TDS Admin Design Documentation

Smarter Balanced Assessment Consortium

Components: Test Delivery System

Smarter Balanced Task Order 17

Revision History

Revision Description	Author / Editor	Date
1.0 - Initial Release	Shajib Khan	May 20, 2016

Table of Contents

Table of Figures

References

Document Name	Location	Version
TDS Admin API document	Source repository: docs or external_release_docs directories	1.0
TDS Admin User Guide	Source repository: docs or external_release_docs directories	1.0
TDS Admin Requirements	Source repository: docs or external_release_docs directories	1.0

1 Overall Application Design

The application has two architectures for the user interface and the REST APIs separated by user authentication methodology. The user interface uses SAML while the REST API uses OAuth 2.0. The following two diagrams depicts the how the application works:

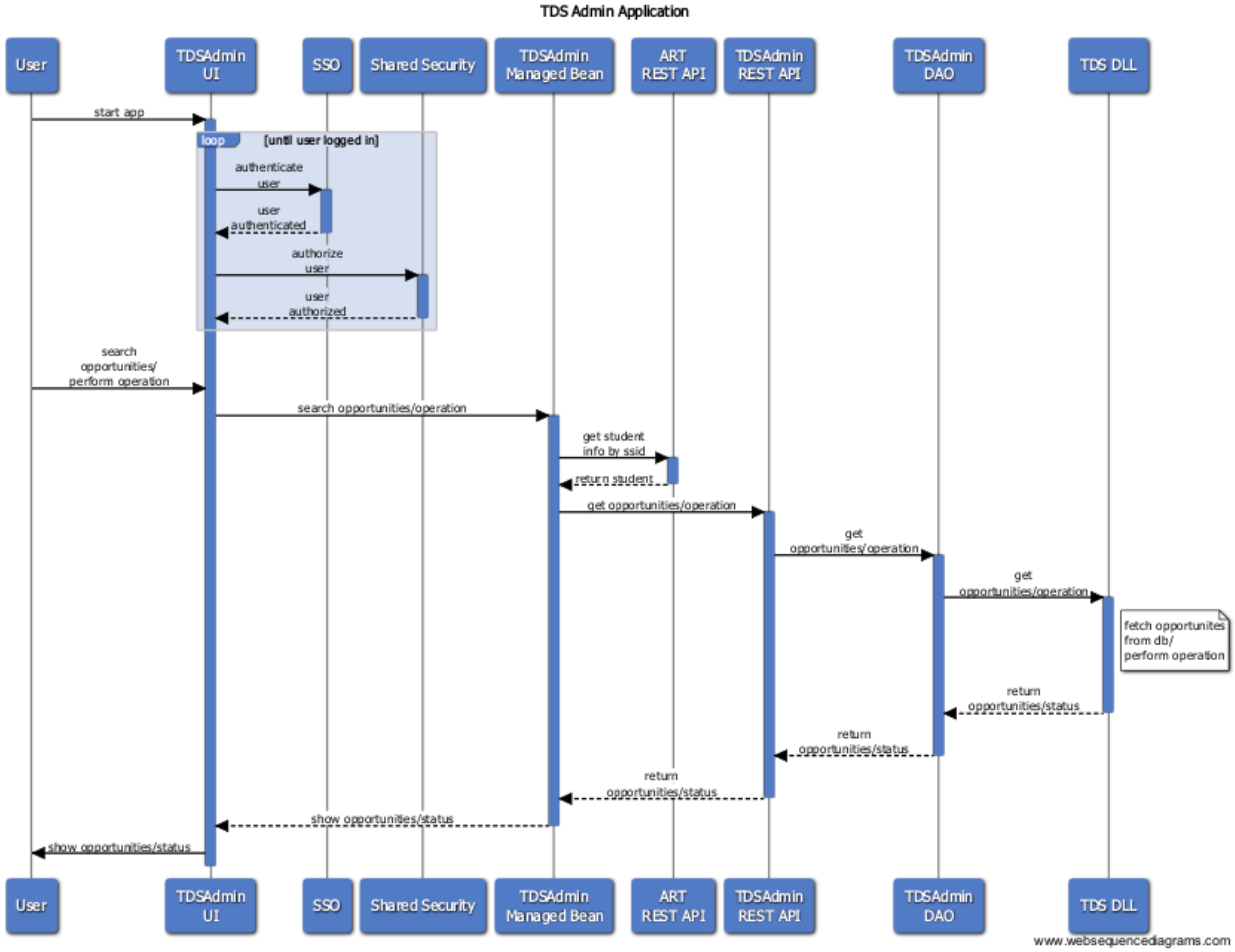

Figure 1: High level design for TDS Admin application with User Authentication

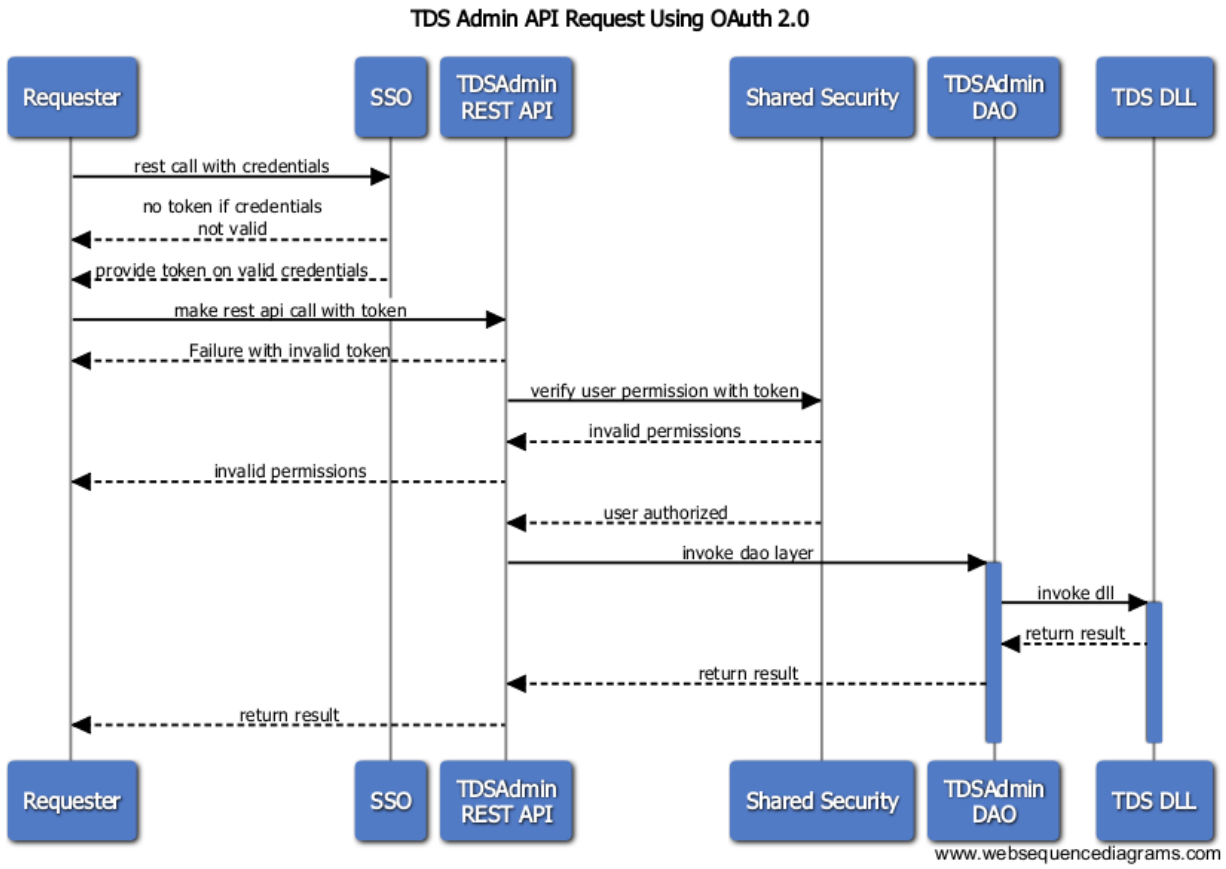

Figure 2: High level design for TDS Admin API Request with OAuth 2.0

2 External Integrations

2.1 Configuration via Program Management

The application needs to be integrated with program management as it contains configuration with following parameters and their values:

