Node.js Server Test Specification (WTS):

Version: 01.0

Date: 28/09/2019

# **How to use this document:**

* Write the requirement of the node server function based on a story.
* Describe this requirement and the specifics of how it is supposed to be implemented.
* Write tests to make sure this requirement is met and list their numbers in the relevant test numbers section of the requirements table.
* If carrying out a review then make sure you add this document and a description of which tests the pull request passed / failed into the pull request approval / comment.
* If this document is updated to account for new features the following should occur:
  + 1. Before anything else go to the review section of word and click “Track Changes”
    2. Increment the version number by 0.1
    3. Update the date to the latest one
    4. The document should be added to the commit for whatever changes you are making ()

# **Requirements:**

|  |  |  |
| --- | --- | --- |
| **Name:** | **Description:** | **Relevant Test Number(s):** |
| **Search using relevant query based on query code request using alldata** | **The server should return the correct query results as json based on the query code input** | **6 component tests**  **7 code tests**  **1 - 13** |
| **Search using relevant query based on query code request using new joining queries and data sets** | **The server should return the correct query results as json based on the query code input, using the new data and the joining queries as to largely speed query time** | **7 code tests**  **14 - 20** |
| **Search using relevant query based on input variables using new joining queries and data sets to ease caching** | **The server should return the correct query results as json based on the input parameters, this will allow far easier caching as it builds on top of the previous query changes and reduces the amount of queries done** | **10 code tests**  **21 -30** |

# **Tests:**

We used Mocha as our testing application, which allowed us to run all test simultaneously and constantly bring corrective error codes which were more useful than the .js errors.

Below is how to run the unit tests as a whole and the testing process

## Unit testing

1. To run all tests navigate to the router folder in the command terminal and run the command “npx mocha”
2. Verify all tests are passing (while only work while able to connect to silva database i.e in university verified wifi or connection

### Test Descriptions

### 1. Basic database connection test using extended file:

1. Get db.js object
2. **Verify** its not null

### 2. Check the install of express:

1. Get express component
2. **Verify** it’s not null

### 3. Check the install of router:

1. Get router component
2. **Verify** it’s not null

### 4. Check the install of msyql:

1. Get mysql component
2. **Verify** it’s not null

### 5. Check the install of cors:

1. Get cors component
2. **Verify** it’s not null

### 6. Check the app creation with express:

1. Use the express component to create an app
2. **Verify** it’s not null

### 7.Check database connection successful in Index:

1. Create a database connection using the create connection mysql function with database host, username, password and database name
2. Set connection to var
3. **Verify** connection state is connected

### 8. Index.Silva method should return the correct results:

1. Use the query object that is set to the indexForTest.js file
2. Call the silva() method and set it a variable “039”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 87th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 9. Index.sortpriceasc method should return the correct results:

1. Use the query object that is set to the indexForTest.js file
2. Call the sortpriceasc() method and set it a variable “039”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 50th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 10. Index.filterzipcode method should return the correct results:

1. Use the query object that is set to the indexForTest.js file
2. Call the filterzipcode() method and set it a variable “039” and “36301”
3. **Verify** the only record is correct by its unique providerid
4. **Verify** its length is one

### 11. Index.filterstate method should return the correct results:

1. Use the query object that is set to the indexForTest.js file
2. Call the filterstate() method and set it a variable “039” and “AL”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 10th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 12. Index.providerinfo method should return the correct results:

1. Use the query object that is set to the indexForTest.js file
2. Call the providerinfo() method and set it a variable “039”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 100th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 13. Index.princerange method should return the correct results:

1. Use the query object that is set to the indexForTest.js file
2. Call the pricerange() method and set it a variable “039”, 0 and 100000
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 78th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 14.Check database connection successful in IndexUpdated:

1. Create a database connection using the create connection mysql function with database host, username, password and database name
2. Set connection to var
3. **Verify** connection state is connected

### 15. IndexUpdated.Silva method should return the correct results:

1. Use the query object that is set to the indexUpdatedForTest.js file
2. Call the silva() method and set it a variable “039”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 87th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 16. IndexUpdated.sortpriceasc method should return the correct results:

1. Use the query object that is set to the indexUpdatedForTest.js file
2. Call the sortpriceasc() method and set it a variable “039”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 87th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 17. IndexUpdated.filterzipcode method should return the correct results:

1. Use the query object that is set to the indexUpdatedForTest.js file
2. Call the filterzipcode() method and set it a variable “039” and “36301”
3. **Verify** the only record is correct by its unique providerid
4. **Verify** its length is one

### 18. IndexUpdated.filterstate method should return the correct results:

1. Use the query object that is set to the indexUpdatedForTest.js file
2. Call the filterstate() method and set it a variable “039” and “AL”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 9th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 19. IndexUpdated.providerinfo method should return the correct results:

1. Use the query object that is set to the indexUpdatedForTest.js file
2. Call the providerinfo() method and set it a variable “039”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 422th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 20. Index.princerange method should return the correct results:

1. Use the query object that is set to the indexForTest.js file
2. Call the pricerange() method and set it a variable “039”, 0 and 100000
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 422th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 21.Check database connection successful in ComboQuery:

1. Create a database connection using the create connection mysql function with database host, username, password and database name
2. Set connection to var
3. **Verify** connection state is connected

### 22. ComboQuery should return the correct results when given only a code:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,null, null, null, null
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 87th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 23. ComboQuery should return the correct results when given only a code and min/max:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,0, 10000, null, null
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 87th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 24. ComboQuery should return the correct results when given only a code and zipcode:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,null, null, 36301, null
3. **Verify** the only record is correct by its unique providerid
4. **Verify** its length is correct

### 25. ComboQuery should return the correct results when given only a code and state:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,null, null, null, “AL”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 11th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 26. ComboQuery should return the correct results when given a code, min/max and state:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,0, 10000, null, “AL”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 11th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct

### 27. ComboQuery should return the correct results when given a code, min/max and zipcode:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,0, 10000, 36301, null
3. **Verify** the only record is correct by its unique providerid
4. **Verify** its length is correct

### 28. ComboQuery should return the correct results when given a code, state and zipcode:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,null, null, 36301, “AL”
3. **Verify** the only record is correct by its unique providerid
4. **Verify** its length is correct

### 29. ComboQuery should return the correct results when given a code, min/max, state and zipcode:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,0, 10000, 36301, “AL”
3. **Verify** the only record is correct by its unique providerid
4. **Verify** its length is correct

### 30. ComboQuery should return the correct results when given a code and “null” strings:

1. Use the query object that is set to the ComboQuery.js file
2. Call the comboQuery() method and set it a variable “039”,”null”, “null”, “null”, “null”
3. **Verify** it first record is correct by its unique providerid
4. **Verify** its 87th record is correct by its unique providerid
5. **Verify** its final record is correct by its unique
6. **Verify** its length is correct