# ASP.NET MVC Retake Exam (October 2015) – StreamPowered

You are assigned to implement a Web-based computer game catalog. The system consists of server-side logic (back-end) and client-side logic (front-end); Use C#, ASP.NET MVC, Entity Framework and Microsoft SQL Server for the back-end and HTML, CSS, Bootstrap, JavaScript and AJAX for the front-end.

## Data Model

* The system contains information about **users**, **games**, **genres**, **ratings** and **reviews**.
* Users can register in the system. After having successfully registered, users have a **username**, **email** and **password**. The username must be unique – there can be no two users with the same username. The password should never be stored as plain text. Users also have some **reviews** and **ratings**. Administrators keep track of their **created games**.
* **Games** have **title**, **description**, **genre**, **system requirements**, some **images** (stored as URLs), **ratings** and **reviews**.
* **Ratings** have **author** and **rating value** (a number between 1 and 5, with steps of 1 unit)
* **Genres** have name and some **games**.
* Game **reviews** have **author**, **creation time** and **content**.
* Users can view information about games, write reviews about them and rate them. Administrators can add new games.

## Functionality

* **Guests** (anonymous users) can register an account with their own username, email and password.
* **Guests** can login by username and password.
* **Logged-in users** can logout.
* **Guests** can view the home page. It holds information about the 5 latest games (ordered from the latest to earliest), 5 highest-rated games (ordered from the highest to lowest average rating), 5 latest reviews (ordered from the latest to earliest).
* **Guests** can view all games in the site (ordered from the latest to earliest, with paging).
* **Guests** can view all games in a certain genre (ordered from the latest to earliest, with paging).
* **Guests** can view game details.
* **Logged-in users** can add reviews to game.
* **Administrators** can add new games.
* Display a **notification message** after each successful operation which changes the database.

## Design the Database

Design **entity classes** and create a **database** to hold the users, games, genres, ratings and reviews. Use Entity Framework code-first. Put your data layer in a separate Visual Studio project (class library).

The data layer should be implemented with some abstraction:

* You can use repository pattern and unit of work.
* You can use repositories and data service layer with database queries.
* You can use only data services layer on the top of the EF context.

10 score

Use **automatic migrations**. At application start-up, **fill some sample data** in the database.

Use the provided **seed.json** file to seed the database.

**Note:** In the seeding files, there are many ratings for a game from the same user. This will be invalid in the application, but use the data "as-is" in order to have more ratings. (Otherwise, you will need to create more users).

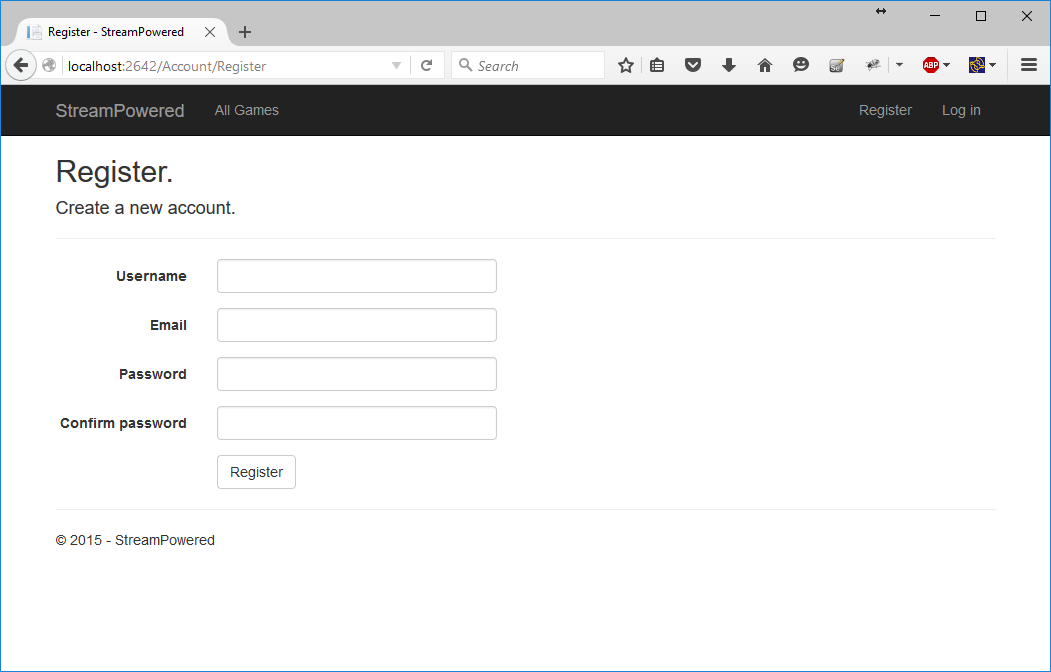
20 score

## Implement User Registration, Login and Logout

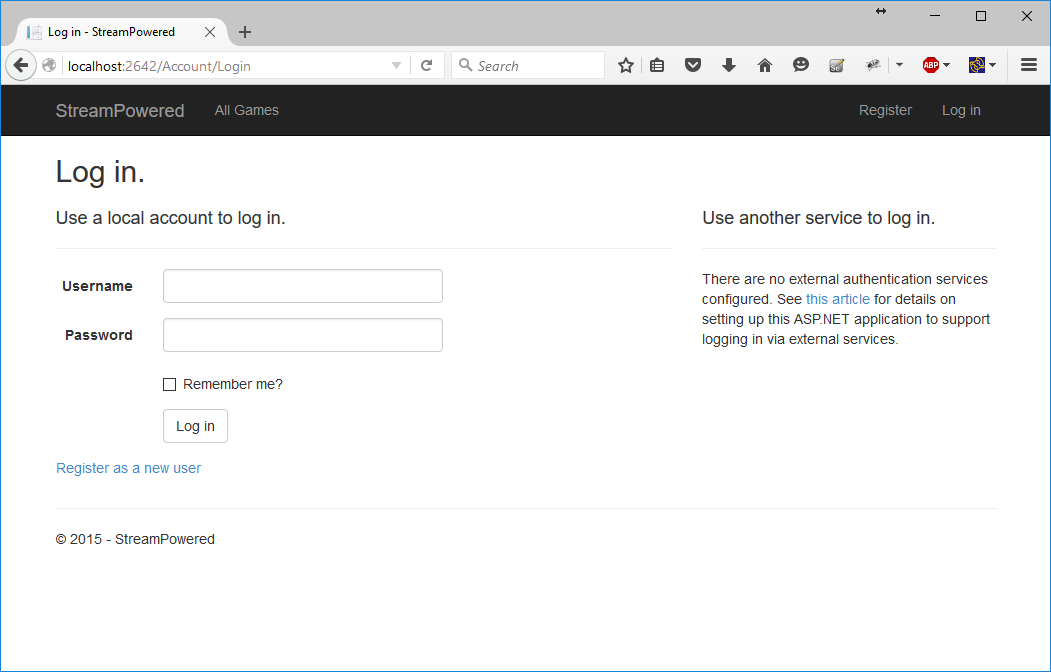
Implement user **registration**, **login** and **logout**.