TDS DATABASE PROPERTIES

- tdsadmin.datasource.url=jdbc:mysql://<url.to.db>:3306/schemaname?useUnicode=true&characterEncoding=utf8&useFastDateParsing=false - The JDBC URL of the database from which connections can and should be acquired. *useUnicode* is required to store unicode characters into the database. Turning off useFastDateParsing allows the column values of DateTime type to be read, given the underlying database is MySQL.
- datasource.username=<db-username> - Username that will be used for the DataSource's default getConnection() method.
- datasource.password=<db-password> - Password that will be used for the DataSource's default getConnection() method.
- datasource.driverClassName=com.mysql.jdbc.Driver - The fully qualified class name of the JDBC driverClass that is expected to provide Connections.
- datasource.minPoolSize=5 - Minimum number of Connections a pool will maintain at any given time.
- datasource.acquireIncrement=5 - Determines how many connections at a time datasource will try to acquire when the pool is exhausted.
- datasource.maxPoolSize=20 - Maximum number of Connections a pool will maintain at any given time.
- datasource.checkoutTimeout=60000 - The number of milliseconds a client calling getConnection() will wait for a Connection to be checked-in or acquired when the pool is exhausted. Zero means wait indefinitely. Setting any positive value will cause the getConnection() call to timeout and break with an SQLException after the specified number of milliseconds.
- datasource.maxConnectionAge=0 - Seconds, effectively a time to live. A Connection older than maxConnectionAge will be destroyed and purged from the pool. This differs from maxIdleTime in that it refers to absolute age. Even a Connection which has not had much idle time will be purged from the pool if it exceeds maxConnectionAge. Zero means no maximum absolute age is enforced.
- datasource.acquireRetryAttempts=5 - Defines how many times datasource will try to acquire a new Connection from the database before giving up. If this value is less than or equal to zero, datasource will keep trying to fetch a Connection indefinitely.
- datasource.idleConnectionTestPeriod=14400 - If this is a number greater than 0, Datasource will test all idle, pooled but unchecked-out connections, every this number of seconds.

TDS ADMIN PROPERTIES

The following parameters need to be added into a Program Management TDSAdmin configuration for TDS Admin itself to function:

- tdsadmin.sessionTimeoutMinutes=15 - Number of minutes before the login session times out. Default is 15. A negative value here will force no timeout.
- tdsadmin.TestRegistrationApplicationUrl=http://<url-to-art-app>:port/rest - URL to ART Application REST context
- tdsadmin.TDSSessionDBName=session - Name of the TDS session schema

The following steps are necessary to integrate Program Management with TDSAdmin:

1. Add following dependencies to pom.xml

```
<properties>
```

```
            <prog.mgmnt.client.version>0.0.3-SNAPSHOT</prog.mgmnt.client.version>
            <sb11-mna-client.version>0.0.4-SNAPSHOT</sb11-mna-client.version>
            <sb11-shared-code.version>0.0.6-SNAPSHOT</sb11-shared-code.version>
        </properties>

        <!-- Start: Program management integration dependency -->
        <dependency>
          <groupId>org.opentestsystem.shared</groupId>
          <artifactId>prog-mgmnt-client</artifactId>
          <version>${prog.mgmnt.client.version}</version>
        </dependency>

        <dependency>
          <groupId>org.opentestsystem.shared</groupId>
          <artifactId>prog-mgmnt-null-client</artifactId>
          <version>${prog.mgmnt.client.version}</version>
        </dependency>
        <!-- End: Program management integration dependency -->
        <!-- Start: Monitoring and Alerting integration dependency -->
        <dependency>
          <groupId>org.opentestsystem.shared</groupId>
          <artifactId>monitoring-alerting.client-null-impl</artifactId>
          <version>${sb11-mna-client.version}</version>
        </dependency>

        <dependency>
          <groupId>org.opentestsystem.shared</groupId>
          <artifactId>monitoring-alerting.client</artifactId>
          <version> ${sb11-mna-client.version}</version>
        </dependency>

        <dependency>
          <groupId>org.opentestsystem.shared</groupId>
          <artifactId>sb11-shared-code</artifactId>
          <version>${sb11-shared-code.version}</version>
        </dependency>
<!-- End: Monitoring and Alerting integration dependency -->
```

2. Add these entries in web.xml

```
<!--Start: Added for Program Management and Monitoring and Alerting dependency -->
        <context-param>
                <param-name>contextInitializerClasses</param-name>    <param-
value>org.opentestsystem.shared.mna.client.listener.ClientSpringConfigurator,org.opentestsystem.shar
ed.progman.init.InitSpringPropertyConfigLoad</param-value>
        </context-param>
<!--End: Added for Program Management and Monitoring and Alerting dependency -->
```

3. Create a new configuration in Program Management and use the config name and environment in step 4 in property -Dprogman.locator="tdsadmin,{env}" (i.e. {env} is local in my computer)

4. Add following to your run configuration

```
-DSB11_CONFIG_DIR={directory having progman} -
Dspring.profiles.active="progman.client.impl.integration,mna.client.integration" -
Dprogman.baseUri=http://pm-dev.opentestsystem.org:8080/programmanagement.rest/ -
Dprogman.locator="tdsadmin,local" -
Djavax.net.ssl.trustStore="c:\Development\security\samlKeystore.jks" -
Djavax.net.ssl.trustStorePassword="nalle123"
```

Here, DSB11_CONFIG_DIR is the directory which has the progman folder, it contains certificates (*_local_sp.xml, *.properties etc.). Change truststore and baseUrl for respective systems.

5. To map variable to Program Management properties use following annotation

```
@Value(value="${mna.logger.level:}")
```

6. Add progman-loader-config-props-context.xml file to the project's resource/spring folder. You can copy the file from Permission or any other project already configured with Program Management.

2.2 Authorization via Permissions

User authorization is accomplished via a REST API call to Permissions. The API URL is provided in the Program Management configuration. This is performed by the Shared Security module.

PERMISSIONS PROPERTIES

- permission.uri=https://<permission-app-context-url>/rest - The base URL of the REST API for the Permissions application.
- component.name=TDS Admin - The name of the component that this tdsadmin deployment represents. This must match the name of the component in Program Management and the name of the component in the Permissions application.

