TravelMatch

Unit Test Plan

Version 1.0

D.J. van den Brand (0772180)
S. He (0810831)
J.M.A.P. van Hoof (0778486)
G.C. Linders (0815449)
M.J.M. Muijsers (0805654)
G.H. Santegoeds (0890429)
L.D. Stooker (0819041)
J.W.J.H. Visser (0828234)

22nd June, 2015
Abstract

This document contains the unit test plan for the TravelMatch application. It describes the environment needed to perform the UT. When this environment is set up, all test cases must be executed according to their corresponding test procedures. This application is developed in the Software Engineering Project at Eindhoven University of Technology. This document complies with the ESA software engineering standard [1].
Contents

Document Status Sheet 3
  General ................................................. 3
  Document history ..................................... 3

Document Change Records 4
  General ................................................. 4
  Changes ................................................ 4

1 Introduction 5
  1.1 Purpose .......................................... 5
  1.2 Overview ......................................... 5
  1.3 Definitions and abbreviations
    1.3.1 Definitions ................................... 5
    1.3.2 List of abbreviations ......................... 7
  1.4 References ....................................... 7

2 Test Plan 9
  2.1 test items ....................................... 9
  2.2 Features to be tested ............................ 9
  2.3 Test deliverables ................................ 9
  2.4 Testing Tasks .................................... 9
  2.5 environmental needs
    2.5.1 Prerequisites for TravelMatch app ............ 10
    2.5.2 Build process ................................ 10
    2.5.3 Prerequisites for the back end ............... 10
    2.5.4 Set up testing environment ................... 10
  2.6 test case pass/fail criteria ...................... 11

3 Test case specification 12
  3.1 Back End ......................................... 12
    3.1.1 Tag ........................................... 12
    3.1.2 Location ...................................... 12
    3.1.3 ImageDimensions .............................. 12
    3.1.4 LocationTag .................................. 13
    3.1.5 TravelDNA ..................................... 13
    3.1.6 ImageTags .................................... 13
    3.1.7 TripOffer ..................................... 13
    3.1.8 ImageBlacklistItem ........................... 14
    3.1.9 LocationBlacklistItem ......................... 14
    3.1.10 RecSys ....................................... 14
    3.1.11 Entropy ....................................... 14
    3.1.12 UserModelMethod ................................ 15
    3.1.13 PasswordTests ................................ 15
  3.2 TravelMatch app ................................ 15
    3.2.1 AuthService .................................. 15
    3.2.2 HotelService .................................. 15
    3.2.3 HttpInjector .................................. 16
    3.2.4 ImageService .................................. 16
3.2.5 LoginService ................................................. 16
3.2.6 RegistrationService ...................................... 17
3.2.7 UserDetailService .................................... 17
3.2.8 VacationDetailService ................................... 18

4 Test procedures ........................................... 19
   4.1 TravelMatch app testing ............................ 19
   4.2 Back end testing .................................. 19

5 Test reports ............................................. 20
   5.1 back end ........................................ 20
   5.2 TravelMatch app ................................. 20
Document Status Sheet

General

Document title: Unit Test Plan
Document identifier: TravelMatch.Doc.SUM/1.0
Authors: D.J. van den Brand (0772180)
S. He (0810831)
J.M.A.P. van Hoof (0778486)
G.C. Linders (0815449)
M.J.M. Muijsers (0805654)
G.H. Santegoeds (0890429)
L.D. Stooker (0819041)
J.W.J.H. Visser (0828234)

Document status: Final document

Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>14-06-2015</td>
<td>L.D. Stooker</td>
<td>Initial setup.</td>
</tr>
<tr>
<td>0.2</td>
<td>15-06-2015</td>
<td>S.He, L.D. Stooker</td>
<td>Adding content.</td>
</tr>
<tr>
<td>1.0</td>
<td>22-06-2015</td>
<td>S.He, L.D. Stooker</td>
<td>Final version.</td>
</tr>
</tbody>
</table>
## Document Change Records

### General

<table>
<thead>
<tr>
<th>Date</th>
<th>22nd June, 2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document title</td>
<td>Unit Test Plan</td>
</tr>
<tr>
<td>Document identifier</td>
<td>TravelMatch.Doc.SUM/1.0</td>
</tr>
</tbody>
</table>

### Changes

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Section</th>
<th>Reason</th>
</tr>
</thead>
</table>
Chapter 1

Introduction

1.1 Purpose

This document is the Unit Test Plan (UTP) of the TravelMatch application. The purpose of this document is to describe the test plan for testing the developed software units against the detailed design, defined in the DDD. The unit test make sure that TravelMatch complies with the design.

