Xbox Game Voting Application

Scope of Work Details

Application Description

The Nerdery needs an application to track the interest in new games for our Xbox 360. This application will display the games we currently own, the games we would like to buy, and the number of votes for each game. Each employee at The Nerdery should be able to vote for their favorite game or add a new game to this list once per day. At the end of the week, if we reach our productivity goals the game with the most votes will be purchased and marked as a game we currently own.

A web service exists to store data with the game and vote information. A user interface will be developed to display the games we own, display games we are voting on, and allow users to submit new titles for voting, and designate a game as owned.

The application must run on all modern browsers: Chrome latest, Safari latest, Firefox latest, IE 7, 8, and 9.

End User Description

The end users of this application will be the programming staff at The Nerdery. It can be assumed that the programmers will only vote for their favorite game from their workstation on the LAN at The Nerdery headquarters, so setting a local flag to determine whether or not the user has voted is one acceptable solution.

Web Application Requirements

The details for this web application are defined below.

Single Page Architecture

Application content should be presented with no page refreshes, and should not require navigation to separate pages.

Display Games

The user interface portion of the application will retrieve the games data via the JSON web service defined in the *Web Service Specification* section below. Games will be separated into those that we currently want, and those than we own. The games that we want will display the current vote count, and must be sorted by this vote count in descending order. The games that we own must be sorted alphabetically.

Vote for a Game

The user interface will provide the user with an ability to vote for any Xbox game that The Nerdery does not currently own, based on the restrictions defined below.

Add New Title/Game

If an Xbox game is not currently in the list of games we want or own the user will have the ability to add that game to the list by providing the game's title. There will be no validation that the title is an actual Xbox game. Application must enforce that there are no duplicate titles.

Voting and Add New Title Restrictions

Users must only be allowed only one action to submit one vote or add one new game title per day (12am to 12am), not both. Additionally, voting or adding a new game title must be prohibited on Saturday and Sunday. Using the date/time from the user's local system is acceptable for this purpose.

Marking a Game as Owned

A separate section will exist to allow a user to indicate a game that we want is now owned. There will be no restrictions as to how many games can be owned.

Code Expectations

The developer is expected to adhere to the following parameters. The code challenge will be scored based on these parameters:

Well Structured

The code should be well structured, maintainable, and should follow best practices.

Well Documented

The application code will be created with the assumption that other developers will work on it in the future, so it must be documented enough to make it easy for others to understand and maintain.

Error Handling

All application errors must be handled gracefully and display user-friendly error messages if necessary.

HTML Coding Style

The quality of the site design will not be judged, but the HTML must be semantic and valid.

Static HTML

Project should be static HTML, CSS and Javascript. Project may run on a web server but cannot have any server-side requirements, such as PHP scripts, .Net projects, Node.js, etc.

Cross Domain Requests

Implementation of a proxy script is not permitted, and all requests are intended to be performed cross-domain from Javascript web requests. The server does not include support for Cross-origin Resource Sharing.

Bonus Features

It is expected that developers build the minimum implementation of the project, in addition, it is required to select one or more of the bonus features below.

Since the features below may not be supported by all browsers, bonus features only need to be implemented in browsers that support them.

Cache Game Data

Users may want their websites to load immediately after page access. Implement a limited cached session solution for storing the game data. This feature should store any data structures that may be used to impact the view. Data should be loaded prior to the first data request from the challenge server.

Mobile Support

Some users may want to use this application on their mobile devices. Implement a user interface that is optimized for mobile users. Interfaces on these devices should support the mobile interaction events, and should not require horizontal scrolling or zooming.

The interface should support iOS or Android phones. Mobile tablets such as iPads or the Xoom are not a requirement.

Web Workers

The developers have been very interested in optimizing our web application's processing speed. In order to push AJAX processing and requests off the main thread, implement all request structures using threads via web workers.

Accessibility

A vision-impaired user has complained that the site is not friendly to screen readers.

Ensure the application is compliant with all ARIA standards. The site should be readable using a common screenreader such as JAWS.

Templating

In order to maintain best practices for separating out view logic from the business logic layer, implement the application using Javascript-based HTML templates.

Render the application view layer by making use of a templating engine. Any template library may be used for this feature.

History / Deep Linking

Because the application does not allow page refreshes between view states, developers want to use the back button to move between sections of the application.

Implement history tracking / push state so the browser back button and URL hash are integrated in the application without requiring page refreshes. This will require that the application presentation layer have on-page navigation through some sort of display view such as a tabbed interface or accordion.

Progress Indicator Animation

Users have requested to have an interesting, small logo animation indicator at the top of the page that will animate based on requests. The animation has little definition, but should be more than a linear fill and must be done using canvas.

Any XBox 360 image may be used for this progress indicator.

Web Service Specifications

The provided web service is developed using the REST standard protocol and implemented using Remote Procedure Calls. Data returned is provided as JSON via JSONP.

Calling a URL endpoint that does not exist should result in an error message.

Service Definition

Please use this document component to obtain service information. An API Key should have been supplied to you.

All web services are RESTful APIs and expect a GET request. The service methods should all be called from http://js.nrd.mn/challenge/.

So an example would be: http://js.nrd.mn/challenge/checkKey?callback=example&apiKey=invalidApiKey.

Check Key

This method checks to verify that the `apiKey` provided is a valid key

URL Endpoint:

`/checkKey`

Input Parameters:

`callback` – Javascript callback method.

`apiKey` – The unique identifier required to access all services.

Output:

TRUE on valid `apiKey`, FALSE on invalid `apiKey`.

Get Games

This method is used to retrieve a list of all games and the number of votes for each. This procedure will return an array of game objects. A game object contains the id, title, status and votes for the game.

URL Endpoint:

`/getGames`

Input Parameters:

`callback` – Javascript callback method.

`apiKey` – The unique identifier required to access all services.

Output:

Array of Game Objects on success, FALSE on invalid `apiKey`.

Example `getgames` response:

```
[
    {
        // Game ID
        "id": "123456",

        // Game Title
        "title": "Mega Man",

        // Number of votes
        "votes": "0",

        // Status for owned state, “wantit” for unowned, “gotit” for owned
        "status": "wantit"
    }
]
```

FALSE on invalid `apiKey`.

Add Vote

This method is used to increment the vote counter for a specific game. The service does not provide any restrictions on how many votes can be added for a title.

URL Endpoint:

`/addVote`

Input Parameters:

`callback` – Javascript callback method.

`apiKey` – The unique identifier required to access all services.

`id` – The integer unique identifier for the game.

Output:

TRUE on success, FALSE on invalid `id` or `apiKey`.

Add New Game

This method is used to add a new game title to the vote list. There are no restrictions on how many titles can be added. The service will add the first vote for the title upon being added. The service will provide no sanitization of input.

URL Endpoint:

`/addGame`

Input Parameters:

`callback` – Javascript callback method.

`apiKey` – The unique identifier required to access all services.

`title` – The name of the game to be added

Output:

TRUE on success, FALSE on invalid apiKey.

Set GotIt

This method is used to set the status of a game to “gotit”.

URL Endpoint:

`/setGotIt`

Input Parameters:

`callback` – Javascript callback method.

`apiKey` – The unique identifier required to access all services.

`id` – The integer unique identifier for the game.

Output:

TRUE on success, FALSE on invalid id or apiKey.

Clear Games

This method is used to clear all games and votes.

URL Endpoint:

`/clearGames`

Input Parameters:

`callback` – Javascript callback method.

`apiKey` – The unique identifier required to access all services.

Output:

TRUE on success, FALSE on invalid apiKey