the app and some dependencies and constraints.
    - Section 3 – To understand the use-cases of the app and the different users supported.
    - Section 4 – To understand the technical and functional requirements of the application. Developers will need to be aware of the different interfaces required for this app.
    - Section 5 – To understand the non-functional requirements.
    - Section 6 – To understand the design approach taken for this project with diagrams and database schema.

Developers will be interested in most of the sections of this document in order to get familiar with the application and its requirements from a client perspective as well as from a software developer perspective.

- Users:
    - Section 1 – To understand purpose of application and this document.
    - Section 3 – To understand what kinds of users are supported with this app and some use-case scenarios.

Users will be interested in these two sections as they outline the purpose of the application and the different use-cases depending on the type of user. This will be important for users as they want to know what some situations in the app may look like.

- Project Managers:
    - Section 3 – To understand the different use-cases that must be met and supported by the development team.
    - Section 4 – To understand the function requirements of the application.
    - Section 5 – To understand the non-functional requirements of the app which could act as a guide for goals of development teams.
    - Section 6 – To understand design approach that must be taken by teams.

Project managers will be interested in these sections because they outline the use-cases and requirements for the application from a functional and non-functional standpoint. This could help managers create goals for development teams.

1.5. Project Scope (Sam George)

The goal of the software is to provide a fun and interactive way of finding/curating new media content. The app will be used by users (those trying to find new content based on their interests), creators/curators (those who create cards to expose their original content to users), and admins (those who monitor curator posts). The goal of the app aligns with the goals of creator's businesses as more views on media content will provide more individuals who are interested in purchasing and consuming creator content.

The benefits of the project to business include:

- Increasing the exposure of new media content
- Increasing pleasure to users as they are able to find new media content that aligns with their interests in a fun and interactive way
- Facilitating the process of curators/creators pushing out content to wide audience.

1.6. Technology Challenges (Jayden Cathcart)

A few technological challenges we faced as a group would be utilizing and learning the new (to us) Spring Boot ThymeLeaf coding structure and its security features. Compared to the Node.JS MVC structure, Spring Boot takes a more complex and well-documented approach to this somewhat simple concept by adding a network of folders and references within its framework as opposed to Node.JS being very straightforward and having named folders for each leg of the MVC. Although the two are very similar, their differences make the other seem foreign to someone who isn't familiar with its structure. The other issue we faced was incorporating the authentication process with the login/ signup feature for our users. We eventually found a makeshift solution to our login problem, but the main issue resided with the hashing feature for the password which also correlates with the signup feature (If you can't properly hash and store a string then try to compare it to another hash that also wasn't hashed properly; it'll always return false).

1.7. References

Getting started. (n.d.). The Movie Database (TMDB). Retrieved May 1, 2024, from https://developer.themoviedb.org/docs/getting-started

2. General Description

2.1. Product Features (Sam George)

The DyVert application is designed to revolutionize the way users interact with entertainment content. Below are the key features and functions that define the essence of the application:

Content Preference Customization:

- Users can select their preferred location to search for content. This will include the local DyVert database and API third-party content.
- Flexible options for content consumption, allowing users to indicate their interest in various media types such as books, movies, and TV shows.

Profile Login:

- Users will be able to access their own content/bucket to ensure that the app stays relevant to their own interests.

Interactive Content Interaction:

- Intuitive swipe-based interaction for discovering and interacting with content.
- Users can swipe right to bookmark content they find interesting, and swipe left to discard content that does not align with their preferences.

Content Creation and Sharing:

- Empowers users to create and share posts about their own content, fostering engagement and interaction within the community.

Content Discovery and Connection:

- Will show user a new card from Local DyVert library after each swipe. Will never show the user the same card twice to ensure that the user stays engaged.

Platform for Content Creators:

- Provides a platform for content creators to publish and showcase their original content, reaching a targeted audience of interested users.
- Opportunities for content creators to gain exposure and build a fanbase within the DyVert community.

Overall Objective: The primary objective of the DyVert web application is to connect users with new and exciting media content while offering an interactive platform for bookmarking and sharing favorite discoveries. By facilitating meaningful connections between users and content creators, DyVert aims to enrich the entertainment experience and expand users' horizons within the world of media and entertainment.

2.2. User Class and Characteristics (Sam George)

The DyVert application caters to a diverse range of users, each with distinct roles, behaviors, and preferences. Below are the categorized user classes and their respective characteristics:

- 1. Standard User:
    - Characteristics: Standard users constitute most of the application's user base. They engage with the platform by swiping on items that align with their content and entertainment preferences.
    - Interaction: Standard users utilize the swipe feature to express interest in content, adding items to their "bucket" for later consumption or marking them as consumed.
- 2. Administrator:
    - Characteristics: Administrators play a crucial role in moderating the platform and ensuring a safe and appropriate environment for all users.
    - Responsibilities: Administrators have the authority to flag explicit content, delete inappropriate or violating content uploaded by creators, and manage user accounts, including the ability to delete accounts if necessary.
- 3. Content Creator:
    - Characteristics: Content creators are users with additional privileges, enabling them to contribute original content to the platform.
    - Special Features: Content creators can create content cards, showing their creations to the user community.
    - Role Relationship: The content creator class is an extension of the standard user class, with added functionalities tailored to facilitate content creation and sharing within the DyVert ecosystem.

- Shared User Actions:
    - Account Creation: All users, regardless of class, are required to create accounts to access the platform.
    - Sign-In: Users must sign in to their accounts to fully utilize the features and functionalities of the DyVert application.

- Overall Objective:
    - By categorizing users into distinct classes based on their roles and functionalities, DyVert aims to provide a tailored experience that meets the diverse needs and preferences of its user base. Whether engaging with content, moderating the platform, or contributing as content creators, each user class plays a vital role in shaping the dynamic and interactive nature of the DyVert community.

2.3. Operating Environment (Jayden Cathcart)

The DyVert application is engineered to operate within a web-based environment, providing accessibility to users across various systems and platforms. Key specifications of the operating environment include:

- Web-Based Platform:
    - DyVert is designed as a web application, accessible through standard web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.
- Compatibility with Multiple Operating Systems:
    - The application is compatible with a wide range of operating systems, including Windows, macOS, Linux, iOS, and Android.

Since this application is a web app, a device with a browser should be sufficient.

2.4. Constraints (Jayden Cathcart)

The development of the DyVert application is subject to certain constraints that may impact its design and implementation. These include:
User Interface Design:

- Visual representation of options for each user class requires careful consideration to ensure simplicity without sacrificing functionality.
- Balancing simplicity with the desired user experience poses a design challenge in creating an intuitive and engaging interface for users.

Limited Development Resources:

- Resource constraints may impact the pace and scope of development, necessitating prioritization of features and functionalities.

Compatibility Considerations:

- Ensuring compatibility with a wide range of web browsers and devices adds complexity to the development process.

2.5. Assumptions and Dependencies (Sergio Guerra

In the development of the DyVert application, certain assumptions have been made regarding the software product and its environment, as well as external dependencies that may affect the project:

- Internet Accessibility: The application assumes that users will have consistent access to the internet to utilize its features and functionalities.
- Web Technologies: The software is dependent on web technologies such as HTML and CSS to create a user-friendly interface and interactive features.
- Backend Frameworks: Dependency exists on backend frameworks and libraries to support server-side functionalities and database management.
- Content APIs: Integration with third-party content APIs will be necessary to access and fetch entertainment content for display within the DyVert platform. These APIs include Google Books API and TMBD API.
- The app will also be dependent on Spring Web and Thymeleaf in order to execute the MVC architecture and integrate HTML with Java.

Overall Objective: By acknowledging these assumptions and dependencies, DyVert aims to proactively address potential challenges and ensure a smooth development process. Collaboration with external providers and careful consideration of environmental factors will be essential in delivering a robust and reliable entertainment application.

