WoW Item API Test Plan

Blizzard Test Engineer Interview

|  |  |
| --- | --- |
| Full Name | Luke Alexander Donne Tomlinson |
| Email Address | luke.ad.tomlinson@gmail.com |
| Date | 22/06/2015 |

# Scope

In this section, you should identify what testing will be covered by this test plan, what will be covered elsewhere, and what will not be tested. Please indicate why you have scoped the test plan as you did. After reading this section, readers should be able to clearly indicate the components for which the test engineer will be responsible.

This test plan is being created to cover the functionality and implementation of the WoW Item Web API. It will test all key functionality of the API, as well as the accuracy of the results returned from its internal database. It will also test some non-functional areas, including performance and reliability.

This plan approaches the API from a user's point of view, without any reference point for design specifications for the API. It is a black-box plan and will only cover functionality available to the end user. This test plan will not cover server-side functionality, for example scalability with increased users, data storage, API usage monitoring and authentication tests beyond checking a single valid key and invalid keys. This plan also does not extend beyond the Item and Item Set APIs, and any testing done will be limited to those APIs only.

The functionality of the Web API to be tested can be described as follows:

1. Item API

* A user with a correct API Key can send a request to the API server to obtain information on an item in WoW. These http API requests are sent with an item identifier ":itemId", as well as a language parameter "locale" and the API Key parameter "apikey". Optionally, a request can specify a JSONP callback function name, which will wrap the returned data in a callback function.
* Correctly-formed requests will return a 200 OK response and a JSON blob describing the requested item, including such information as the item name, stats, description, icon information and so on. Included in this is also item set information, relating each item to an item set by the item set ID.
* Incorrect requests can happen if one or more of the following is true:
  + The API Key is invalid or not provided
    - The API should respond with a 403 Forbidden
    - The response body should include the JSON:

{"code":"403", "type":"Forbidden", "detail":"Account Inactive"} Note: this "detail" is the same whether an invalid key is provided or no key is provided - might be more clear to differentiate the two cases. "Account Inactive" is also misleading as a reason for the failure.

* + The locale is not one from the pre-existing list. Requests still work and default to english (en\_US).
  + The requested item number does not refer to an existing item
    - The API should respond with a 404 Not Found
    - The response body should include the JSON:

{"status":"nok", "reason":"unable to get item information"} Note: this is inconsistent with the above JSON response, and also doesn't give a reason for the failure other than the 404

* + The itemId parameter is not valid input
    - The API should respond with a 404 Not Found
    - The response body should include the JSON:

{"status":"nok", "reason":"When in doubt, blow it up. (page not found)"}

2. Item Set API

* A user with a correct API Key can send a request to the API server to obtain information on an item set in WoW. These http API requests are sent with an item set identifier ":setId", as well as a language parameter "locale" and the API Key parameter "apikey". Optionally, a request can specify a JSONP callback function name, which will wrap the returned data in a callback function.
* Correctly-formed requests will return a 200 OK response and a JSON blob describing the requested item set, including the item set name, buff data and the item set information, relating each item set to the items in the set by their item ID.
* Incorrect requests can happen if one or more of the following is true:
  + The API Key is invalid or not provided
    - The API should respond with a 403 Forbidden
    - The response body should include the JSON:

{"code":"403", "type":"Forbidden", "detail":"Account Inactive"} Note: this "detail" is the same whether an invalid key is provided or no key is provided - might be more clear to differentiate the two cases. "Account Inactive" is also misleading as a reason for the failure.

* + The locale is not one from the pre-existing list. Requests still work and default to english (en\_US).
  + The requested item set number does not refer to an existing item
    - The API should respond with a 404 Not Found
    - The response body should include the JSON:

{"status":"nok", "reason":"unable to get item set information"} Note: this is inconsistent with the above JSON response, and also doesn't give a reason for the failure other than the 404

* + The setId parameter is not valid input
    - The API should respond with a 404 Not Found
    - The response body should include the JSON:

{"status":"nok", "reason":"When in doubt, blow it up. (page not found)"}

3. Account limits

* Accounts are rate-limited on requests to 100 per second or 36000 per hour. The API will function normally for requests below this limit, and when exceeding this limit it will respond with a 403 error:
  + '{"code":403, "type":"Forbidden", "detail":"Account Over Queries Per Second Limit"}'
* This limit is tracked inside the response headers using the X-Plan-Qps-Allotted and x-Plan-Qps-Current headers for the per second limits, and the X-Plan-Quota-Allotted and X-Plan-Quota-Current headers for the per-hour limits. It resets the hourly limit at the given time in the X-Plan-Quota-Reset header.

# Test Strategy

In this section, you should describe your approach of testing. Please include the types of tests to be performed, and base your strategy on project goals and assessed risk. After reading this section, readers should have a clear understanding of the types of tests to be performed.

Tests are divided into functional and non-functional tests, and subsequently into priority groups.

