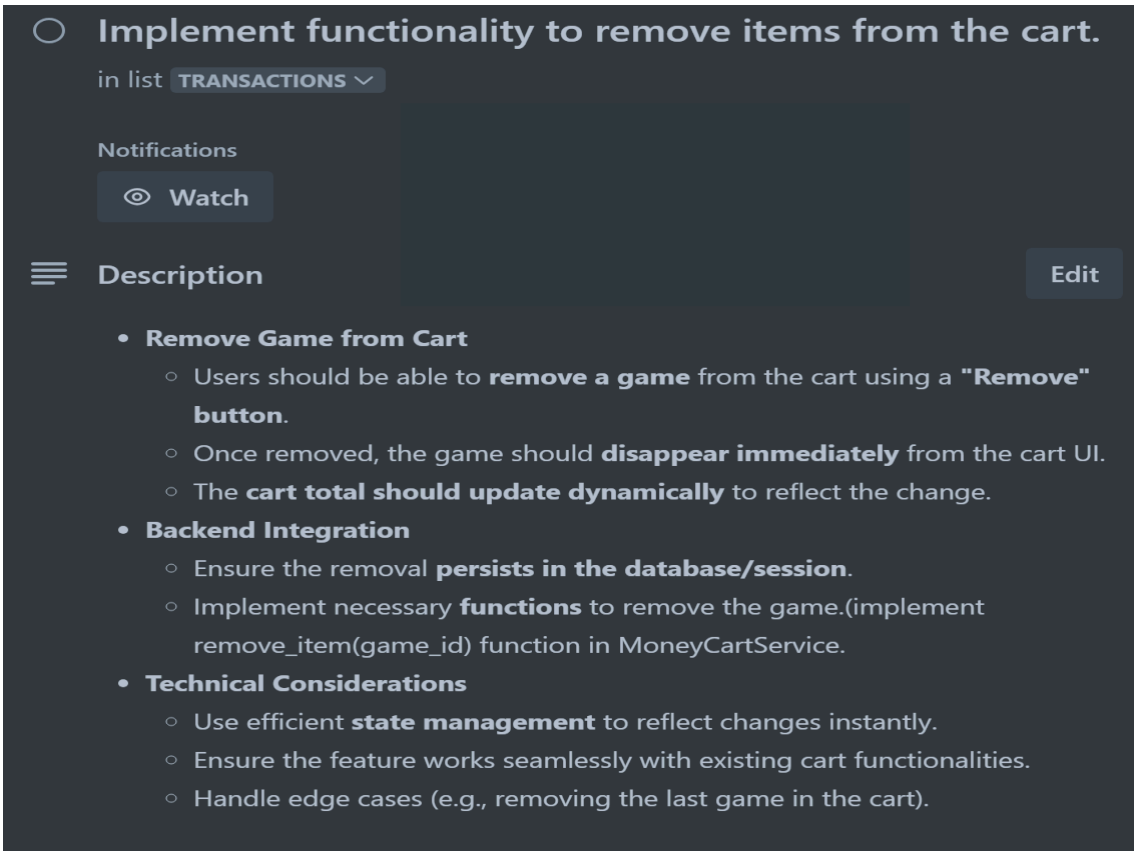
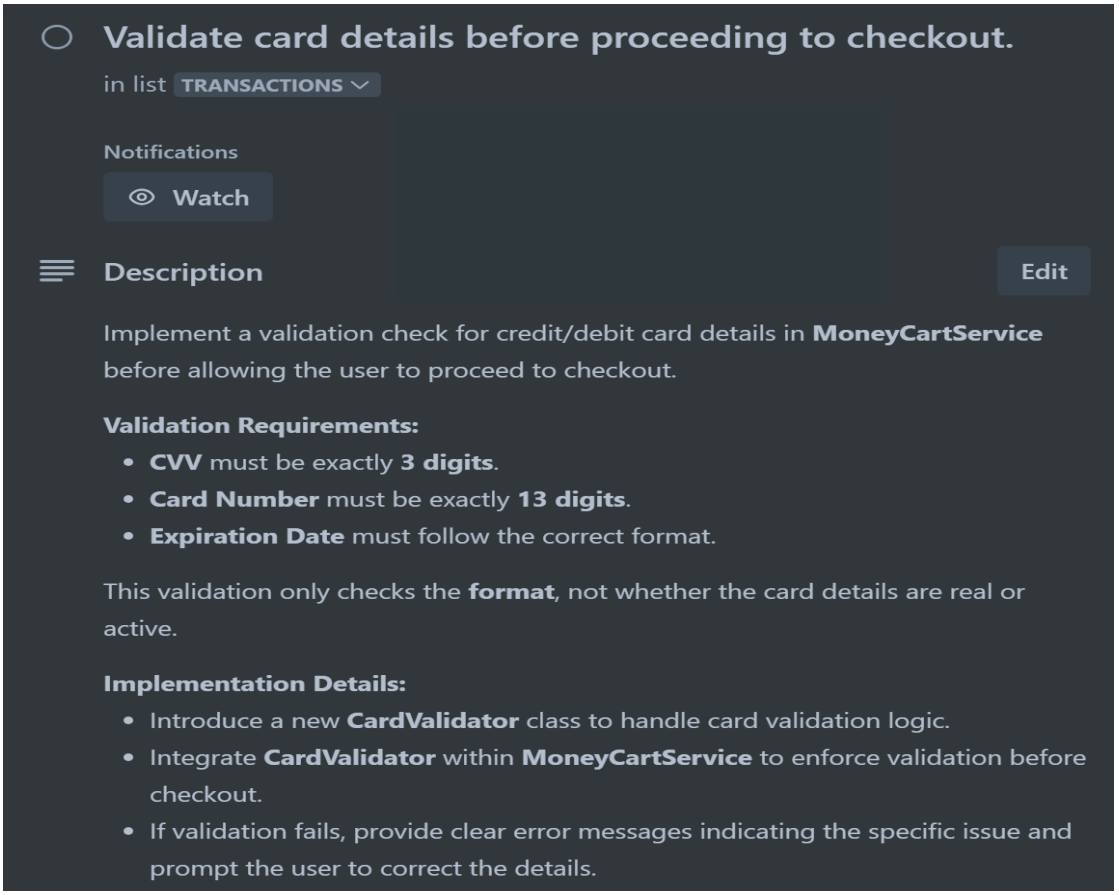
1.

