Medicaid Provider Screening Portal Services Design Specification

Author	Revision Number	Date
architect	1.0	August 10, 2012

Application Design Specification

1. Design

The United States Center for Medicaid Services ("CMS") is running a Challenge (a series of TopCoder contests) through the NASA Center of Excellence for Collaborative Innovation (CoECI) and TopCoder to develop a web-based portal to support provider enrollment and screening. Medicaid is a healthcare program that is funded by the US federal government and by individual states. It is operated by individual states. The portal developed in this Challenge will allow medical service providers to register themselves in their individual state for compensation from the Medicaid program. While this alone will provide a very welcome new procedure for providers, who must often fill out many paper forms today, the true goal of the portal will be to automatically screen potential providers for fraud risk before they are approved for Medicaid payments. The portal will assign a potential level of risk to each registrant, and will do this by applying a suite of dynamic business rules. Once a level of risk has been assigned, the registrant's application will be sent to the appropriate sub-system for additional screening and processing.

1.1 Work Flow Description

The services essentially provide management of user and provider/enrollment data, as services to secure the portal, export enrollment data, send notifications.

The primary focus of the app is to provide services to manage users, profiles, enrollments, roles, help, and provider types. There are also other services used by the application for auditing, lookup retrieval. All of these services use persistence to store the data.

Also, there are services to export enrollment data to a PDF document, send notifications, perform token-based security with Oracle OAM.

There are also services for performing background checks, verify business ownership, screen enrollments, check NPIs. These services interact with external services that are not in scope of this module.

The architecture is designed to fit into the Enterprise Service Bus, and conforms to MITA 3.0 standards as requested by the client. The EBS requires that services be able to be deployed in a flexible, independent manner to allow other services to use them in a transparent manner. To this end, all services are deployed as web services.

1.2 Component Requirements

1.2.1 TopCoder Software Components

1.2.1.1 Assemblies

- Medicaid Provider Screening Portal Services 1
  - Provides the all entities and exceptions.
  - Provides these services and their implementations, as well as parallel DAOs and their implementations
    - PermissionService
    - RoleService
    - LookupService
    - HelpService
    - ProviderProfileService
- Medicaid Provider Screening Portal Services 2
  - Provides these services and their implementations, as well as parallel DAOs and their implementations
    - AuditService

- UserService
- ExportService
- NotificationService
- ScreeningService
- NPIRegistryService
- SecurityService

1.2.1.2 *Existing Generic Components*

- Base Exception 2.0
  - Used for exceptions
- Logging Wrapper 2.0
  - Used for logging
- PDF Builder 1.0
  - Used for assembly of a PDF document
- JSON Object 1.0
  - Used for entity JSON serialization

1.2.2 *External Components*

- Spring 3.0.5: http://www.springsource.org
- Hibernate 3.6.3

1.3 Application Management

1.3.1 *Transactions*

The creation, update and deletion operations in the backend module must be transactional and thread-safe. EJB container transaction management will be used.

Each service class requiring transactions can be annotated like the following to manage transactions:

```
@TransactionManagement(value = TransactionManagementType.CONTAINER)
```

Each service method that performs any modifying actions will be annotated like the following to require a transaction:

```
@TransactionAttribute(value = TransactionAttributeType.REQUIRED)
```

1.3.2 *Threading*

The services will be effectively thread-safe. It can be assumed that the entities being persisted won't change during a persistence operation (for example, only one thread will work with a given Rule entity instance at any given time), but with that exception, the rest of the code will be able to be called from multiple threads.

1.3.3 *Configuration*

The service implementations will use setter injection for configuration with the use of Spring. The init method will be declared with @PostConstruct. The init method will throw PortalServiceConfigurationException if there is a configuration error.

1.3.4 *Persistence*

The services will use JPA backed by Hibernate to manage all data in persistence.

1.3.5 *Paging and Sorting*

The application services will provide the means to get lists of data that are pageable and sortable. The paging will require the provision of a page number and page size. The page number should be 1-based, with page of 0 meaning there should be no paging and all data should be returned. Page size will be a positive number.

Sorting requires the provision of the name of the column to be sorted, and the sort order. In most cases, the service will provide some default sorting, so the methods can be called with a null column and sort order, which would be interpreted to mean the sorting is to be whatever the service deems to be as default.

1.3.6 *Logging*

The application will log activity and exceptions using the LoggingWrapper.

It will log errors at Error level, and method entry/exit information at DEBUG level.

Specifically, logging will be performed as follows, if logging is turned on.