* **User registration** should create a new user account and display "**User registered**" **notification message**. In case of error, display an error message. **Validate** the registration form. **All passwords must have at least one lowercase letter and one digit and should be at least 6 symbols long** (e.g. "password123").
* The **user registration form** should have **username,** **email**, **password** and **confirm password** fields (mandatory).

8 score



* **User login** should login an existing user by **username** and **password**. After login, display the site home page. In case of a problem, display an error message. **Validate** the login form.



4 score

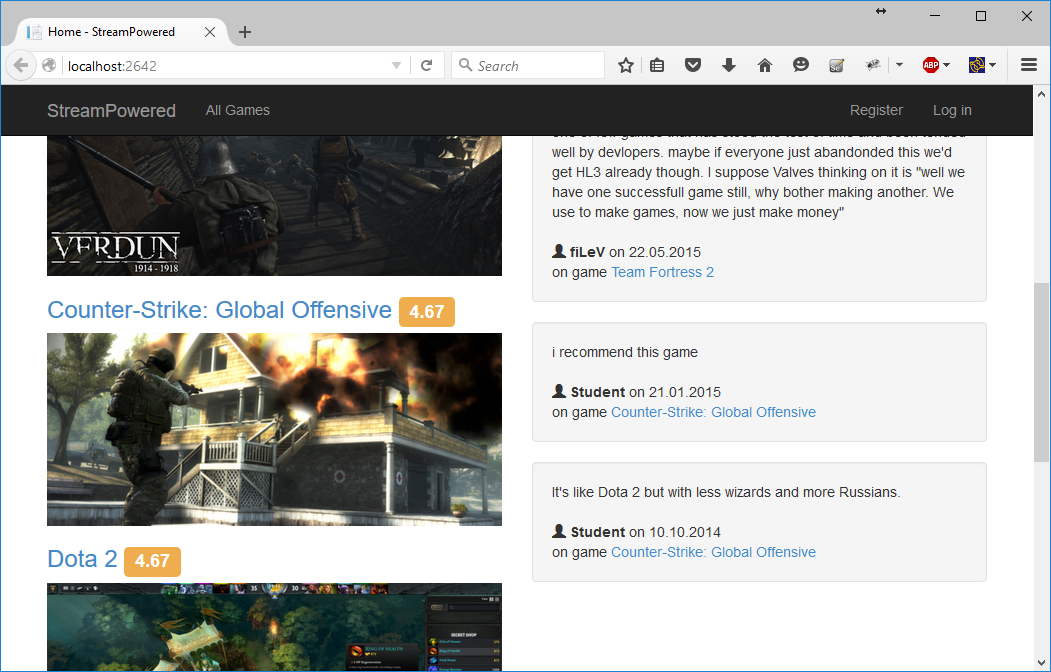
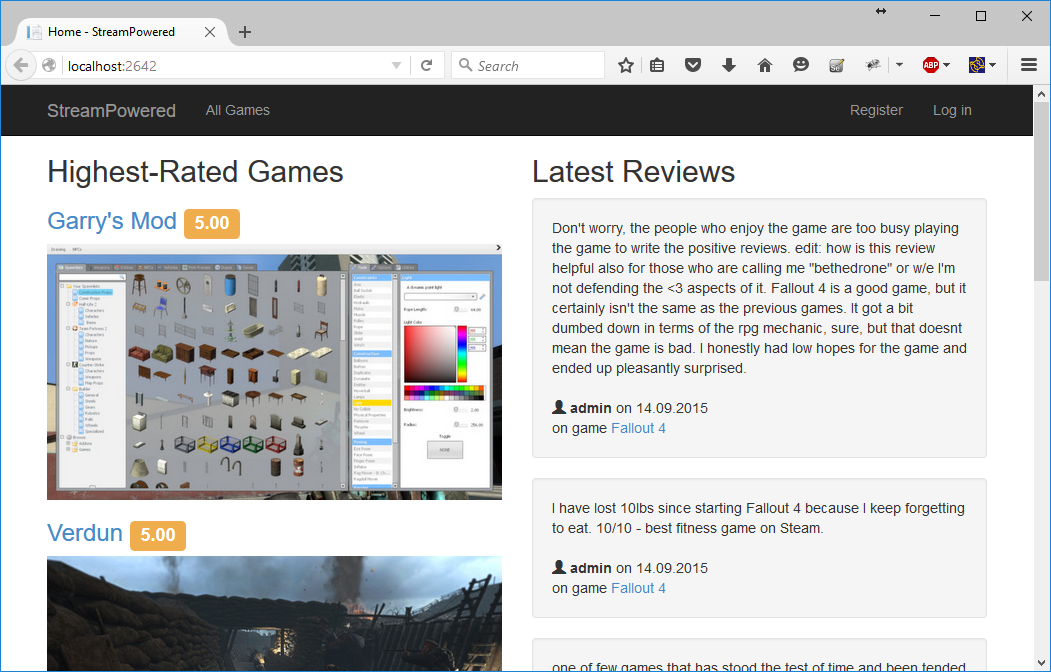
* **User logout** should invalidate user's session and redirect to the application home page.

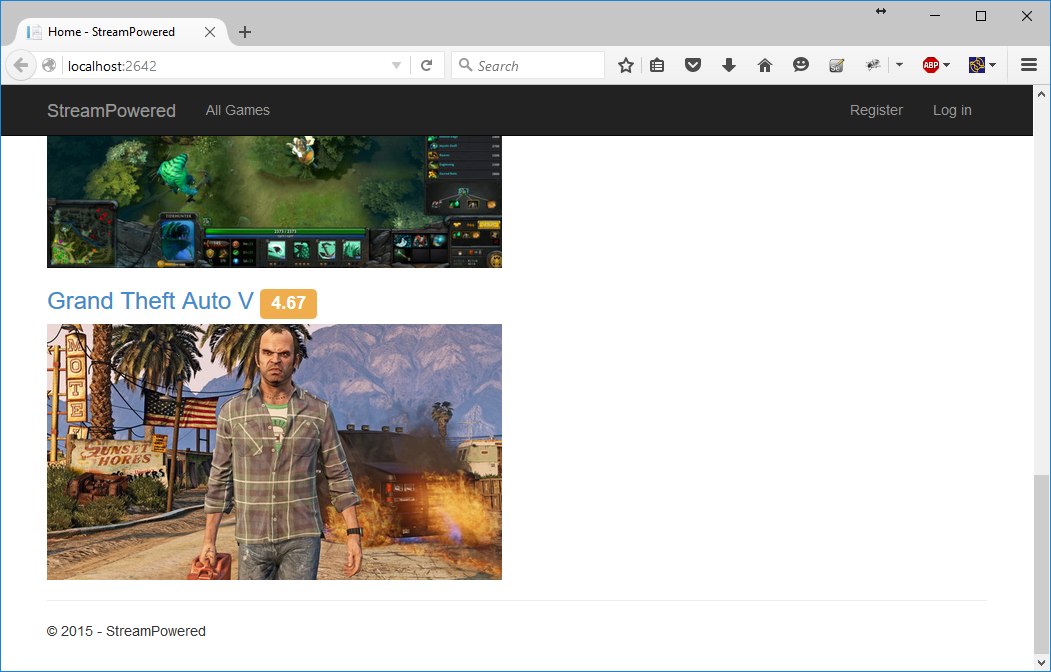
2 score

## Implement the Home Page

Anonymous users can view the **home page**.