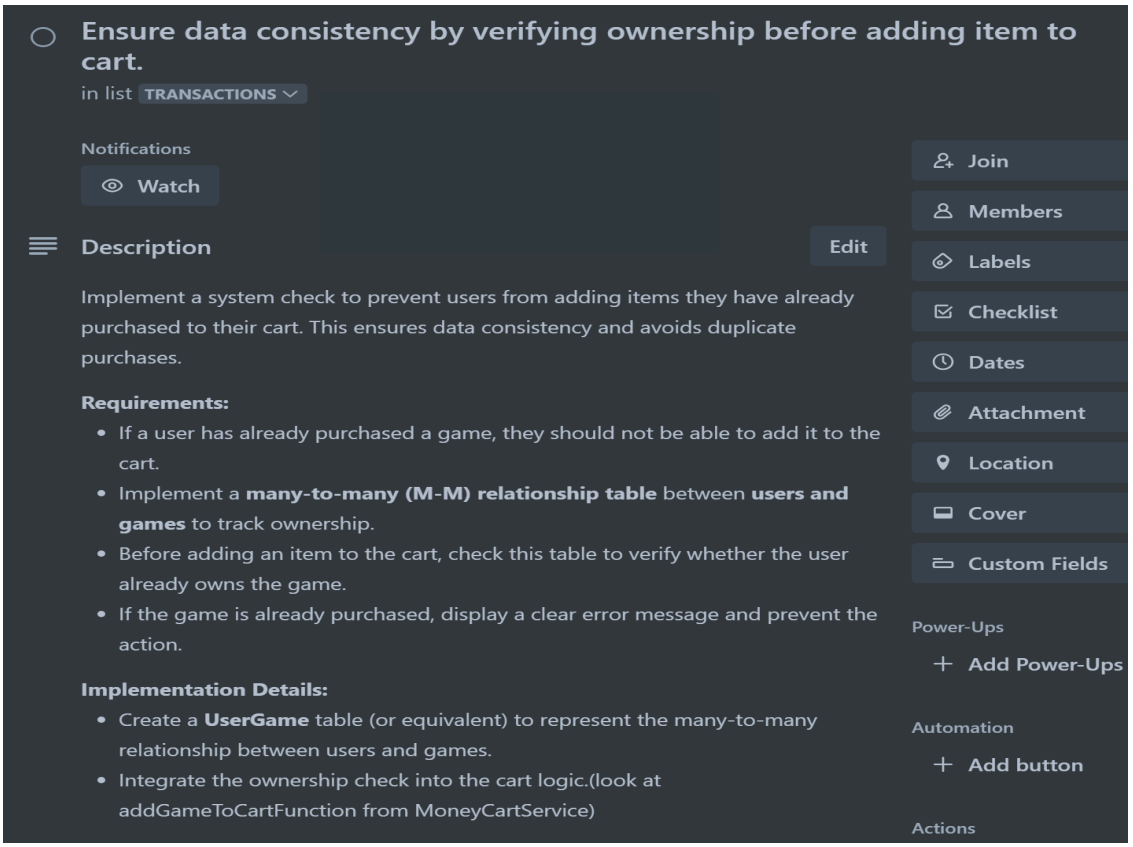

Implement functionality to remove items from the cart.
in list TRANSACTIONS
Notifications
Watch
Description
Edit
Remove Game from Cart
Users should be able to remove a game from the cart using a "Remove" button.
Once removed, the game should disappear immediately from the cart UI.
The cart total should update dynamically to reflect the change.
Backend Integration
Ensure the removal persists in the database/session.
Implement necessary functions to remove the game.(implement remove_item(game_id) function in MoneyCartService.
Technical Considerations
Use efficient state management to reflect changes instantly.
Ensure the feature works seamlessly with existing cart functionalities.
Handle edge cases (e.g., removing the last game in the cart).

2.

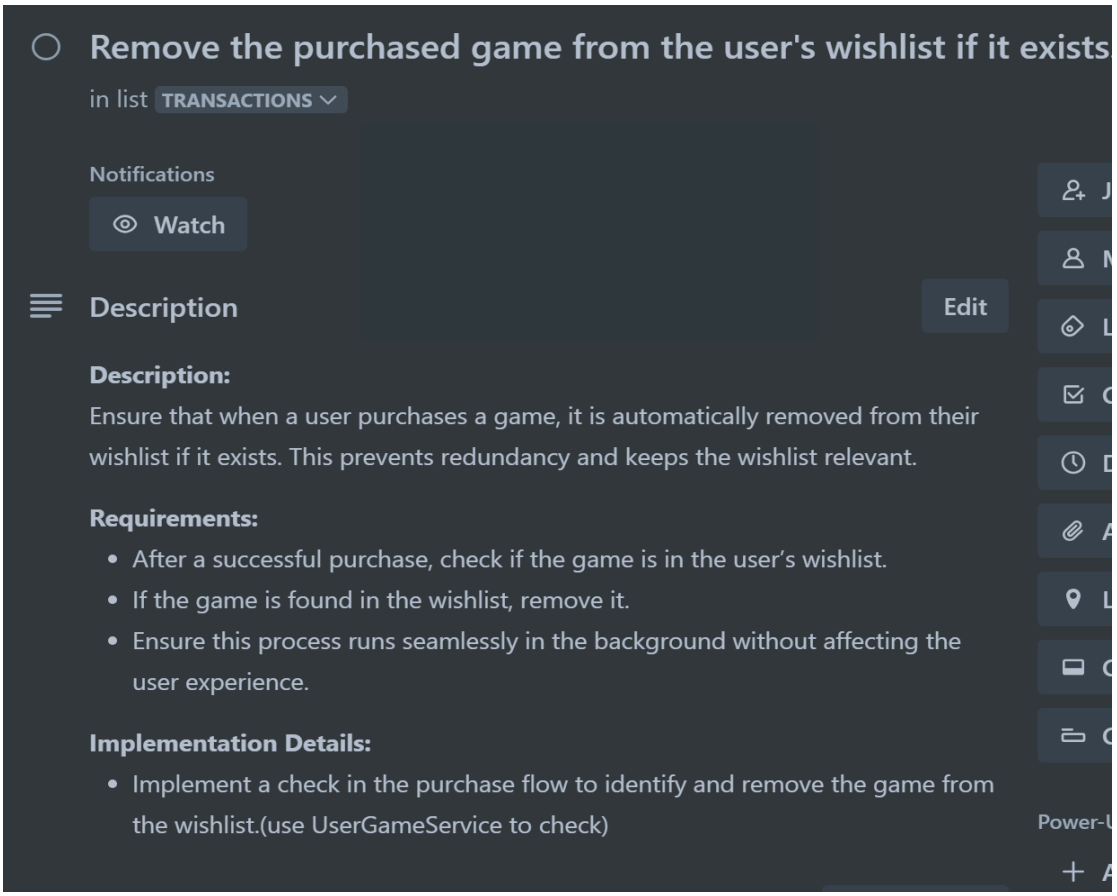

Validate card details before proceeding to checkout.
in list TRANSACTIONS
Notifications
Watch
Description
Edit
Implement a validation check for credit/debit card details in MoneyCartService before allowing the user to proceed to checkout.
Validation Requirements:
CVV must be exactly 3 digits.
Card Number must be exactly 13 digits.
Expiration Date must follow the correct format.
This validation only checks the format, not whether the card details are real or active.
Implementation Details:
Introduce a new CardValidator class to handle card validation logic.
Integrate CardValidator within MoneyCartService to enforce validation before checkout.
If validation fails, provide clear error messages indicating the specific issue and prompt the user to correct the details.

.

