BBRB Biorepositories and Biospecimen Research Branch

leidos
Leidos Biomedical Research, Inc.

IT-9012 Comprehensive Data Resource-Lite (CDR-Lite) Architecture & Design

Version 1.01

11/7/2016

Version History

This document combines information from a variety of existing sources in describing the design and context under which the Comprehensive Data Resource operates. This document is consistent with version 1.0 of the CDR Lite.

Version	Author	Effective Date	Description of Change
1.00	William Lander	11/4/2016	Initial Version
1.01	William Lander	11/7/2016	Administrative Update

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The CDR-Lite is an open-source extension of the Comprehensive Data Resource (CDR), intended for researchers to customize to record the metadata around specimen and tissue creation, processing, and storage.

The CDR was developed to meet the challenges of collection of data regarding tissues gathered in the early stages of the biospecimen lifecycle.[1] These data include information about potential candidates; their eligibility criteria and consent; medical/surgical procedures used; and acquisition handling, processing, and storage. As the focus for biospecimen-based studies of cancer has turned to the molecular level, applying stringent specimen annotation techniques in the lifecycle is more important than ever.

The initial design of the CDR reflects the changing requirements of real-world data collection.

- As the collection process progresses, the need for different or more detailed information may arise. Changing needs necessitate rapid changes (on the order of days) in forms, Web pages, workflows, and database structure.
- The collection process must support various review cycles, including reviews for data entry or processing errors (data management) and specimen quality (both internal and external pathology review).
- The process must support geographic separation between those working on the supported projects. The collection and processing sites may change during the course of the project.
- To minimize the learning curve for users, the interface must reflect the historical paper-based forms process.

The first project producing requirements for the CDR was the National Institutes of Health (NIH) Common Fund's Genotype-Tissue Expression (GTEx) program.[2] GTEx collected 961 cases with more than 24,000 individual specimens.

The CDR code was later extended to cover a second project, the Biospecimen Pre-Analytical Variables (BPV) Program.[3] BPV collected information on more than 300 cases, with specimens covering four different cancer types (and associated normal tissue) from four different tissue source sites (hospitals and university medical centers).

[1] http://biospecimens.cancer.gov/researchnetwork/lifecycle.asp

[2] https://commonfund.nih.gov/GTEx/index

[3] http://biospecimens.cancer.gov/programs/bpv/default.asp

Following the successful execution of the BPV, GTEx, and other projects, NCI BBRB made the decision for extracting the features of the CDR that might lend themselves to other studies. The purpose of the CDR-Lite is to allow researchers to take advantage of the best practices of biospecimen collection that have been worked out over the course of several years, with much expert consultation, and apply the software in their own environments without a lengthy development cycle.

1.1 Purpose of This Document

This document describes the CDR-Lite's architecture, giving the teams at each implementing institute guidance on architecture if the need arises for further changes, updates, or extensions. Its intended audience is the project manager, project team, and development team. Some portions of this document, such as the user interface (UI) (see 5.3), may on occasion be shared with users and with other stakeholders whose input on or approval of the UI is needed.

2 High-Level View of CDR-Lite

2.1 CDR-Lite Capabilities

The CDR-Lite is a Web application, custom-built to support specimen collection, clinical data entry, coordination of specimen logistics, and curation of study data. Its capabilities reflect the best practices for a good out-of-the-box study management system and include the following:

- Allow remote users (e.g., researchers, clinicians, support staff) to securely enter, revise, and review data about biospecimen collection through a standard (HTTPS) Web interface using a series of electronic forms with a sophisticated role-driven workflow
- Trigger responses to automatically communicate timely information to project managers and data analysts
- Assist with quality assurance by auditing process flows through data management and pathology teams
- Control display of personally identifiable information (PII) based on user entitlements and roles

2.2 Using the CDR-Lite

Users are provided with accounts based on predefined study roles.

Setting up a study does not require custom coding and configuration. The CDR-Lite's core data model is robust and can be extended to accommodate new studies. The level of effort needed to modify the CDR-Lite will vary with a new project's requirements.

Grails was chosen as the implementation language because it reduced the time needed in development by supporting automated generation of Web interfaces and automated management of the database schema, persistence, and searching. No database administrators are needed once the supporting Oracle or PostgreSQL instance is up and available; changes in the schema are managed internally by Grails Object Relational Mapping (GORM). All database access is based on Hibernate 3, which manages table definition updates as well as content updates.

3 General Overview and Design Guidelines/Approach

This section describes the principles and strategies to use as guidelines when modifying CDR-Lite.

3.1 Assumptions, Constraints, and Standards

The CDR-Lite architecture is designed for maximum flexibility to meet changing requirements. As such, it is constrained to use standard services (e.g., RESTful Web service interfaces, XML, JSON data exchanges).

The CDR-Lite is designed to be compliant with the Health Insurance Portability and Accountability Act (HIPAA) with a limited data set (LDS).[4] In the government arena, this greatly simplifies such things as security approval and Federal Information Security Modernization Act (FISMA) compliance. The architecture reflects the need to store PII and protected health information (PHI) in an environment where not all users have a need to know all information. For this reason, it includes a variety of roles that can be assigned to users, limiting their ability to access sensitive information. Section 4.5 discusses this aspect of the architecture in detail.

[4] http://www.hopkinsmedicine.org/institutional_review_board/hipaa_research/limited_data_set.html

4 Architecture Design

The CDR-Lite is an enterprise-level application built around the motto "Science First." The technologies chosen to build the CDR-Lite, including the open-source Grails framework and other proven open-source technologies, facilitated a faster development time through rapid and agile software methods.

4.1 Logical View

The CDR-Lite provides secure user access to case and biospecimen sample data based on pre-defined roles and privileges. It uses dynamic content redaction to restrict PII and PHI to an LDS with access given only to authorized users. Intuitive graphic UIs for the biospecimen source sites (BSSs) streamline data entry workflow by strictly following standard operating procedures (SOPs) for sample collection and processing. Contextual automated data checks and business rule validations confirm data integrity and adherence to biospecimen collection and preservation SOPs.

Users of the Web forms interface at collaborating institutions are granted access through application-specific user accounts to enter data and access existing data entered under their institutions' activities. BSS, Administrative, and Data Manager roles are assigned as needed.

4.2 Hardware Architecture

The CDR-Lite is hosted on two virtual machines located at the Frederick National Laboratory for Cancer Research. As shown in Figure 1, separate virtual machines are used to host the CDR-Lite and the Apache front-end. This architecture was chosen as a best practice. In a typical Web application deployment, the Apache server would be isolated in the DMZ with the Tomcat and database assets behind the firewall.