2.3 Monitoring and Alerting (MNA)

The following steps are required for integration with the Monitoring and Alerting application:

1. Follow all the steps as shown above for Program Management.
2. Add logback.xml file inside your project's resource folder and add following. Change the value of app.base.package.name and app.context.name inside logback.xml file:

```
<configuration scan="true">
        <property scope="local" name="app.context.name" value="tdsadmin" />
        <property scope="local" name="app.base.package.name" value="tds.tdsadmin" />
```

```
        <property scope="local" name="mna.appender.active" value="true" />
        <include resource="logback-included-common-config.xml" />
</configuration>
```

The following parameters need to be added into a Program Management TDSAdmin configuration for integration with MNA:

MNA PROPERTIES

- mna.mnaUrl=http://<mna-context-url>/rest/ - URL of the Monitoring and Alerting client server's REST context.
- mnaServerName=tdsadmin - Used by the MNA clients to identify which server is sending the log/metrics/alerts.
- mnaNodeName=prod - Used by the MNA clients to identify who is sending the log/metrics/alerts. There is a discrete mnaServerName and an mnaNodeName to provide the ability to search across clustered nodes by server name or specifically for a given node. It's being stored in the db for metric/log/alert, but not displayed.
- mna.logger.level=INFO - Used to control what is logged to the Monitoring and Alerting system. Logging Level values include (ALL - Turn on all logging levels, TRACE, DEBUG, INFO, WARN, ERROR, OFF - Turn off logging). IMPORTANT: The audit log of TDSAdmin activity is provided at the INFO level. Setting this value to anything higher will prevent successful actions from being logged in MNA.
- mna.oauth.batch.account=<mna-client-user>
- mna.oauth.batch.password=<password>
- mna.oauth.client.secret=<password> - OAuth Client secret/password configured in OAM to allow get an OAuth token for the "batch" web service call to MnA.
- mna.oauth.client.id=mna - OAuth Client id configured in OAM to allow get an OAuth token for the "batch" web service call to MnA.

2.4 Authentication via Single Sign On (SSO)

Integration with Program Management and Monitoring & Alerting modules is required before Single Sign On can work. Assuming they are already integrated, use these steps to add Single Sign On capability to TDSAdmin:

1. Copy security folder from src/main/resource of the Permissions application into the project's src/main/resource directory.

2. Open samlmetadata-context.xml and change the properties starting with ${} to your project's configuration in Program Management. For example, since we are copying from Permissions to TDSAdmin, replace ${permission.security.saml.keystore.cert} with ${tdsadmin.security.saml.keystore.cert}. This is something that should be mapped to your Program Management configuration.

3. Open securityContext.xml and follow the same steps as in (2), above.

4. Inside pom.xml add the following dependency:

```
<properties>    <org.springframework.security-version>3.2.9.RELEASE</org.springframework.security-version>
</properties>
<!-- Start: SSO integration dependency -->
                <dependency>
```

```
            <groupId>org.opentestsystem.shared</groupId>
            <artifactId>sb11-shared-security</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-core</artifactId>
            <version>${org.springframework.security-version}</version>
        </dependency>
        <dependency>
            <groupId>xerces</groupId>
            <artifactId>xercesImpl</artifactId>
            <version>2.10.0</version>
        </dependency>
<!-- End: SSO integration dependency →
```

5. Add following filter mapping into web.xml

```
<!--Start:  Following filter is added for integrating SSO  -->
        <filter>
            <filter-name>springSecurityFilterChain</filter-name>
            <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        </filter>
        <filter-mapping>
            <filter-name>springSecurityFilterChain</filter-name>
            <url-pattern>/*</url-pattern>
        </filter-mapping>
<!--End:  Following filter is added for integrating SSO  -->
```

6. Add the following entries in the beginning of the Spring context xml file of your project

```
<!-- Start : Added for SSO Integration -->
        <beans:import resource="classpath:security/securityContext.xml"/>
        <beans:import resource="classpath:security/samlmetadata-context.xml"/>
<!-- End : Added for SSO Integration -->
```

To complete the SSO integration, the following parameters must be added to the TDS Admin configuration in Program Management:

SSO PROPERTIES

- tdsadmin.security.profile=prod - The name of the environment the application is running in. For a production deployment this will most likely be "prod. (it must match the profile name used to name metadata files).
- tdsadmin.security.idp=https://<idp-url> - The URL of the SAML-based identity provider (OpenAM).
- tdsadmin.webapp.saml.metadata.filename=tdsadmin_prod_sp.xml - OpenAM Metadata file name uploaded for environment and placed inside server directory.
- tdsadmin.security.dir=file:////<sp-file-location-folder> - Location of the metadata file.
- tdsadmin.security.saml.keystore.cert=<cert-name> - Name of the keystore cert being used.

- tdsadmin.security.saml.keystore.pass=<password> - Password for keystore cert.
- tdsadmin.security.saml.alias=tdsadmin_webapp - Alias for identifying web application.
- oauth.access.url=https://<oauth-url> - OAuth URL to OAM to allow the SAML bearer workflow to POST to get an OAuth token for any "machine to machine" calls requiring OAUTH
- tdsadmin.oauth.resource.client.secret=<password> - OAuth Client secret/password configured in OAM to allow get an OAuth token for the "batch" web service call to core standards.
- tdsadmin.oauth.resource.client.id=tdsadmin - OAuth Client id configured in OAM to allow get an OAuth token for the "batch" web service call to core standards.
- tdsadmin.oauth.checktoken.endpoint=http://<oauth-url> - OAuth URL to OAM to allow the SAML bearer workflow to perform a GET to check that an OAuth token is valid.

2.5 Administration and Registration Tool (ART)

The ART API is called when searching for opportunities via SSID. To enable this capability, the following parameters must be added to the TDS Admin Program Management configuration:

ART PROPERTIES

- oauth.testreg.client.id=art - OAuth test client ID for ART
- oauth.testreg.client.secret=<client-secret> - OAuth client secret for ART
- oauth.testreg.client.granttype=password - OAuth grant type for ART, should be password
- oauth.testreg.username=<email> - OAuth username for test registration
- oauth.testreg.password=<password> - OAuth password for test registration

2.6 Build Dependencies

The TDSAdmin application has dependencies on the following projects:

1. shared-db
2. tds-dll-api

tds-dll-mysql

3 User Interface

3.1 Skinning

The TDS Admin user interface was skinned to match the other Smarter Balanced applications. This was accomplished via inline CSS in Default.xhtml and two external CSS file tdsadmin.css and core.css. The same images as other Smarter apps were used.

3.2 Managing Table of Opportunities

Uses JSF PrimeFaces library (version 4.0 under Apache license 2.0) for datatable which works with Java object of type LazyDataModel<T> (It has generic type of list within it, TestOpportunity in our case) and uses custom column sorting using Java reflection.

4 Fetch Opportunity

4.1 Sequence Diagrams

Figure 3 is a sequence diagram representing the flow for searching opportunities by external SSID. Figure 4 is a similar diagram, but for searching by SSID. Note that in this case, SSID search requires a call to ART since the SSID is not stored in the TDS DB.