3. Functional Requirements

3.1. Primary

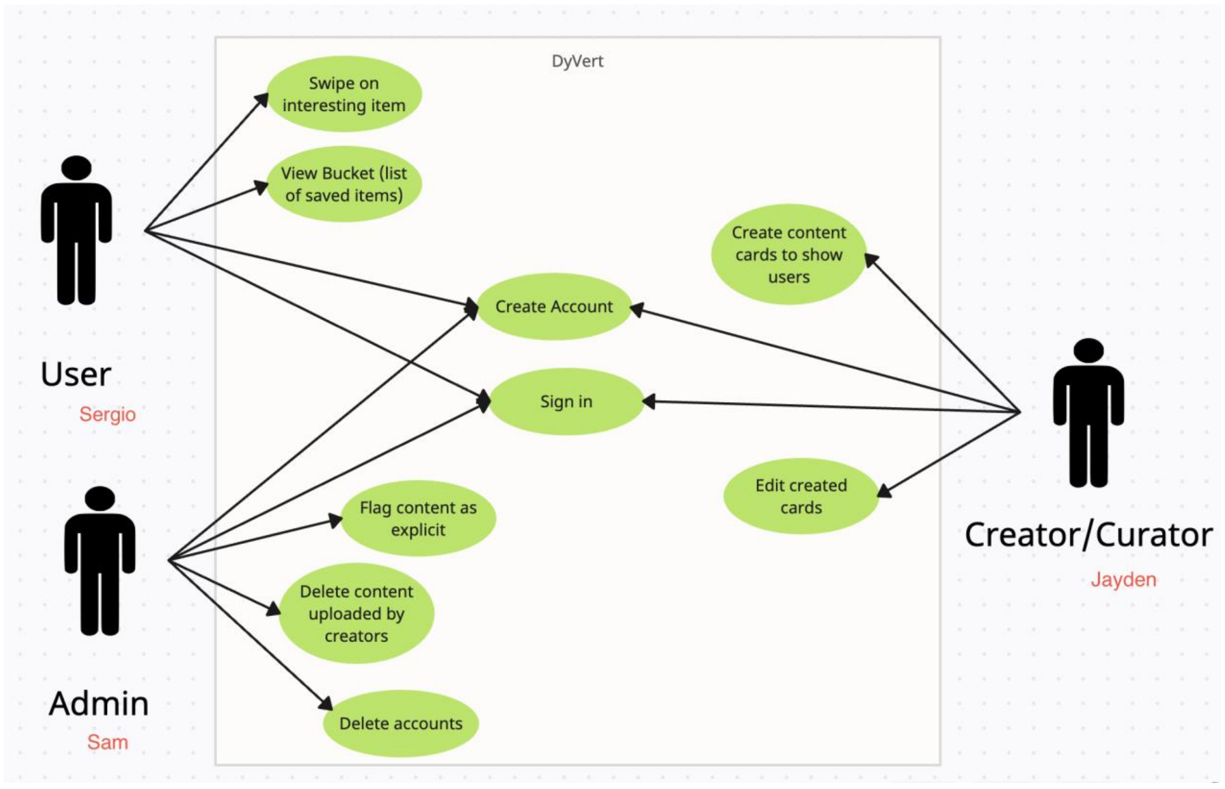
- FR0: The app will allow the user to filter the content based on movies, shows, and books.
- FR1: The app will allow the user to swipe left or right on cards displaying a summary of the given media to save it or discard it.
- FR2: The app will allow the user to view their saved content.
- FR3: The app will allow curators to create cards for their own, original media.
- FR4: The app will allow administrators to flag content and delete cards and accounts.

3.2. Secondary

- Accounts will be protected by login pages with passwords.
- Administrators will be assigned.

3.3. Use-Case Model

3.3.1. Use-Case Model Diagram


DyVert
Swipe on interesting item
View Bucket (list of saved items)
Create content cards to show users
Create Account
Sign in
Edit created cards
Flag content as explicit
Delete content uploaded by creators
Delete accounts
User
Sergio
Admin
Sam
Creator/Curator
Jayden

3.3.2. Use-Case Model Descriptions (Everyone developed their own)

3.3.2.1. Actor: Admin (Sam)

– **Flag Content:** Admins can flag content creator cards that are not necessarily appropriate for all audiences. This flag will appear for all users that come across the card and cover the card asking the user if they want to view it.

– **Delete Content Cards:** Admins can delete content cards that are not suitable for viewing.

– **Delete Accounts:** Admins can delete accounts that post inappropriate or offensive cards.

3.3.2.2. Actor: User (Sergio)

– **Swipe/DyVert:** The user can swipe left or right on a card that previews a piece of media. A swipe left will discard the item (uninterested) and a swipe right will save the item in the user's bucket (interested).

– **View Bucket**: The user can open their bucket with a menu option and visit the cards that they saved to take a second look at them.

3.3.2.3. Actor: Creator (Jayden)

- **Create Content Card**: The creator can choose to create a content card from a menu option. The creation process will involve choosing what kind of media it is and a synopsis, image, and genre(s).
- **Edit Content Card**: The creator can go back and edit content cards that they have created and will have access to view all cards.

3.3.3. Use-Case Model Scenarios

3.3.3.1. Actor: Admin (Sam)

- **Use-Case Name**: Flag Content
    - **Initial Assumption**: The user has access to the web app and has the role of admin.
    - **Normal**: The admin will be able to go into a dashboard of unreviewed cards and click a Flag button to flag any cards that contain explicit content.
    - **What Can Go Wrong:** n/a
    - **Other Activities**: Can unflag content.
    - **System State on Completion**: The database will flag content for swipers and ask them if they wish to see explicit content.
- **Use-Case Name**: Delete Content Cards
    - **Initial Assumption**: The user has access to the web app and has the role of admin.
    - **Normal**: The user can delete unreviewed cards from their dashboard if deemed inappropriate.
    - **What Can Go Wrong**: Can accidentally delete a card. There will be an undo button to retrieve the most recently deleted card.
    - **Other Activities**: Select multiple cards to delete.
    - **System State on Completion**: The card will be removed from the DyVert database.
- **Use-Case Name**: Delete Accounts
    - **Initial Assumption**: The user has access to the web app and has the role of admin.
    - **Normal**: The user can go to their dashboard and click on an account under an unreviewed card. There, they will be able to find a delete button to remove the account.
    - **What Can Go Wrong**: There could be accidental deletes, so the system will have 2 steps where admin must confirm deletion.
    - **Other Activities**: n/a
    - **System State on Completion**: Account will be removed, and creator will no longer be able to sign in.

3.3.3.2. Actor: User (Sergio)

- **Use-Case Name**: Swipe/DyVert
    - **Initial Assumption**: The user has access to the web app and is on the main page of the DyVert website.
    - **Normal**: The user will be presented with a card that they will be able to swipe left (uninterested) or right (interested).

- **What Can Go Wrong**: The user can accidentally discard an item when they meant to save it. There will be an undo button on the page that will take the user back to the previous card.
- **Other Activities**: Client can click the refresh button to get new cards generated.
- **System State on Completion**: A new card with new content will be shown after user swipes on a card.

- **Use-Case Name**: View Bucket
  - **Initial Assumption**: The user has access to the web app and has clicked on Bucket option in menu.
  - **Normal**: The user will see a list of the different cards that they have swiped right on. They will be able to view the cards again in order to do more research on items. They can also remove items from Bucket.
  - **What Can Go Wrong**: A user can accidentally delete an item from Bucket. There will be an undo button that brings back one item.
  - **Other Activities**: A user will also be able to re-order Bucket list.
  - **System State on Completion**: Bucket will save any changes made by user and be available the next time it is clicked.

3.3.3.3. Actor: Creator/Curator (Jayden)

- **Use-Case Name**: Create Content Card
  - **Initial Assumption**: The user has access to the web app and has a creator account.
  - **Normal**: The creator will see a Create + button on their menu and clicking that will allow the creator to create a new card and fill in their summary, genre, and image. They will then be able to publish it.
  - **What Can Go Wrong**: A user could publish a card before completion. They can go back and edit cards.
  - **Other Activities**: Can save a card to drafts for more editing in the future before publishing.
  - **System State on Completion**: The card is uploaded to DyVert database where it will be stored and queried at random for users to swipe based on the genre.
- **Use-Case Name**: Edit Content Card
  - **Initial Assumption**: The user has access to the web app and has a creator account.
  - **Normal**: The creator will have two options in their menu: Drafts and Published. The creator can choose either of these options to edit created cards.
  - **What Can Go Wrong**: The creator could accidentally delete a created card. There will be an undo button that can retrieve deleted cards.
  - **Other Activities**: The user can unpublish cards which will send them to the drafts.
  - **System State on Completion**: The DyVert database will store the new card in place of the old version.

4. Technical Requirements

4.1. Interface Requirements (Sam George)

4.1.1. User Interfaces

The user interface of the web app will consist of a home page that will display the title of our app, a button for the user to access their profile (or create a profile if need be), the body will broadcast the DyVert content pulled from the content APIs depending on user filters, and lastly, there will be a nav bar of some sort to display a navigation system with buttons: Home (for the user to reset/ refresh their DyVert content feed), Create (for the user to create content for DyVert), View Bucket (for the user to view their list of saved items), and Admin (for the user to change their preferences or appearance of the web app). There will also be other options for Admins once they access that page to view their Dashboard to review content cards and creator profiles.

4.1.2. Hardware Interfaces

This web application will operate on most any device that has access to the internet and the ability to interact with online web pages. The supported devices include PCs, laptops, smartphones, and tablets. The application will use the HTTPS protocol to communicate with the content APIs and with the user to display images and text.

4.1.3. Communications Interfaces

The web app must be always connected to the internet to be able to access the TMBD (The Movie Database) API for the content centered around motion pictures and TV shows.

4.1.4. Software Interfaces

For the web app, we will be incorporating Spring Boot, Thymeleaf, and HTML/CSS for the building of the frontend. Spring will communicate with our database for the backend of the app and interaction will be done with Java.

5. Non-Functional Requirements

5.1. Performance Requirements (Sergio Guerra)

- FR0(R): The user will be able to choose their filters instantly.
- FR1(R): Content cards will be pulled randomly from the card_data table and, after ensuring that the user has not seen the card, will display the card to the user. This avoids the need for a long list of cards.
- FR2(R): A user can locate their bucket in less than 10 seconds.
- FR3(R): The card creation will have 4 editable areas: name, image, synopsis, genre, and type. Card areas will have a limit on the max size such as a max image file size to support fast loading.
- FR4(R): Flagging and deleting cards should take less than 10 seconds.

5.2. Safety Requirements (Sam George)

- Admin will moderate newly made content coming from recently made accounts to ensure that the user behind the account isn't encouraging/ attempting any malpractice (or unethical behavior).
- The UI will restrict the user from entering any inappropriate phrases.
- The admin users will have the option to delete their cards from the database if they deem it inappropriate.

