# KPMP Participant Portal – Application Design

**Components**

There are three deployed components in this system:

1. The Account Management page, used by research coordinators to administer accounts for participants at their site.
2. The Participant Registration page, which is used by participants to complete their registration and set their account password. This is also used later to reset the password as needed.
3. The Login system, including the login form, where participants log in to the system to gain access to the Participant Portal.

**Software Projects**

There are three software projects used to package the code. These projects do not correspond directly to the three components listed above (for reasons described below). Two of these are quite small, with most of the code in the second package described below.

1. **portal-enc**: contains two classes. This is used to encrypt on disk the passwords and access tokens used to connect to systems containing PHI. The build artifact for this project is a jar file containing the classes.
2. **portal-acc**: contains all the code for the Account Management page. Much of this code is reused by the Participant Registration page since that needs to connect to the same underlying data sources. This project also includes the password validation code used behind the login form. The build artifacts from this project are the war file for the Account Management page, as well as a jar file containing the Java code.
3. **portal-reg**: contains only the web pages used specifically for the Participant Registration page. The build artifact for this project is the war file for the Participant Registration page, which includes the jar from the previous project.

The mapping between deployed components and projects is as follows:

1. The Account Management page uses the war file from project #2, plus the encryption jar from project #1.
2. The Participant Registration page uses the war file from project #3, plus the jar files from projects #1 and #2.
3. The login page uses the jar file from project #2. In addition, the login system is deployed as a Shibboleth IDP installation with some custom views and configuration that define the login page and the workflow.

**Data Sources**

There are three data sources accessed by this system:

1. The KPMP User Portal contains data on users and provides an API to access this data. The Account management page makes an API call with the user’s Shibboleth identifier to get information on the user, including if they are authorized to access this page.
2. The KPMP Redcap site contains data on participants. It also contains the data access group for each user. The Account Management page queries this site for participant data, as well as to look up the data access group for the user. The Participant Registration page also accesses this site to verify participant data provided by the person registering for their account.
3. A local MySql database stores data specifically needed to manage participant accounts. Records for a participant are created in this database when a study coordinator uses the Account Management page to create an account for that participant. The Participant Registration page also reads and writes this database, and the login system reads the user’s password and writes audit data from/to this database.

**Account Management code design**

Since most of the code is in the account management project, this section describes the code components that exist in this project.

The source is organized as a standard Maven build. The directory src/main/java contains the Java code, while src/main/webapp contains the web application components. The pom.xml file defines the build configuration and dependencies.

There are five Java packages within this project:

* org.kpmp.portal: This package contains general shared classes. This includes data classes representing the user and participant, as well as various utilities.
* org.kpmp.portal.data: This package includes all the classes for data access to the three data sources listed above.
* org.kpmp.portal.accounts: This package contains the code that implements the business logic for the Account Management page. It includes the main controller as well as several helper classes.
* Org.kpmp.portal.registration: This package contains the controller for the participant registration page.
* org.kpmp.portal.passwords: This package contains the code for computing the hash of a plaintext password and for validating a login. This is used in the registration page to store the password and in the login system to validate the input password.

**Individual classes**

Package org.kpmp.portal:

* AccountsUser: represents the data for the user (study coordinator) of the Account Management page.
* Config: reads and makes available the values in the configuration file. This depends on the encryption library to decrypt the passwords and access tokens that are stored encrypted in the configuration file.
* ControllerUtils: defines various static utilities shared between the main controllers for the two web applications.
* DataModel: this data class contains an instance of the AccountsUser, Participant, and ParticipantAccount objects. An instance of this class is passed from the controller to the jsp as the data to be used to generate the user interface.
* JspException: used to log exceptions thrown by the jsp page, if any.
* ParticipantAccount: this represents the data representing the participant’s account in the local account database.
* Participant: this represents the participant data pulled from RedCap.

Package org.kpmp.portal.data:

* DataAccess: this is the main class used for data access. It includes all of the code for accessing the local account database. It delegates to the other classes in this package for calls to the User Portal and RedCap.
* ExportParticipantFromRedcap: contains the code to get the participant data from RedCap.
* ExportUserDataAccessGroup: contains the code to call RedCap to get the user’s data access group.
* ExportUserDataFromUserPortal: contains the code to get the user data from the User Portal.
* ParticipantEvent: this is a helper class used by ExportParticipantFromRedcap to address the fact that the RedCap data is stored in different events. An instance of this class represents one event as stored in RedCap.