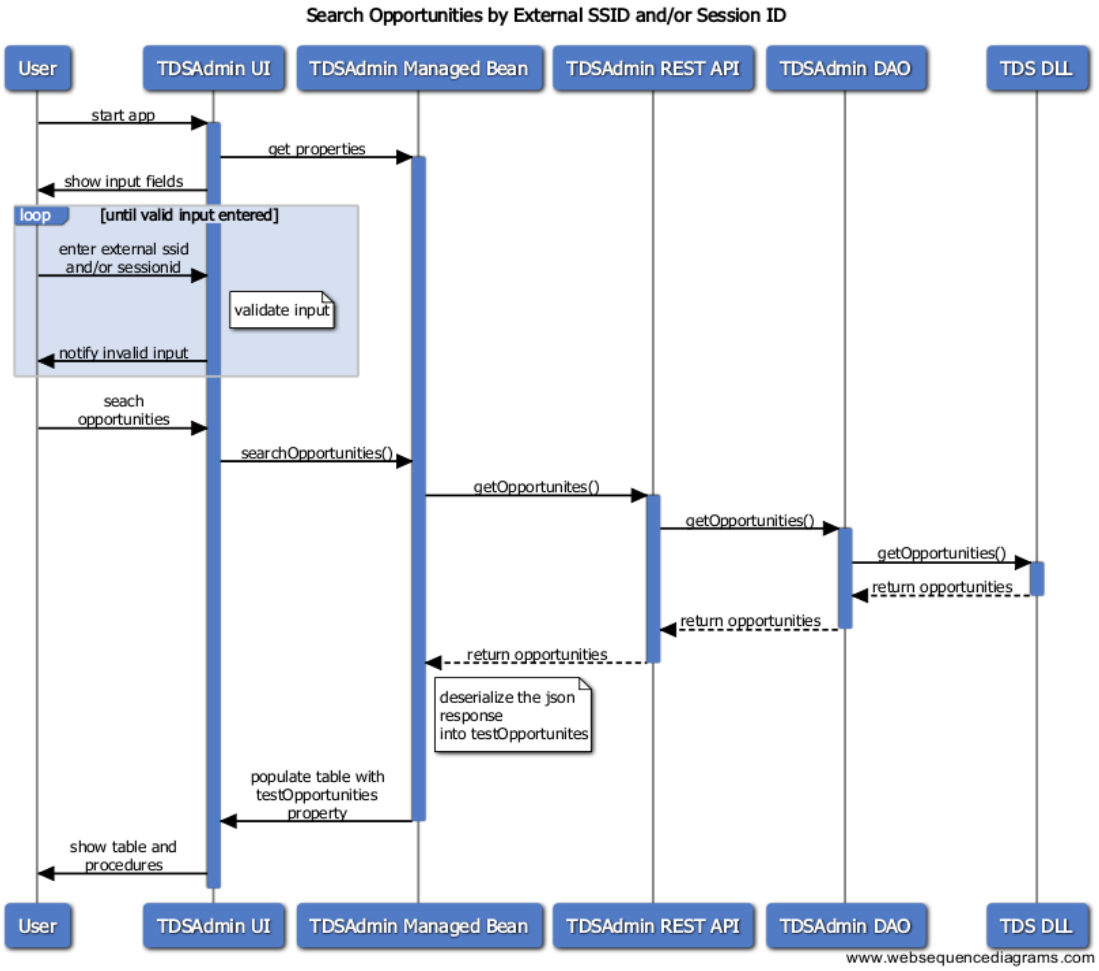

Figure 3: Search Opportunities by External SSID and/or Session ID

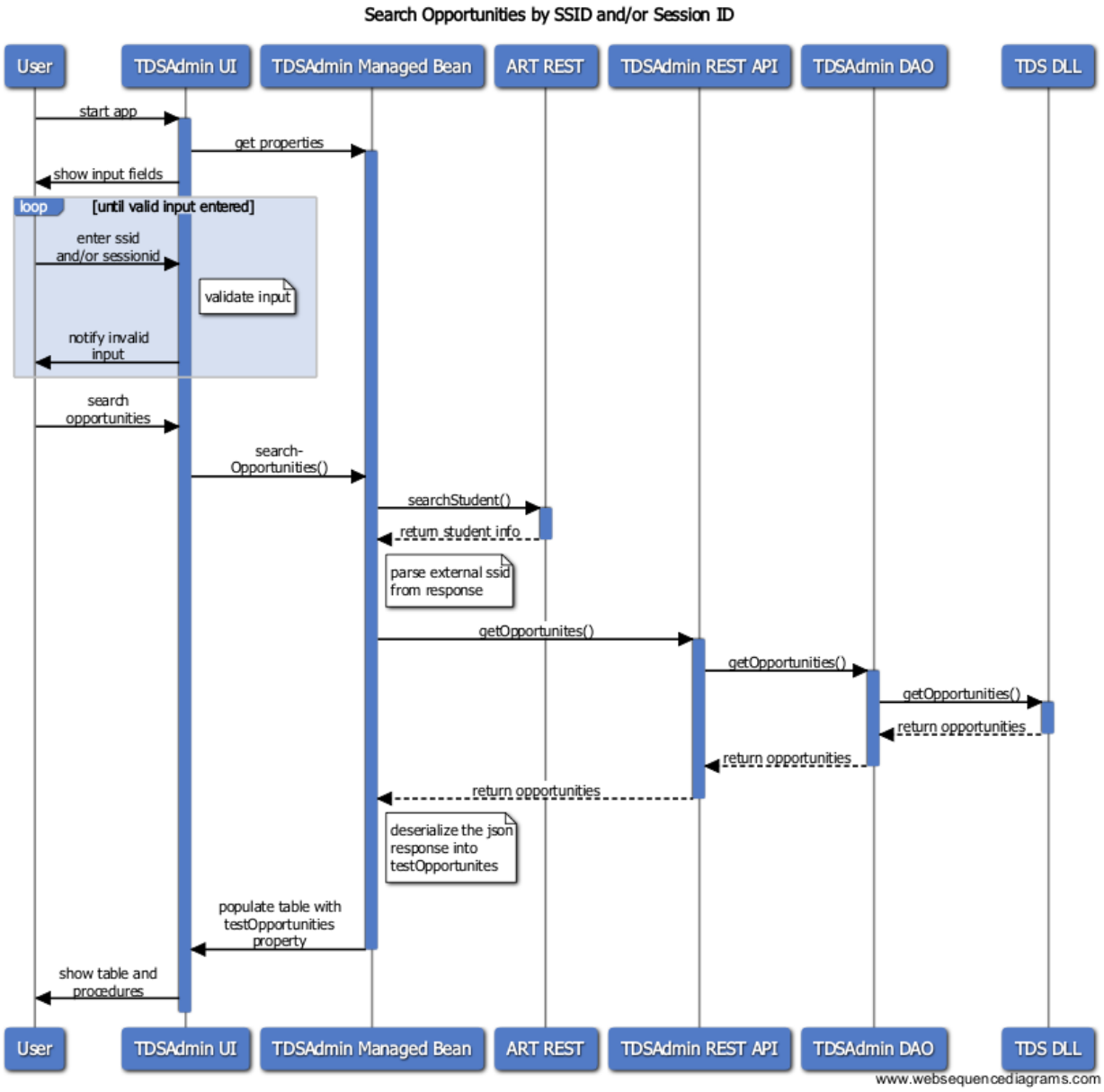

Figure 4: Search Opportunities by SSID and/or session ID

4.2 Diagram Components

1. TDS Admin UI: This will be one JSF page providing the interface to user for fetching opportunities, select database operations and modify opportunities.

2. TDS Admin Backing Bean: This is the Java code working behind the UI, the class is called DefaultBacking.

3. TDS REST API: These are the REST web service called from TDS Admin Backing Bean to interact with the database layer. TDS Admin can't call the database layer directly.

4. ART REST API: The ART REST API is used to obtain the student's External SSID based on their SSID. This call is secured via OAuth2 token authentication (not shown in this Figure, but is analogous to the flow described in Figure 2).

5. TDS DLL: This component provides the API to execute database operation in raw level.

6. SSO: This manages user (SAML) and application (OAuth2) authentication (Not shown in this Figure, for clarity).

7. MNA: This manages logging of activities within the TDS Admin application (Not shown in this Figure, for clarity). MNA calls are secured via OAuth2 token authentication (not shown in this Figure, but is analogous to the flow described in Figure 2).