5.3. Security Requirements (Jayden Cathcart)

- The system will only be usable by authorized users who have signed up for our application
- Our application will also have password protection and password authentication with usernames.
- All users' private and personal data will be protected. Personal data will not be distributed.

5.4. Software Quality Attributes (Jayden Cathcart)

5.4.1. Availability

All users who can access the internet and have an account will have this app available.

5.4.2. Correctness

The software will be correct because it will behave as specified. This can be seen when our app communicated with the APIs and user created cards.

5.4.3. Maintainability

The app will be maintainable by using similar classes for all content cards and specializing them based on their media type. By using parent classes, changes for all cards will only need to be made once.

5.4.4. Reusability

The app's assets will be reusable since all the content retrieved from the APIs will be formatted into similar cards. The app will also be reusable since these APIs will be updated with new data and so will our app.

5.4.5. Portability

This app will be portable since it will be running on the web and accessible by any operating system with access to the internet and a browser. No other ports will need to be created.

5.5. Process Requirements (Sergio Guerra)

5.5.1. Development Process Used

The development process used will be a Scrum approach.

5.5.2. Time Constraints

The project is due April 30th which will require new features to be developed regularly. The team will meet once a week for discussion on what has been completed and what needs to be completed.

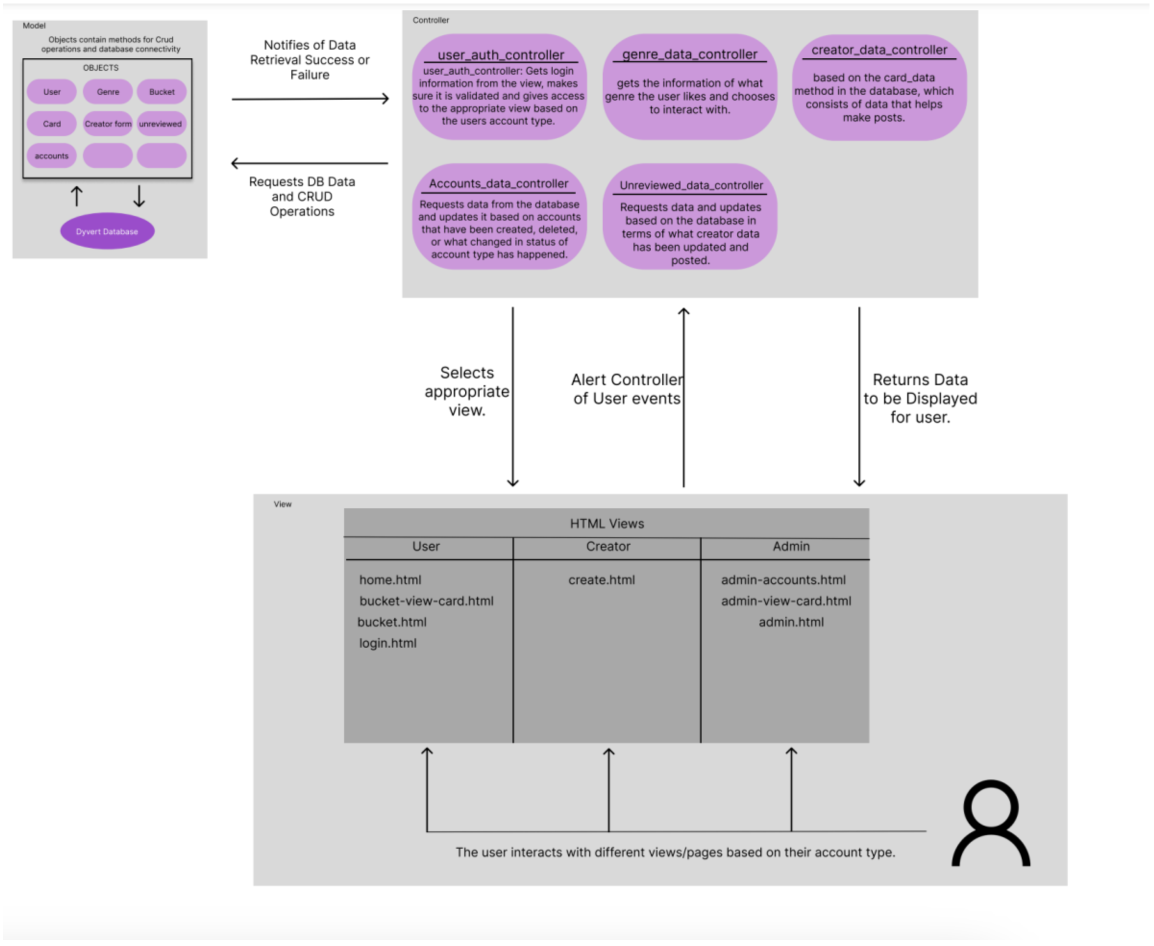
5.5.3. Cost and Delivery Date

The project is due Tuesday, April 30 with the functionality of in this document. No associated costs except time.

5.6. Other Requirements

6. Design Documents

6.1. Software Architecture (Sam George)


Model
Objects contain methods for Crud operations and database connectivity
OBJECTS
User
Genre
Bucket
Card
Creator form
unreviewed
accounts
Dyvert Database
Notifies of Data Retrieval Success or Failure
Requests DB Data and CRUD Operations
Controller
user_auth_controller
genre_data_controller
creator_data_controller
Accounts_data_controller
Unreviewed_data_controller
Selects appropriate view.
Alert Controller of User events
Returns Data to be Displayed for user.
View
The user interacts with different views/pages based on their account type.

user_auth_controller: Gets login information from the view, makes sure it is validated and gives access to the appropriate view based on the users account type.

gets the information of what genre the user likes and chooses to interact with.

based on the card_data method in the database, which consists of data that helps make posts.

Requests data from the database and updates it based on accounts that have been created, deleted, or what changed in status of account type has happened.

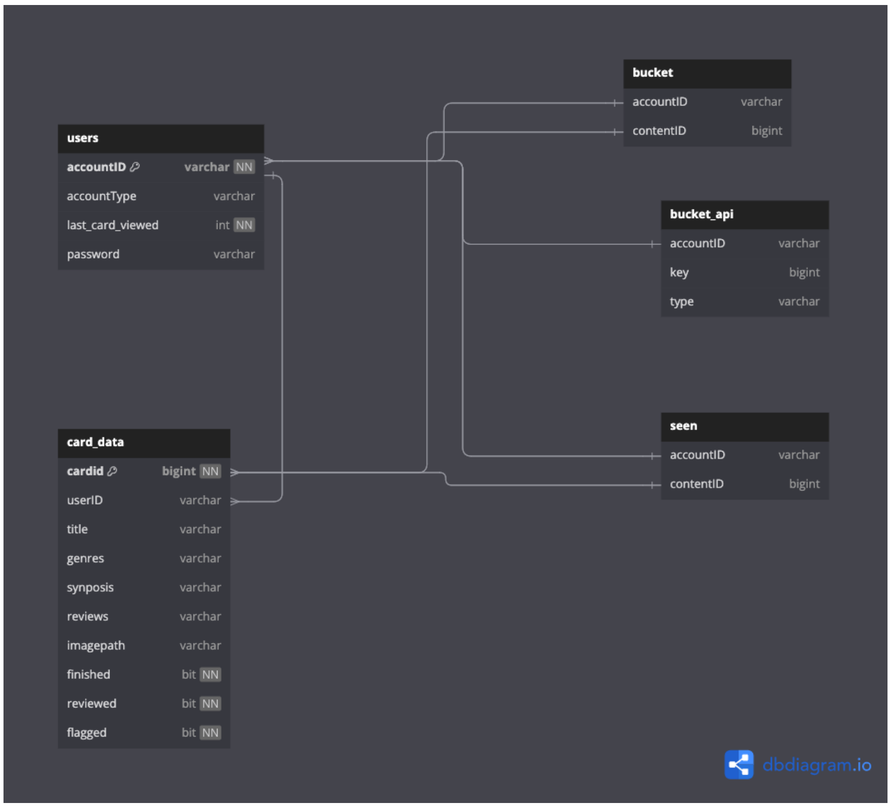
Requests data and updates based on the database in terms of what creator data has been updated and posted.

HTML Views

User	Creator	Admin
home.html	create.html	admin-accounts.html
bucket-view-card.html		admin-view-card.html
bucket.html		admin.html
login.html		

There are several assumptions that the MVC system architecture we have chosen makes. Firstly, it assumes that the database will be running and available during the duration of the application's operation. Without the ability to query the database and perform CRUD operations, the system cannot function. The controllers also assume that the data in the database will be stored and formatted in a standard way. One example of these standards is the use of integers 1 and 0 to mark a card as flagged or finished. Finally, our MVC architecture assumes that all possible user events are able to be managed by the controllers.