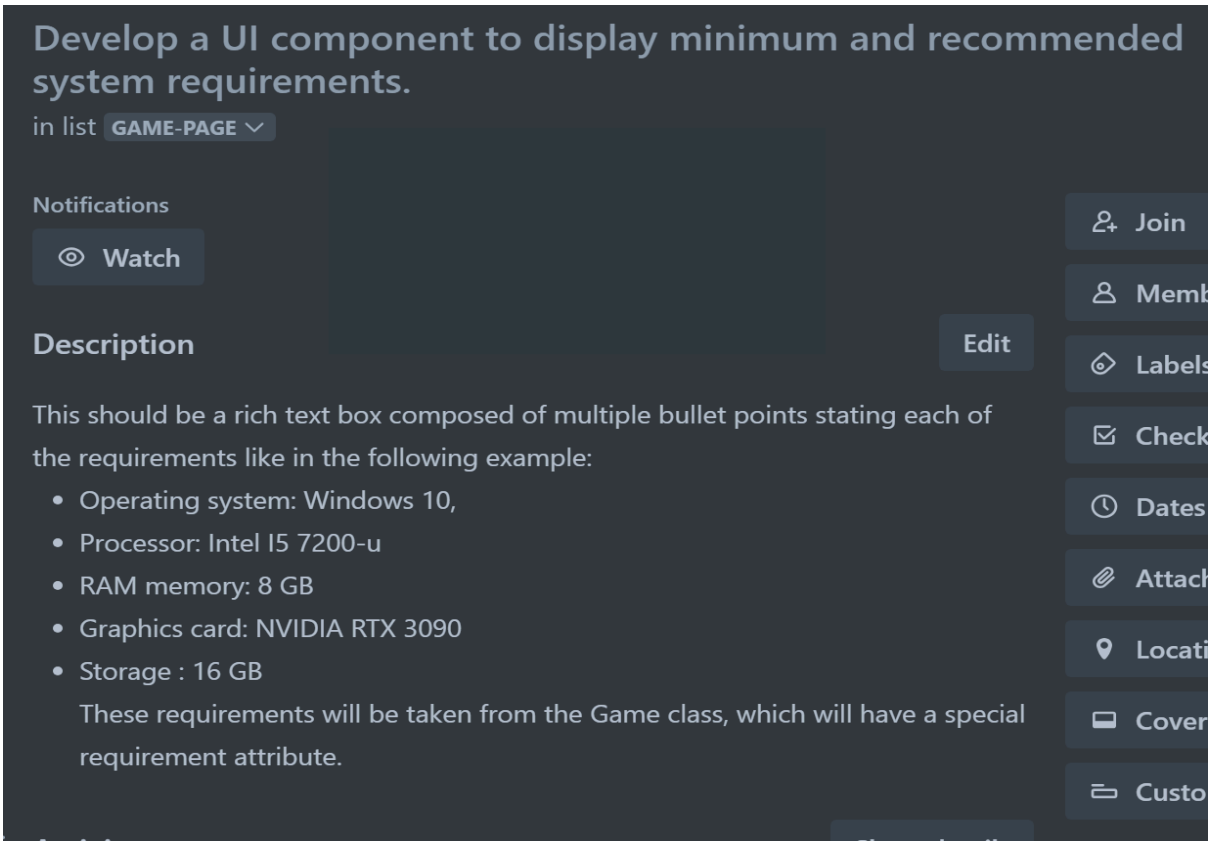
3


Ensure data consistency by verifying ownership before adding item to cart.
in list TRANSACTIONS
Notifications
Watch
Description
Edit
Implement a system check to prevent users from adding items they have already purchased to their cart. This ensures data consistency and avoids duplicate purchases.
Requirements:
If a user has already purchased a game, they should not be able to add it to the cart.
Implement a many-to-many (M-M) relationship table between users and games to track ownership.
Before adding an item to the cart, check this table to verify whether the user already owns the game.
If the game is already purchased, display a clear error message and prevent the action.
Implementation Details:
Create a UserGame table (or equivalent) to represent the many-to-many relationship between users and games.
Integrate the ownership check into the cart logic.(look at addGameToCartFunction from MoneyCartService)
Join
Members
Labels
Checklist
Dates
Attachment
Location
Cover
Custom Fields
Power-Ups
Add Power-Ups
Automation
Add button
Actions

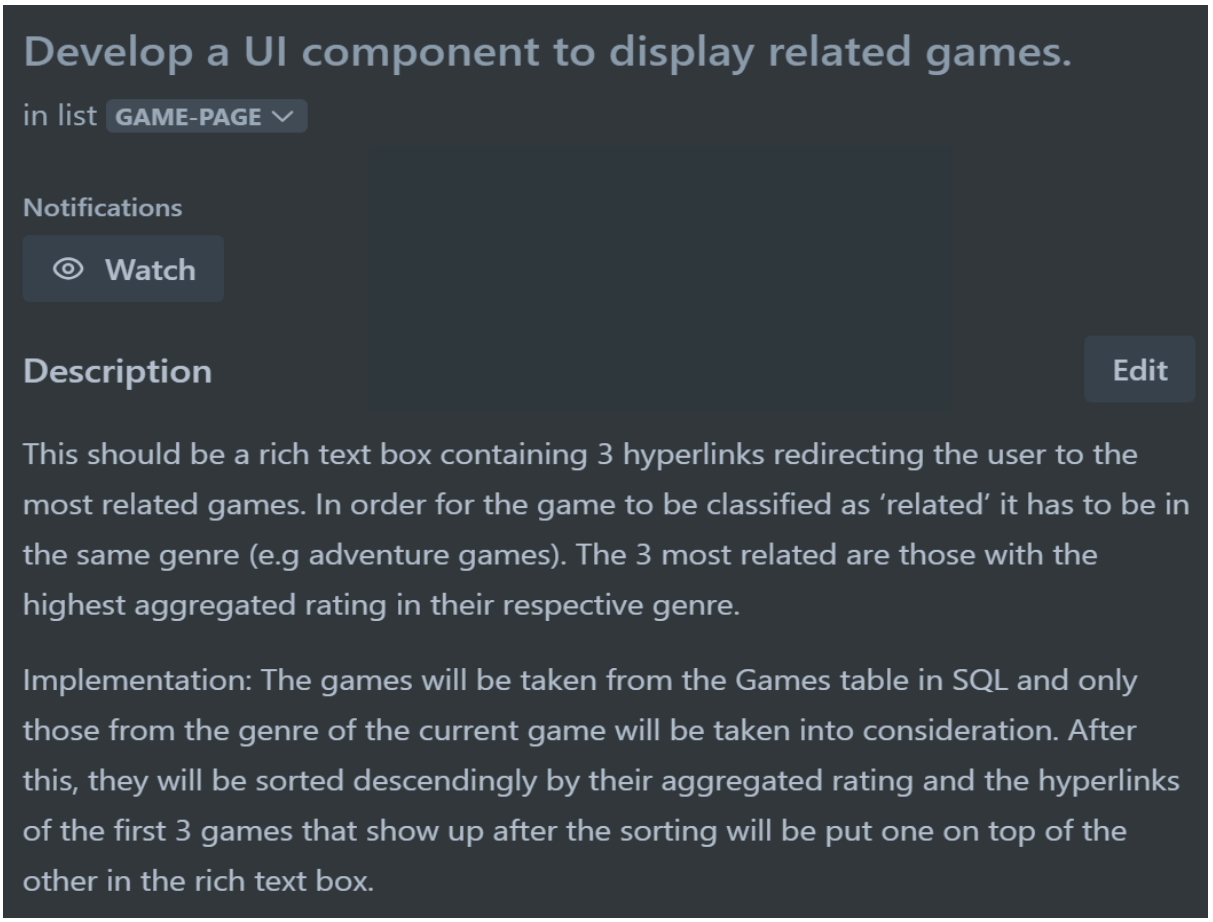
4


Remove the purchased game from the user's wishlist if it exists
in list TRANSACTIONS
Notifications
Watch
Description
Edit
Description:
Ensure that when a user purchases a game, it is automatically removed from their wishlist if it exists. This prevents redundancy and keeps the wishlist relevant.
Requirements:
After a successful purchase, check if the game is in the user's wishlist.
If the game is found in the wishlist, remove it.
Ensure this process runs seamlessly in the background without affecting the user experience.
Implementation Details:
Implement a check in the purchase flow to identify and remove the game from the wishlist.(use UserGameService to check)