4.3 Class Diagram

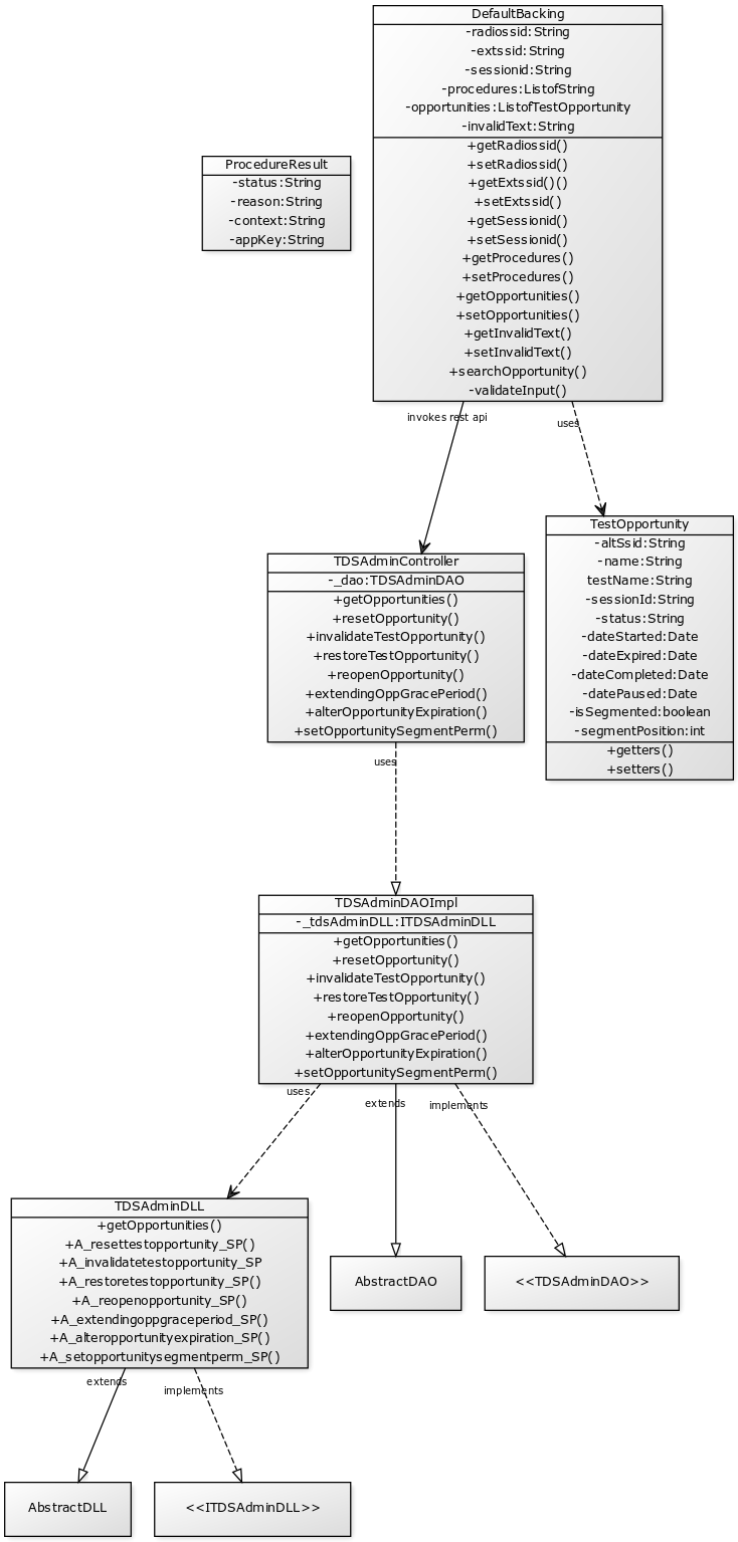

Figure 5: Class Diagram

The class diagram in Figure 5 shows the core classes and methods in the application. There are additional helper methods in some of the classes.

DefaultBacking: The backing bean for the default page, the primary and only user interface.

TestOpportunity: This is the model class storing information about an opportunity.

TDSAdminController: This is the application's controller class which defines the REST endpoints and handles client requests for getting and updating information about opportunities.

TDSAdminDAO: Interface for the data access object (DAO) used by controller.

TDSAdminDAOImpl: Implementation for TDSAdminDAO.

AbstractDAO: Resides in shared-db module inside sharedmultijar. It's extended by TDSAdminDAOImpl.

ITDSAdminDLL: Interface providing the skeleton for accessing database and calling stored procedures. It is the tds-dll-api project.

TDSAdminDLL: Implementation of ITDSAdminDLL. It is in tds-dll-mysql project.

AbstractDLL: Resides in shared-db module inside sharedmultijar and extended by TDSAdminDLL.

ProcedureResult: Returns the result of procedure execution to the user.

5 Test Opportunity Operations

The following sequence diagrams show the post -authentication and -authorization workflow of the seven APIs to perform the operations on the test opportunities within the TDS Admin application. However, the workflow differs for the REST API calls, which is depicted in Figure 2.

5.1 Set Opportunity Segment Permeability

Figure 6 is a sequence diagram for setting segment permeability on an identified opportunity.

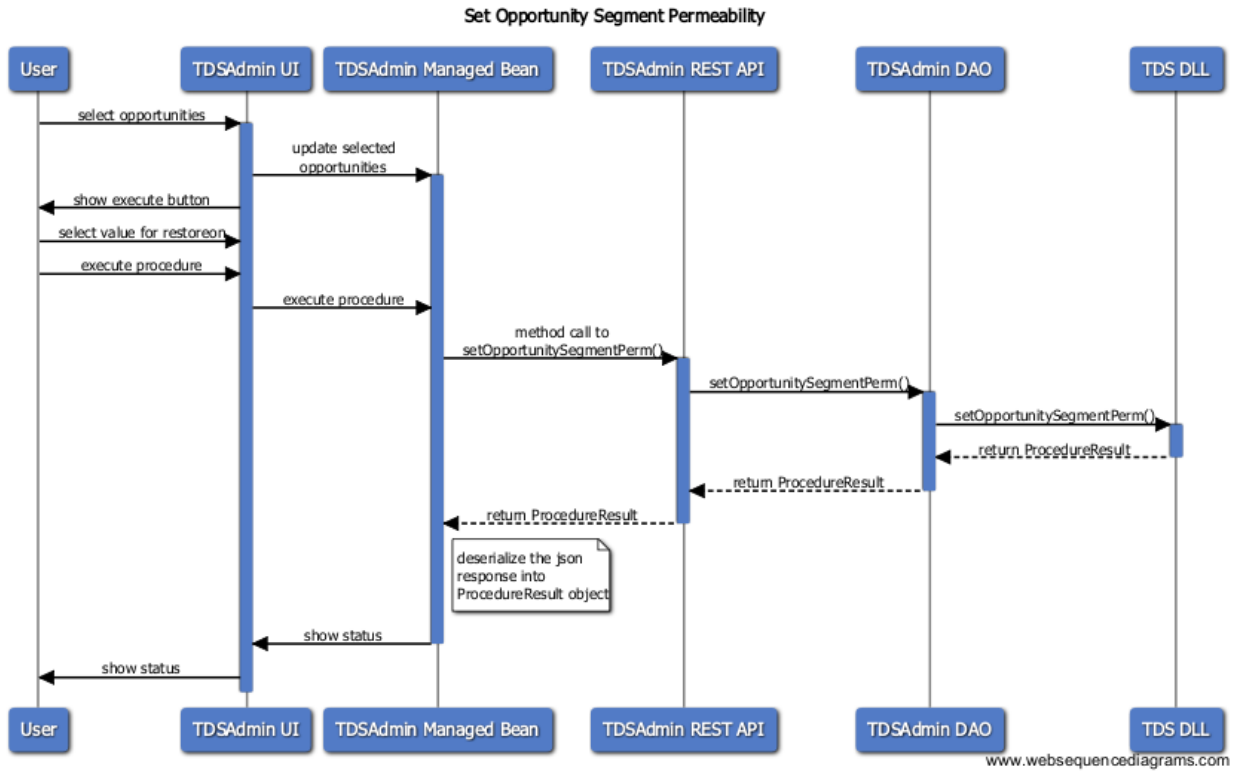

Figure 6: Set Opportunity Segment Permeability

5.2 Restore Test Opportunity

Figure 7 is a sequence diagram for restoring test opportunity. Restore reverses a *reset* action.