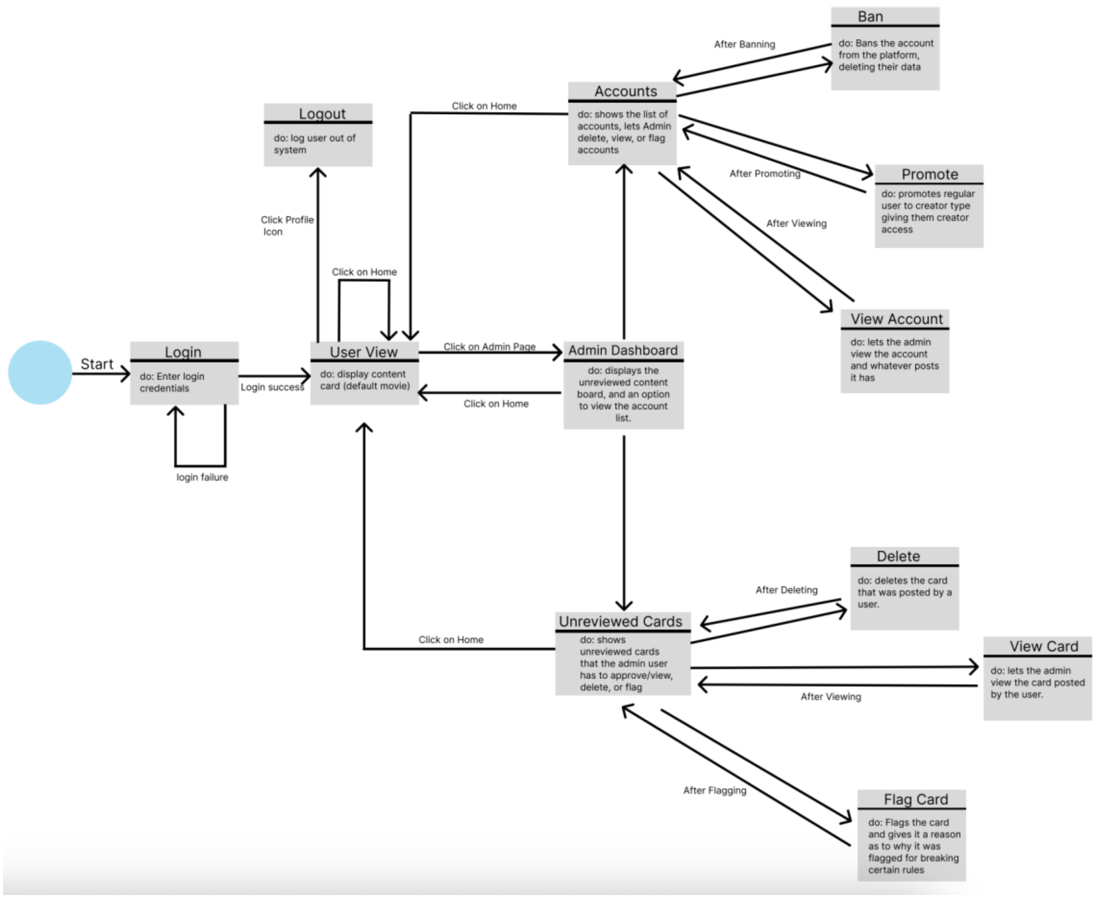
6.2. High-Level Database Schema (Jayden Cathcart)



This High-Level Database Schema is comprised of 5 entities that represent our database for our web application “DyVert”. We will implement this schema into MySQL for our application to utilize. The Card Data entity is the created card information: cardid increments after each card is made so each card is unique and never null, the userID is a foreign key and references the users entity set’s accountID to link the creator’s card to the user, and finished/ flagged/ reviewed will be indicators respectively for their roles (0 for flagged = meaning the card is safe for work; 1 being NSFW, etc). The Bucket entity is the user’s card bucket which stores their saved cards: accountID is a foreign key that references the user entity set’s accountID to link the user to their bucket, the contentID is a foreign key that references the card_data entity set’s cardid to ensure that the saved cards appear into the bucket. The Bucket_API entity is the user’s card bucket which stores their saved api cards and has the same functionality as its sister entity Bucket with two subtle differences: key is a bigint attribute that stores the media’s specific key generated from the api for storage and type is a varchar attribute that stores the type of media the saved card is. The Users entity is the user’s information: accountID is the user’s username, accountType is an indicator (admin for admins, creator for creators, and user for users), last_card_viewed was supposed to be used to refrain the user from seeing cards they’ve previously viewed but the implementation was done on the frontend so this attribute was no longer needed, password is a not null varchar attribute which stores the user’s password. The Seen entity is a sequence table used for logging the user’s last seen card which replaces the since extinct attribute “last_card_viewed”.

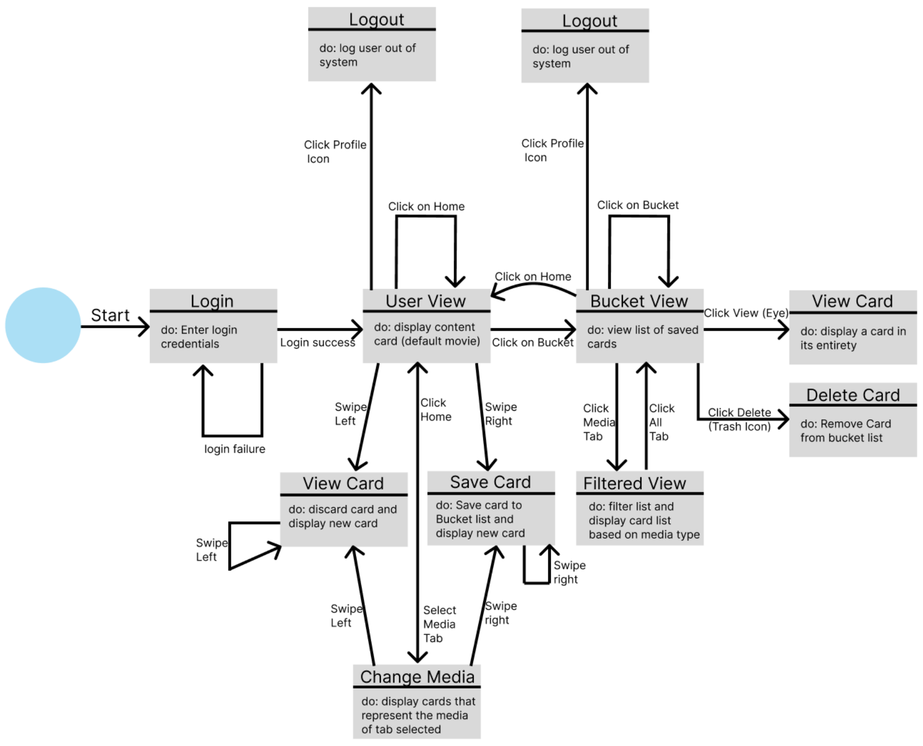
6.3. Software Design

6.3.1. State Machine Diagram: Admin (Sam George)



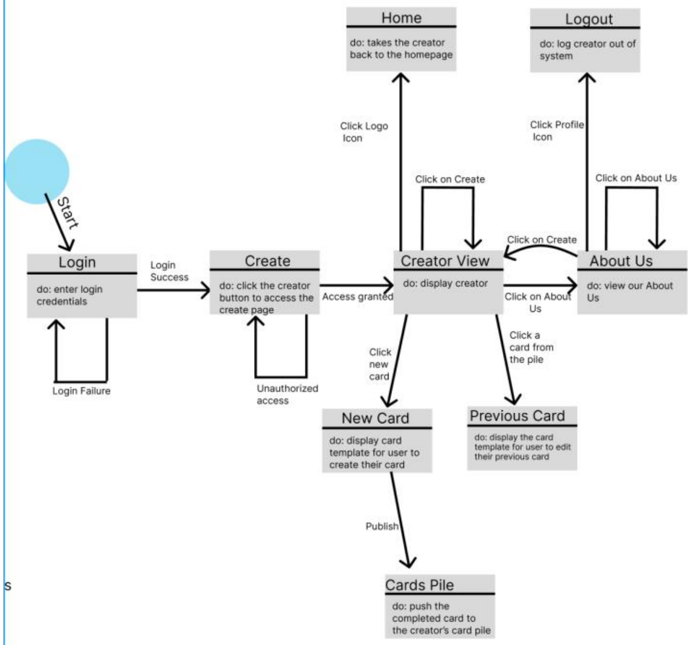
Start
Login
do: Enter login credentials
login failure
Login success
User View
do: display content card (default movie)
Click on Home
Click Profile Icon
Logout
do: log user out of system
Click on Home
Click on Admin Page
Click on Home
Admin Dashboard
do: displays the unreviewed content board, and an option to view the account list.
Accounts
do: shows the list of accounts, lets Admin delete, view, or flag accounts
After Banning
Ban
do: Bans the account from the platform, deleting their data
After Promoting
Promote
do: promotes regular user to creator type giving them creator access
After Viewing
View Account
do: lets the admin view the account and whatever posts it has
Unreviewed Cards
do: shows unreviewed cards that the admin user has to approve/view, delete, or flag
Click on Home
After Deleting
Delete
do: deletes the card that was posted by a user.
After Viewing
View Card
do: lets the admin view the card posted by the user.
After Flagging
Flag Card
do: Flags the card and gives it a reason as to why it was flagged for breaking certain rules