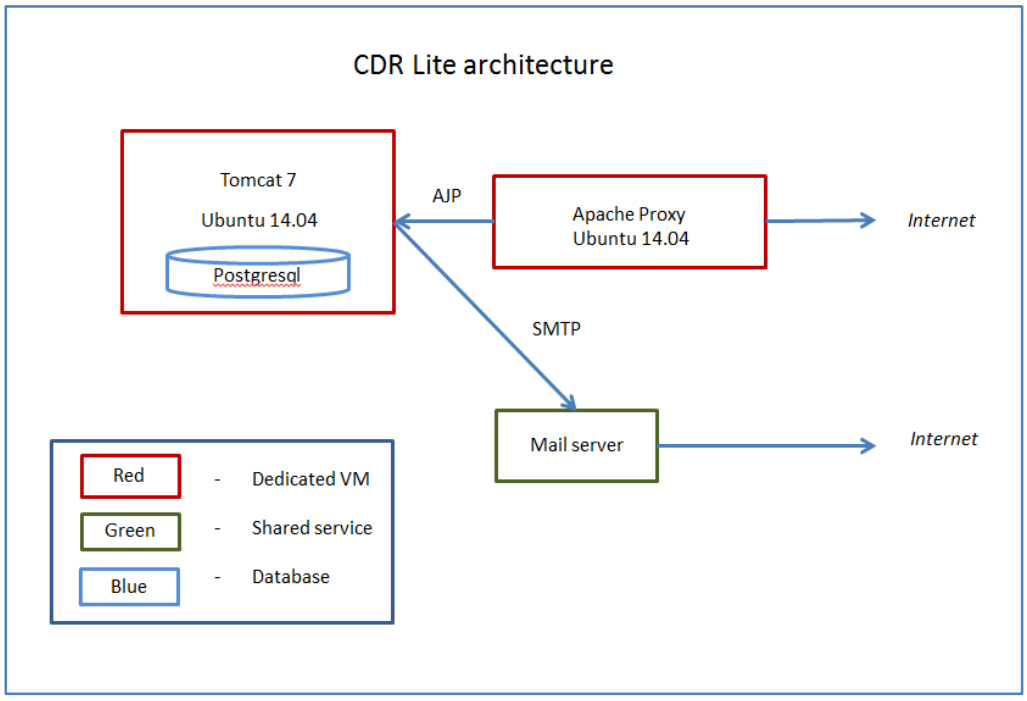

Figure 1: Virtual Machine Network Architecture

The Simple Mail Transfer Protocol (SMTP) mail server is a shared mail server for all of NIH and is not dedicated to CDR-Lite activities. Mail alert messages are sent to various email groups when associated

events are detected in the CDR-Lite. This messaging is discussed in more detail in sections 4.6.2 and 5.3.2.

Table 1 shows the current virtual machine configurations for the CDR-Lite.

Table 1: Current Virtual Machine Configurations

VMWare ESX Host	RAM	CPU	Disk storage capacity
CDR-Lite	4,096 MB	2—Westmere Xeon Core i7	68 GB
Apache	2,048 MB	1—Westmere Xeon Core i7	15 GB

4.3 Software Architecture

Grails provides the software architecture as a framework, shown in Figure 2 and Figure 12. The definition of Grails is maintained at http://grails.github.io/grails-doc/latest/guide/spring.html. Like other Java languages, Groovy, the compiled Grails language, runs in a Java virtual machine. For Web applications, this Java virtual machine is shared with the Tomcat application container. Spring Security supports elements of the enterprise service layer and interactive UI; SiteMesh is a decorator engine, supporting view layouts in generating Web page displays.

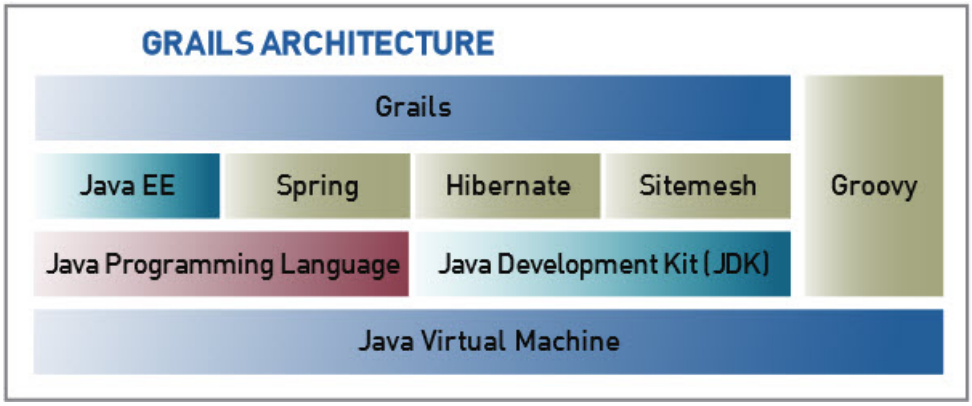

Figure 2: Grails Framework Architecture

4.4 CDR-Lite Domain Class Model

The heart of the CDR-Lite is its domain model, the set of domain classes and their relationships. Domain classes hold the persistent objects in the database. They are linked together through relationships: one-to-one, one-to-many, or many-to-many. In Figure 2, these domain classes reside in the Grails level.

A domain class represents persistent data and, by default, is used to create a table in a database. The name of the domain class (the “model” in model-view-controller [MVC]) is the same as the names of the corresponding controller and views. The location of the file in the CDR-Lite directory structure, along with the domain class name, gives the intent of the code in a file. One of the development benefits of Grails is that, by default, all the capabilities of a basic CRUD (create, read, update, and delete)

application are already available. When a domain class is defined in NetBeans, right click on it, and choose "generate all." Grails will generate the controller and default views for your domain class.

Table 2 enumerates and describes the domain classes making up the CDR-Lite.

Table 2: CDR-Lite Domain Classes

Package Name	Domain Classes	Description
cdrlite		
	ErrorsController.groovy	Customizable error handler generated by Grails; not modified for the CDR-Lite
nci\bbrb\cdr\authservice		
	CdrRole.groovy CdrUser.groovy CdrUserRole.groovy	CDR users and roles generated by Spring Security plugin and renamed to "CdrUser," "CdrRole," and "CdrUserRole." In Oracle, "user" is a reserved word, and using it as a table name causes problems.
nci\bbrb\cdr\datarecords		
	CandidateRecord.groovy	Candidate record
	CaseRecord.groovy	Case record
	ImageRecord.groovy	Image record (for whole-slide images)
	PhotoRecord.groovy	Photo record (for gross tissue preparation)
	ProcessingEvent.groovy	Processing event
	SlideRecord.groovy	Slide record
	SpecimenRecord.groovy	Specimen record
nci\bbrb\cdr\forms		Forms package for the CDR-Lite
	CancerHistory.groovy	Cancer History form
	ClinicalDataEntry.groovy	Clinical Data Entry form
	ConsentVerification.groovy	Consent Verification form
	Demographics.groovy	Demographics form
	GeneralMedicalHistory.groovy	General Medical History form
	HealthHistory.groovy	Health History form
	MedicationHistory.groovy	Medication History form
	ScreeningEnrollment.groovy	Screening and Enrollment form