- Method entrance and exit will be logged with DEBUG level.
  - Entrance format: [Entering method {*className.methodName}*]
  - Exit format: [Exiting method {*className.methodName}]*. Only do this if there are no exceptions.
- Method request and response parameters will be logged with INFO level
  - Format for request parameters: [Input parameters[{request_parameter_name_1}:{ request_parameter_value_1}, {request_parameter_name_2}:{ request_parameter_value_2}, etc.)]]
  - Format for the response: [Output parameter {response_value}] . Only do this if there are no exceptions and the return value is not void.
- All exceptions will be logged at ERROR level, and automatically log inner exceptions as well.
  - Format: Simply log the text of exception: [Error in method {*className.methodName}*: Details {*error details*}]
  - The stack trace of the error and a meaningful message.

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry
3. Log method input parameters
4. If error occurs, log it and skip to step 7
5. Log method exit
6. If not void, log method output value
7. Method exit

All entities will provide a JSON version of their data, to be supplied in a toJSONString():String method. This will standardize the logging of input and out information of entities.

Note the application should not log sensitive data like user password, medical data etc...

1.3.7 *Exception Handling Overview*

The architecture defines a top-level exception PortalServiceException that all services will use.

For configuration, the services will use the PortalServiceConfigurationException if anything goes wrong during configuration.

For all service update and delete methods, if the affected entity is not found, EntityNotFoundException is thrown.

The exceptions have been defined in Exceptions Class Diagram.

1.3.8 Internationalization

None

1.3.9 Auditing

Several services/DAOs will be responsible for managing the auditable entities. In the create method, it will set the creation and update date to the timestamp of the action, and during the update, it will do this just to the update date.

The use of the AuditService will be done only by the clients of this module. No service uses AuditService directly.

1.3.10 Security

The authentication will be done with Oracle OAM. The user will be authenticated, and upon successful authentication, a session token will be obtained that will be used subsequently to authorize user actions.

The access to the site itself will be done using HTTPS. This will be configured in the web server.

1.3.11 Risks

There are no risks and the application doesn't use new, exotic software.

1.3.12 Scalability

None.

1.3.13 New technology

All used technologies are very stable and well known.

1.3.14 Performance

There are no specific performance issues

1.3.15 JPA search algorithms

This section shows the general algorithm for implementing the paged/sorted/filtered search methods. The first algorithm will be for the ProviderProfile search, and subsequent algorithms will state what changes for the other entity searches.

1.3.15.1 ProviderProfile search

This search is done in the ProviderProfileDAOBean.search method

1. Create result object: searchResults:SearchResult<ProviderProfile> = new SearchResult<ProviderProfile>()
1.1. Set page number: searchResults.pageNumber = criteria.pageNumber
1.2. Set page size: searchResults.pageSize = criteria.pageSize

2. Query for count of all entities for the provided filters (here there is no paging applied, and sorting is not relevant)
2.1. Create query string: entityCountQueryString:String = "SELECT count(entity) FROM ProviderProfile entity"
2.2. If at least one criteria parameter is provided then entityCountQueryString += " WHERE "
2.3. If criteria.enrollmentNumber provided then entityCountQueryString += "enrollmentNumber=:enrollmentNumber AND"
2.4. If criteria.providerType provided then entityCountQueryString += "entity.providerType.name=:providerType AND"
2.5. If criteria.requestTypes provided with at least one entry then entityCountQueryString += "(entity.requestType.name IN (" + comma-delimited list of the provided request types + ") AND"
2.6. If criteria.statuses provided with at least one entry then entityCountQueryString += "(entity.enrollment.status.name IN (" + comma-delimited list of the provided statuses + ") AND"
2.7. If criteria submissionDateStart provided, then entityCountQueryString += "entity.requestEffectiveDate>=:submissionDateStart AND"
2.8. If criteria submissionDateEnd provided, then entityCountQueryString += "entity.requestEffectiveDate<=:submissionDateEnd AND"
2.9. Remove the last "AND" from entityCountQueryString
2.10. Get query object: query:Query = entityManager.createQuery(queryString)
2.11. If criteria.enrollmentNumber provided then set this parameter: query.setParameter(":enrollmentNumber",enrollmentNumber)
2.12. If criteria.providerType provided then set this parameter: query.setParameter(":providerType",providerType)
2.13. If criteria submissionDateStart provided, then set this parameter: query.setParameter(":submissionDateStart",submissionDateStart)
2.14. If criteria submissionDateEnd provided, then set this parameter: query.setParameter(":submissionDateEnd",submissionDateEnd)
2.15. Execute query: results:List = query.getResultList()
2.16. Get total count: total:int = ((Integer)results.get(0)).intValue()