6.3.2. State Machine Diagram: User (Sergio Guerra)

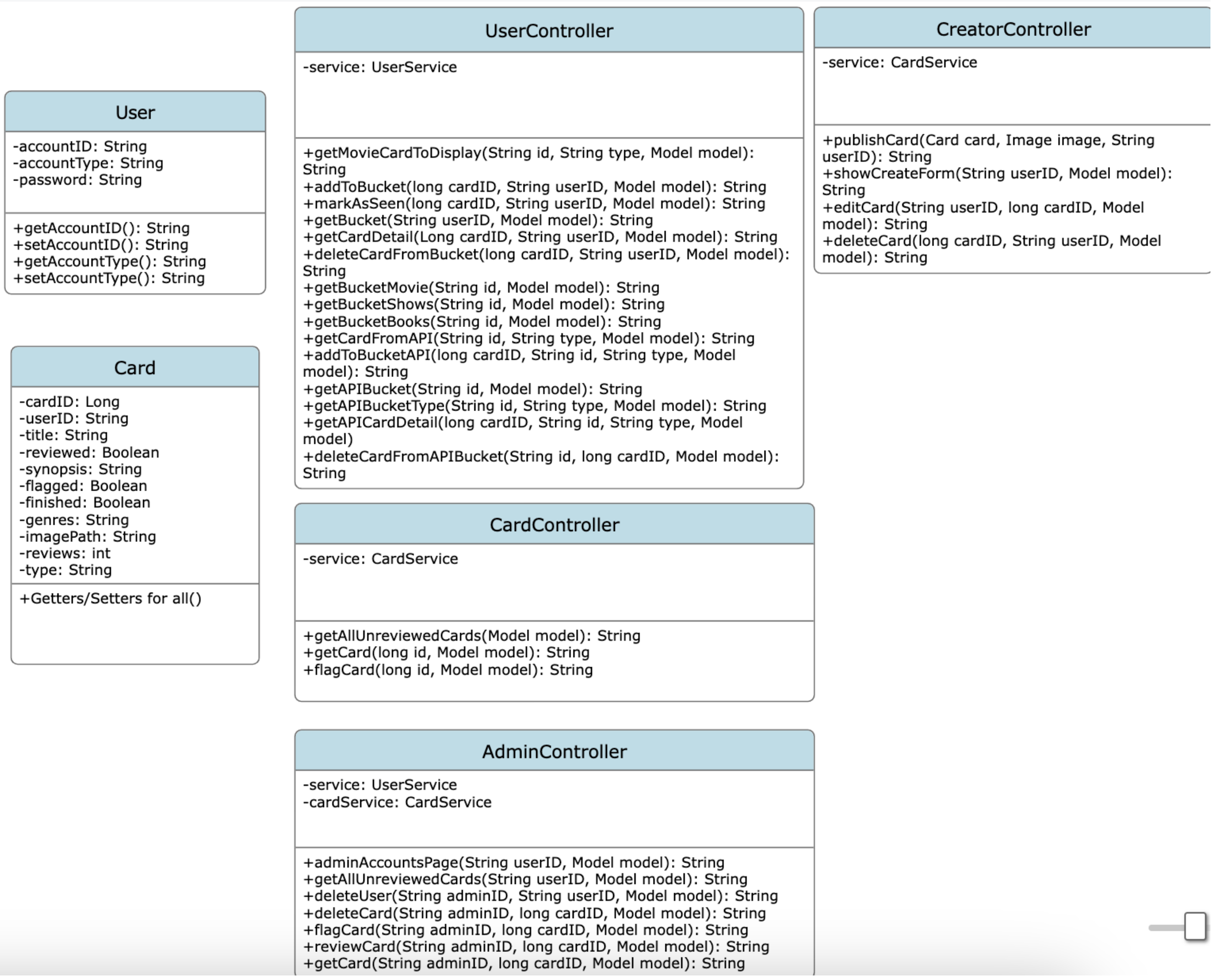


Start
Login
do: Enter login credentials
login failure
Login success
User View
do: display content card (default movie)
Click on Home
Click Profile Icon
Logout
do: log user out of system
Logout
do: log user out of system
Click Profile Icon
Click on Bucket
Click on Home
Click on Bucket
Bucket View
do: view list of saved cards
Click View (Eye)
View Card
do: display a card in its entirety
Click Delete (Trash Icon)
Delete Card
do: Remove Card from bucket list
Swipe Left
Click Home
Swipe Right
Click Media Tab
Click All Tab
View Card
do: discard card and display new card
Swipe Left
Save Card
do: Save card to Bucket list and display new card
Filtered View
do: filter list and display card list based on media type
Swipe right
Swipe Left
Select Media Tab
Swipe right
Change Media
do: display cards that represent the media of tab selected

6.3.3. State Machine Diagram: Creator/ Curator (Jayden Cathcart)


Home
do: takes the creator back to the homepage
Logout
do: log creator out of system
Click Logo Icon
Click Profile Icon
Click on Create
Click on About Us
Start
Click on Create
Login
do: enter login credentials
Login Success
Create
do: click the creator button to access the create page
Access granted
Creator View
do: display creator
Click on About Us
About Us
do: view our About Us
Login Failure
Unauthorized access
Click new card
Click a card from the pile
New Card
do: display card template for user to create their card
Previous Card
do: display the card template for user to edit their previous card
Publish
Cards Pile
do: push the completed card to the creator's card pile
s

6.4. UML Class Diagram (Sergio Guerra)



```
User
-accountID: String
-accountType: String
-password: String
+getAccountID(): String
+setAccountID(): String
+getAccountType(): String
+setAccountType(): String
```

```
Card
-cardID: Long
-userID: String
-title: String
-reviewed: Boolean
-synopsis: String
-flagged: Boolean
-finished: Boolean
-genres: String
-imagePath: String
-reviews: int
-type: String
+Getters/Setters for all()
```

```
UserController
-service: UserService
+getMovieCardToDisplay(String id, String type, Model model):
String
+addToBucket(long cardID, String userID, Model model): String
+markAsSeen(long cardID, String userID, Model model): String
+getBucket(String userID, Model model): String
+getCardDetail(Long cardID, String userID, Model model): String
+deleteCardFromBucket(long cardID, String userID, Model model):
String
+getBucketMovie(String id, Model model): String
+getBucketShows(String id, Model model): String
+getBucketBooks(String id, Model model): String
+getCardFromAPI(String id, String type, Model model): String
+addToBucketAPI(long cardID, String id, String type, Model
model): String
+getAPIBucket(String id, Model model): String
+getAPIBucketType(String id, String type, Model model): String
+getAPICardDetail(long cardID, String id, String type, Model
model)
+deleteCardFromAPIBucket(String id, long cardID, Model model):
String
```

```
CreatorController
-service: CardService
+publishCard(Card card, Image image, String
userID): String
+showCreateForm(String userID, Model model):
String
+editCard(String userID, long cardID, Model
model): String
+deleteCard(long cardID, String userID, Model
model): String
```

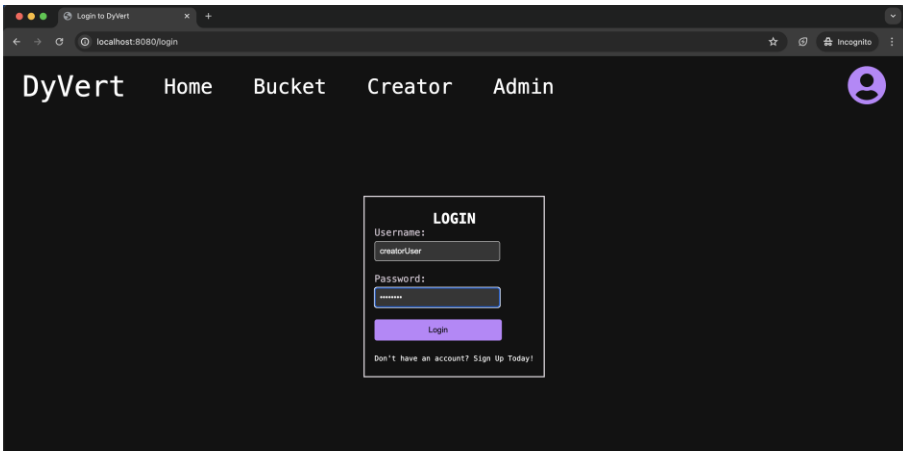
```
CardController
-service: CardService
+getAllUnreviewedCards(Model model): String
+getCard(long id, Model model): String
+flagCard(long id, Model model): String
```

```
AdminController
-service: UserService
-cardService: CardService
+adminAccountsPage(String userID, Model model): String
+getAllUnreviewedCards(String userID, Model model): String
+deleteUser(String adminID, String userID, Model model): String
+deleteCard(String adminID, long cardID, Model model): String
+flagCard(String adminID, long cardID, Model model): String
+reviewCard(String adminID, long cardID, Model model): String
+getCard(String adminID, long cardID, Model model): String
```

7. Scenario

7.1. Brief Written Scenario with Screenshots (Sergio Guerra)

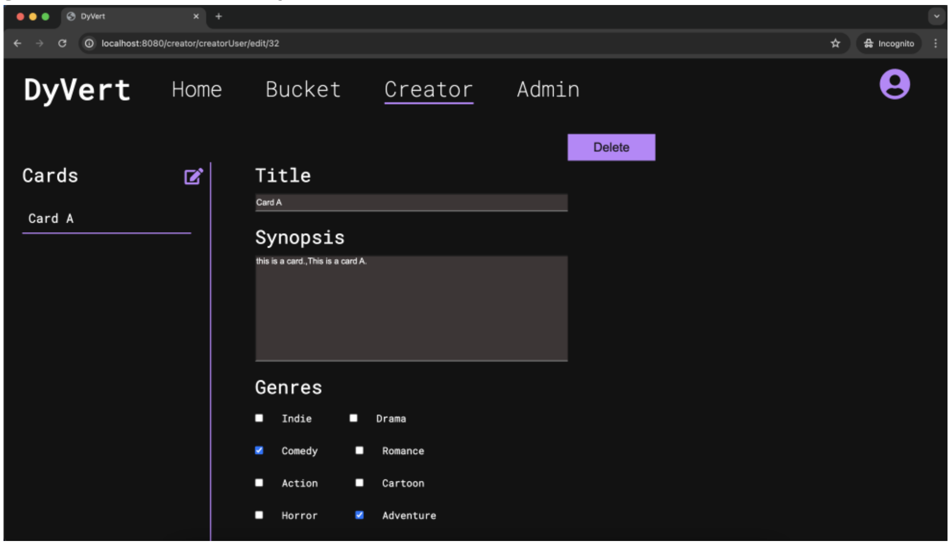
Creator: Create and Delete Card Use Case