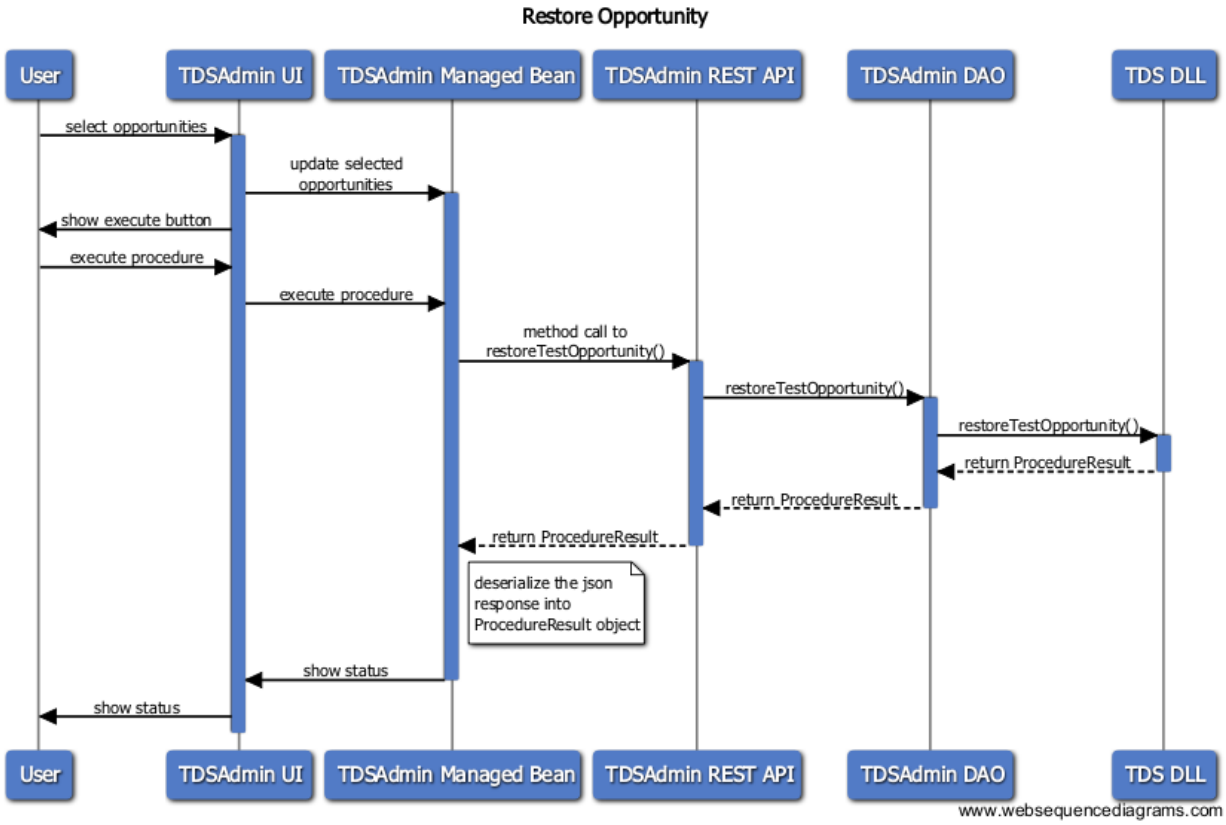

Figure 7: Restore Test Opportunity

5.3 Reopen a Test Opportunity

Figure 8 diagram allows reopening of a test opportunity to complete a previously expired test opportunity.

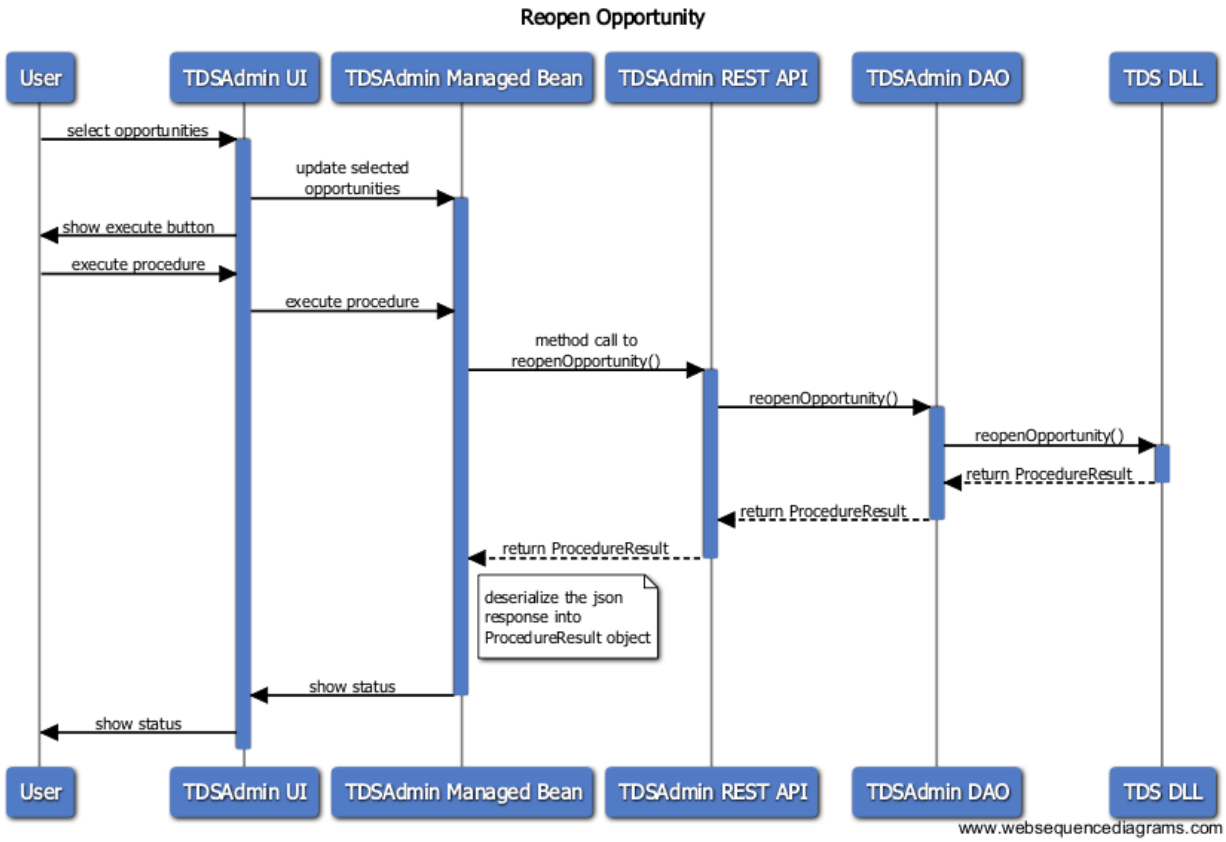

Figure 8: Reopen Test Opportunity

5.4 Invalidate a Test Opportunity

Figure 9 is a sequence diagram for invalidating an existing test opportunity.