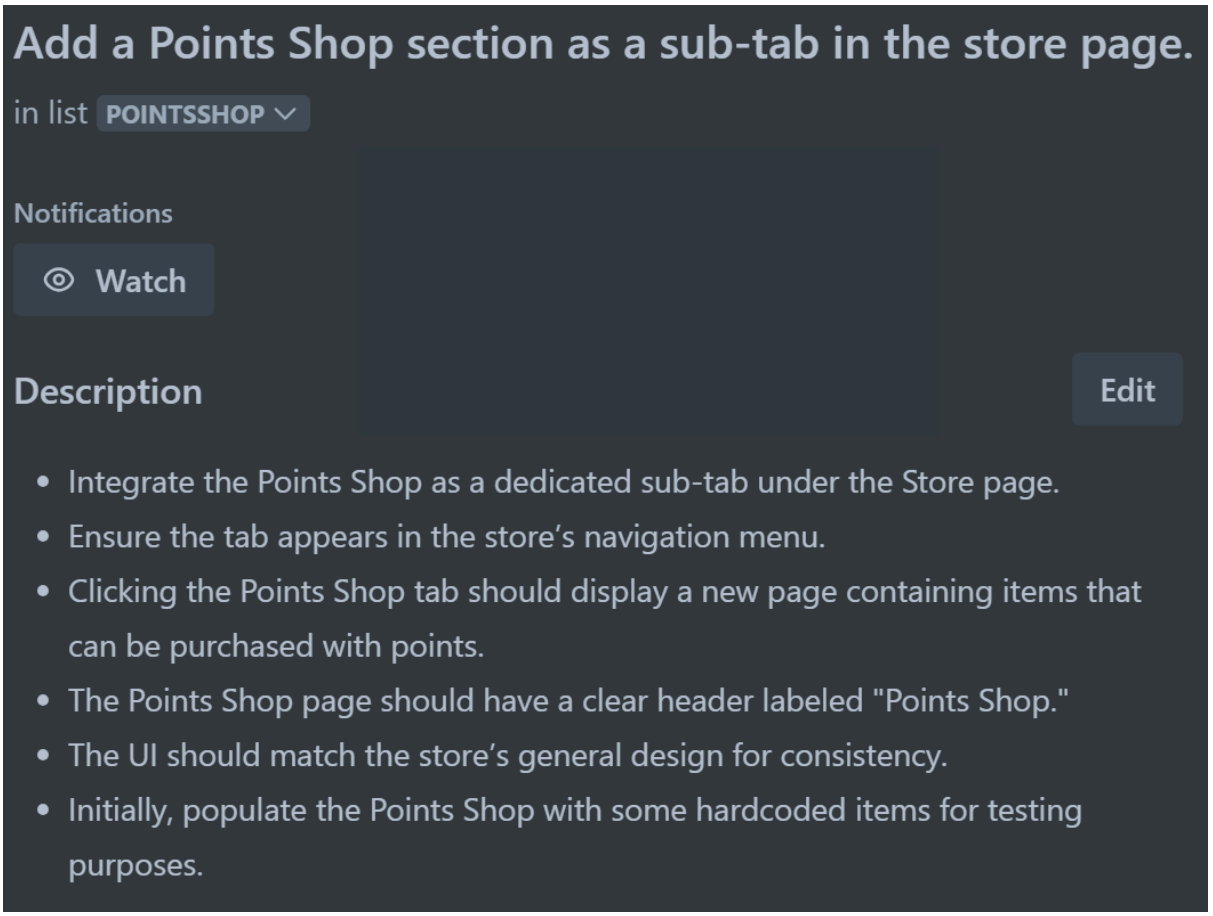
5


Develop a UI component to display minimum and recommended system requirements.
in list GAME-PAGE
Notifications
Watch
Description
Edit
This should be a rich text box composed of multiple bullet points stating each of the requirements like in the following example:
Operating system: Windows 10,
Processor: Intel I5 7200-u
RAM memory: 8 GB
Graphics card: NVIDIA RTX 3090
Storage : 16 GB
These requirements will be taken from the Game class, which will have a special requirement attribute.
Join
Memb
Labels
Check
Dates
Attach
Locati
Cover
Custo

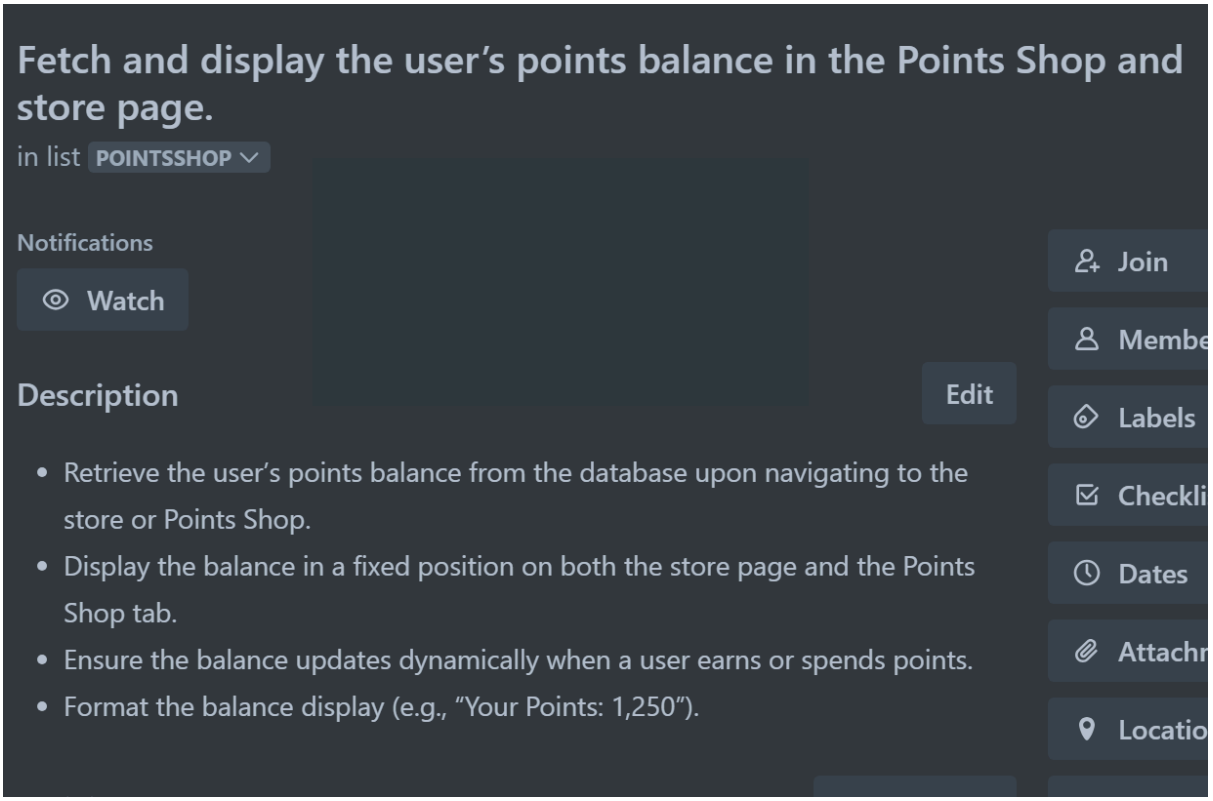
6


Develop a UI component to display related games.
in list GAME-PAGE
Notifications
Watch
Description
Edit
This should be a rich text box containing 3 hyperlinks redirecting the user to the most related games. In order for the game to be classified as 'related' it has to be in the same genre (e.g adventure games). The 3 most related are those with the highest aggregated rating in their respective genre.
Implementation: The games will be taken from the Games table in SQL and only those from the genre of the current game will be taken into consideration. After this, they will be sorted descendingly by their aggregated rating and the hyperlinks of the first 3 games that show up after the sorting will be put one on top of the other in the rich text box.