- Creator logs in:
  - ○ 

localhost:8080/login

Username: creatorUser

Password: ••••••••

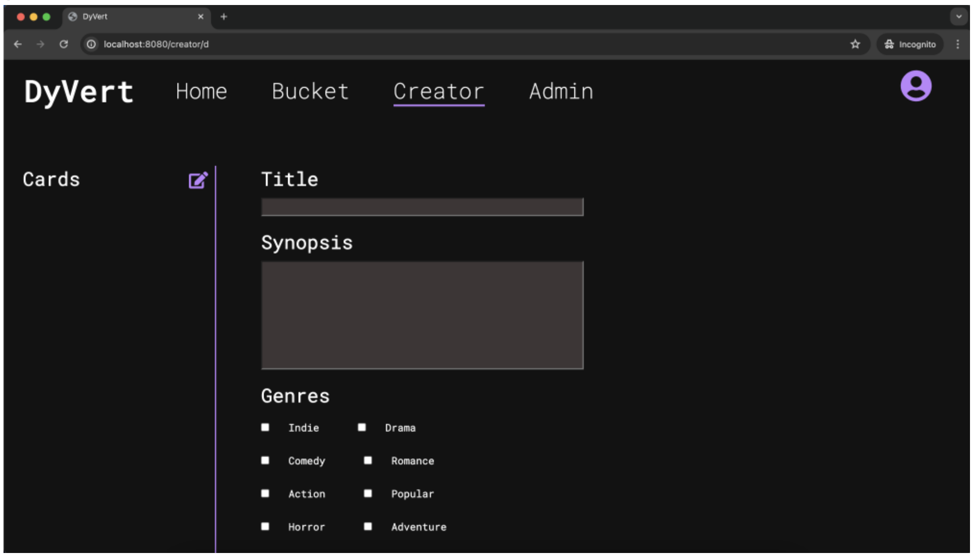
- Creating a card and the ability to delete:
  - ○ 

localhost:8080/creator/creatorUser/edit/32

Title: Card A

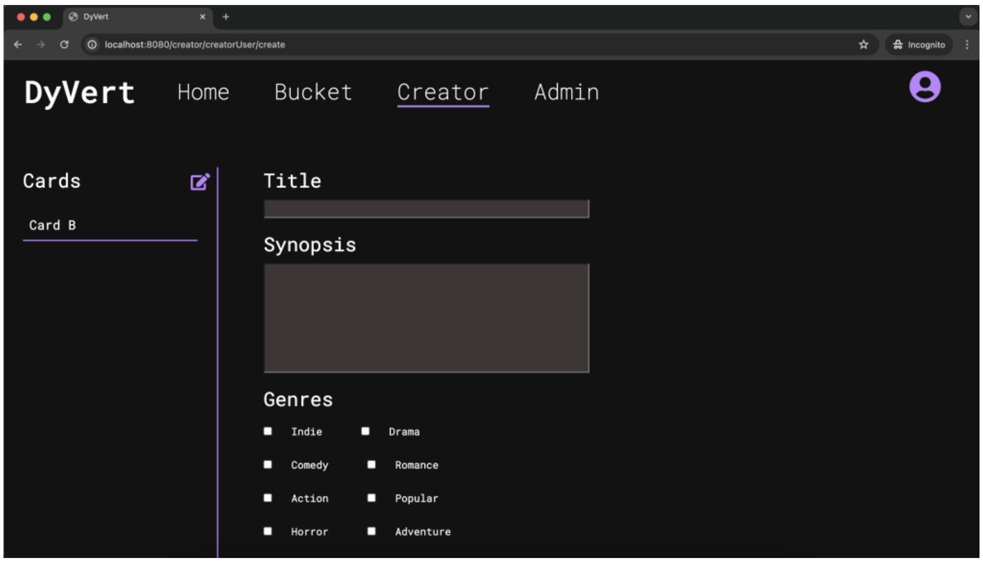
Synopsis: this is a card.,This is a card A.

Genres: Indie ☐, Drama ☐, Comedy ☑, Romance ☐, Action ☐, Cartoon ☐, Horror ☐, Adventure ☑

- Clicking delete removes from list:
  - ○ 

localhost:8080/creator/d

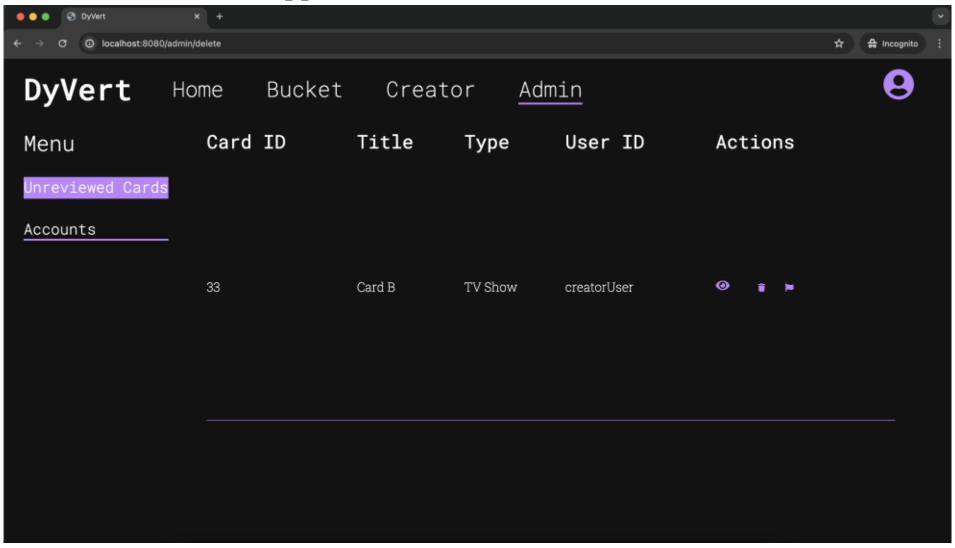
- Creating a second card and logging out:



localhost:8080/creator/creatorUser/create

○

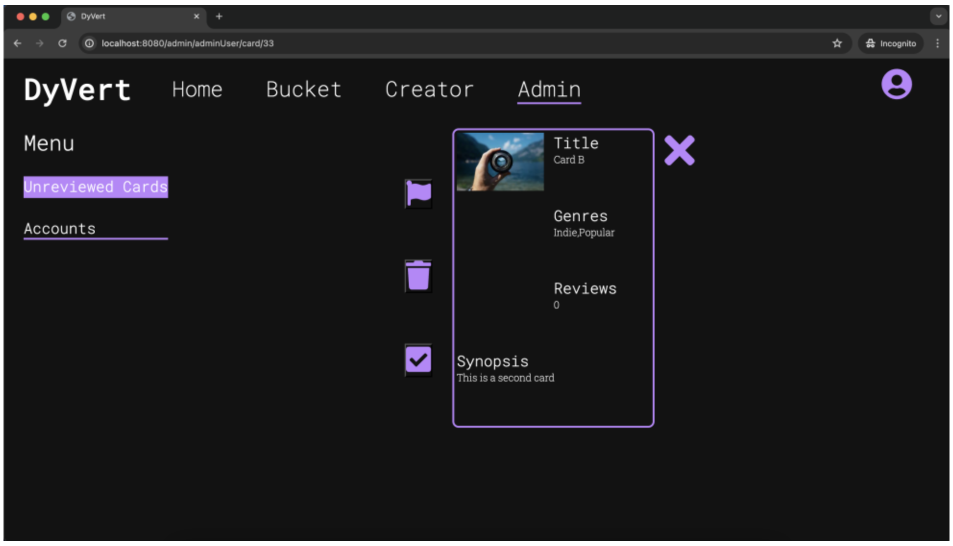
- Admin User logs in and views unreviewed cards. Locates Card B created by creatorUser and views it. Click the check button to approve the card and remove from Unreviewed Cards list.