1.2 Overview

Chapter 2 gives an overview of all items be tested and general criteria for the UT. Chapter 3 specifies how tests are defined, chapter 4 specifies the test procedures and chapter 5 specifies how test results are reported.

1.3 Definitions and abbreviations

1.3.1 Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affiliate Network</td>
<td>A network that enables you to receive money from customer redirection [18]</td>
</tr>
<tr>
<td>Analytics Data</td>
<td>The log of analytics events that is recorded and stored on the database.</td>
</tr>
<tr>
<td>Android</td>
<td>A popular open-source operating system for embedded devices, including smartphones and tablets, created by Google.</td>
</tr>
<tr>
<td>Angular JS</td>
<td>An open-source web application framework maintained by Google.</td>
</tr>
<tr>
<td>Cosine similarity</td>
<td>A measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them.</td>
</tr>
<tr>
<td>Destination advice</td>
<td>The city, and selection of hotels, that is advised to a user after performing one or more interest analyses.</td>
</tr>
<tr>
<td>Destination attributes or tags</td>
<td>Each destination will have one or more destination attributes with an associated numerical relative value, those attributes cover the same preferences as the DNA attribute.</td>
</tr>
<tr>
<td>DNA attribute or tags</td>
<td>These are the attributes that the client wants to use to compose the DNA of. In the beginning 10 attributes are chosen and each image shall have a relative numerical value on one or more of the attributes. Attributes can be added or removed later for new and existing images and DNA.</td>
</tr>
<tr>
<td>Google Play Store</td>
<td>A public repository of free and paid apps for Android, managed by Google.</td>
</tr>
<tr>
<td>Guest user</td>
<td>An user that does not provide login details but still uses the TravelMatch app.</td>
</tr>
<tr>
<td>Hotelstars rating</td>
<td>A hotel classification with common criteria and procedures in participating countries to rate a hotel's quality. See [21].</td>
</tr>
<tr>
<td>iLysian</td>
<td>Short for iLysian B.V., a software engineering company situated in Eindhoven, Netherlands. The client for the TravelMatch project.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Interest analysis</td>
<td>The action the user will do of judging the images.</td>
</tr>
<tr>
<td>iOS</td>
<td>A popular closed-source operating system for smartphones and tablets created by Apple.</td>
</tr>
<tr>
<td>iOS App Store</td>
<td>A public repository of free and paid apps for iOS, managed by Apple.</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token: a compact URL-safe means of representing claims to be transferred between two parties, and used in TravelMatch as authentication token, since it is self-validating.</td>
</tr>
<tr>
<td>Relational database management system (RDBMS)</td>
<td>A database management system (a piece of computer software that interacts with users, other applications and a database to capture and analyze data) based on the relational model (commonly based on the relational database model)</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>A computer networking model and set of communication protocols used on the internet and similar computer networks, including the Transmission Control Protocol (TCP) and the Internet Protocol (IP)</td>
</tr>
<tr>
<td>Tinder</td>
<td>A popular dating application for smartphones and tablets featuring a swipe based interface, where a swipe to the left indicates a dislike and a swipe to the right indicates a like.</td>
</tr>
<tr>
<td>Travel DNA</td>
<td>A collection of information about vacation preferences of a specific user or, more specifically, one vacation of that user. This information is stored on the server in a table with values representing the respective gain per attribute for each image the user has swiped.</td>
</tr>
<tr>
<td>TravelMatch</td>
<td>An application for smartphones and tablets that assists users in planning a vacation. The subject of this project.</td>
</tr>
<tr>
<td>TravelMatch team</td>
<td>A team of Computer Science students at Eindhoven University of Technology who will design and implement the TravelMatch application.</td>
</tr>
<tr>
<td>User</td>
<td>The user of the app.</td>
</tr>
<tr>
<td>Waverunner</td>
<td>Waverunner Search Service by Pyton Communication Services; a search service that provides vacation offers and prices of participating travel agencies.</td>
</tr>
</tbody>
</table>
1.3.2 List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>Architectural Design</td>
</tr>
<tr>
<td>ADD</td>
<td>Architectural Design Document</td>
</tr>
<tr>
<td>Android</td>
<td>Operating system of an Android device</td>
</tr>
<tr>
<td>AT</td>
<td>Acceptance Test</td>
</tr>
<tr>
<td>ATP</td>
<td>Acceptance Test Plan</td>
</tr>
<tr>
<td>DD</td>
<td>Detailed Design</td>
</tr>
<tr>
<td>DDD</td>
<td>Detailed Design Document</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>iOS</td>
<td>Operating system of Apple products</td>
</tr>
<tr>
<td>TU/e</td>
<td>Eindhoven University of Technology</td>
</tr>
<tr>
<td>SCMP</td>
<td>Software Configuration Management Plan</td>
</tr>
<tr>
<td>SEP</td>
<td>Software Engineering Project</td>
</tr>
<tr>
<td>SPMP</td>
<td>Software Project Management Plan</td>
</tr>
<tr>
<td>SQAP</td>
<td>Software Quality Assurance Plan</td>
</tr>
<tr>
<td>SR</td>
<td>Software Requirements</td>
</tr>
<tr>
<td>SRD</td>
<td>Software Requirements Document</td>
</tr>
<tr>
<td>STD</td>
<td>Software Transfer Document</td>
</tr>
<tr>
<td>SUM</td>
<td>Software User Manual</td>
</tr>
<tr>
<td>SVVP</td>
<td>Software Verification and Validation Plan</td>
</tr>
<tr>
<td>UR</td>
<td>User Requirements</td>
</tr>
<tr>
<td>UT</td>
<td>Unit test</td>
</tr>
<tr>
<td>UTP</td>
<td>Unit test plan</td>
</tr>
<tr>
<td>URD</td>
<td>User Requirements Document</td>
</tr>
</tbody>
</table>