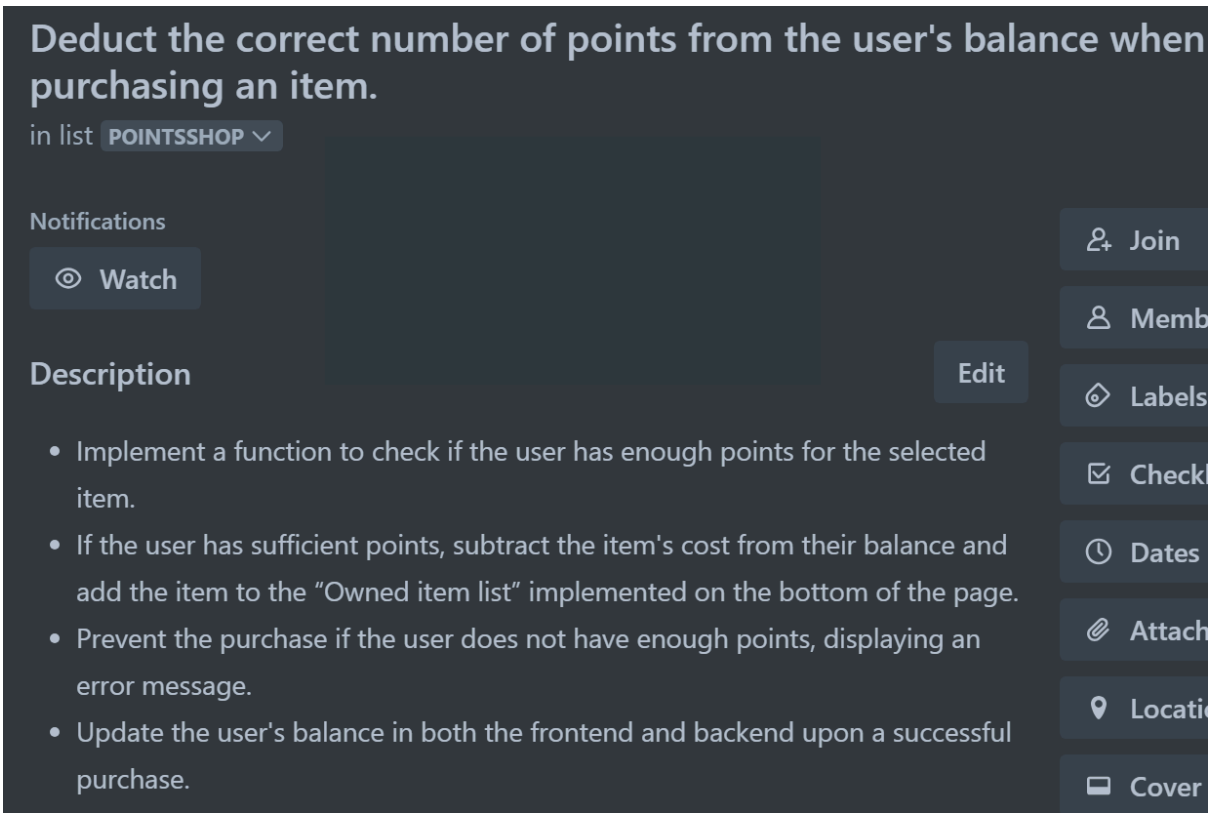
7


Add a Points Shop section as a sub-tab in the store page.
in list POINTSSHOP
Notifications
Watch
Description
Edit
Integrate the Points Shop as a dedicated sub-tab under the Store page.
Ensure the tab appears in the store's navigation menu.
Clicking the Points Shop tab should display a new page containing items that can be purchased with points.
The Points Shop page should have a clear header labeled "Points Shop."
The UI should match the store's general design for consistency.
Initially, populate the Points Shop with some hardcoded items for testing purposes.

8


Fetch and display the user's points balance in the Points Shop and store page.
in list POINTSSHOP
Notifications
Watch
Description
Edit
Retrieve the user's points balance from the database upon navigating to the store or Points Shop.
Display the balance in a fixed position on both the store page and the Points Shop tab.
Ensure the balance updates dynamically when a user earns or spends points.
Format the balance display (e.g., "Your Points: 1,250").
Join
Labels
Dates

9


Deduct the correct number of points from the user's balance when purchasing an item.
in list POINTSSHOP
Notifications
Watch
Description
Edit
Implement a function to check if the user has enough points for the selected item.
If the user has sufficient points, subtract the item's cost from their balance and add the item to the "Owned item list" implemented on the bottom of the page.
Prevent the purchase if the user does not have enough points, displaying an error message.
Update the user's balance in both the frontend and backend upon a successful purchase.
Join
Labels
Dates
Cover

10


Create a UI component that displays details of items available in the Points Shop.
in list POINTSSHOP
Notifications
Watch
Description
Edit
Develop a UI component that displays each item's details in the Points Shop.
The item details should include:
Item name
Item image
Item description
Points cost
A "Purchase" button
Ensure items are displayed in a grid or list format for easy browsing.
Hovering over an item should slightly highlight it to improve user experience.
Clicking on an item should open a modal or expanded view with additional details.
Ensure the "Purchase" button is disabled if the user does not have enough points.
Fetch item details dynamically from the database and update the UI accordingly.
Join
Members
Labels
Checklist
Dates
Attachme
Location
Cover
Custom F
Power-Ups
Add Powe
Automation

GUI: Game Lists

in list HOMEPAGE

Notifications

Watch

Description

Edit

Prerequisites:

- StoreView class template
- Game class from the domain
- A way to display a single game (see “Displaying a Single Game” task)
- **No connection to the back-end** is required other than being able to use the Game class from the domain

