Nerdery Assessment Test

PHONE (877) 664.6373
FAX (952) 948.1611
EMAIL info@nerdery.com
WEB http://nerdery.com

JANUARY 18, 2013 / PHP NAT VERSION 4.3

NAT SPECIFICATION
PHP

DOCUMENT OBJECTIVE

The purpose of this document is to provide detailed documentation that clearly defines the application that an applicant will create for the Nerdery Assessment Test.

NOTES TO CHALLENGER

The Nerdery Assessment Test (NAT) is a chance for you to show off your development capabilities and prove your experience. While this is a relatively simple web application, we expect you, the applicant, to think of the end result as a finished project that we would eventually launch for internal use. It should be maintainable, and reasonably free from errors and defects. We don't expect a high-design, high-security/high-performance driven application.

You should spend upwards of 10 hours on this challenge. We understand that this is typically not enough time to develop a fully robust application. You may feel free to spend additional time on it if you wish, but just know that it's not necessary to do so as long as you meet the spec's requirements. Due to the time goal and simplicity constraints, if you make a development decision that goes against what you would normally perform, we encourage you to include notes with your submission.

If you have any questions regarding the spec, please feel free to email your candidate advocate or the developer assigned to you for questions. We will be able to answer general questions about the challenge or make clarifications about the spec, but cannot offer any specific advice for completing it.

JANUARY 18, 2013 / PHP NAT VERSION 4.3

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.

END USER DESCRIPTION

The end users of this application will be the programming staff at The Nerdery. The majority of these users prefer Google Chrome release channel for a web browser and are running Windows 7 Professional for an Operating System. 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 cookie to determine whether or not the user has voted is an acceptable low tech solution.

WEB APPLICATION REQUIREMENTS

The details for this web application are defined below.

DISPLAY GAMES

The user interface portion of the application will retrieve the games data via the SOAP web service defined in the Web Service Specification section below. Games will be separated into those that we currently want, and those that 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.

The list of games and current votes must not be cached on the web server, and therefore must always use current data from the Web Service.

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 but to keep the list clean duplicate titles will not be allowed.

VOTING AND ADD NEW TITLE RESTRICTIONS

Users must only be allowed to submit one vote **or** add one new game title per day (midnight to midnight). Additionally, voting or adding a new game title must be prohibited on Saturday and Sunday.

MARKING A GAME AS OWNED

A separate page 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 guidelines:

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 readable and commented enough to make it easy for others to understand and maintain.

FRAMEWORK AND TOOLSET

The application code may pull in any third party libraries or tools, open or closed source, as needed, as long as they do not require a paid license.

PHP CODING STYLE

The application code must be written in PHP 5 and be E_STRICT-compliant. Code style should be consistent throughout the application. Code should adhere to modern PHP programming practices and style conventions.

OBJECT ORIENTED

The code produced is expected to follow industry accepted object oriented principles and patterns.

ERROR HANDLING

All application errors must be handled gracefully and display user-friendly error messages if necessary. Submission should avoid common web security vulnerabilities and exploits.

HTML CODING STYLE

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

WEB SERVICE SPECIFICATIONS

The provided web service is developed using the SOAP standard protocol and implemented using Remote Procedure Calls.

SERVICE DEFINITION

Web Service Definition:

http://xbox.sierrabravo.net/v2/xbox.wsdl

Web Service URL:

http://xbox.sierrabravo.net/v2/xbox.php

CHECK KEY

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

Method:

checkKey

Input Parameters:
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.

Method:

getGames

Input Parameters:
apiKey – The unique identifier required to access all services.

Output:
Array of game objects on success, 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.

Method:

addVote

Input Parameters:
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.

Method:

addGame

Input Parameters:
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 GOT IT

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

Method:

setGotIt

Input Parameters:
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.

Method:

clearGames

Input Parameters:
apiKey – The unique identifier required to access all services.

Output:
TRUE on success, FALSE on invalid apiKey