* Display the 5 games with highest average rating (ordered by average rating, the highest one being on top, then by title alphabetically) and the 5 latest reviews (from the latest to the earliest).
* All game titles must be clickable. Upon clicking, the user should see the game details.
* Next to each game title, display the first image for the game which has been saved in the database. In case there is no image for the game, display a placeholder image. Display the average game rating too (rounded to two decimal places).

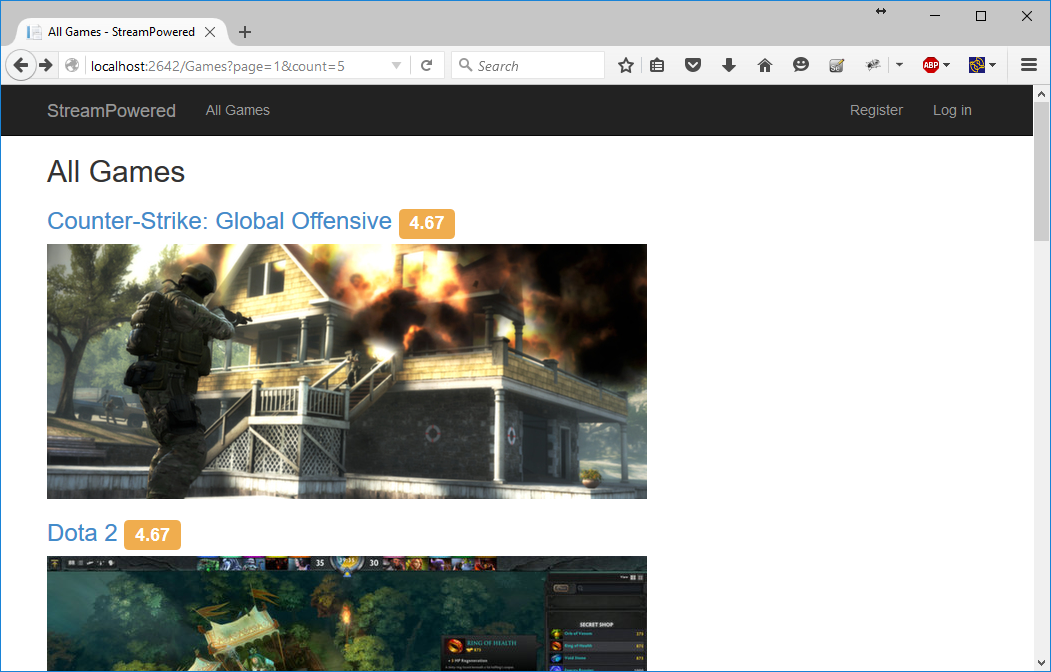


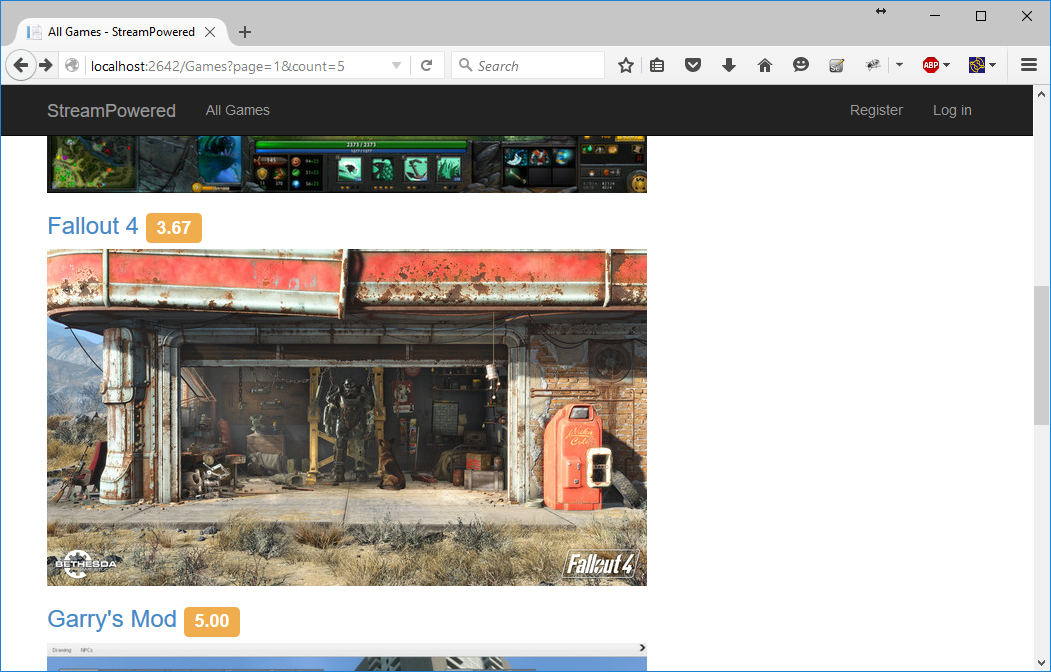


12 score

## Implement the View All Games Page

Anonymous users can **view all games**.





* Display all games, ordered by title alphabetically.
* The game titles should lead to the "game details" page.