Package Name	Domain Classes	Description
	SlidePrep.groovy	Slide Prep form
	SlideSection.groovy	Slide Sectioning and Staining form
	SocialHistory.groovy	Social History form
	SurgeryAnesthesia.groovy	Surgery Anesthesia form
	TherapyRecord.groovy	Therapy Record form
	TissueGrossEvaluation.groovy	Tissue Gross Evaluation form
	TissueProcessEmbed.groovy	Tissue Processing and Embedding form
	TissueReceiptDissection.groovy	Tissue Receipt and Dissection form
nci\bbrb\cdr\forms\blood		A special package for the blood form
	Aliquot.groovy	Blood aliquots
	Blood.groovy	The main Blood form
	CollectionTube.groovy	Collection tubes
	Draw.groovy	Blood draws
nci\bbrb\cdr\prc		PRC Reports
	PrcReport.groovy	PRC Report form
nci\bbrb\cdr\staticmembers		Static members are where the “controlled vocabulary” items are stored, maintained, and modified. These are usually presented to the user in forms as a drop-down list. The contents can be modified in the Back Office (available only to Admins).
	ActivityType.groovy	Various Activity Types can trigger events, such as sending an Alert email.
	BloodAliquotType.groovy	Used for the Blood form
	BloodCollectionReason.groovy	Used for the Blood form
	BloodDrawTech.groovy	Used for the Blood form
	BloodDrawType.groovy	Used for the Blood form
	BloodTubeType.groovy	Used for the Blood form
	BSS.groovy	Static list of BSSs
	CaseAttachmentType.groovy	Attachment Types for files uploaded and attached to a case
	CaseCollectionType.groovy	Collection type

Package Name	Domain Classes	Description
	CaseStatus.groovy	Case status
	ContainerType.groovy	Container type for specimen collection
	Fixative.groovy	Fixative for specimen collection
	Organization.groovy	Organization; some organizations are BSSs
	PrcAcceptability.groovy	PRC acceptability status
	QueryStatus.groovy	Query status for the query tracker
	QueryType.groovy	Query type
	StorageTemp.groovy	Storage temperature
	Study.groovy	Study
	TissueCategory.groovy	Tissue category
	TissueLocation.groovy	Tissue location
	TissueType.groovy	Tissue type
nci\bbrb\cdr\util		Utility domain classes for the CDR-Lite
	ActivityEvent.groovy	Holds a record of activities that trigger events
	AppSetting.groovy	Dynamic application settings that contain, for example, the Login Bulletin message to display and lists of users who can receive email event notifications
	FileUpload.groovy	Files uploaded and their path on the server
	UserLogin.groovy	Record of users logged in and login history
nci\bbrb\cdr\util\querytracker		
	Deviation.groovy	Used to record deviations from approved SOPs
	Memo.groovy	Memos attached to a case record for approved changes
	Query.groovy	Query records of data management activities resolving data discrepancy issues
	QueryAttachment.groovy	File attachments to a query
	QueryResponse.groovy	BSS responses to queries

The CDR-Lite Grails solution uses a Spring MVC Web application framework. Spring MVC is an extensible MVC, making it perfect for Grails. The Grails servlet extends Spring's DispatcherServlet to bootstrap the Grails environment; then a single Spring MVC controller called org.codehaus.groovy.grails.web.servlet.mvc.SimpleGrailsController handles all Grails controller requests.

The SimpleGrailsController delegates to a class called org.codehaus.groovy.grails.web.servlet.mvc.SimpleGrailsControllerHelper that actually handles the request. This class breaks the handling of the request down into a number of steps. The entry point for the class is the handleUri method, which uses the following steps:

1. Parse the universal resource identifier (URI) into its components (controller name, action name, id, etc.).
2. Look up a GrailsControllerClass instance for the URI (see Figure 4).
3. Create a new Grails controller instance.
4. Configure the controller instance's dynamic methods and properties.
5. Retrieve a scaffolder if scaffolding is enabled for the controller.
6. Get a reference to the closure action to execute for the URI.
7. Increment flash scope, moving the scope on to its next state.
8. Get the view name for the URI.
9. Execute any interceptors that have been registered (e.g., Spring Security).
10. Execute the closure that is the controller action if the "before" interceptor did not return false.
11. Create a Spring MVC ModelAndView instance from the view name and the model returned by the closure action.
12. Execute any "after" interceptors registered, passing the returned model to the interceptor.
13. Return the Spring MVC ModelAndView instance.

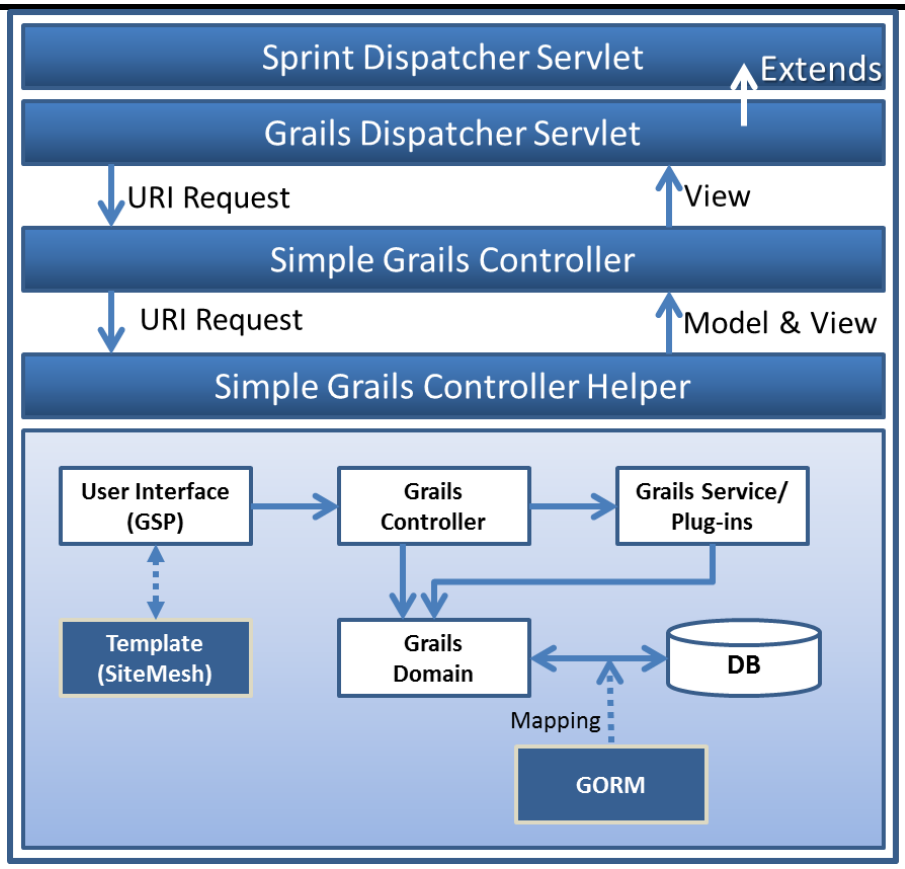

Figure 3: Grails Framework in Context of Spring MVC

CDR-Lite Grails is implemented as a standard Grails 2.4.4 project. CDR-Lite forms are implemented as Groovy Server Pages (GSPs); data records and static members are Grails domain classes (see Table 2); and XML interfaces are defined as Grails Services. In Figure 5, the directory structure is described, showing how the elements of the CDR-Lite map into Figure 3.