1.4 References

[1] ESA PSS-05-0 Issue 2, Software requirements and architecture engineering process, February 1991


Chapter 2

Test Plan

2.1 test items
The software to be tested is the TravelMatch application and the TravelMatch back end. Information about the detailed design of TravelMatch can be found in the DDD.

2.2 Features to be tested
TravelMatch must meet the design as stated in the DDD. Each component should adhere to the interfaces given in the DDD.

2.3 Test deliverables
Before testing starts, the following documents must be delivered:

• SVVP [14]
• DDD [5]
• UTP (Current document)
• UT input data.

After completing the testing, the following documents must be delivered:

• UT report (Chapter 5)
• UT output data
• Problem reports (if any)

2.4 Testing Tasks
Before any testing, the following tasks need to be done

• Designing the unit tests
• All components mentioned in the ADD need to be covered by test cases
• Creation of the UT input data
• Ensuring that all environmental needs for the UT have been satisfied

When these tasks have been done, a UT can be performed according to the procedures described in chapter 4.

2.5 environmental needs
There are different environments for the TravelMatch app and the back end.
2.5.1 Prerequisites for TravelMatch app

The following are prerequisites for the testing environment of the TravelMatch application:

1. A computer connected to the Internet
2. NPM is installed.
3. Git is installed.

2.5.2 Build process

1. Clone the TravelMatch Git repository.
2. Open a console window with admin/superuser privileges and go to the src folder:
   - cd src
3. Create the output directory:
   - mkdir www
4. Use NPM to install gulp, Bower, Cordova and Ionic:
   - npm install gulp bower cordova ionic
5. Install karma-cli globally.
   - npm install karma-cli -g
6. Install all development dependencies:
   - npm install
7. Install all app dependencies:
   - gulp cook
8. Running the unit tests:
   - karma start