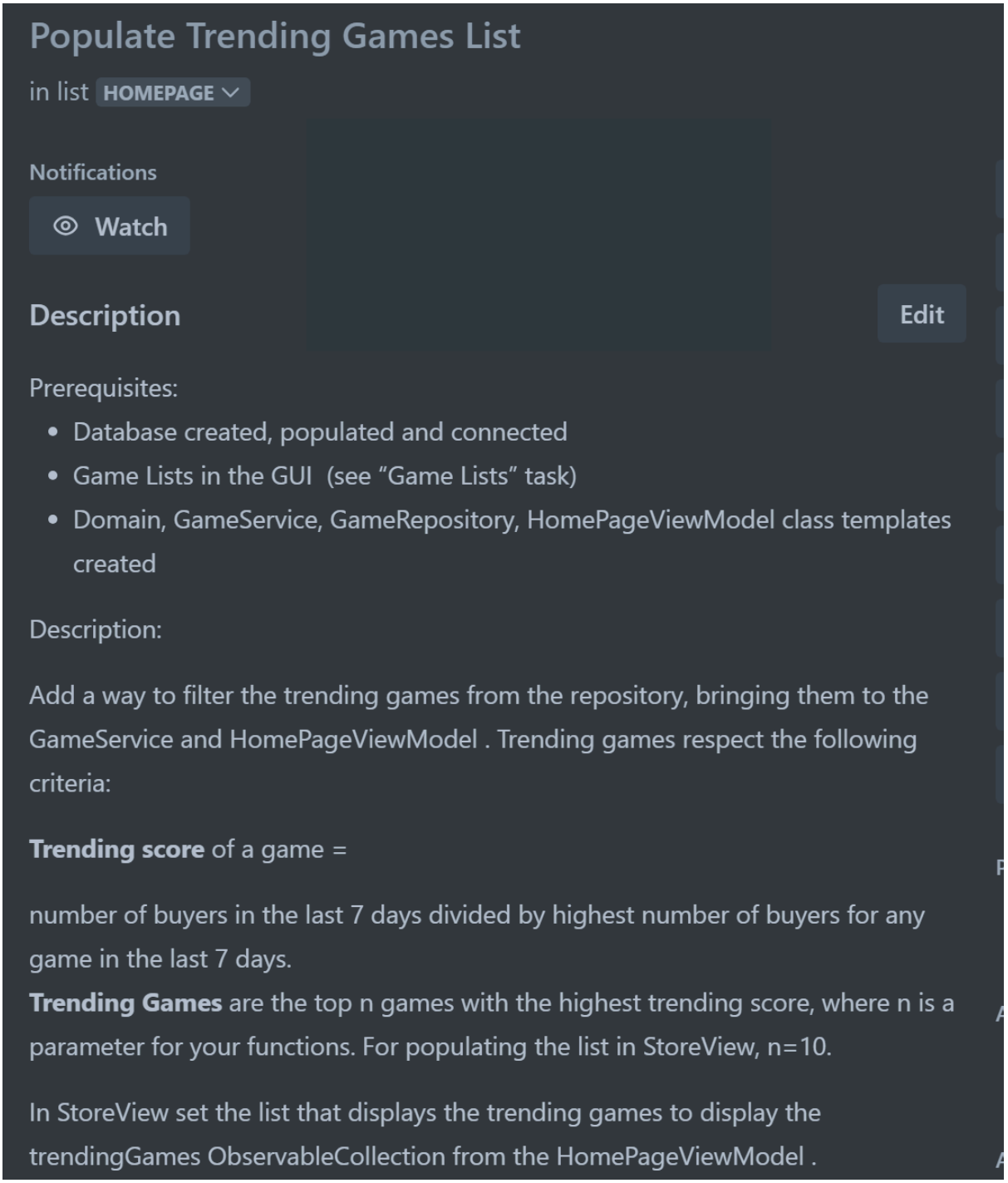
Description:

The new methods written for this task will be added in the StoreView class.

On the homepage, add 3 lists in the GUI that are able to display 5 Games at a time. Arrow(pointing outwards from the center) buttons on the left(previous) and right(next) move the displayed games to the previous/next 5 games that are available to be displayed. If there are less than 5 games on a page to be displayed, the remaining slots in the list will be left blank, displaying just a pre-defined empty slot model. For now, put some hardcoded games in those lists.

The 3 lists will be stacked vertically, each list displaying its respective games horizontally.

The 3 lists will be the lists for trending, on sale and recommended games in the future, when they will be populated with actual entities from the repository. The category for each list shall be displayed in a text box above the list in an easy to read font and large letter size.


Populate Trending Games List
in list HOMEPAGE
Notifications
Watch
Description
Edit
Prerequisites:
Database created, populated and connected
Game Lists in the GUI (see "Game Lists" task)
Domain, GameService, GameRepository, HomePageViewModel class templates created
Description:
Add a way to filter the trending games from the repository, bringing them to the GameService and HomePageViewModel . Trending games respect the following criteria:
Trending score of a game =
number of buyers in the last 7 days divided by highest number of buyers for any game in the last 7 days.
Trending Games are the top n games with the highest trending score, where n is a parameter for your functions. For populating the list in StoreView, n=10.
In StoreView set the list that displays the trending games to display the trendingGames ObservableCollection from the HomePageViewModel .

13.

Populate Games on Sale List

in list HOMEPAGE

Notifications

Watch

Description

Edit

Prerequisites:

- Database created, populated and connected
- Game Lists in the GUI (see “Game Lists” task)
- Domain, GameService, GameRepository, HomePageViewModel class templates created
- Trending Score computation (Included in “Populating Trending Games List” task

Description:

Add a way to bring games on sale from the repository to the GameService and HomePageViewModel.

Games on sale are games which have the discount field > 0. You should filter the games that are on sale and be able to return the top n games on sale ordered by their **trending score.** For populating the lists in store view, n=10.

In StoreView set the list that displays the games on sale to display the gamesOnSale ObservableCollection from the HomePageViewModel .

14.