The configuration file, shown in Figure 4, controls which roles have access to which controllers. The restrictions that this configuration file imposes limit data access by both roles and controllers.

Any URL not in this list is denied.

```
/appSetting/** etc., is the URL
['ROLE_ADMIN','ROLE_SUPER'], is the Role
grails.plugin.springsecurity.controllerAnnotations.staticRules = [
   //system setting controllers
   '/user/**': ['ROLE_ADMIN','ROLE_SUPER'],
   '/role/**': ['ROLE_ADMIN','ROLE_SUPER'],
   '/userRole/**': ['ROLE_ADMIN','ROLE_SUPER'],
   '/securityInfo/**': ['ROLE_ADMIN','ROLE_SUPER'],
   '/controllers.gsp':['ROLE_ADMIN','ROLE_SUPER'],
   '/backoffice/**':['ROLE_ADMIN','ROLE_SUPER'],
   '/auditLogEvent/**':['ROLE_ADMIN','ROLE_SUPER'],
   '/userLogin/**': ['ROLE_ADMIN','ROLE_SUPER','ROLE_DM'],
   '/privilege/**':['ROLE_ADMIN','ROLE_SUPER','ROLE_DM','ROLE_PRC','ROLE_LDS'],
   '/tissueType/**':['ROLE_ADMIN','ROLE_SUPER','ROLE_DM'],
```

```
    //leave these alone. these rules are needed for everything to work properly.
    '/login/**': ['IS_AUTHENTICATED_ANONYMOUSLY'],
    '/logout/**': ['IS_AUTHENTICATED_FULLY'],
    '/register/**': ['IS_AUTHENTICATED_ANONYMOUSLY'],
    '/plugins/**': ['IS_AUTHENTICATED_ANONYMOUSLY', 'IS_AUTHENTICATED_FULLY'],
    '/images/**': ['IS_AUTHENTICATED_ANONYMOUSLY'],
    '/css/**': ['IS_AUTHENTICATED_ANONYMOUSLY'],
    '/js/**': ['IS_AUTHENTICATED_ANONYMOUSLY'],

    //webapp controllers
    '/home/**': ['IS_AUTHENTICATED_FULLY'],
    '/appSetting/**': ['IS_AUTHENTICATED_FULLY'],
    '/caseRecord/**': ['IS_AUTHENTICATED_FULLY'],
    '/candidateRecord/**': ['IS_AUTHENTICATED_FULLY'],
    '/specimenRecord/**': ['IS_AUTHENTICATED_FULLY'],
    '/slideRecord/**': ['IS_AUTHENTICATED_FULLY'],
    '/study/**': ['IS_AUTHENTICATED_FULLY'],
    '/organization/**': ['ROLE_DM', 'ROLE_ADMIN'],
    '/bss/**': ['IS_AUTHENTICATED_FULLY'],
    '/user/**': ['IS_AUTHENTICATED_FULLY'],
    '/role/**': ['IS_AUTHENTICATED_FULLY'],
    '/activityType/**':['IS_AUTHENTICATED_FULLY'],
    '/activityEvent/**': ['IS_AUTHENTICATED_FULLY'],
    '/activitycenter/**': ['IS_AUTHENTICATED_FULLY'],
    '/textSearch/**': ['IS_AUTHENTICATED_FULLY'],
    '/textSearch/index_all': ['ROLE_ADMIN'],
    '/query/**':['ROLE_BSS_UUU','ROLE_BSS_CCC',
  'ROLE_DM','ROLE_SUPER','ROLE_ADMIN','ROLE_ORG_VARI','ROLE_ORG_BROAD','ROLE_ORG_MBB'],
    '/fileUpload/**': ['ROLE_ADMIN','ROLE_BSS', 'ROLE_DCC'],
    '/caseAttachmentType/**':['ROLE_ADMIN'],
    '/prcReport/**': ['ROLE_PRC','ROLE_ADMIN'],
    '/prcReport/view': ['ROLE_PRC','ROLE_ADMIN', 'ROLE_DCC'],
    '/healthHistory/**': ['ROLE_BSS', 'ROLE_DCC'],
    '/socialHistory/**': ['ROLE_BSS', 'ROLE_DCC'],
    '/surgeryAnesthesia/**': ['ROLE_BSS', 'ROLE_DCC','ROLE_ADMIN'],
    '/tissueGrossEvaluation/**': ['ROLE_BSS', 'ROLE_DCC','ROLE_ADMIN'],
     '/generalMedicalHistory/**': ['ROLE_BSS', 'ROLE_DCC'],
     '/cancerHistory/**': ['ROLE_BSS', 'ROLE_DCC'],
     '/medicationHistory/**': ['ROLE_BSS', 'ROLE_DCC'],
     '/screeningEnrollment/**': ['ROLE_DCC','ROLE_BSS'],
     '/consentVerification/**': ['ROLE_DCC','ROLE_BSS'],
     '/demographics/**': ['ROLE_DCC','ROLE_BSS'],
     '/blood/**': ['ROLE_DCC','ROLE_BSS'],
     '/tissueReceiptDissection/**': ['ROLE_DCC','ROLE_BSS'],
     '/rest/**': ['ROLE_ADMIN']
  ]
```

Figure 4: Spring Security Configuration File `Config.groovy`, Showing Roles and Restrictions

Figure 5 provides a high-level view of the directory structure. This is basically an extension of the default Grails directory structure.

```
\---cdrlite
    +---grails-app
    |   +---conf
    |   +---controllers
    |   +---domain
    |   +---i18n
    |   +---jobs
    |   +---services
    |   +---taglib
    |   +---utils
    |   \---views
    +---lib
    +---scripts
    +---src
    \---web-app
```

Figure 5: High-Level View of CDR-Lite Directory Structure

The paragraphs below describe the conf, controllers, domain, jobs, and services directories.

The `cdrlite/web-app` directory is the location of the standard directory structure defined in the J2EE specification.