2.5.3 Prerequisites for the back end

1. Linux Ubuntu distribution.
2. Python is installed.

2.5.4 Set up testing environment

1. Install django via pip with following command:
   - sudo python get-pip.py
   - git clone git://github.com/django/django.git django-trunk
   - sudo pip install -e django-trunk/
   - sudo pip install djangorestframework
2. Install related django packages
   - sudo pip install django_facebook
3. Execute the test cases
   - python manage.py.test
2.6 test case pass/fail criteria

Every test should describe the criteria that should be met to pass a specific test. An overall UT pass can only be achieved when all tests described in chapter 3 have been performed and passed.
Chapter 3

Test case specification

3.1 Back End

Due to the technical difficulty, the SwipeImage object can only be manually tested, thus unit tests does not cover swipe image related functions.

3.1.1 Tag

Test target: Tag model

- It should check if creation of a new non-existing tag is possible.
- It should give IntegrityError if the new tag already exist in the database
- It should give IntegrityError if the new tag’s primary key (tag_id) is not unique in the database.
- It should store all the criteria from the new tag.
- It should be able to update the properties with the .save() function.

3.1.2 Location

Test target: Location model

- It should check if creation of a new non-existing location is possible.
- It should give IntegrityError if the new location already exist in the database
- It should give IntegrityError if the new location’s primary key (locid) is not unique in the database.
- It should store all the criteria from the new location.
- It should be able to update the properties with the .save() function.

3.1.3 ImageDimensions

Test target: ImageDimensions model

- It should check if creation of a new location with a unique super key (width and height) is possible.
- It should give IntegrityError if the new ImageDimension’s super key (width and height) is not unique in the database.
- It should be able to perform .to_s() function that returns the dimensions in a string format.
- It should store all the criteria from the new ImageDimensions.
- It should be able to update the properties with the .save() function.
3.1.4 LocationTag

Test target: LocationTag model

- It should check if creation of a new LocationTag with a unique super key (tag_id and loc_id) is possible.
- It should give IntegrityError if the new ImageDimension’s super key (tag_id and loc_id) is not unique in the database.
- It should be able to update the values of existing LocationTag by the .put() method.
- It should store all the criteria from the new LocationTag.
- It should be able to update the properties with the .save() function.

3.1.5 TravelDNA

Test target: TravelDNA model

- It should check if creation of a new TravelDNA with a unique super key (img and vacation) is possible.
- It should give IntegrityError if the new ImageDimension’s super key (img and vacation) is not unique in the database.
- It should store all the criteria from the new TravelDNA.
- It should be able to update the properties with the .save() function.

3.1.6 ImageTags

Test target: ImageTag model

- It should check if creation of a new ImageTag with a unique super key is possible.
- It should give IntegrityError if the new ImageTag’s super key (img and vacation) is not unique in the database.
- It should store all the criteria from the new ImageTag.
- It should be able to update the properties with the .save() function.

3.1.7 TripOffer

Test target: TripOffer model

- It should check if creation of a new TripOffer with a unique primary key is possible.
- It should give IntegrityError if the new TripOffer primary key is not unique in the database.
- It should store all the criteria from the new TripOffer.
- It should be able to update the properties with the .save() function.
3.1.8 ImageBlacklistItem

Test target: ImageBlacklistItem model

- It should check if creation of a new ImageBlacklistItem with a super key (img and vac) key is possible.
- It should give IntegrityError if the new ImageBlacklistItem super key (img and vac) is not unique in the database.
- It should store all the criteria from the new ImageBlacklistItem.
- It should be able to update the properties with the .save() function.

3.1.9 LocationBlacklistItem

Test target: LocationBlacklistItem model

- It should check if creation of a new LocationBlacklistItem with a super key (loc and vac) key is possible.
- It should give IntegrityError if the new LocationBlacklistItem super key (loc and vac) is not unique in the database.
- It should store all the criteria from the new LocationBlacklistItem.
- It should be able to update the properties with the .save() function.

3.1.10 RecSys

Test target: Functions from recommender system.