Validate Games from other Devs

in list **DEVELOPERSPAGE**

Notifications

Watch

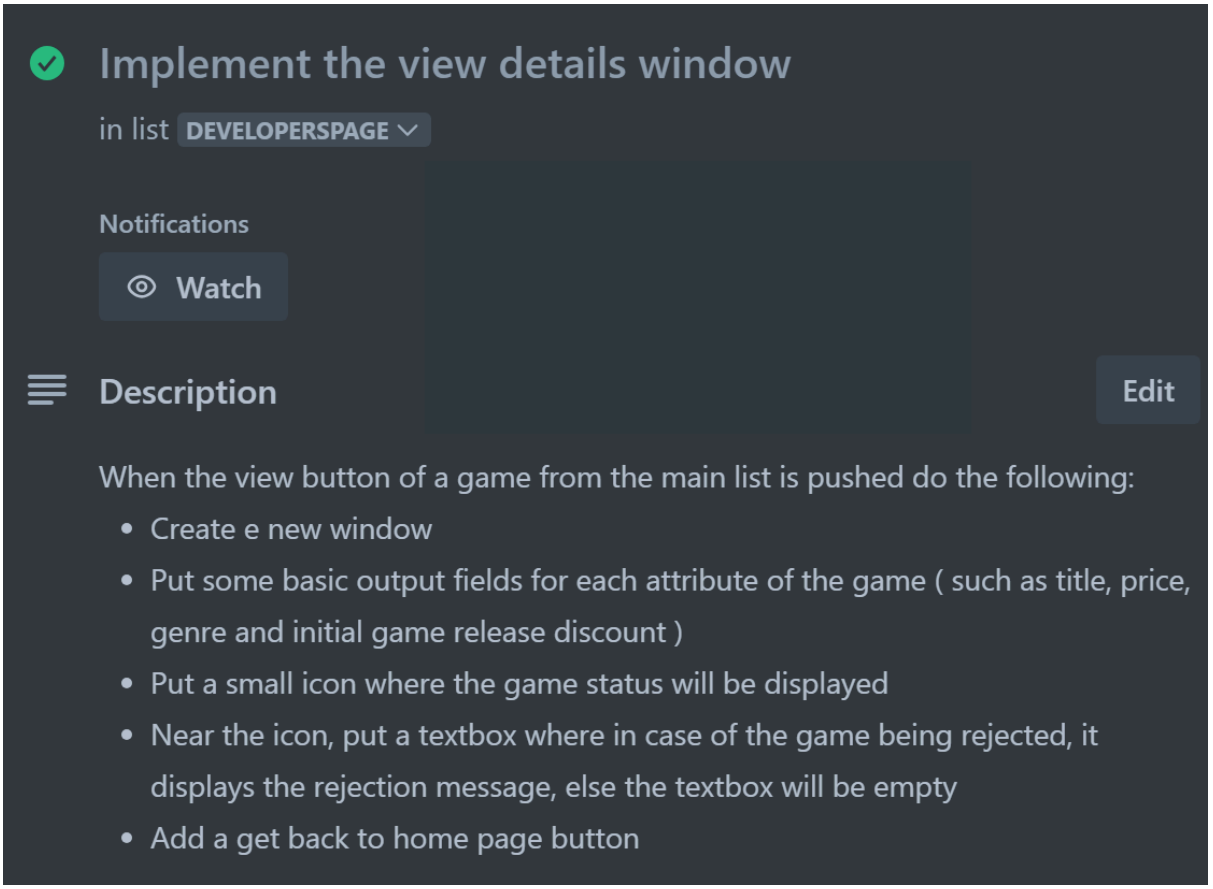
Description

Edit

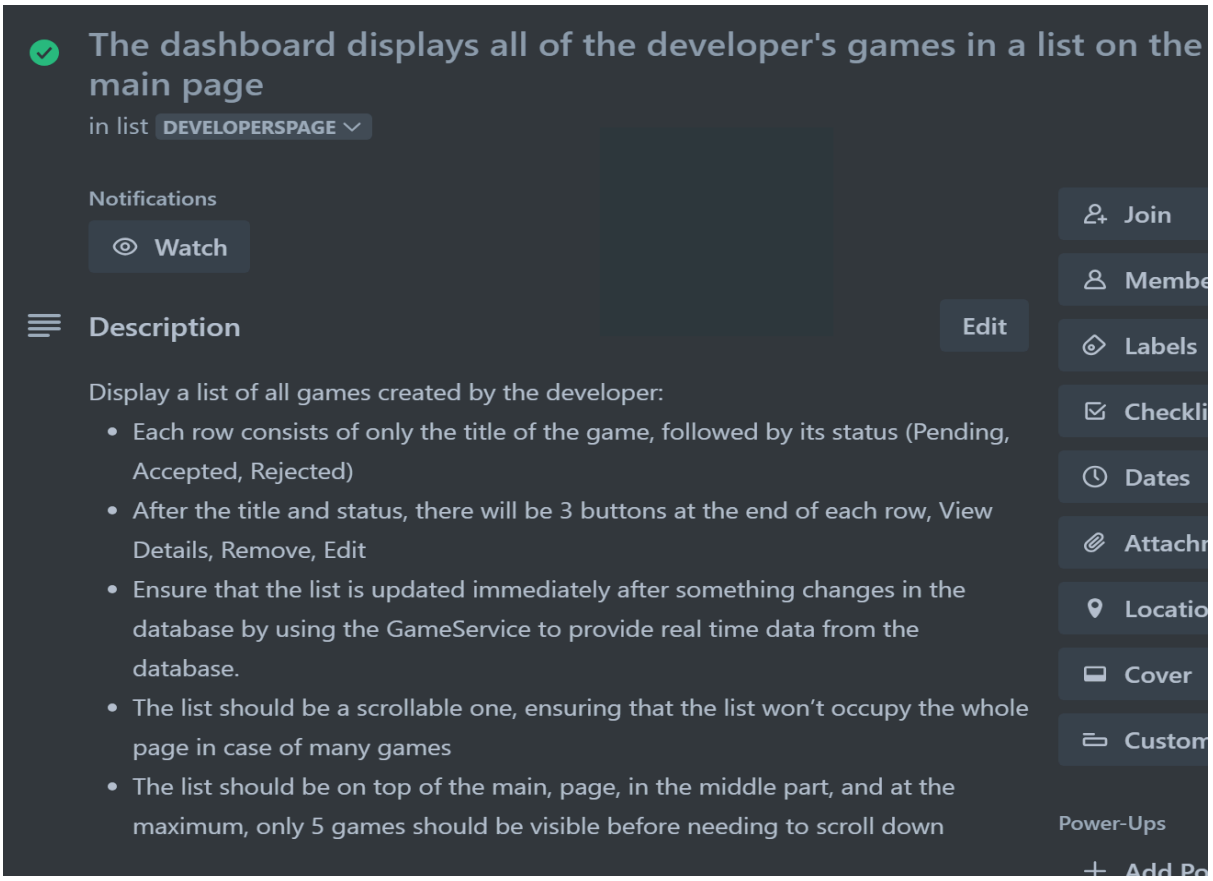
This is related to the page where developers can accept or reject games from other developer, using the already implemented validation page:

- If the approve button that has been previously implemented is pressed, then the game's state goes to approved and the game is published on the steam store
- In this case, the DB should immediately be updated to contain the game, and it should instantly be available on the developer page.
- If the reject button is pushed, then make sure that the developer also inputs a message for the rejection reason
- If the reason is written, and the reject button is pressed, then make sure that the status is set to rejected, and the game is not published on the store
- The DB should also update to contain the game, but with the rejected status, the change needs to also be visible on the developer game list

15.


Implement the view details window
in list DEVELOPERSPAGE
Notifications
Watch
Description
Edit
When the view button of a game from the main list is pushed do the following:
Create e new window
Put some basic output fields for each attribute of the game (such as title, price, genre and initial game release discount)
Put a small icon where the game status will be displayed
Near the icon, put a textbox where in case of the game being rejected, it displays the rejection message, else the textbox will be empty
Add a get back to home page button

16.


The dashboard displays all of the developer's games in a list on the main page
in list DEVELOPERSPAGE
Notifications
Watch
Description
Edit
Display a list of all games created by the developer:
Each row consists of only the title of the game, followed by its status (Pending, Accepted, Rejected)
After the title and status, there will be 3 buttons at the end of each row, View Details, Remove, Edit
Ensure that the list is updated immediately after something changes in the database by using the GameService to provide real time data from the database.
The list should be a scrollable one, ensuring that the list won't occupy the whole page in case of many games
The list should be on top of the main, page, in the middle part, and at the maximum, only 5 games should be visible before needing to scroll down
Join
Labels
Dates
Cover
Power-Ups