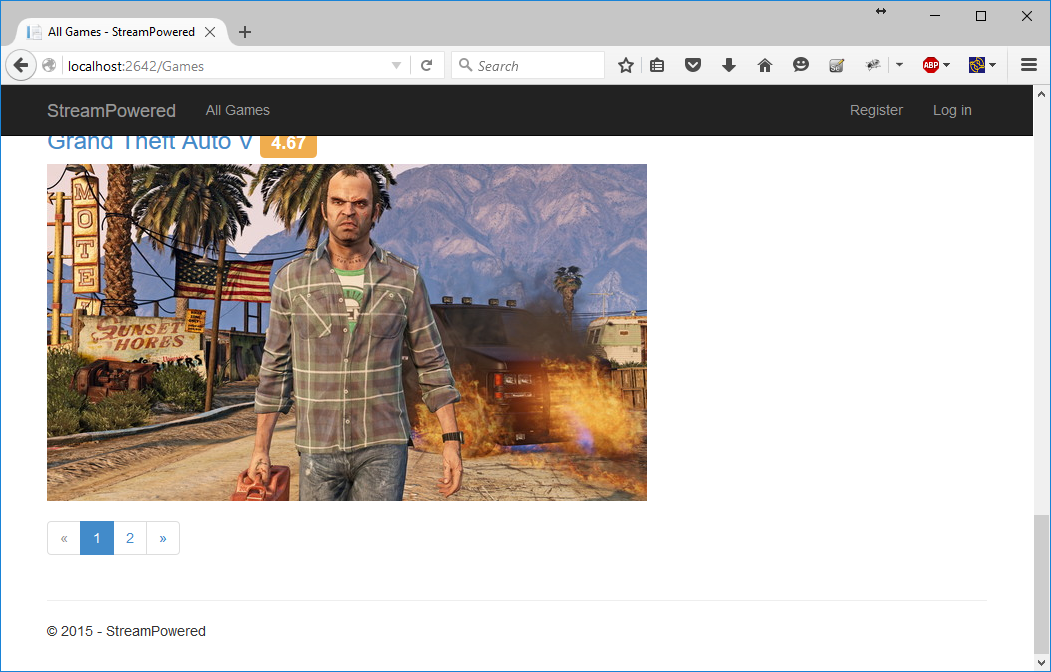
4 score

* Display the first image saved in the database next to each game title. In case there is no image for the game, display a placeholder image. Display the average game rating too.

2 score

* Use pagination. Display 5 games per page by default. The current page should be highlighted. There should also be links to the first and last pages (« and »). When the user is on the first page, the « link should be disabled. The same applies for the last page (and »).

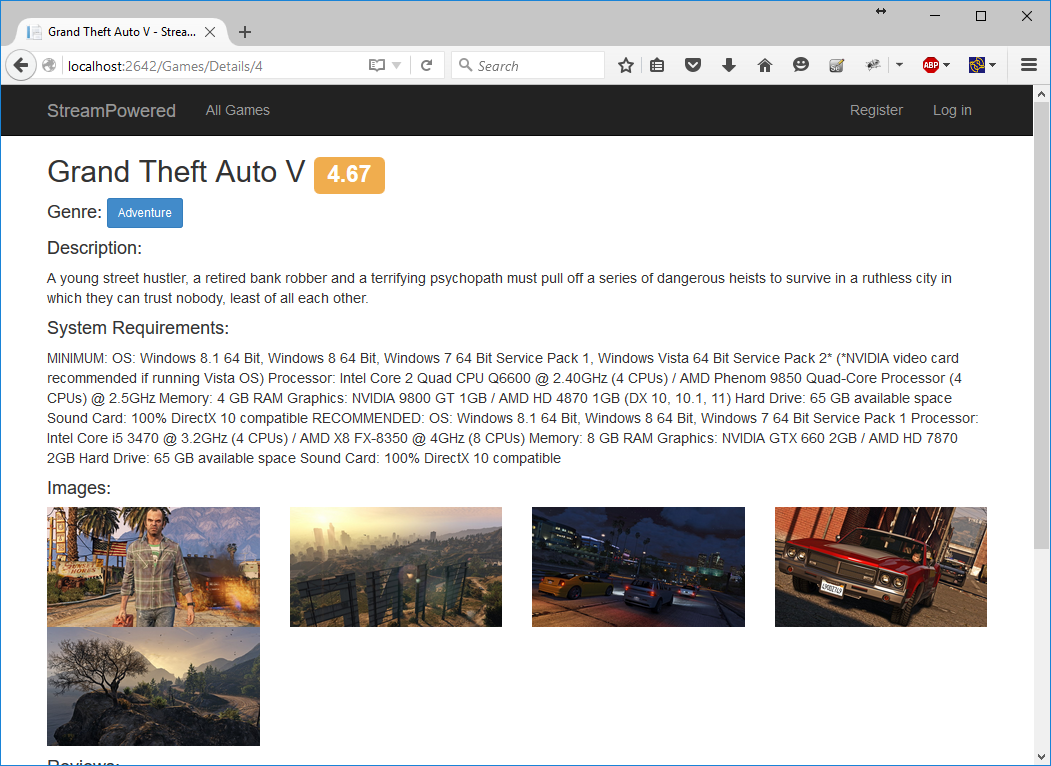
You **ARE NOT allowed** to use any JavaScript pagination plugins, HTML helpers or server-side code (such as PagedList) except Bootstrap pagination.



**BONUS:** 8 score

## Implement the Game Details Page

Guests can view details about all games.