3. Query for relevant entities for the provided filters, page, and sorting
3.1. Create query string: entityQueryString:String = "SELECT entity FROM ProviderProfile entity"
3.2. If at least one criteria parameter is provided then entityQueryString += " WHERE "
3.3. If criteria.enrollmentNumber provided then entityQueryString += "enrollmentNumber=:enrollmentNumber AND"
3.4. If criteria.providerType provided then entityQueryString += "entity.providerType.name=:providerType AND"
3.5. If criteria.requestTypes provided with at least one entry then entityQueryString += "(entity.requestType.name IN (" + comma-delimited list of the provided request types + ") AND"
3.6. If criteria.statuses provided with at least one entry then entityQueryString += "(entity.enrollment.status.name IN (" + comma-delimited list of the provided statuses + ") AND"
3.7. If criteria submissionDateStart provided, then entityQueryString += "entity.requestEffectiveDate>=:submissionDateStart AND"
3.8. If criteria submissionDateEnd provided, then entityQueryString += "entity.requestEffectiveDate<=:submissionDateEnd AND"
3.9. Remove the last "AND" from entityQueryString but add the sorting: entityQueryString += "ORDER BY entity." + sortColumn + {criteria.ascending; " ASC": " DESC"}
3.10. Get query object: query:Query = entityManager.createQuery(queryString)
3.11. If criteria.enrollmentNumber provided then set this parameter: query.setParameter(":enrollmentNumber",enrollmentNumber)
3.12. If criteria.providerType provided then set this parameter: query.setParameter(":providerType",providerType)
3.13. If criteria submissionDateStart provided, then set this parameter: query.setParameter(":submissionDateStart",submissionDateStart)

3.14. If criteria submissionDateEnd provided, then set this parameter: query.setParameter(":submissionDateEnd",submissionDateEnd)
3.15. Execute query: searchResults.records = query.getResultList()
3.16. Set total page count: If page > 0, then searchResults.totalPages = (int)Math.ceil(total/searchResults.pageNumber); else searchResults.totalPageCount = 1

4. return searchResults

1.3.15.2 User search

This search is done in the UserDAOBean.search method.

This method will use the above algorithm, but will operate on the User entity instead of ProviderProfile. Then the search filters in steps 2 and 3 will be for filters in UserSearchCriteria.

1.3.15.3 HelpItem search

This search is done in the HelpDAOBean.search method.

This method will use the above algorithm, but will operate on the HelpItem entity instead of ProviderProfile. Then the search filters in steps 2 and 3 will be for filters in HelpSearchCriteria.

1.3.15.4 User search

This search is done in the ProviderTypeBean.search method.

This method will use the above algorithm, but will operate on the ProviderType entity instead of ProviderProfile. Then the search filters in steps 2 and 3 will be for filters in ProviderTypeSearchCriteria.

1.3.16 Validation

Input data will be validated as the first step in calling a method, and will throw IllegalArgumentException if validation fails.

No specific validation rules have been stated by system architecture, so only the rules for null objects, empty string, etc, are currently to be used (and these are defined in the method docs).

One set of more complex validation has to do with the creation and modification of the provider profile where the type's required fields are checked for presence.

SQL injection is prevented during search queries by using parameterized queries.

1.4 Deployment Constraints

The services will be deployed as an EAR on a Web sphere Application Server, and use JPA/Hibernate to access the database. Spring is used to give necessary configuration.

The complete technology overview is below.

The application can be deployable in JBOSS J2EE container. The Frontend tier will incorporate HTMLs with JSP(which is not in scope). This application will be packaged as a single jar file and used be frontend application. The services will be deployed on a JBoss Application Server, and use Hibernate to access the database. Spring is used to give necessary configuration.

1.4.1 Technology overview

- J2SE 1.6
- J2EE 1.6
- Spring 3.0.5
- Log4 1.2.17: http://logging.apache.org/log4j/1.2/
- Oracle 11g http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html
- JBoss 5.1.0.GA http://www.jboss.org/jbossas/downloads/
- JBoss ESB 4.10 http://www.jboss.org/jbossesb/downloads/
- Hibernate 3.6.3 http://planet.jboss.org/post/hibernate_core_3_6_3_release

1.5 Development Standards:

The assembly solutions must adhere to the guidelines as outlined here: http://apps.topcoder.com/wiki/display/tc/Assembly+Competition+Tutorial

1.6 Interfaces Classes Overview

See the TCUML file

1.7 Changes to Existing System

None

2. User Interface

Not specified.

3. Included Documentation

3.1 Architecture Documentation

- Assembly Diagram
- Class Diagrams
- Sequence Diagrams
- Application Design Specification
- Assembly Specifications
- ERD
- JPA mapping file

4. Future Enhancements

None