The `conf` directory, shown in Figure 6, stores various files used in configuring the CDR-Lite and the data source (`DataSource.groovy`). When needed, other configuration files go in this directory. Grails removes the need to add configuration in XML files for the CDR-Lite. Instead, the framework uses a set of conventions while inspecting the code of Grails-based applications. For example, a class name that ends with "Controller" (e.g., `LoginController`) and is in the `Controllers` folder is a controller. The controllers related to base classes are stored in the `Controller` directory, shown in Figure 7.

```
    +---grails-app
    |   +---conf
    |   |   |   BootStrap.groovy
    |   |   |   BuildConfig.groovy
    |   |   |   Config.groovy
    |   |   |   DataSource.groovy
    |   |   |   DefaultQuartzConfig.groovy
    |   |   |   GrailsMelodyConfig.groovy
    |   |   |   QuartzBootStrap.groovy
    |   |   |   Searchable.groovy
    |   |   |   SecurityFilters.groovy
    |   |   |   UrlMappings.groovy
    |   |   |
    |   |   \---spring
    |   |           resources.groovy
```

Figure 6: CDR-Lite `conf` Directory Details

Figure 7 shows the `controllers` directory, which stores various files mapping between the database model and the views displayed in the Web interface. Grails uses controllers to implement the behavior of Web pages. For each CDR-Lite domain class, the associated controller file has the domain name concatenated with "Controller" and is of type .groovy. For example, the login controller is `LoginController.groovy`. Table 3 shows controllers unrelated to domain classes.

```
    \---cdrlite
        +---grails-app
        |   +---controllers
        |   |   \---nci
        |   |       \---bbrb
        |   |           +---cdr
        |   |           |   +---datarecords
        |   |           |   +---forms
        |   |           |   |   \---blood
        |   |           |   +---prc
        |   |           |   +---staticmembers
        |   |           |   \---util
        |   |           \---cdrlite
```

Figure 7: CDR-Lite `controllers` Directory Structure

Table 3: Controllers Not Associated with Domain Classes

Controller	Description
`backoffice`	Supports monitoring of system functioning, activities, and RESTful services
`errors`	Supports the "Page not found" screen that appears when requested information is not available
`help`	Supports the system help screens that are available to users
`helpFileUpload`	Supports the upload of user manuals, SOPs, and other documents for the users
`home`	Supports the page that users see when they log in. This home page is a function of the individual user and their role.
`login`	Supports the authentication of users when they attempt to access the website
`logout`	Destroys the session
`userLogin`	Supports the reporting on users such as the information stored about those users, such as affiliation, and when they last logged in

The `grails-app/domain` directory, shown in Figure 8, stores the domain classes. Table 2 describes the individual domain classes. Each file is a .groovy file, and the file name is the name of the domain class. The directory structure divides the files into core functionality (e.g., data records, pathology resource center (prc), utility (util)) and forms. GORM manages all domain classes found in the domain. Methods are dynamically added to aid in persisting the class's instances. The files in Figure 8 map into the "Grails Domain" box in Figure 3.

```
\---cdrlite
    +---grails-app
    |   +---domain
    |   |   \---nci
    |   |       \---bbrb
    |   |           \---cdr
    |   |               +---authservice
    |   |               +---datarecords
    |   |               +---forms
    |   |               |   \---blood
    |   |               +---prc
    |   |               +---staticmembers
    |   |               \---util
```

Figure 8: CDR-Lite `domain` Directory Structure

Figure 9 shows a collection of directories defined in the Grails scaffolding.

```
\---cdrlite
    +---grails-app
        +---i18n
        +---jobs
        |   \---cdrlite
        +---services
        |   \---nci
        |       \---bbrb
        |           \---cdr
        +---taglib
        |   \---cdrlite
        +---utils
        \---views
```

Figure 9: CDR-Lite General Purpose Directories

The `grails-app/i18n` directory holds internationalization information for the website. This directory contains the file `messages.properties`, which contains settings for error messages used in data validation. Only English-language messages are available.

The `grails-app/jobs/cdrlite` directory contains Groovy scripts runnable on a periodic basis ("cron" jobs) via the Quartz package. These scripts are listed in Table 4: CDR-Lite Quartz Scheduler Jobs.

CDR-Lite Quartz Jobs	Description
`TextIndexJob.groovy`	Updates indexing of text fields for Lucene searching

The `grails-app/services/nci/bbrb/cdr` directory is the repository for the CDR-Lite Grails service classes. A service class is a Plain Old Groovy Object (POGO), frequently with a name starting with

a domain class and ending with "Service.groovy." These service classes are Spring-loaded Groovy beans. Table 5 describes the CDR-Lite services that are not associated with a domain class.

Table 5: Service Classes Not Associated with a Single Domain Class

Service	Description
`AccessPrivilegeService`	Supports Spring in checking the access privileges of the user and role
`ActivityEventService`	Implements user email notification when various events take place (such as a case collected)
`BloodService`	Supports the Blood form
`CandidateService`	Supports creation of candidates
`CaseStatusService`	Supports setting and changing of case status
`HealthHistoryService`	Supports saving the Health History form
`HubIdGenService`	Generates unique identifiers
`LocalPathReviewService`	Supports saving the Local Pathology Review form
`PrcReportService`	Supports the PRC Report form
`ProcessingService`	Supports shipping and processing events as XML payloads when they are received from Van Andel via REST over HTTPS
`QueryService`	Supports the query tracker
`SendMailService`	Supports email notification of users when triggers occur
`SlideSectionService`	Supports the Slide Sectioning form
`TextSearchService`	Supports the Lucene searches of records
`TissueGrossEvaluationService`	Supports the Tissue Gross Evaluation form

The `grails-app/taglib/cdrlite` directory, shown in Table 6, contains custom Groovy code that dynamically generates the HTML associated with GSP tags in the forms. These tags take the form `<g:{tagname} {attributes} />` in the GSP pages.

Table 6: Custom Groovy Tags

Classes for Implementing Dynamic Customized Code	Description
`CaseRecordLinkTagLib.groovy`	Implements customized GSP tags for the Case Record display with a link to the CaseRecordController by ID
`JqueryDatePickerTagLib.groovy`	Implements a customized date picker
`MedicationAdminTagLib.groovy`	Implements customized GSP tags for the BPV Medication display
`QueryTagLib.groovy`	Implements customized GSP tags for the query tracker
`RadioButtonTagLib.groovy`	Implements customized GSP tags for single-select

Classes for Implementing Dynamic Customized Code	Description
	buttons

All contents of the directory `grails-app/util` are deprecated.

The `grails-app/view`, shown in Table 7, has one sub-directory for each domain class. Each entry under that directory is one or more GSP (.gsp) files. Each file's name describes a method in the corresponding controller class. Table 7 lists the typical methods that are automatically generated whenever you create a domain class (you can create other views and controller methods; the names *should*, by convention, match).

Table 7: Typical/Default Grails Server Pages

Grails Server Page	Description
`create.gsp`	Produces a UI to enter all values for a domain class and creates a new instance
`edit.gsp`	Produces a UI to change the values of an instance of a domain class
`index.gsp`	The default method for a domain class, like `index.html`
`list.gsp`	Produces a list of persistent domain class objects
`show.gsp`	Produces a UI showing the contents of an instance of the domain class
`_form.gsp`	An include file with all the domain class attributes, used by the `create.gsp` and `edit.gsp` files

Figure 10 shows the `cdrlite/src` directory containing Groovy, Java, and template code.