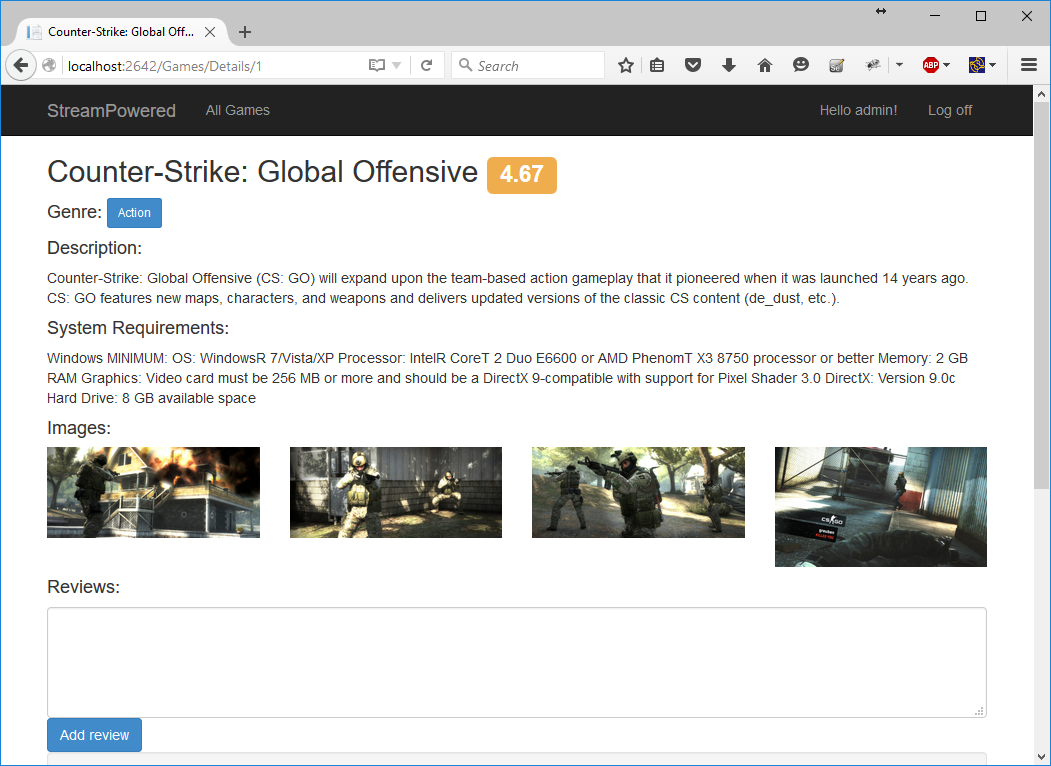


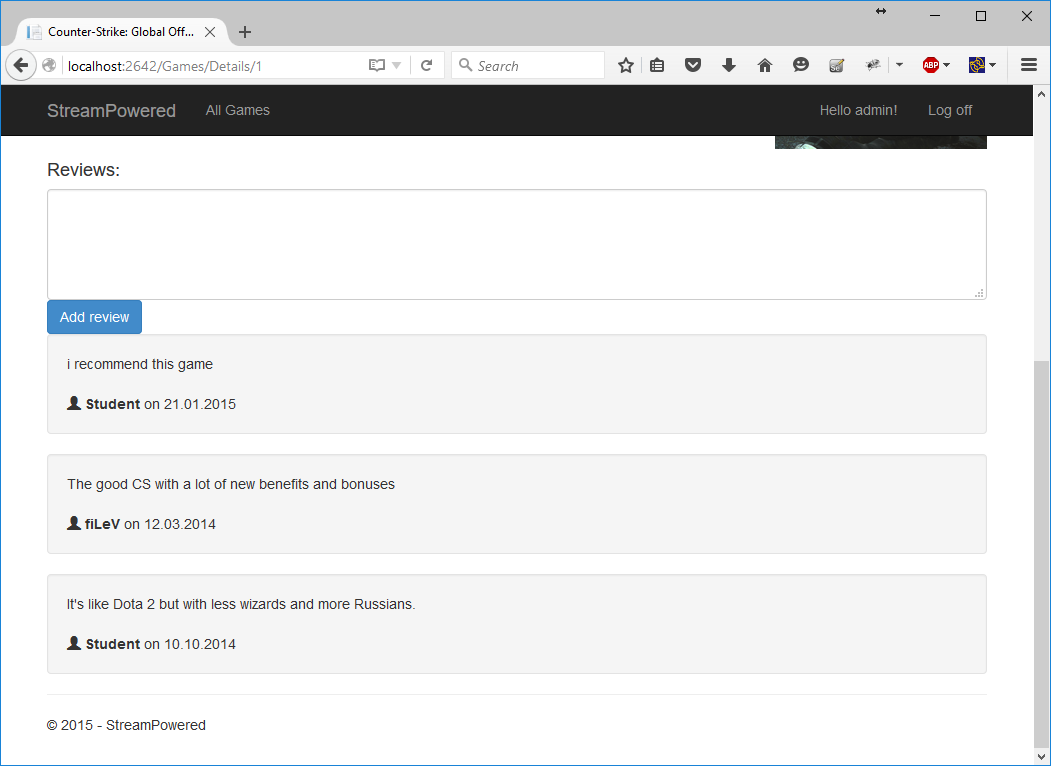
* The genre name should be a link to display all games in the genre.

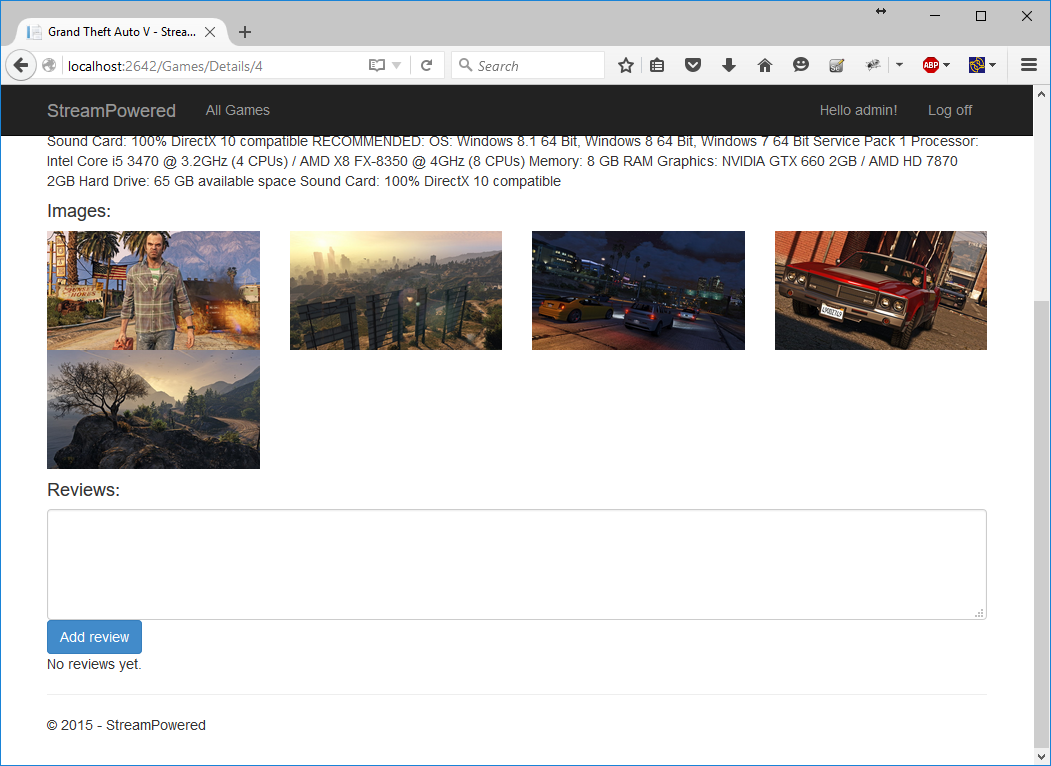
2 score

* In case there are no reviews, display a message. In case there are already some reviews, display them from the latest to the earliest.

2 score







## Implement Reviewing Games

Logged-in users can review any game.

* Implement a form to add a review. The form should have review content and should be large enough to contain a multi-paragraph review. Display the form only when there is a logged-in user.