Package org.kpmp.portal.accounts:

* AccountsController: this is the main controller that processes all requests for this webapp.
* Actions: this implements the business logic for handling each of the actions requested by the user, such as sending a registration email, deactivating an account, etc.
* EmailAction: this implements the logic for handling the send email action. It builds the message and then delegates to an instance of the Mailer class to send the message.
* Mailer: this implements the actual sending of a pre-generated email message.

Package org.kpmp.portal.passwords:

* PasswordHash: implements the logic for generating a random seed and for computing the hashed value of a password given a seed and a plaintext password. This class can be run as a stand-along application to test the password hash function.
* VerifyPassword: this implements the logic to check an input login id and password.

Package org.kpmp.portal.registration:

* RegistrationController: this is the controller for the registration webapp which processes participant registrations. This should probably be moved to the registration webapp; it is here for now so that all the code is in one place and can be revised more easily.

**Other files**

The src/main/webapp directory contains the files for the web application. The accounts directory contains accounts.jsp, which is the main page for the application. The other pages are error.jsp, displayed when an error occurs, and notAuthorized.jsp which is used when the application is accessed by an unauthorized user.

The WEB-INF directory contains several supporting files that are described further below.

Finally, the pom.xml file defines the build and dependencies on third-party libraries.

**Third Party libraries**

The application uses the Apache log4j2 libraries to log errors and other relevant information from the application. The configuration of the logging is controlled by the configuration file at src/main/webapps/WEB-INF/classes/log4j2.xml.

The application uses the Spring JDBC libraries to assist in the database connection, and the Spring webmvc library to assist with the control flow as described below.

The application depends on the MySql JDBC library to connect to the MySql database, and the Apache HTTP Core and HTTP Client libraries to connect to the RedCap and User Portal APIs. The Google gson library is used to parse the JSON data responses.

The final third-party dependency is the Java servlet library, which is deployed with Tomcat.

**Other projects**

As noted above, the encryption project contains two classes.

org.kpmp.portal.encryption.Encryption: implements the encryption of connection credentials. This is further described below in the deployment section.

Org.kpmp.portal.encryption.KpmpDataSourceFactory: this is used by Tomcat to create database connections to the MySql database; it used the encryption class to decrypt the encrypted password.

The registration project is organized similarly to the accounts project. The registration page which is as src/main/webapps/registration/registration.jsp. The webapp directory also contains the same files in WEB-INF that control the application as described below.

**Request handling**

The input URLs for the application are (in dev)

<https://accounts-dev.mydata.kpmp.org/accounts/accounts.htm> and

<https://registration-dev.mydata.kpmp.org/registration/registration.htm>

The handling of these is similar for both applications.

The Apache web server receives the request and delegates to the accounts or registration webapp on the Tomcat server. The web.xml file (in WEB-INF) declares that the X.htm patterns map to the Spring DispatcherServlet. In turn, the dispatcher-servlet.xml file (also in WEB-INF) maps the path to the specific controller class associated with the page (either AccountsController or RegisrationController).

The controller first prepares for the request by loading the application configuration and establishing a user session. It then uses the data loading classes to load data for the user, and for the participant if an appropriate identifier was requested. If the request is a post indicating the submission of a form, control is delegated to other code that detects which action was requested and performs that action. After performing the action, the ParticipantAccount object is reloaded since most actions update the account data. Finally, the user and participant data are bundled into a DataModel object and passed to the jsp. The jsp builds the user interface using core jsp tags for conditional logic and calls methods on the model to get the data for display.

**Build**

All projects use the standard Maven Central repository for third party libraries.

First, build the accounts project using the standard ‘mvn package’ command. This will produce accounts-X.Y.war and accounts-X.Y.jar in the target directory. The jar file should then be installed in the local Maven repository so it can be included in the registration project:

cd $HOME/.m2/repository

mvn install:install-file -Dfile=${ACCOUNTS.HOME}/target/accounts-X.Y.jar -DgroupId=org.kpmp.portal -DartifactId=accounts -Dversion=X.Y -Dpackaging=jar -DgeneratePom=true

In the command above, fill in the path to the project home and the current version number.