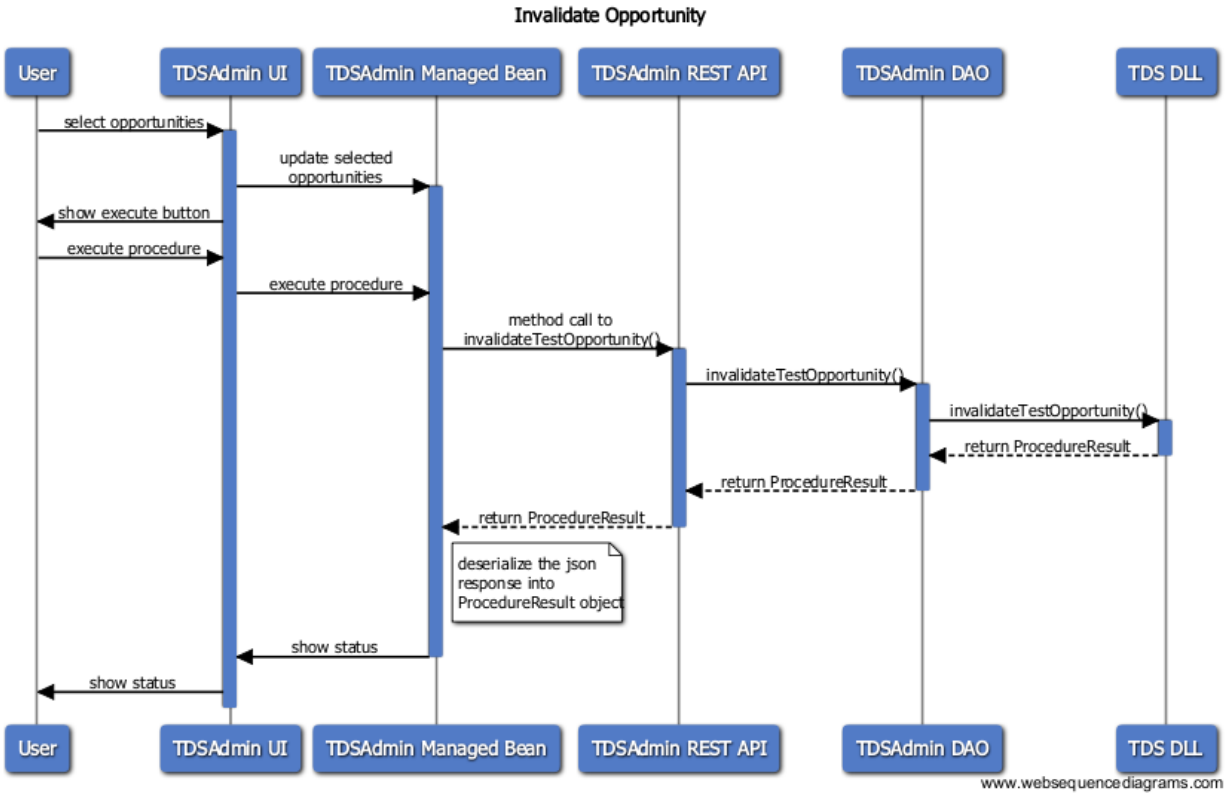

Figure 9: Invalidate Test Opportunity

5.5 Extend an Opportunity's Grace Period

Figure 10 below is a sequence diagram for extending grace period on an identified opportunity.

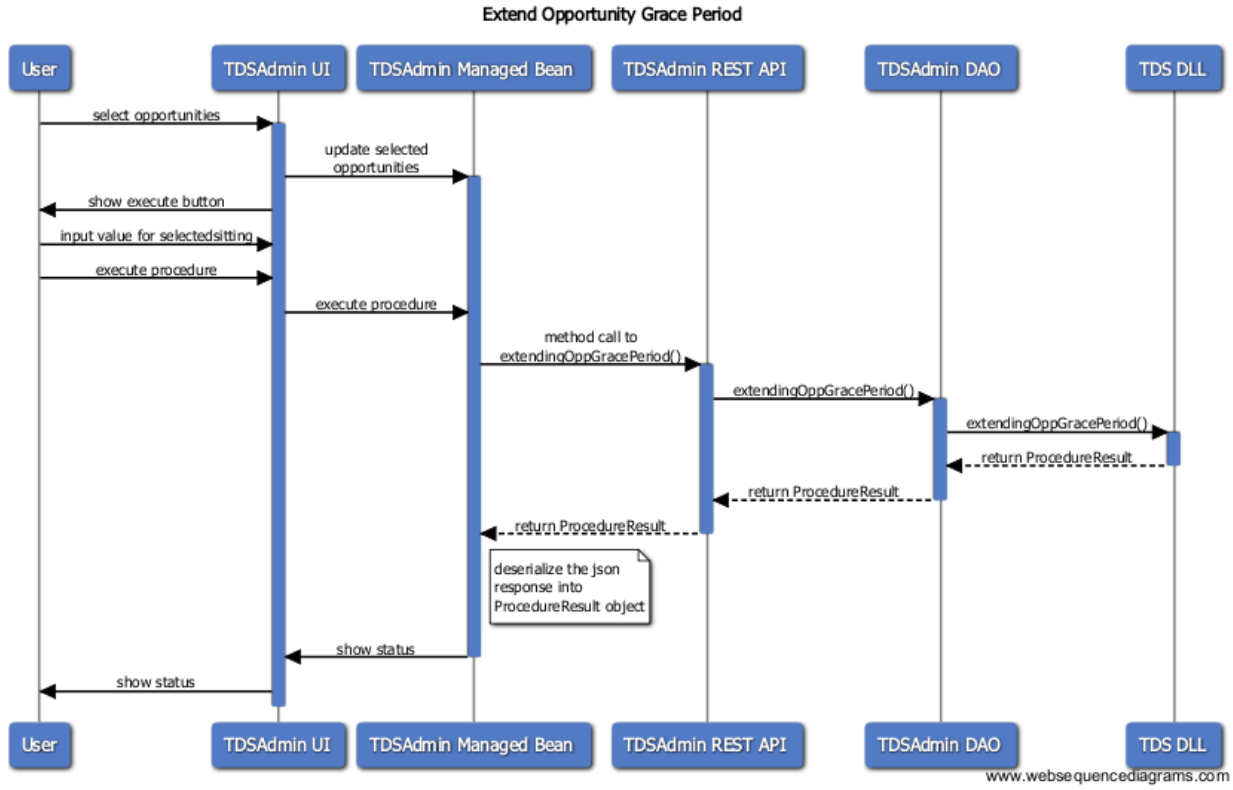

Figure 10: Extend Opportunity Grace Period

5.6 Alter an Opportunity's Expiration

Figure 11 sequence diagram provides the ability to Alter an Opportunity's Expiration on an identified opportunity. For test opportunities that have already expired, Extend Expiration allows the student to resume the test and be able to go back and change previous responses.