The `cdrlite/src/groovy/nci/bbrb/cdr` directory contains one file, `CDRBaseClass.groovy`, which is the base class for all CDR-Lite Groovy classes. The CDR-Lite developers use it to create an abstract class that the domain classes have all inherited. Extending the `CDRBaseClass` when implementing a domain class causes the domain class to inherit the "auditable" attribute, which logs all changes, inserts, updates (with old value and new value), and deletes, all with username and timestamp, in a special audit Log table.

The `cdrlite/src/groovy/nci/bbrb/cdr/context` directory contains one file, `CDRApplicationEvent.groovy`, which responds to all Spring class events. The code in this file is an interceptor class that fires automatically whenever a Spring Security event (such as login or update to a domain class) is triggered.

The `cdrlite/src/groovy/nci/bbrb/cdr/datarecords` directory contains one commonly used file, `DataRecordBaseClass.groovy`, which is the base class for all database classes. `DataRecordBaseClass` extends `CDRBaseClass,` but it implements no new attributes.

The `cdrlite/src/groovy/nci/bbrb/cdr/staticmembers` directory contains one file, `StaticMemberBaseClass.groovy`, which serves as the base class for all CDR-Lite static classes. Inheriting the StaticMemberBaseClass gives every static member a Name and a Code attribute and the auditable property.

The `cdrlite/src/groovy/nci/bbrb/cdr/utility` directory contains files for recording user access and event logging.

```
\---cdrlite
    +---grails-app
    +---src
    |   +---groovy
    |   |   \---nci
    |   |       \---bbrb
    |   |           \---cdr
    |   |               +---context
    |   |               +---datarecords
    |   |               +---forms
    |   |               +---security
    |   |               +---staticmembers
    |   |               \---util
    |   |                   +---appaccess
    |   |                   \---pogo
    |   +---java
    |   \---templates
```

Figure 10: CDR-Lite `src` Directory Structure

Figure 11 shows the `web-app` directory, which uses a standard web-app directory structure from J2EE. The standard `js` directory includes OpenSeadragon[5] (seefootnote), which is used in visualizing whole-slide images of specimens. OpenSeadragon was released under the new BSD license. It was used for the CDR but is not used in the CDR-Lite. The rest of the directories under `web-app` contain custom Java script written for the CDR-Lite application, in most cases specific to each form. The `WEB-INF` and `META-INF` directories are also located here, but they are not shown by default in the NetBeans IDE project view. `applicationContext.xml` is generated and managed by Grails, not the programmer.

```
\---cdrlite
    +---grails-app
    \---web-app
        +---css
        +---images
        +---js
            +---countdown
            +---ext
            +---openseadragon-bin-1.1.1
            \---timeentry
```

Figure 11: CDR-Lite `web-app` Directory Structure

Grails uses the concept of "convention over configuration." This means that, typically, it uses the name and location of files instead of explicit configuration. Therefore, you will need to familiarize yourself with the directory structure provided by Grails 2.2.4.

[5] https://openseadragon.github.io

`cdrlite` is the main application directory and contains the following sub-directories:

- `Configuration`: Contains Grails, Hibernate, and Spring configuration files and directories.
- `Controllers`: Holds the controller classes, the entry points into a Grails application. Grails subclasses Spring's DispatcherServlet, used for delegating to CDR controllers.
- `Domain`: Holds the domain classes, which represent the persistent data for CDR, such as cases and specimens.
- `i18n`: Supports internationalization.
- `Scripts`: Holds Groovy scripts.
- `Services`: Holds the server classes, which are Spring-managed beans.
- `taglib`: Contains GSP custom tag libraries.
- `utils`: Holds a variety of codec classes.[6]
- `Views`: Contains GSPs—the V (view) in MVC.

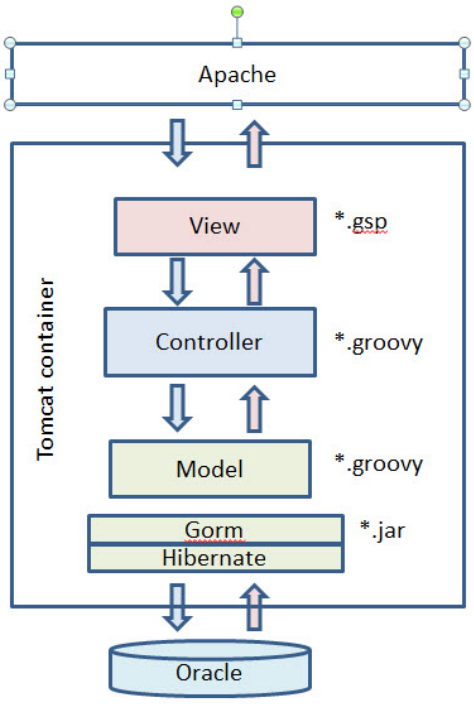

Figure 12: Grails Application in MVC Context and Server Aspect

Figure 12 shows the file types (*.gsp, *.groovy, and *.jar) that are used in various layers of the Grails version of an MVC application.

[6] See http://grails.org/doc/2.3.1/guide/single.html#codecs.

4.5 Security Architecture

As shown in the high-level diagram, Figure 13, the CDR is secured by Spring Security (formerly Acegi), using dependency injection and aspect-oriented programming (AOP). Every Web request for a resource (page) is filtered through Spring Security. Before each request, via user or RESTful interface, is fulfilled, Spring Security decides whether the requesting user is (a) authenticated and (b) authorized.

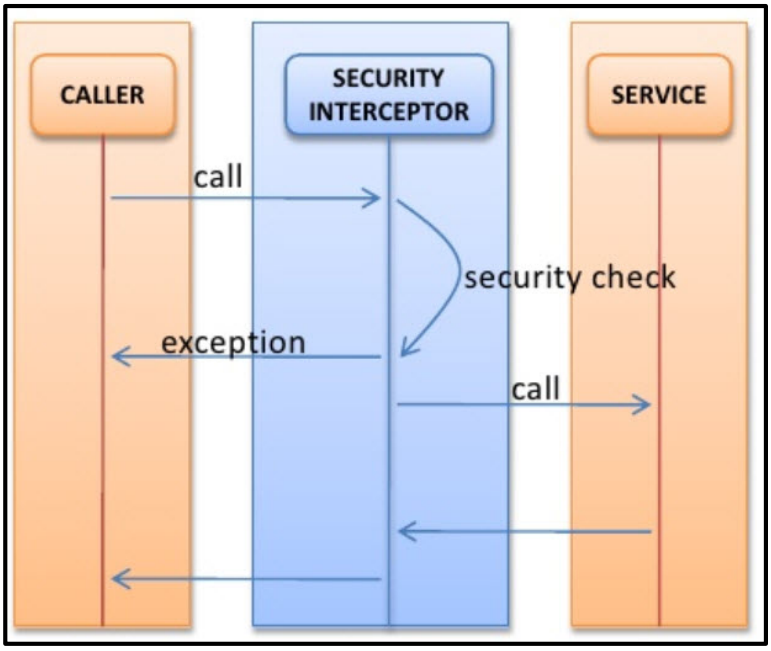

Figure 13: CDR-Lite Spring Security Model

The configuration of the Spring Security model is shown in Figure 4.