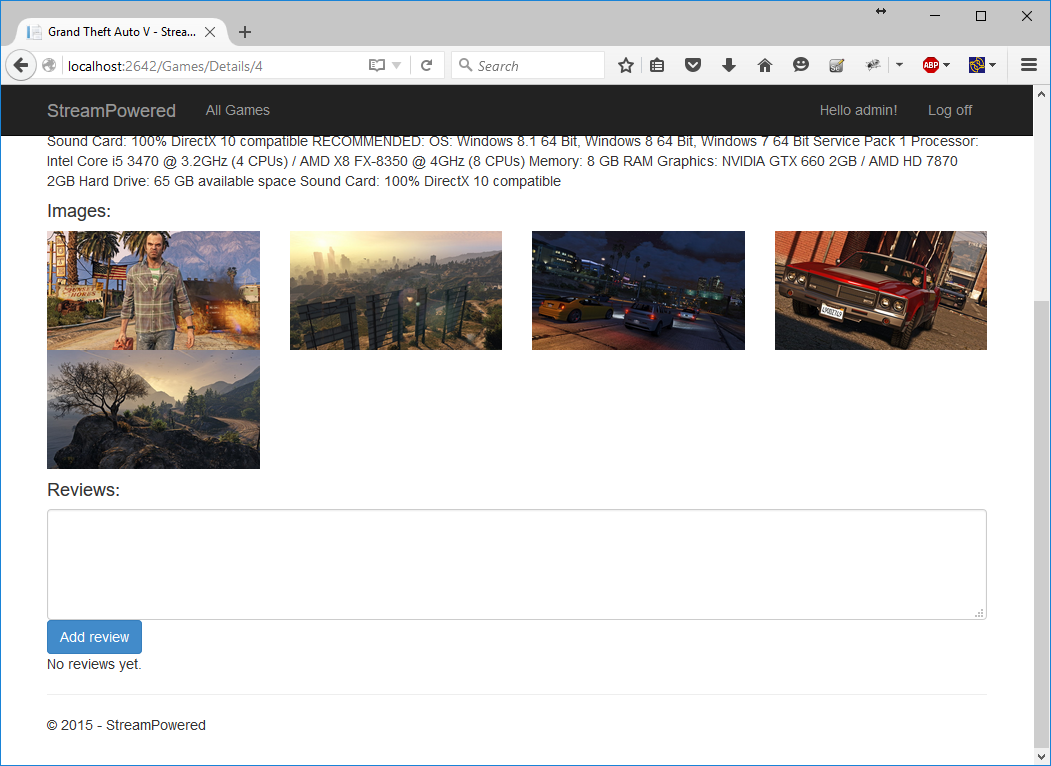
1 score

* Implement standard (non-AJAX) reviewing. Upon review submission, reload the page and display the new review where it belongs (it should be on top or near the top).

2 score

* Implement reviewing using an AJAX form. When the review has been submitted, show it at the top of all reviews (since it’s the newest now, and the newest reviews should be displayed at the top). Do not consider race conditions (there may be some new comments added after the current one, and you will miss displaying them).
* Use a CSRF token to prevent request forgery attacks.
* **Note:** To get the full possible score (including the bonus score) for this problem, you will not need to have the non-AJAX form.

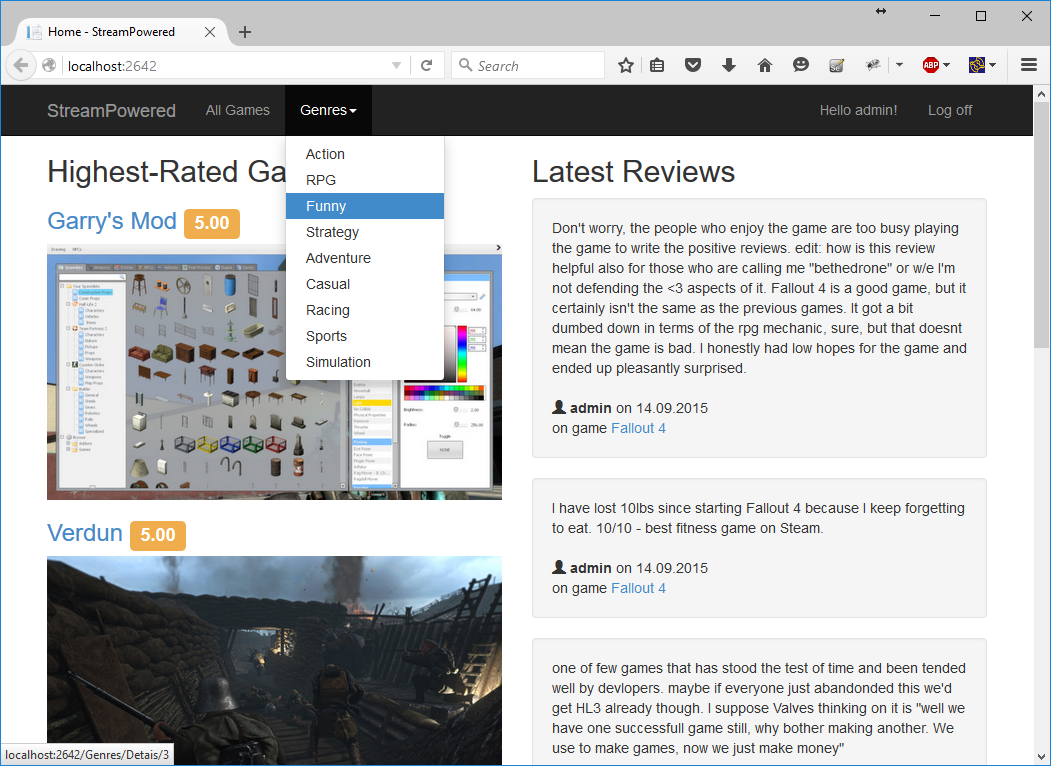
**BONUS:** 6 score



## Implement the Games in a Genre Page

Guests can view all games in a specified genre.

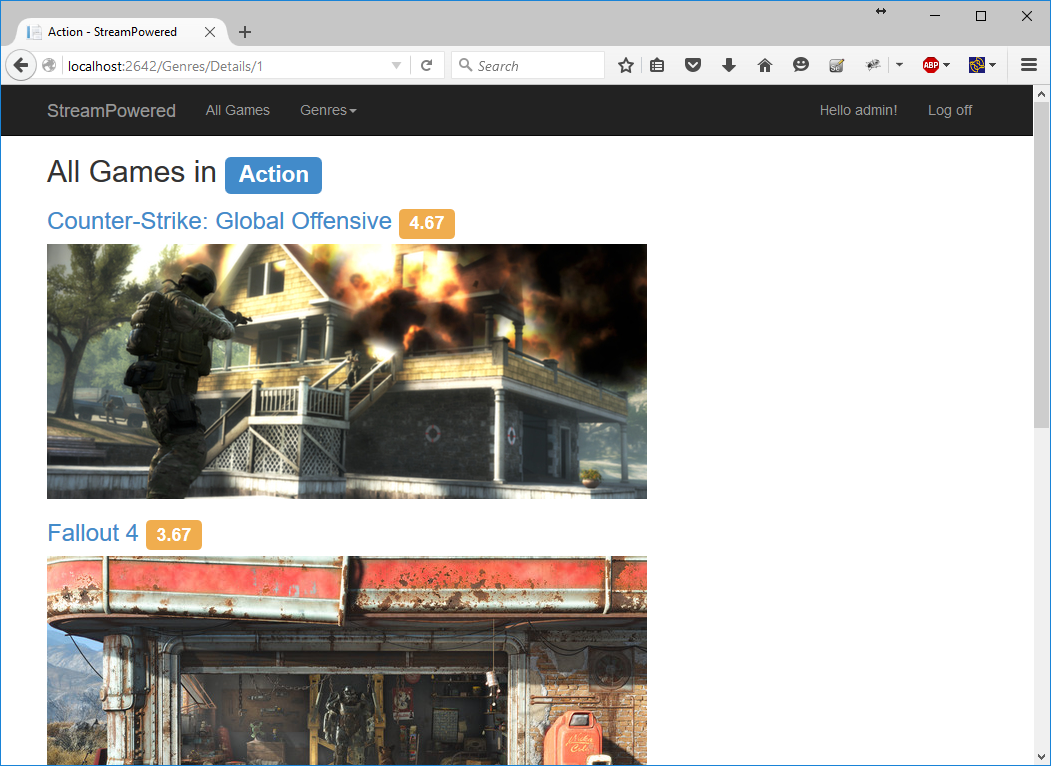
* Display a top menu containing all genres. Upon clicking a genre, the user can see all games in it.