localhost:8080/admin/delete

Card ID	Title	Type	User ID	Actions
33	Card B	TV Show	creatorUser	

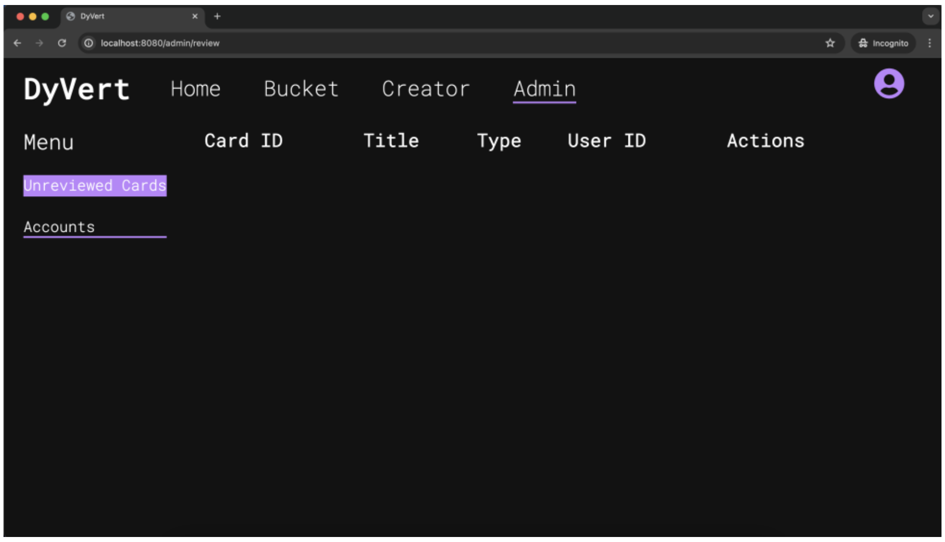
○



localhost:8080/admin/adminUser/card/33

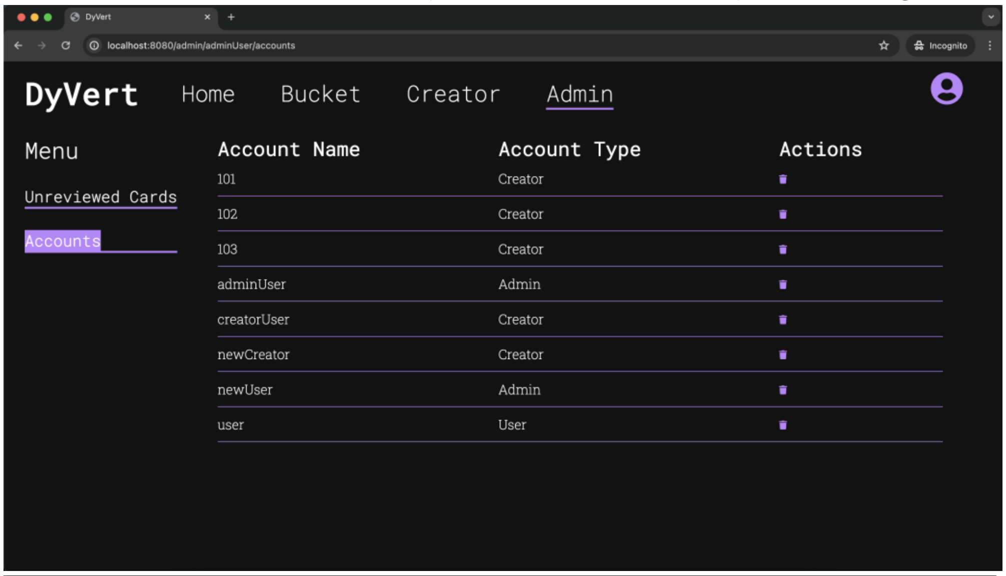
Title
Card B
Genres
Indie,Popular
Reviews
0
Synopsis
This is a second card

○

- ○ 

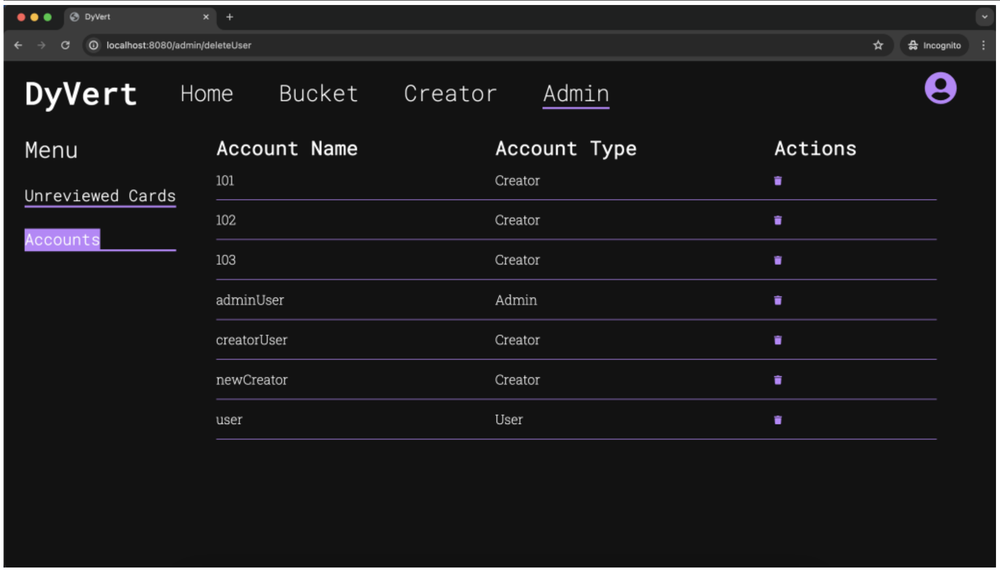
localhost:8080/admin/review

Card ID	Title	Type	User ID	Actions

- View account list and delete a newUser (second from the bottom) and then logs out:
  - ○ 

localhost:8080/admin/adminUser/accounts

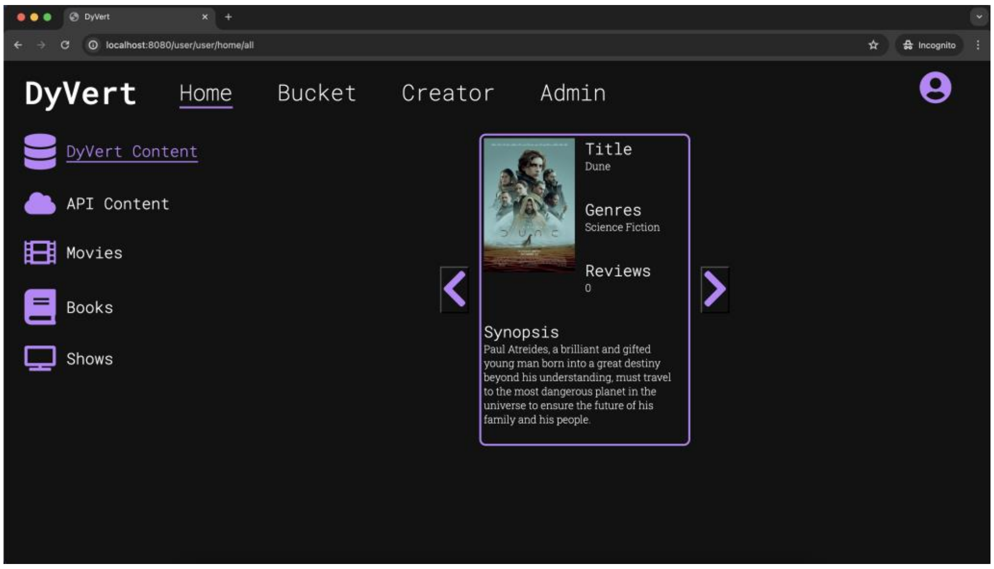
Account Name	Account Type	Actions
101	Creator	
102	Creator	
103	Creator	
adminUser	Admin	
creatorUser	Creator	
newCreator	Creator	
newUser	Admin	
user	User	

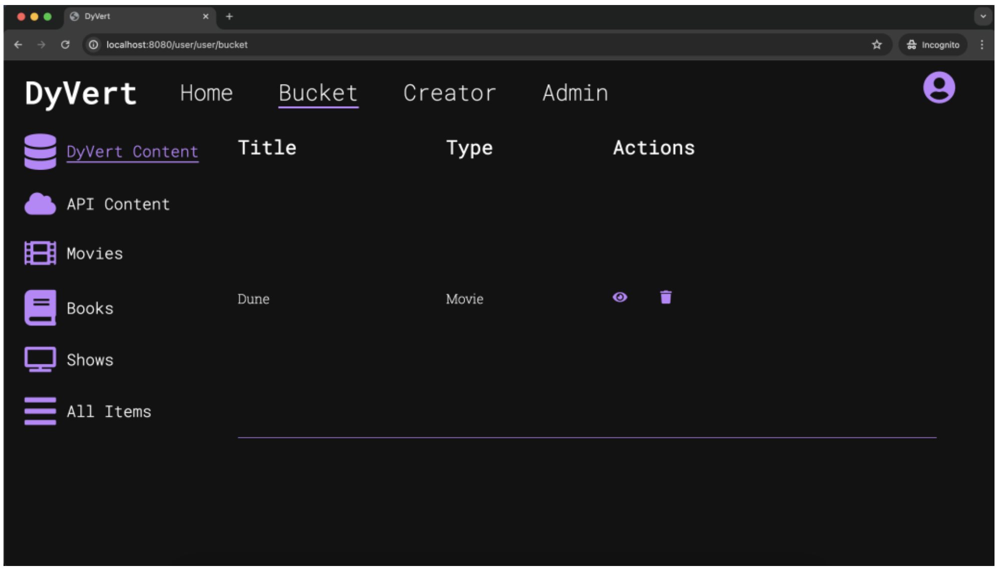
  - ○ 

localhost:8080/admin/deleteUser

Account Name	Account Type	Actions
101	Creator	
102	Creator	
103	Creator	
adminUser	Admin	
creatorUser	Creator	
newCreator	Creator	
user	User	