4.6 Communication Architecture

Communication with systems outside of the CDR-Lite happens in the following ways:

- Web services
- Email notifications
- Database CRUD operations

The following sections discuss these communications mechanisms.

4.6.1 Web Services

The CDR-Lite invokes and accepts external communications by RESTful Web services. REST is an industry standard built on top of the HTTP protocol as defined by Roy Thomas Fielding, Ph.D., using HTTP GET and POST. Table 8 gives the list of REST interfaces available in CDR-Lite. Data exchanged via the RESTful Web services are XML documents.

Table 8: RESTful Services Available with CDR-Lite

RESTful Service Type	Purpose
Processing Event	Processing event triggers an email to a configurable distribution list.

4.6.2 Email Notifications

The CDR-Lite sends email notifications using the industry-standard SMTP protocol. When the CDR detects triggers, it sends notifications to pre-defined email groups. Members of the individual groups receive notifications at the same time. The text of each message describes what triggered the email, giving specifics (not containing PII or PHI) so that the recipients can take the appropriate action. Recipients are defined by Exchange distribution lists and in the application settings, which system administrators can modify.

4.6.3 Database CRUD Operations

The CDR-Lite uses Java Database Connectivity to execute CRUD operations on a dedicated, local instance of PostgreSQL. By default, and in `DataSource.groovy`, the CDR-Lite expects to find PostgreSQL running on the same server, localhost:5432. This layer is encapsulated by the GORM layer, as described in section 4.3, Software Architecture. There is no external direct access to the CDR-Lite database, nor does the CDR-Lite access any other database instances.

5 System Design

5.1 Database Design

Figure 14 provides a high-level overview of the database design.

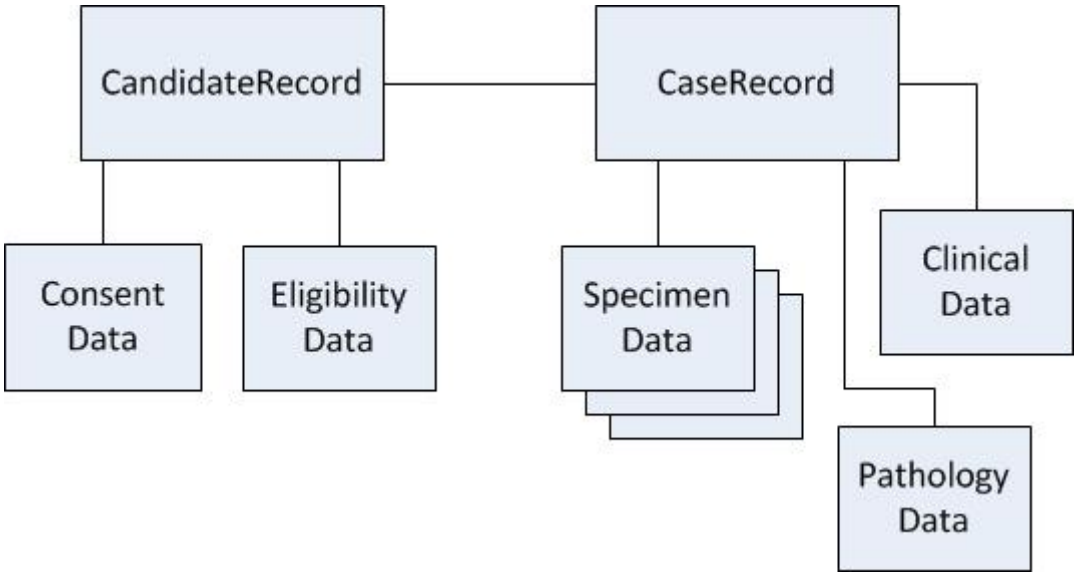

Figure 14: High-Level Overview of CDR-Lite Tables and Relations

A Candidate Record (domain class: candidateRecord) contains the basic information about someone who may be involved with a study. Associated with the Candidate Record is the Consent Data, which contains the candidate's consent, or lack thereof, to participate in a study. The Eligibility Data records specific information about the candidate, including specifics of their disease and history. All of this information determines whether the candidate meets the study's requirements (e.g., if the candidate's age is 65, but the study requires 20- to 40-year-olds, the candidate is ineligible). Details of consenting and eligibility vary greatly between studies, so these classes are implemented differently for each study.

For eligible candidates whose consent has been acquired, a Case Record (domain class: caseRecord) is created. The Case Record records specific information about the individual's involvement in the study. Associated with the case is Specimen Data (domain class: specimenRecord), which includes information about the collection and processing of the surgical products (e.g., blood, tissue). In all studies, a pathologist must review the specimens to confirm specimen type and quality; that review is recorded in the Pathology Data (domain class: prcReport). Clinical information related to the participant's care is recorded in the Clinical Data (domain class: clinicalDataEntry). Critical clinical information is also study specific, so this class is often sub-classed, reflecting study-specific details.

Grails uses GORM and Hibernate to automatically map between domain class objects and records in the underlying database. The table design, relations, and implementation details are automatically generated once the domain classes are defined.

Table 9 shows the mapping between tables in the database and domain classes.

Table 9: Mapping Between Database Tables and Domain Classes in the CDR-Lite

Database Table	Domain Class
ACTIVITY_EVENT	activityEvent
AUDIT_LOG	auditLog
blood_aliquot	bloodAliquots
blood_collection_tube	collectionTubes
blood_draw	bloodDraw
cdr_role	Spring Security: role
cdr_user	Spring Security: user
cdr_user_role	Spring Security: user to role many-to-many mapping table
DEVIATION	deviation
DR_CANDIDATE	candidateRecord
DR_CASE	caseRecord
DR_IMAGE	ImageRecord
Dr_photo	PhotoRecord
DR_PROCESSEVT	processingEvent
DR_SLIDE	SlideRecord
DR_SLIDE_PROCESS_EVENTS	slideProcessingEvent
DR_SPECIMEN	specimenRecord
DR_SPECIMEN_PROCESS_EVENTS	specimenProcessingEvents
FORM_BLOOD	BloodForm
FILE_UPLOAD	fileUpload
QUERY	Query
QUERY_ATTACHMENT	queryAttachment
QUERY_RESPONSE	queryResponse
form_cancer_history	CancerHistory
form_clinical_data_entry	clinicalDataEntry
form_consent_verification	ConsentVerification
form_demographics	Demographics
form_general_medical_history	GeneralMedicalHistory
form_health_history	HealthHistory
form_med_history	MedicationHistory
form_prcReport	prcReport
form_screening_enrollment	ScreeningEnrollment
form_slide_prep	SlidePrep

Database Table	Domain Class
form_slide_prep_dr_slide	*Generated by GORM*
form_slide_section	SlideSection
form_slide_section_dr_specimen	*Generated by GORM*
form_social_history	SocialHistory
form_surgery_anesthesia	SurgeryAnesthesia
form_therapy	Therapy
form_tissue_gross_evaluation	TissueGrossEvaluation
form_tissue_process_embed	TissueProcessEmbed
form_tissue_receipt_dissect	TissueReceiptDissection

Table 9 shows the tables in the CDR-Lite for basic information handling. When new forms are entered into the CDR-Lite, it automatically adds tables and fields through the Hibernate mechanism. This flexibility allows the types of data entered into the CDR-Lite to dynamically change, meeting developing requirements. However, Hibernate will not delete columns in the database. If an attribute is deleted in a domain class, the associated field must be deleted manually at the database level.