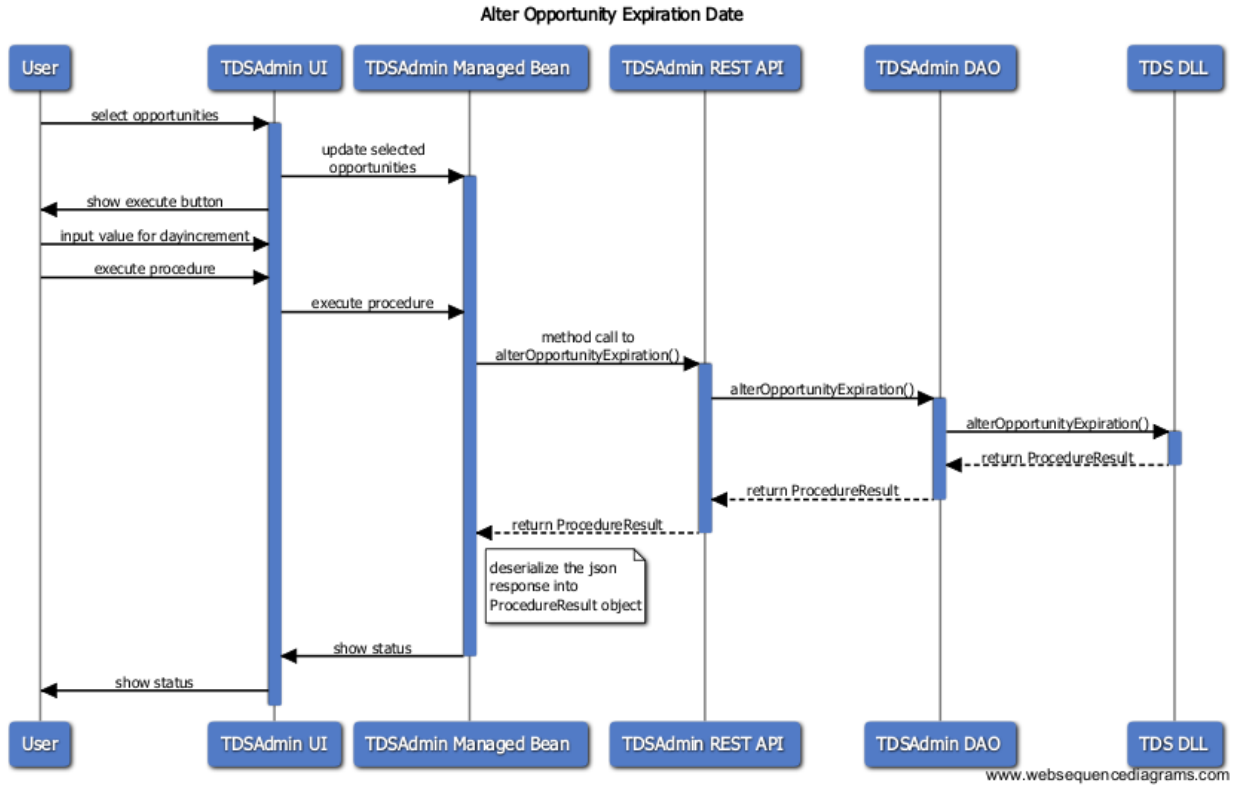

Figure 11: Alter Opportunity Expiration

5.7 Reset a Test Opportunity

Figure 12 – Reset a test opportunity, wipes out an existing student's test opportunity, allowing the student to take an opportunity again.

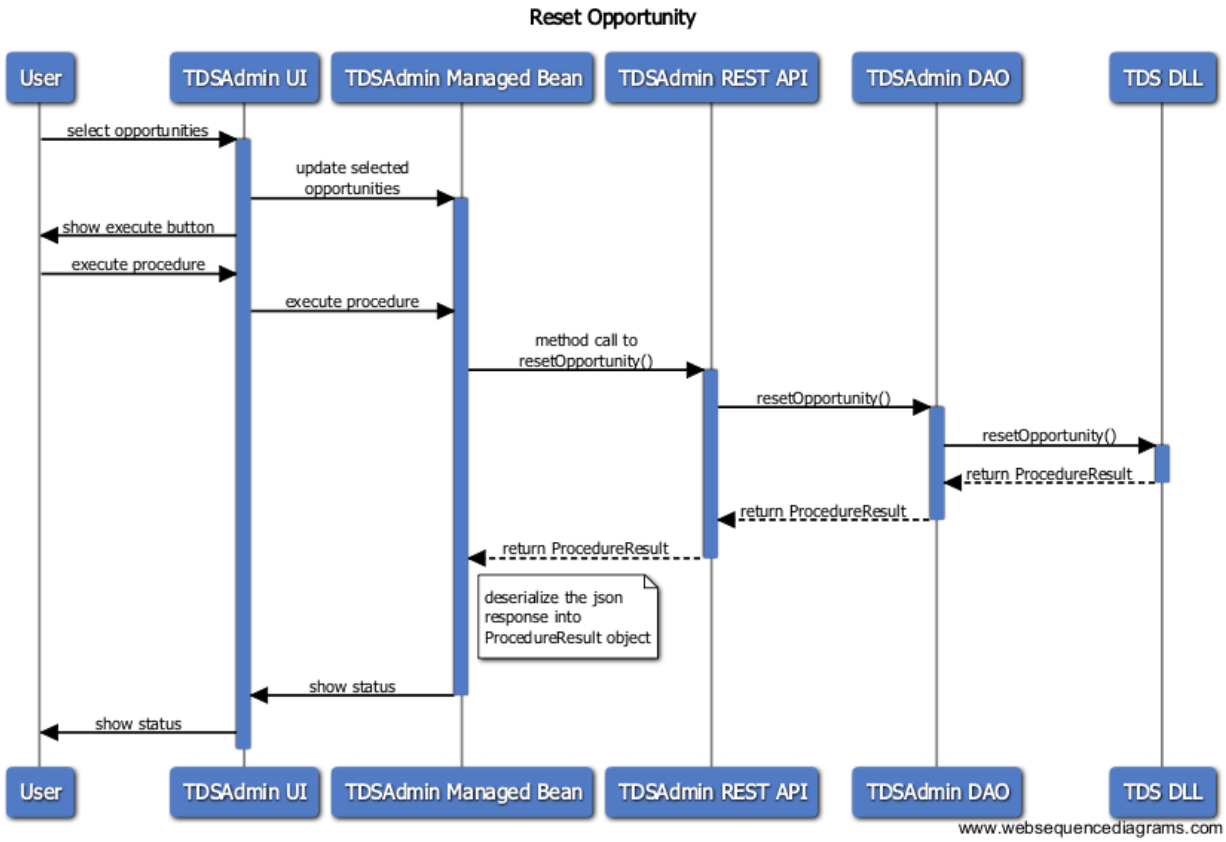

Figure 12: Reset Test Opportunity

5.8 Show Result of Procedure Execution

This information is extracted from ProcedureResult object returned after API call for each procedure and shown by the managed bean (DefaultBacking). ProcedureResult has a property called *status*, which has value either "success" or "failed" and another property *reason*, which stores a text string with failure details.

6 Session Operations

6.1 Login

Login is managed by the common SSO system used by Smarter Balanced (OpenAM/OpenDJ). A user is authenticated and authorized using the same, standard SAML mechanisms used by other Smarter applications (see Figure 1).

6.2 Logout of Current Session

The user interface provides a logout button which allows the user to logout. The functionality is provided by the Shared Security module.

6.3 Expire Session after a Period of Inactivity

A class called TdsAdminSessionListener is registered in the deployment descriptor which handles the session expiration time due to inactivity. The number of minutes before expiration is configured in TDSAdmin's Program Management config (see Section 2.1) and defaults to 15 minutes.