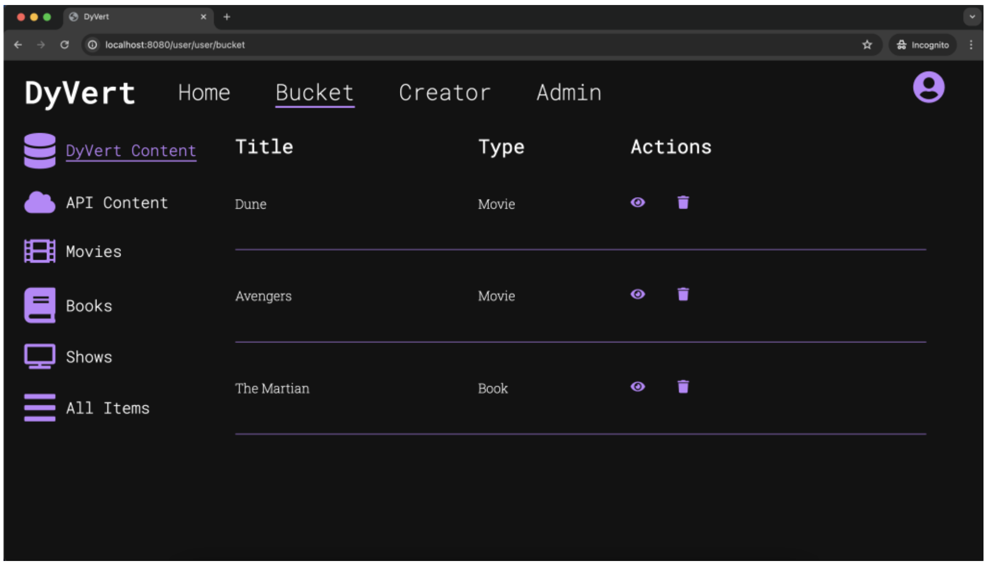
- User logs in and swipes left and right on cards. Swiping right adds to bucket:

○ 
DyVert
localhost:8080/user/user/home/all
Incognito
DyVert
Home
Bucket
Creator
Admin
DyVert Content
API Content
Movies
Books
Shows
Title
Dune
Genres
Science Fiction
Reviews
0
Synopsis
Paul Atreides, a brilliant and gifted young man born into a great destiny beyond his understanding, must travel to the most dangerous planet in the universe to ensure the future of his family and his people.

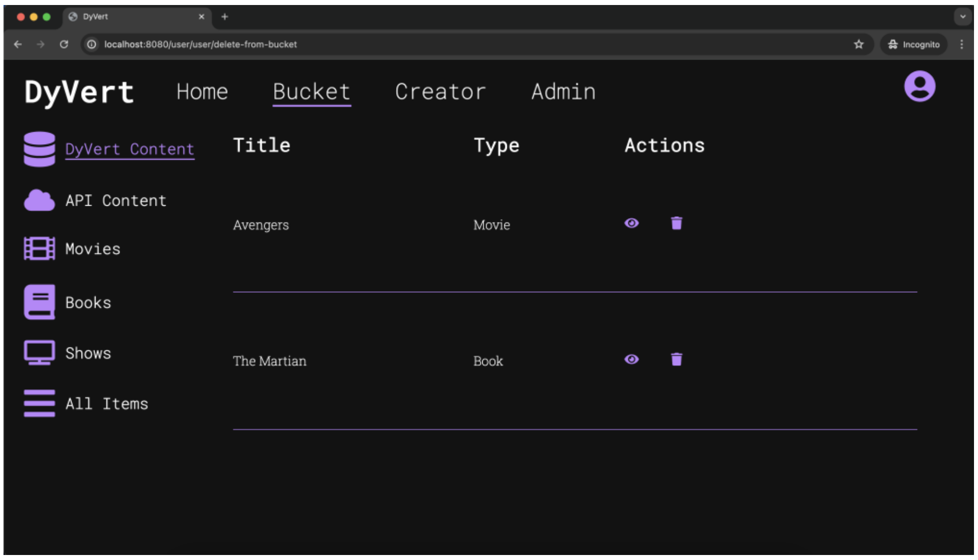
○ 

Title	Type	Actions
Dune	Movie	

- User can delete from bucket as well:

○ 

Title	Type	Actions
Dune	Movie	
Avengers	Movie	
The Martian	Book	



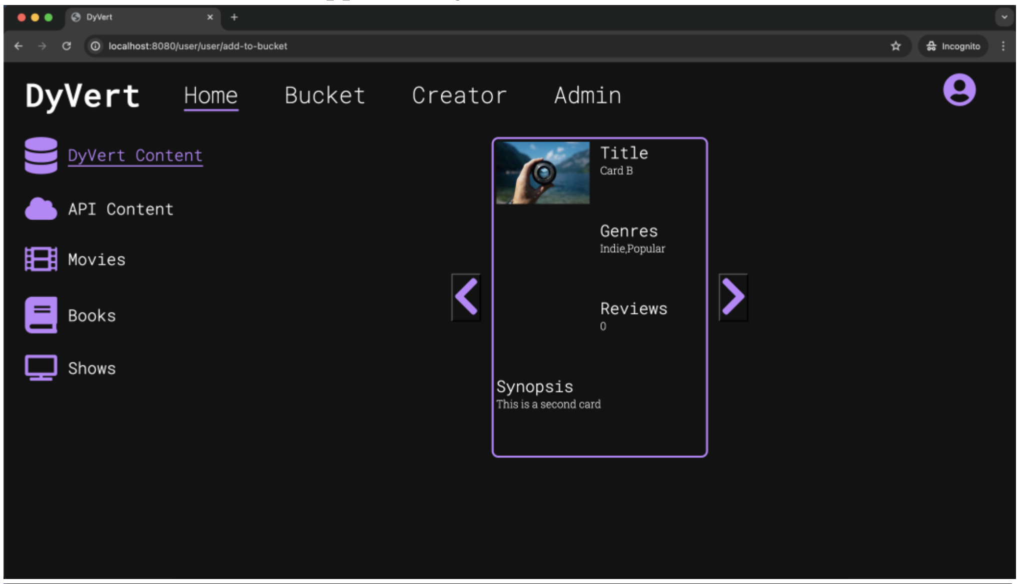
localhost:8080/user/user/delete-from-bucket

DyVert Home Bucket Creator Admin

DyVert Content
API Content
Movies
Books
Shows
All Items

Title	Type	Actions
Avengers	Movie	
The Martian	Book	

- User can see the reviewed card approved by the adminUser and can save it:



localhost:8080/user/user/add-to-bucket

DyVert Home Bucket Creator Admin

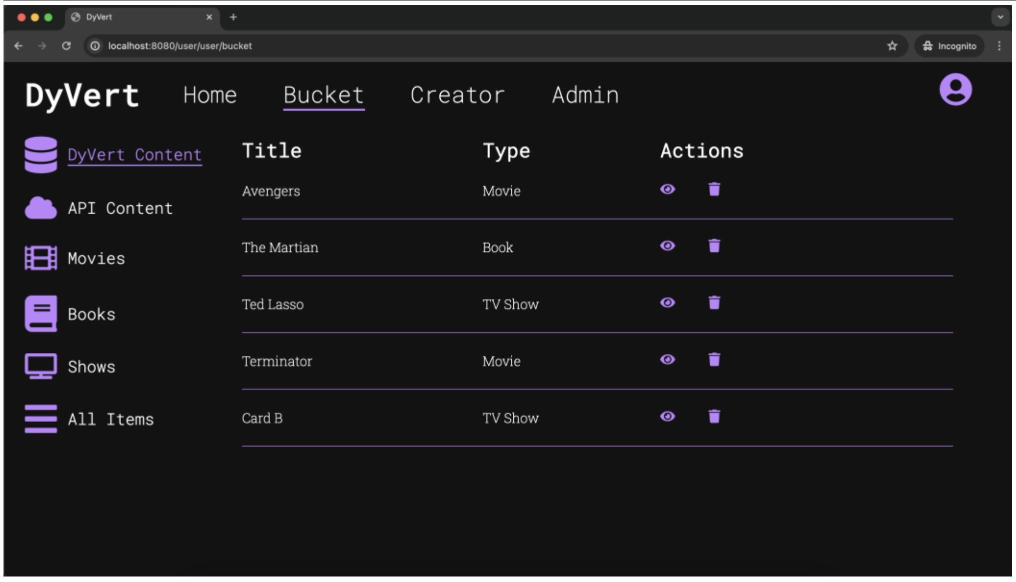
DyVert Content
API Content
Movies
Books
Shows

Title
Card B

Genres
Indie,Popular

Reviews
0

Synopsis
This is a second card



localhost:8080/user/user/bucket

DyVert Home Bucket Creator Admin

DyVert Content
API Content
Movies
Books
Shows
All Items

Title	Type	Actions
Avengers	Movie	
The Martian	Book	
Ted Lasso	TV Show	
Terminator	Movie	
Card B	TV Show	

Creator: Create and Delete Card Use Case

Creator logs in and creates a card, "Card A", then decides to delete "Card A".
Creator also creates another card, "Card B".
Creator logs out.
Admin: Review and Manage Content Use Case:

Admin logs in and reviews the cards submitted.
Admin approves "Card B".
Admin identifies an inactive or problematic account and deletes it.
Admin logs out.
User: Interact with Cards Use Case:

User logs in and begins swiping through a bucket of cards.
User accidentally adds an unwanted card to their bucket and removes it.
User continues to swipe until they find "Card B" and interact with it.
User logs out.