Functional testing is more closed-ended, with a reasonably small specification.

1. High Priority - required for the API to be used and secure:

* Correct input returns usable item and item set data. Given a valid ID and API Key we retrieve valid results from the API.
* Results returned from the API have the correct and expected values.
* Altering the locale parameter correctly changes the language of the returned data.
* Incorrect API Keys will return a 403 error instead of being accepted or giving an unexpected result.

2. Medium Priority - required for the API to be reasonable and user-friendly:

* Incorrect or invalid item or item set IDs will return a 404 error, instead of a different item or unexpected result.
* Incorrect locale values will default to en\_US.
* The jsonp parameter correctly wraps the data in a JavaScript function with the name given.

3. Low Priority - required for the API to be more manageable server-side but not directly impacting use:

* The API will reject requests over 100 per second with a 403 error.
* The API will reject requests over 36000 per hour with a 403 error.

Non-functional testing performed in this plan cover reliability and performance, and is more open-ended. This API must be able to serve a large number of users and respond correctly to each request.

1. High Priority

* API can handle two simultaneous requests without hanging or halting.
* API returns results within a reasonable time.

2. Medium Priority

* The API does not respond badly to malformed requests, for example returns 404 errors and the correct error message when passed a unicode string as the item ID, locale or JSONP parameters.

3. Low Priority

* The API can handle large numbers of requests without failing to respond to any of them.

# Entry and Exit Criteria

In this section, you should identify the criteria for test entry as well as the criteria that identify when to stop testing. Be as specific as possible. After reading this section, readers should be able to identify when the product is ready to be tested and when enough testing has been performed.

The Entry criteria for the product has already been met - the API is already at feature-complete status and deployed, and is not waiting for any more development. Tests can begin as soon as the test plan is complete and automated tests are created.

The Exit criteria for the test plan can be broken down into several stages. Once the High and Medium priority tests are complete, it will be possible to release the API for small-scale use, such as a beta test.

# Test Cases

In this section, you should include a list of test cases for each test type defined in the Test Strategy section. Test cases should include input values, conditions, and expected results. After reading this section, readers should know precisely what is to be tested and the expected results of each test case.