- It should pass if the recsys.cosine_similarity function gives correct mathematical answer.
- It should not pass if the cosine_similarity function gives incorrect mathematical answer.
- It should pass if the get_best_match function returns the city that matches inputing tag data according to the mathematical property of the cosine similarity method.
- It should pass if the get_match_bl returns the correct result, fail otherwise.

3.1.11 Entropy

Test target: Entropy related functions.

- It should always pass if .get_best_tags_on_total_score returns the correct total score for given a dictionary of a tags.
- It should pass if entropy.get_first_n_random it should return given number of images, and images are not from the ImageBlacklistItem model.
- It should pass if entropy.get_best_tags_on_priority functions returns the tag in the test data with the lowest score xor the best priority.
- It should pass if entropy.get_images_sorted_on_best_tag_val returns the set of correct images with maximum tag_value in the given tag list.
- It should pass if entropy.get_best_image_on_max_sum_values returns the max sum value of the give tag lists if they have different sum value.
- It should pass if `entropy.get_best_image_on_max_sum_values` returns the max sum value of the
given tag lists if they have same sum value.

- It should pass if `entropy.get_best_image` should give the best images according the entropy
calculation if there is only 1 image as input.

- It should pass if `entropy.get_best_image` should give the best images according the entropy
calculation if there are only multiple images as input, and the result is not in ImageBlacklistItem.

### 3.1.12 UserModelMethod

Test target: AppUser model

- It should check if creation of a new non-existing tag is possible.
- It should be able to give correct JSON web token.

### 3.1.13 PasswordTests

Test target: encode and verify function for the password.

- It checks when you have the correct plain password entered, verify function will return `True`.
- It checks when you have the incorrect plain password entered, verify function will return `False`.

### 3.2 TravelMatch app

The TravelMap application test cases are grouped per Service with functions specified in the DDD. The
test cases are self explanatory and can be easily read in the repository. (src/test) Below the
descriptions of the test cases are given, when a test case fails this description is shown as identifier for
the test case together with the Service name.

#### 3.2.1 AuthService

- it should check if user info exists and a user can logout
- it should set and get user info
- it it should remove user info
- it should check if a user is authenticated and checks for an existing user

#### 3.2.2 HotelService

- it gets new recommendations with all fields valid
- it fails to get new recommendations
- it fails to find new recommendations
- it gets new trip offers with all fields valid
- it saves a recommendation
- it checks for the length of the recommendation
- it check setRecommendation for precondition
- it saves a recommendation but fails in the back end
• it saves a recommendation but cannot find in the back end
• it loads a recommendation
• it fails to load a recommendation
• it cannot find the recommendation
• it deletes a recommendation
• it deletes a recommendation but returns an error
• it deletes a recommendation but cannot find
• it deletes a recommendation but fail precondition
• it validates offers fail precondition and different fields

3.2.3 HttpInjector
• it should add the Content-Type header
• it should add and remove the Authorization header
• it should base64-encode POST strings
• it should catch HTTP 0
• it should catch HTTP 401
• it should catch HTTP 500
• it should catch unknown HTTP errors

3.2.4 ImageService
• it should return an empty array
• it should remove all images
• it should return one image
• it should return two images
• it should return two images and remove the other two
• it should resolve
• it should error out

3.2.5 LoginService
• it processes 204 result
• it processes e-mail and facebook
• it aborts if already logged in
• it aborts if credentials are wrong or missing
• it processes 401 result
• it processes 403 result
• it processes 404 result
• it processes no server response
• it processes unexpected errors

### 3.2.6 RegistrationService

• it processes 201 result
• it processes 404 result
• it processes 409 result
• it processes no server response
• it processes internal server error
• it processes unknown errors
• it aborts if credentials are wrong or missing
• it fails FBRegister by input
• it fails FBRegister by API return of 403
• it fails FBRegister by 400 result
• it fails FBRegister by wrong API return
• it fills in the last email

### 3.2.7 UserDetailService

• it GET 201 with all fields valid
• it GET 201 with missing or invalid fields
• it GET 201 and sanitize user info
• it PUT 204 with all fields valid
• it PUT 204 with missing fields
• it PUT 204 with invalid fields
• it GET/PUT 500
• it PUT 412
• it GET/PUT 401
• it GET/PUT server offline
• it GET/PUT Unknown error
• it fails preconditions for put
3.2.8 VacationDetailService

