Test plan

Smarter Balanced Assessment Consortium
Test Delivery System

Components:
Test Delivery System Administration

Smarter Balanced Task Order 17

Revision History

Revision Description	**Author**	**Reviewer**	**Date**
0.1 – First Draft	Peter Tereshchenko		Mar. 31, 2016
0.2 – Second Draft	Peter Tereshchenko	Rami Levy	Apr. 14, 2016
1.0 – Final version	Peter Tereshchenko	Rami Levy	Apr. 24, 2016

Table of Contents

Tables

1 Introduction and Background of System

The purpose of this document is to define testing scope and approach for Task Order 17. This task order seeks to create and provide a user interface for several stored procedures which have to be used to adjust TDS test opportunity parameters like status and expiration date. To avoid administrator's extensive access and use of database, a new component called Test Delivery System Administration (TDSAdmin) was created.

2 References

	Reference	Location	Version
1	TDSAdmin-TestCases.xlsx	TDS Admin source code repository documentation folder	1.0
2	TDSAdmin-Requirements.docx or pdf	TDS Admin source code repository documentation folder	1.0

Table 1: References

3 Scope of testing and preconditions

There are 17 requirements for TO17, which can be divided in to 2 categories:

a) Requirements describing user interface and behavior of the TDSAdmin component (1-7, 15-17)
b) Requirements describing the stored procedures which the TDSAdmin component requests to execute in Test Delivery System (8-14)

The TDSAdmin interface and behavior is relatively simple and doesn't require specific information and preconditions other than those already described in test cases.

Each stored procedure from section b) can be executed by TDSAdmin user interface and by API request. There is one additional API request which is not executing a procedure but returns a list of search results. APIs provide access to almost all functionality available in the UI. That's why in general APIs will use same parameters for procedure execution. For example, if in UI to execute "Extend Grace Period", the user has to specify "selected sitting", then the same parameter has to be specified in UI. However there are 2 fundamental differences between UI and API use:

1) In the UI, the user selects from the available opportunities returned by the search. In the API request, the oppKey used to identify an opportunity for execution is an internal representation of the opportunity identifier available in the UI.
2) There are parameters which some APIs require as user input but UI does not. For example "Set Opportunity Permeability" requires user to input segment ID and segment position. UI on the other hand asks for only segment position and extracts segment ID on its own. Since APIs are

supposed to be used in component to component communications, having extra parameters to input is not considered as an issue.

Stored procedures have several restrictions as to which type of opportunities they can be applied as well as rules for procedure parameters. The QA team should verify that the application doesn't allow users to break these rules and restrictions. Rules are specific to each procedure and are described in designated test cases. Additionally, test cases will provide information about which test opportunities QA team should create in advance to test various scenarios for specific stored procedures. UI of TDSA by itself protects the user from many mistakes, for example not allowing to input "selected sitting" outside of allowed range. But there are many rules that can be verified only before procedure execution, in that case application will return status of execution as fail if rule is violated and provide information about what is violated. For APIs there is no way to prevent input of wrong data prior procedure execution, and when for many mistakes APIs will return proper error explanation, there are cases which are not covered and API will return generic 400 syntactic error. Additionally, the UI doesn't allow selecting and thus executing a procedure for an opportunity which is not in proper status; for instance, *reset* cannot be performed on paused opportunities. The API doesn't check this, so it is the API caller's responsibility to verify in advance that selected procedure for specific opportunity is appropriate. Testing of results for running procedures with wrong opportunities is out of scope.

All procedures modify database information for opportunities, which in general will change TDS Student application behavior. These stored procedures were developed prior to the TDS Admin application. TDS Admin is a user interface and API handler which allows administrators and components to execute these procedures without logging directly into the database. Because of that, the QA team can make an assumption that the stored procedures are working as expected and the main focus of testing should be to verify not only that the UI or API execution results shows confirmation of a procedure's successful completion, but also that the procedure was executed. It is unnecessary to verify each change made by procedure to check that it was applied, so testing scope is limited to verifying parameters from "session.testopportunity" table in TDS DB and also verifying changes for proctor and student UI where applicable.

Depending on test environment and configuration testing may require modifying opportunities directly in database to make them suitable for procedure execution. For instance several opportunities with expired status are required for testing. QA team can configure some tests to have very small expiration period, but it cannot be smaller than 1 day, so testers will have to wait for 24 hours to get expired opportunities. Other option is to execute specific query in DB which will make opportunity expired, but in that case QA should be careful and update several other fields which in normal workflow will be updated by system. Also if test environment does not have downstream components running, getting some cases required for testing opportunities will not be possible without direct DB data modifications.