1. Item API
   1. General Note: These simple tests are designed to catch basic faults that block the remainder of the tests.
      1. (HIGH) Send an Item API request with a valid API Key, item ID and locale. The API must return information for that item.
      2. (HIGH) Send an Item API request and time its response. It must respond within a sensible amount of time.
      3. (HIGH) Send multiple Item API requests simultaneously. The API must respond correctly to all of them.
      4. (HIGH) Send a large number of requests from different sources. The API must respond correctly to all of them. Note: Your definition of "large" may vary - I don't know how many users this API is designed to serve.
   2. Item ID
      1. (MED) Send an Item API request with a numeric ID that does not refer to an existing item (but correct Key and locale). The API must return a 404 error, giving the reason as "unable to get item set information".
      2. (MED) Send an Item API request with an ID that is not numeric. It must return a 404 Not Found error giving the reason as " When in doubt, blow it up. (page not found)".
      3. (MED) Send Item API requests with as many different ID strings as possible (a fuzz test). It must respond to all of them with either correct information or the above 404 errors.
   3. Localization
      1. (MED) Send an Item API request with an omitted locale. It must default to en\_US and return data in English (US).
      2. (MED)Send an Item API request with an invalid locale. It must default to en\_US and return data in English (US).
      3. (HIGH) Send an Item API request with each valid locale. It must return data in the requested language. Check all valid locales (en\_US, es\_MX, pt\_BR).
   4. API Key
      1. (HIGH) Send an API request with a random string of characters for the API Key. It must not return a valid result, and instead return a 403 Forbidden error with "detail":"Account Inactive" in the JSON.
   5. JSONP
      1. (MED) Send an API request with the "jsonp" parameter set. Compare it to the same request without the "jsonp" parameter. Make sure the same information is received and the JSONP syntax is correct (wrapping the data in a callback JavaScript function).
      2. (MED) Send API requests with the JSONP parameter set to strings that are invalid for JavaScript functions. The API should not allow this and should either return an error or default to a non-JSONP response. Note: this test currently fails!
   6. Returned data
      1. For each item in the database: Note: this would take a long time, so perhaps might be easier to do without the 36000/hour restriction.
         1. (HIGH) Send an API request for the data for that item. Compare that data to the latest known-good information used in the WoW game. Every stat and property of the item must match exactly. Ideally, do this in all languages. Note: This is only automatable with a regularly-updated database of known-good information - without this, the best you can do is check that each value is of the correct type (integer, string, list, etc) and inside the range of valid values.
         2. (HIGH) If an item has the "itemSet" key in the returned item data, send an API request to the Item Set API with the Set ID from that data. The itemSet data for the item must match the retrieved item set data exactly.
   7. API Throttling
      1. (LOW) Send valid API requests at a rate of over 100 per second. Each second, we must have 100 correctly returned requests and any more than that must be responded to with a 403 Forbidden error and give the detail as "Account Over Queries Per Second Limit". The message headers must track the number of 200 OK responses in the X-Plan-Qps-Allotted and X-Plan-Qps-Current headers.
      2. (LOW) Send valid API requests at a rate of over 36000 per hour. Each hour, we must have 36000 correctly returned requests and any more than that must be responded to with a 403 Forbidden error and give the detail as "Account Over Queries Per Second Limit". The message headers must track the number of 200 OK responses in the X-Plan-Quota-Allotted and X-Plan-Quota-Current headers. Note: this test can take a long time to do!
      3. (LOW) Ensure that if the per-hour quota is exceeded, the quota is correctly reset at the given time in the X-Plan-Quota-Reset header.
2. Item Set API
   1. General Note: These simple tests are designed to catch basic faults that block the remainder of the tests.
      1. (HIGH) Send an Item Set API request with a valid API Key, set ID and locale. The API must return information for that item set.
      2. (HIGH) Send an Item Set API request and time its response. It must respond within a sensible amount of time.
      3. (HIGH) Send multiple Item Set API requests simultaneously. The API must respond correctly to all of them.
      4. (HIGH) Send a large number of requests from different sources. The API must respond correctly to all of them. Note: Your definition of "large" may vary - I don't know how many users this API is designed to serve.
   2. Set ID
      1. (MED) Send an Item Set API request with a numeric ID that does not refer to an existing item set (but correct Key and locale). The API must return a 404 error, giving the reason as "unable to get item set information".
      2. (MED) Send an Item Set API request with an ID that is not numeric. It must return a 404 Not Found error giving the reason as " When in doubt, blow it up. (page not found)".
      3. (MED) Send Item Set API requests with as many different ID strings as possible (a fuzz test). It must respond to all of them with either correct information or the above 404 errors.
   3. Localization
      1. (MED) Send an Item Set API request with an omitted locale. It must default to en\_US and return data in English (US).
      2. (MED)Send an Item Set API request with an invalid locale. It must default to en\_US and return data in English (US).
      3. (HIGH) Send an Item Set API request with each valid locale. It must return data in the requested language. Check all valid locales (en\_US, es\_MX, pt\_BR).
   4. API Key
      1. (HIGH) Send an Item Set API request with a random string of characters for the API Key. It must not return a valid result, and instead return a 403 Forbidden error with "detail":"Account Inactive" in the JSON.
   5. JSONP
      1. (MED) Send an Item Set API request with the "jsonp" parameter set. Compare it to the same request without the "jsonp" parameter. Make sure the same information is received and the JSONP syntax is correct (wrapping the data in a callback JavaScript function).
      2. (MED) Send API requests with the JSONP parameter set to strings that are invalid for JavaScript functions. The API should not allow this and should either return an error or default to a non-JSONP response. **Note: this test currently fails!**
   6. Returned data
      1. For each item set in the database: Note: this would take a long time, so perhaps might be easier to do without the 36000/hour restriction.
         1. (HIGH) Send an Item Set API request for the data for that item set. Compare that data to the latest known-good information used in the WoW game. Every stat and property of the item set must match exactly. Ideally, do this in all languages. Note: This is only automatable with a regularly-updated database of known-good information - without this, the best you can do is check that each value is of the correct type (integer, string, list, etc) and inside the range of valid values.
         2. (HIGH) For each of the items in the "items" list in the returned data, send an API request to the Item API with the Item ID from that data. The itemSet data for the item must match the item set data exactly.
         3. (MED) If an Item Set has threshold requirements, make sure that the number of items in the "items" list is greater than or equal to the highest threshold value.
   7. API Throttling
      1. (LOW) Send valid API requests at a rate of over 100 per second. Each second, we must have 100 correctly returned requests and any more than that must be responded to with a 403 Forbidden error and give the detail as "Account Over Queries Per Second Limit". The message headers must track the number of 200 OK responses in the X-Plan-Qps-Allotted and X-Plan-Qps-Current headers.
      2. (LOW) Send valid API requests at a rate of over 36000 per hour. Each hour, we must have 36000 correctly returned requests and any more than that must be responded to with a 403 Forbidden error and give the detail as "Account Over Queries Per Second Limit". The message headers must track the number of 200 OK responses in the X-Plan-Quota-Allotted and X-Plan-Quota-Current headers. Note: this test can take a long time to do!
      3. (LOW) Ensure that if the per-hour quota is exceeded, the quota is correctly reset at the given time in the X-Plan-Quota-Reset header.

# Dependencies

In this section, you should identify any significant constraints on testing as well as identify any tools on which your test cases will be dependent.

* At least one valid API Key is necessary for this plan.
* The example test cases require the Python Requests library.