3 score

* Display all games for the given genre. Display all games ordered alphabetically. You can omit paging if you like. The page must include the game titles, ratings and cover images. In case there are no images, display a placeholder image.

2 score



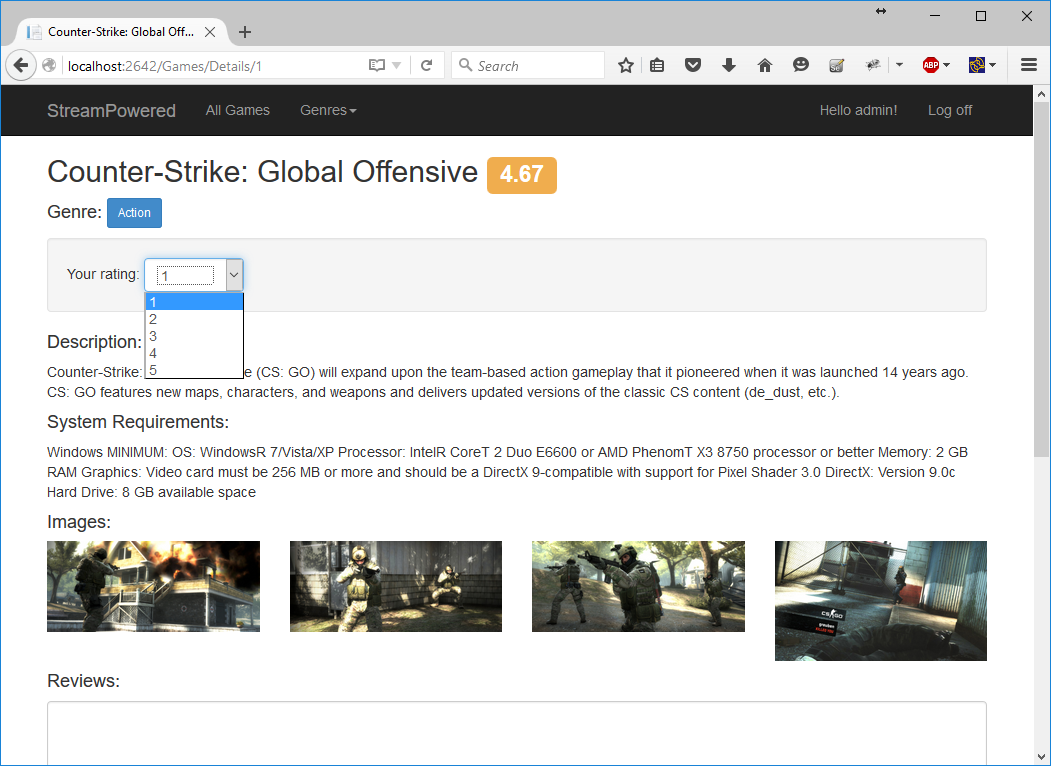
## Implement Rating Games

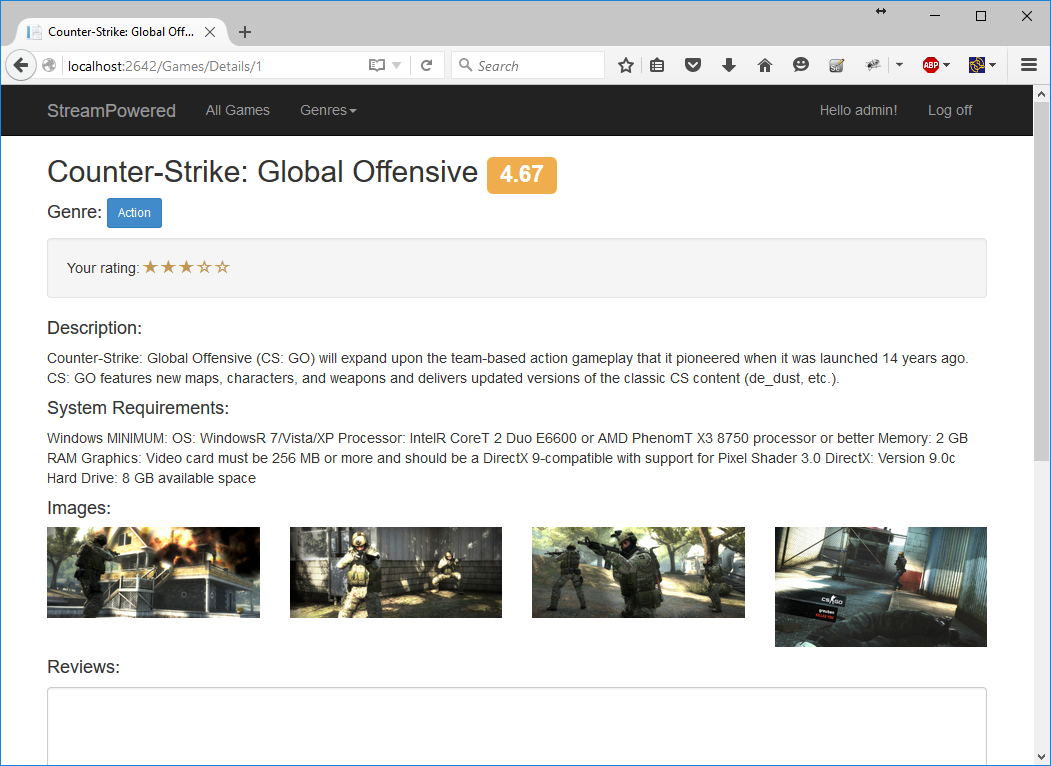
Logged-in users can rate any game.