Next, the registration project can be built with the same build command, producing target/registration-X.Y.war.

The encryption project is built using the standard ‘mvn package’ command. This will generate target/encryption-X.Y.jar, where X.Y is the version number. This is only needed if encrypted passwords/access tokens are used. If used, it should be installed in the Tomcat lib directory.

**Deployment**

The war file for each webapp should be copied to the Tomcat webapp directory.

After this, the supporting configuration files must be put in place: see below.

There are two different modes of deployment. The normal case is that user and participant data are loaded from RedCap and the user portal. In this case, the URLs for those connections and the API access tokens must be defined in the configuration file. Also, an encryption library must be built and installed so that the application can decrypt the secure tokens that are stored encrypted in the configuration file.

Alternatively, to support local development and testing, the user and participant data can be loaded from text files instead of the data repositories. In this case the configuration file must contain the paths to the data files. In this case the encryption library is not required, however Tomcat must be configured with the plaintext password used to connect to the MySql database.

**Supporting Tomcat files**

${tomcat.home}/conf/portalConfig.prop: this is the application configuration file in standard Java properties format. The supported values are defined below.

${tomcat.home}/conf/context.xml: this file contains the connection parameters for the connection to the MySql account database.

${tomcat.home}/lib/encryption.jar: This is the jar file for the encryption library as described above. It must be installed here as Tomcat uses this to decrypt the encrypted database password in context.xml.

${tomcat.home}/conf/keystore: this file contains the encryption key used to decrypt connection passwords.

**Application configuration file**

There are two versions of the application configuration file. The first version is used when connecting to secured data repositories and contains the encrypted access passwords/tokens needed to connect to those resources. This mode of operation requires the encryption module to be installed in order to decrypt the encrypted values.

The second version of the configuration file is used when using local data files for the user and participant data. This version does not require the encryption library but does require paths to the files that hold the data.

Both versions are a standard Java properties file where each line is of the form

KEY=VALUE

**Configuration file for secure access**

The following are the fields in the configuration file used when connecting to secure repositories.

KEYSTORE\_PATH=<path to the keystore holding the encryption key; see above)

REDCAP\_API\_URL=<URL for the RedCap API>

REDCAP\_API\_TOKEN=<the encrypted token to connect to REeCap>

USERPORTAL\_API\_URL=<URL for the user portal>

USERPORTAL\_API\_TOKEN=<the encrypted token to connect to RedCap>

**Configuration file for data files**

The following are the fields in the configuration file used when reading user/participant data from files.

DATA\_FROM\_FILES=true

ACCOUNTS\_USER\_FILENAME=<path to file containing user data>

PARTICIPANT\_FILENAME=<path to file containing participant data>

The specified files then should contain a user object and a participant object in JSON format. Sample data content is included below, noting that if any fields are added to the user or participant objects, the data files would also need to updated to include those new fields.

AccountsUser:

{"shibId":"markiel@uw.edu","email":"markiel@uw.edu","firstName":"Andrew","lastName":"Markiel","isAccessAllowed":true,"isAdmin":true,"dataAccessGroup":"testGroup"}

Partiicpant:

{"studyId":"1","firstName":"Test","lastName":"Participant","dateOfBirth":"1970-01-01","emailAddress":"test@kpmp.org","emailWho":"1","emailOwner":"","dataAccessGroup":"testGroup"}

Note that this use case will still require credentials to connect to a local MySql account database. It is assumed that this would occur in a non-prod domain where those credentials could be stored unencrypted in the context.xml file.

**Generating the keystore and encrypted strings**

This section describes how to generate the encryption key used to encrypt values, and how to calculate the encrypted values to put in the configuration file.

The encryption key is generated with the following command:

keytool -genseckey -alias kpmp-portal -keyalg AES -keysize 128 -storetype PKCS12 -keystore ${tomcat.home}/conf/keystore

This command will prompt for a keystore password. The password is hard-coded in the code for the Encryption class. In this way an attacker would need access to both the source code and the installation directory in order to get the encryption key.

Once the key is generated, a string is encrypted using the following command:

java org.kpmp.portal.encryption.Encryption ${tomcat.home}/conf/keystore encrypt

This command will prompt for the string to encrypt and will output the encrypted value.