# TEST PLAN IDENTIFIER SMS-STP2.0

# REFERENCES

Documents that support this test plan include the requirement document and mockup designs.

# INTRODUCTION

## PURPOSE

The purpose of this test plan is to validate the requirements in School Management System (SMS) such that a school administrator can view the students’ information on their scores. It defines the tools to be used during the testing period. The techniques of performing the tests are also described in this plan. This document will also mention the scope and priorities.

## PROJECT OVERVIEW

The SMS is a project which allows a school administrator to view the scores of all students in the school so that he/she can analyse the performance of them.

# IN SCOPE

The following items are to be tested:

1. Login by a school administrator: A school administrator should be able to login into the system by email and password. The school administrator must be registered beforehand.
2. As a school administrator, viewing the lists of students: A school administrator can view the list of students and their scores in tabular form.
3. As a school administrator, viewing the details of each student: A school administrator can view the details of each student. The details include, photo, name, student id, ACT score, and SAT score.
4. As a school administrator, exporting the details of a student in CSV format: A school administrator can export the details of each student in CSV (Comma-Separated Values) file.
5. As a school administrator, searching a student by name and score: A school administrator can search a student using name and score through a search input box in student list view.
6. As a school administrator, sorting students by their scores: A school administrator can sort the students according to their ACT and SAT scores. To sort the students, there are up and down arrows in the end of score columns. If ‘up’ key is set, the sorting is done in ascending order, else if ‘down’ key is set, the sorting is done in descending order, otherwise the default sorting is done.

# OUT OF SCOPE

The following feature is not considered in this test plan:

* Security Testing

The above testing are not considered for this sprint but may have to be done in the future.

# TEST STRATEGY

## TEST DATA PREPARATION

The test data is created manually with the valid values of scores which bear a resemblance to the actual data of students of the school. The name can be obtained from data list of nouns and scores have their own ranges i.e. for ACT, it is 1 to 36, and for SAT, it is 400 to 1600. A default photo can be used for every entry. For example, a student of name John Doe with id std1, having ACT score 28 and SAT score 1200.

## FUNCTIONAL TESTING

For functional testing following strategies are included:

1. Black box testing: Black box testing examines the functionality of the system based on the specification. For example, on entering a valid email and valid password, the student list view page should be displayed, otherwise the proper error messages should be displayed. Likewise, on clicking export to CSV button, a CSV file should be downloaded, with the details of the students separated by commas.
2. White box testing: White box testing examines the program structure and derive test data from the program logic. It also finds the redundant, and inefficient code. For example, the implementation of logic of the program if the data is empty can be tested using this techniques. Similarly, the pagination logic can be tested using this technique.
3. Unit testing: Each module in the system is tested. For example, for login module, different combinations of valid and invalid emails and passwords are used to validate the functionality. Similarly, the student detail module is tested by clicking on a student row in student list view and verify if proper data is received or not.
4. Regression testing: Regression testing examines the ripple effect created by new modules in previous modules. For example, checking if the signup module is affected by the introduction of login module. For this, the previous tests are re-executed and the current and previous results are compared.

## API TESTING

The testing is performed to test the API from which the data are obtained. For this following approaches are taken:

1. Verify the response status code: The response status code gives the information of how the request is handled. For example, a valid login request should give a status of 200 i.e. OK, and an invalud login request should give a status of 403 i.e. Forbidden. Similarly, a 200 status for a GET request retrieves the data whereas, a 404 status for a GET request cannot get any resources on the endpoint.
2. Verify the response payload: The response payload should be a proper format. For example, the response should be in proper JSON format so that the data can be extracted. For example, if a GET request is sent on api.domain.com/students, a JSON string should be obtained with the different entries of students and their values to be displayed on student list view.
3. Verify the response headers: The response headers are verified for performance and security reasons. For example, if a header is sent which has no use in this application, it can be removed.
4. Verify the response time: For large amount of data, retrieving all the data can be time consuming. So, the response time plays a crucial role in the performance of the system.

## UI TESTING

The approaches can be used for UI Testing:

1. Check if proper messages are displayed: Messages are important for a good user experience. For example, if a user is not able to login to the system, a proper error message should be displayed. For instance, instead of “something went wrong”, “the email is invalid” is preferred.
2. Check if proper colours are used: Colours can be used to highlight key things. For example, the error should be displayed in red colour, warning should be displayed in orange like colour and so on.
3. Check if contrast of texts is good: The inspect tool in developer console can be used to check the contrast of texts. A good contrast is easy to read.

# TOOLS

The following software are required for testing:

1. POSTMAN

* An API testing tool.
* Provides automation.
* Performs the integration test.
* Stores data for future use.
* Tests are structured properly.

1. CYPRESS

* A testing framework for Javascript.
* Tests are managed in a structured way.
* Test runner and test execution is easy.

# ASSUMPTIONS / RISKS

## ASSUMPTIONS

1. The requirements will not change during the sprint.
2. The test data will be sufficient to validate all features of this system.
3. All the tests will be performed within the time period of the sprint.
4. There will be sufficient resources to perform the testing.

## RISKS

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risk** | **Impact** | **Mitigation Plan** |
| 1 | Test data might not be sufficient to test all the features. | Medium | Import data from existing database by anonymising. |
| 2 | The requirement might change. | High | Communicate with stakeholders. Make proper sprint plan. |
| 3 | The tests will not be finished on time. | Medium | High priority tests will be performed at first such that low priority tasks will be left which can be performed later in another sprint and will not have much impact on functionality. It should be communicated with the stakeholders. |
| 4 | The resources might not be sufficient to complete the test. | High | Negotiate with the project manager and stakeholders. |

# TEST DELIVERABLES

* Test plan document
* Test cases
* Summary Reports
* Bug Reports

# TESTING ENVIRONMENT

|  |  |
| --- | --- |
| Environment | Roles |
| Dev | Developers |
| QA | QA, PM, TL |
| UAT | QA, Client |

# BUG REPORT

A bug report consists of the detailed information of a bug and the steps to reproduce it. The bugs will be added to the issues tracking system called JIRA.