* Display a rating component (it could be, for example, a dropdown or five stars, or whatever). You are not allowed to use any external components.

**UI BONUS:** 5 score

* Implement non-AJAX rating: display a button and reload the page after the user has clicked the button.
* After the user has rated a game, display his / her current rating and **do not allow** second rating.





10 score

* Implement AJAX rating – when the user clicks the Rate button, display his / her current rating and do not allow second rating. Do not refresh the page.
* When receiving a new rating, update the average game rating too.
* Use a CSRF token to prevent request forgery attacks.

**BONUS:** 6 score

## Implement the New Game Page

Administrators can create new games.

**Note:** You are not obliged to create an admin area for this page but feel free to do it if you wish.

* If the currently logged in user is administrator, display a link to create a new game. Non-administrators should not see this link. Even if they know the link and copy it, they should not be able to create a new game.
* A new game has a title, description, system requirements, genre and images (optional).

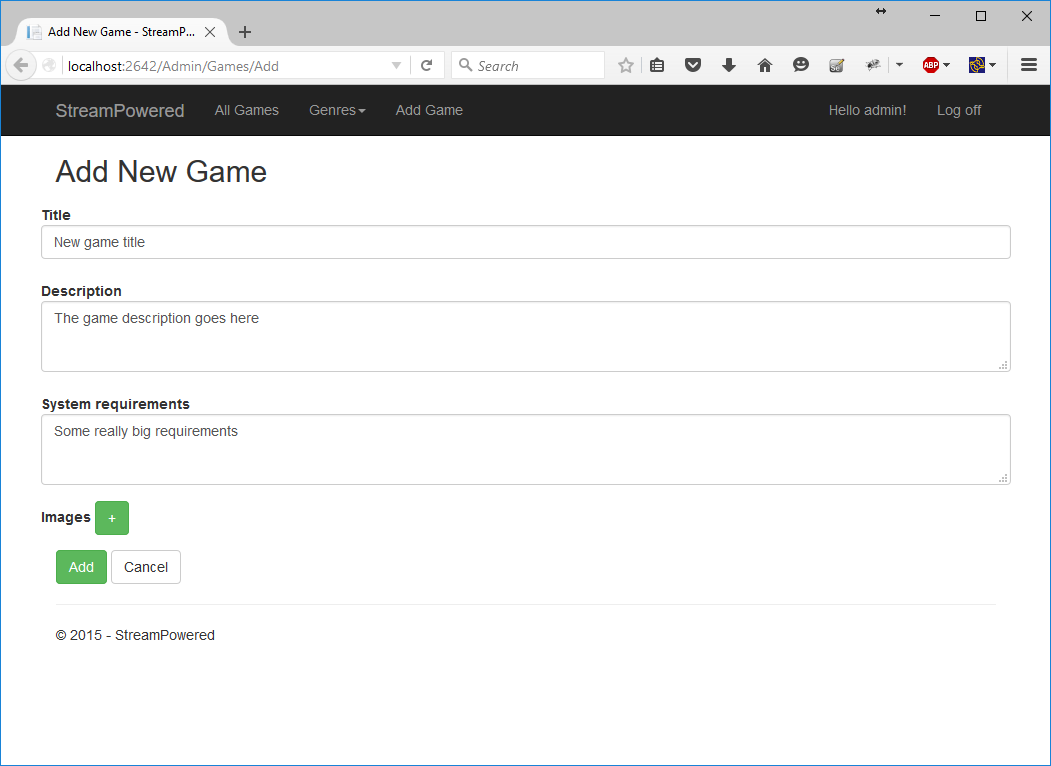
5 score

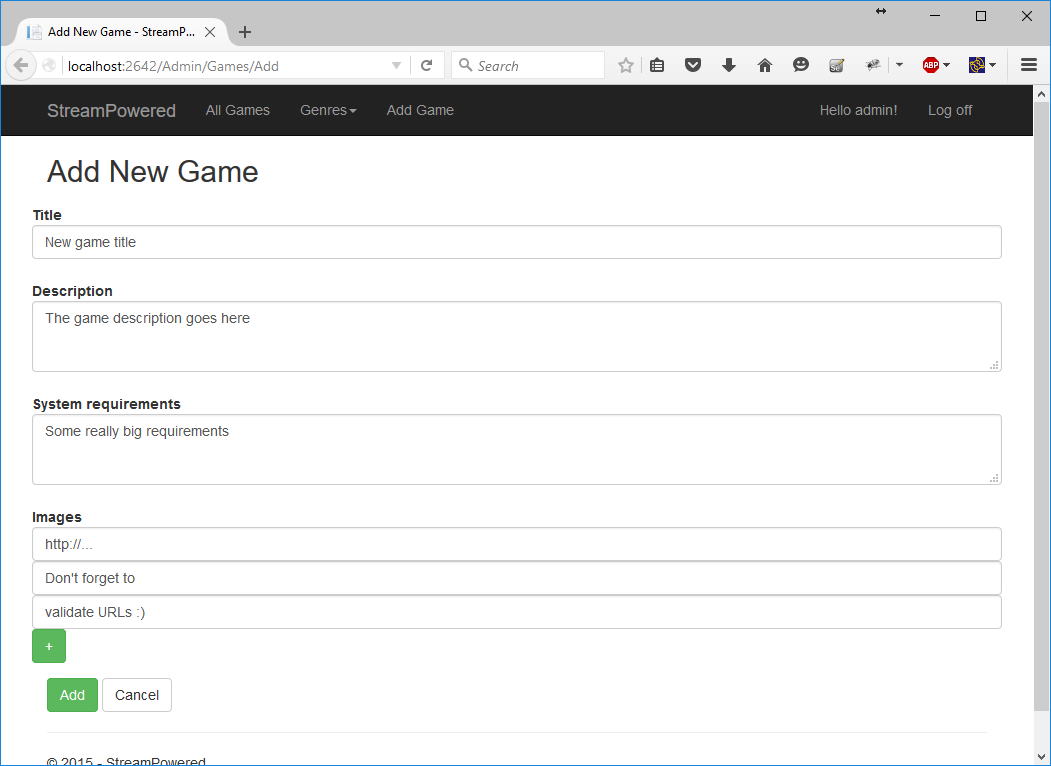
* Display an "Add image" button. An image for a game is defined via its URL. After an image has been added, display a button to add another one.
* Display a dropdown for the existing genres.

2 score

* The administrator who is creating the game should be able to add as many images as he / she wants.

7 score





* When the new game is created successfully, the application should display the newly created game.

2 score

## Project Infrastructure Bonus

Bonus points code quality / good application structure / additional effort.

* Bonus points for implementing **separate data layer**.
* Bonus points for using **dependency inversion**.
* Bonus points for using **AutoMapper**.
* Bonus points for using **services** for different jobs.

**BONUS:** 5 score