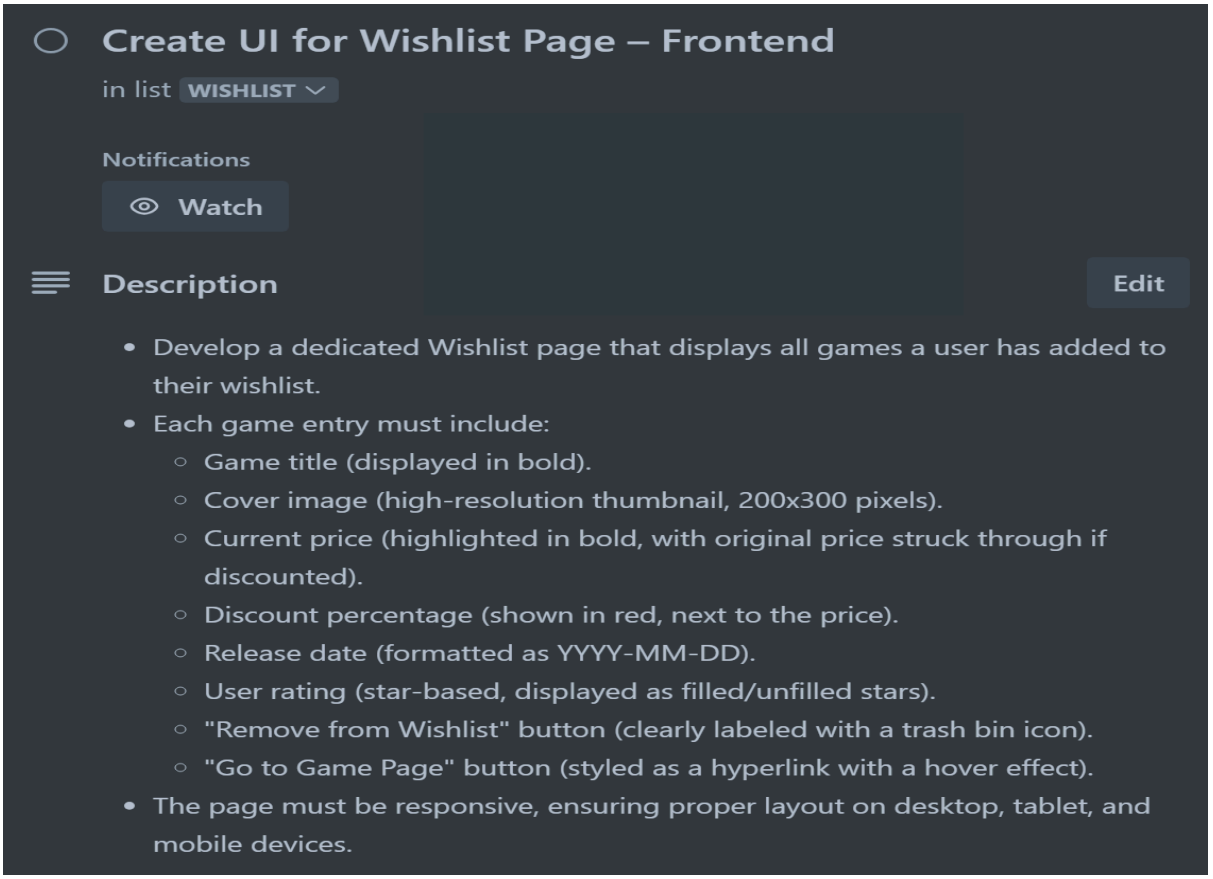
17


Implement the delete button
in list DEVELOPERSPAGE
Notifications
Watch
Description
Edit
This task is related to the delete button displayed after each game:
When pressed the delete button should open a small pop-up asking the developer if he is sure he wants to delete the game, with two options, yes or no
If yes is pressed, then the game shall be deleted from the developer page
If yes is pressed, then the game shall be deleted from the store page
If yes is pressed, then the game shall be deleted from all of the active users who own it, after the dev will be shown with the appropriate warning that the game is currently owned by other players.
Make sure that the update is reflected immediately on the developer list (i.e it instantly gets deleted from the DB and then from the list)

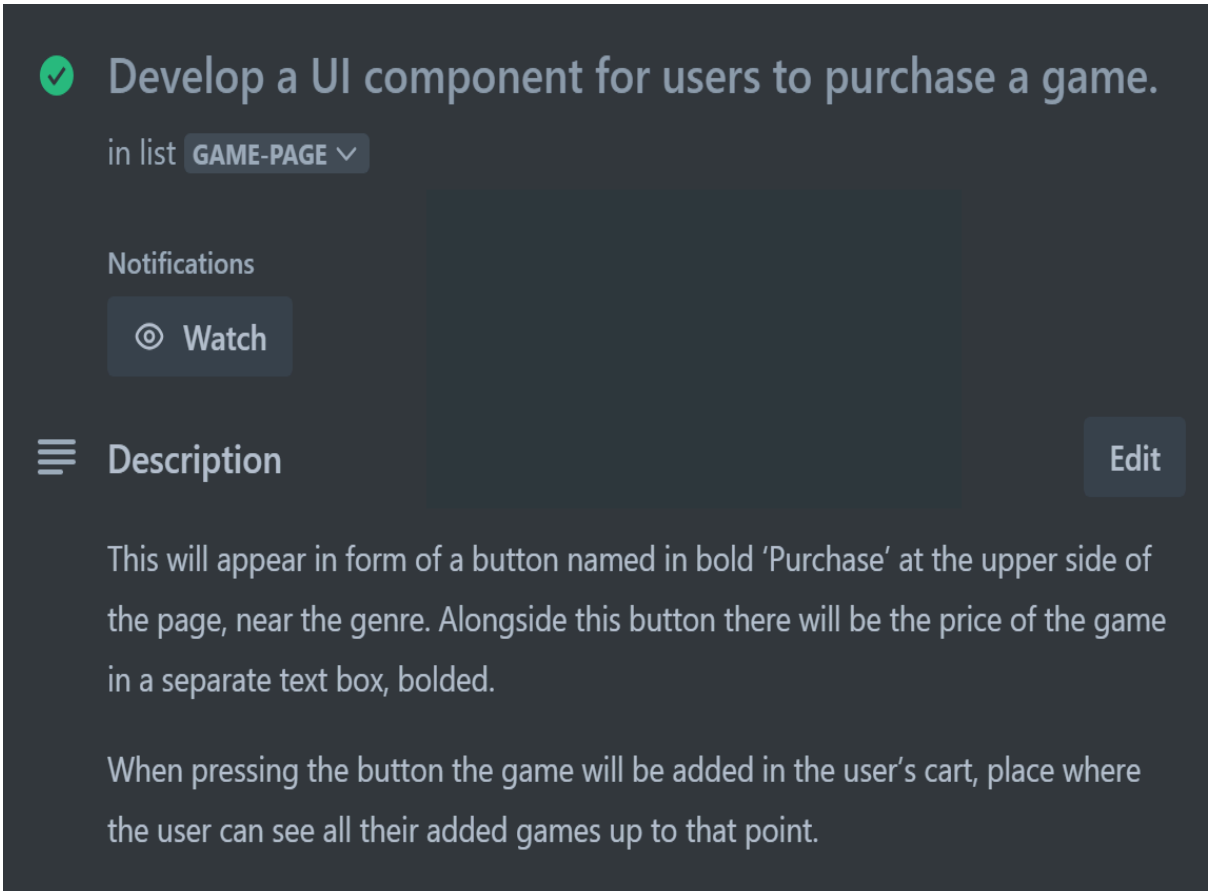
18.


Display Wishlisted Games
in list WISHLIST
Notifications
Watch
Description
Edit
Fetch and render the user's wishlisted games dynamically
Ensure the list updates in real time when games are added/removed.
The system should retrieve and print a list of wishlisted games for the current user.
The display should include key information such as:
Game ID (for reference)
Game Title
Current Price
Discount Percentage (if applicable)
User Rating (as stars or numeric values)
Release Date
The output should be structured and easy to read.

19


Create UI for Wishlist Page – Frontend
in list WISHLIST
Notifications
Watch
Description
Edit
Develop a dedicated Wishlist page that displays all games a user has added to their wishlist.
Each game entry must include:
Game title (displayed in bold).
Cover image (high-resolution thumbnail, 200x300 pixels).
Current price (highlighted in bold, with original price struck through if discounted).
Discount percentage (shown in red, next to the price).
Release date (formatted as YYYY-MM-DD).
User rating (star-based, displayed as filled/unfilled stars).
"Remove from Wishlist" button (clearly labeled with a trash bin icon).
"Go to Game Page" button (styled as a hyperlink with a hover effect).
The page must be responsive, ensuring proper layout on desktop, tablet, and mobile devices.

20


Develop a UI component for users to purchase a game.
in list GAME-PAGE
Notifications
Watch
Description
Edit
This will appear in form of a button named in bold 'Purchase' at the upper side of the page, near the genre. Alongside this button there will be the price of the game in a separate text box, bolded.
When pressing the button the game will be added in the user's cart, place where the user can see all their added games up to that point.