- it gets vacations and the latest vacation from the back end
- it gets vacations and fails getting latest vacations
- it fails to get latest vacations
- it clears cached vacations
- it sets Vacation to anything and finds empty vacations
- it creates a new vacation in the back end
- it gets vacation, but the api returns empty
- it trying to send an empty object to api
- it does an empty get on vacations for both calls
- it fails to create a new vacation in the back end
- it fails to create a new vacation in the back end by conflict
- it updates an existing vacation
- it fails to update an existing vacation
- it deletes an existing vacation
- it gets the ID of the latest vacation from the back end
- it fails to get latests vacations because it does not exist
- it fails to get latests vacations default
Chapter 4

Test procedures

The author responsible for the class is also responsible for the testcases. This means tests should be written, specified and run, possibly several times if bugs are found. Only if all tests in a branch pass, we can merge it with the development branch.

4.1 TravelMatch app testing

**Test procedure identifier**  TravelMatch app

**Purpose**  all TravelMatch application functionality off AngularJS

**Procedure steps**  After the build procedure, be in the working directory and fire the following command from the command line:

- `karma start`

4.2 Back end testing

**Test procedure identifier**  Back end

**Purpose**  All python scripts

**Procedure steps**  Execute the build procedure to execute the test cases.
Chapter 5

Test reports

A test report is generated after each execution of a test procedure for the TravelMatch app or the back end. The test suite will print a simple message to the screen when all tests pass. Any errors that occur are printed in the console. For each test failed test that is performed, a description is given in the console to find which test failed.

5.1 back end

Below is the coverage of the back end code, given the file location in the directory. The file structure can be find in Section [4.3.1] from the ADD. The coverage is on a scale from 0 to 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Stmts</th>
<th>Miss</th>
<th>Cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>affiliate/affiliate_parser</td>
<td>89</td>
<td>14</td>
<td>0.842696629</td>
</tr>
<tr>
<td>affiliate/models</td>
<td>86</td>
<td>19</td>
<td>0.779069767</td>
</tr>
<tr>
<td>ai/entropy</td>
<td>133</td>
<td>1</td>
<td>0.992481203</td>
</tr>
<tr>
<td>ai/models</td>
<td>277</td>
<td>110</td>
<td>0.602888087</td>
</tr>
<tr>
<td>ai/recommender_system</td>
<td>82</td>
<td>4</td>
<td>0.951219512</td>
</tr>
<tr>
<td>ai/serializers</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>appusers/models</td>
<td>157</td>
<td>70</td>
<td>0.554140127</td>
</tr>
<tr>
<td>appusers/serializers</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>travelmatch/settings</td>
<td>17</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TOTAL</td>
<td>855</td>
<td>218</td>
<td>0.74502924</td>
</tr>
</tbody>
</table>

5.2 TravelMatch app

Below is the coverage of the TravelMatch app code, given the file location in the directory. The file structure can be find in Section [4.3.2] from the ADD. The coverage shown is per component. We used some custom component for the analytics and a debug service. These are not tested as of yet.
### Figure 5.1: TravelMatch app coverage per component

<table>
<thead>
<tr>
<th>File</th>
<th>Statements</th>
<th>Branches</th>
<th>Functions</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>/app/scripts/analytic/</td>
<td>14.29%</td>
<td>0%</td>
<td>0%</td>
<td>14.29%</td>
</tr>
<tr>
<td>/app/scripts/debug/</td>
<td>14.29%</td>
<td>0%</td>
<td>0%</td>
<td>14.29%</td>
</tr>
<tr>
<td>/app/scripts/details/</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>/app/scripts/hotelOverview/</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>/app/scripts/login/</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>/app/scripts/main/</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>/app/scripts/register/</td>
<td>55.07%</td>
<td>64.29%</td>
<td>30.30%</td>
<td>55.07%</td>
</tr>
<tr>
<td>/app/scripts/user/</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>