5.2 Data Conversion and DE-Identification

The CDR-Lite does not perform data conversion. Users enter data in one or more electronic forms, and data are persisted in the database unchanged. Where applicable, input values are tested for acceptable ranges, either absolutely (e.g., the height of a person cannot be negative) or based on the values entered in other fields.

PHI may be entered on several of the forms managed by CDR-Lite. The access level of the personal health information stored is restricted via an LDS. The LDS data are stored in the underlying database. Access to areas of the CDR-Lite containing LDS data is controlled both by user entitlements and roles, and by validation against Spring Security. If authorized to view LDS data, full LDS data will be displayed on screens and reported; otherwise, data are de-identified through dynamic content redaction. Examples of de-identified elements include birth dates, dates of procedures, and dates in relation to which procedures were performed or may be deduced. This dynamic redaction is performed in the GSP. This dynamic redaction is performed on the fly by the server at the stage of rendering of Web service payloads or as HTML screens are generated.

Data may be aggregated in various ways for reporting and analytics. These aggregations may rely on or contain LDS data. Access to these reports is also validated against Spring Security.

If a field may contain PHI, it must be coded as such using the custom `jQueryDatePicker` tag library. See below:

```
<g:jqDatePicker LDSOverlay="${bodyclass ?: ''}" name="surgeryDate"
value="${surgeryAnesthesiaInstance?.surgeryDate}"/>
```

5.3 User Interface Design

The CDR-Lite's UI design is based on standard Web templating, using SiteMesh, scripting, and Cascading Style Sheets (CSS). HTML, JavaScript, and CSS are embedded in GSPs or included from the `WEB-INF/js` and `WEB-INF/css` folders. The rendered output is a standards-compliant, cross-browser–compatible HTML page. All pages in the CDR-Lite include the `cahubTemplate.` Changes to the overall look and feel, banners, and footers are handled in the `cahubTemplate`, which can be found under `Views and Layouts` in the `layouts` folder.

The UI requirements regarding how the CDR-Lite was to look to the BSS users were minimal. The UI is designed to look and act like an electronic version of existing paper forms. The primary UI includes tables, lists, and dynamic elements populated based on the response to user input. The Web forms used to capture clinical data were based on the project SOPs for data capture. All users share the same style of interface, but fields and entire pages are restricted from some users.

5.3.1 Users, Roles, and Audiences

The CDR-Lite has to support different user types, roles, and privileges. Some users are external to the NCI, and some are internal. Users have either read-write or read-only access to the data. Some roles can see only certain aspects of the programs supported by the CDR and CDR-Lite, whereas others can see only data generated by their organization. Table 10 shows the roles and privileges that the CDR-Lite supports. Each role is configured and validated against Spring Security upon login. Users' privileges and access levels are determined by their roles and organizations.

Table 10: CDR-Lite's User Roles and Privileges

Role	Write	LDS (access to PII)	Global Access	Notes
BSS	Y	Y	N	Biospecimen source site staff role
DM	Y	Y	Y	Data manager
PRC	Y	N	Y	Pathology Resource Center pathologist
LDS	N	Y	Y	Read-only with access to HIPAA identifiers
R/O	N	N	Y	Basic read-only account
External Org	N	N	Y	External organization limited to a subset of read-only data
Super	Y	Y	Y	CDR-Lite super user account
Service (API)	Y	N	Y	Machine to machine accounts for Web Service APIs

Access to study-specific and functional areas of the CDR-Lite is available through a user's home page, as shown in Figure 15. Depending on their entitlements, users may see a different home page or be restricted to certain areas. Users with some power user roles, such as DM, LDS, and Super, have the ability to raise and lower their privileges as needed.

Figure 15: Example of a CDR-Lite Home Page

5.3.2 Triggers

The CDR-Lite uses a mechanism called "triggers," which send email messages to the appropriate users when some predefined event happens or a business rule is applied. Each trigger contains a number of elements, including code to check for the business logic to see if a matching event is happening, a pre-defined mail body text, and a list of email addresses for those who should be notified when the event occurs. Section 4.6.2 discusses the communications mechanism for emails. Individual triggers may be customized to include specific information in the body of the message (e.g., case ID, individual field values not containing PII or PHI).

Table 11 contains the full list of triggers and a description of the event associated with each trigger. When a trigger fires, it creates an SMTP message specific to the triggering event. That SMTP message hands off to an external mail server, which delivers the message to the appropriate experts. The users then take the appropriate action. Which users get the message depends on the SMTP mail list description; adding a person to a distribution list ensures that the person automatically gets all future messages for that list.

Table 11: Mail Distribution Lists for Various Triggers

Name of Trigger Distribution List	Triggering Event
APERIO_IMAGE_DISTRO	Notification: Whole-slide images are available at the CBR for a given study.
CDRLITE_ADMIN_DISTRO	Notification: A case was created or case status changed.
NEW_QUERY_TRACKER_DISTRO	Notification: A new query was created by Data Management.

APPENDIX A. KEY TERMS

The following table defines and explains terms and acronyms relevant to this document.

Term	Definition
ABCC	Advanced Biomedical Computer Center – an NCI facility located in the FNLCR
AOP	Aspect-oriented programming: A programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns
BPV	Biospecimen Pre-Analytical Variables: A study sponsored by the Biorepositories and Biospecimen Research Branch that used the CDR to manage study-specific data
BSD	Berkeley Software Distribution
BSS	Biospecimen source site: An institute from which human tissue is initially collected
CDR	Comprehensive Data Resource
CDR-Lite	The revised and generalized Comprehensive Data Resource publically released
DM	Data management: The people and activities preserving data integrity
FNLCR	Frederick National Laboratory for Cancer Research
GORM	Grails Object Relational Mapping
Grails	A powerful computer software framework based on the Groovy programming language and emphasizing rapid software development of Web-based applications
GTEx	Genotype-Tissue Expression program: A project of the NIH Common Fund
HHS	U.S. Department of Health and Human Services
LDS	Limited data set: A reflection of the central data where PHI and PII have been protected
LIMS	Laboratory information management system
PHI	Protected health information: Health information, including demographic information, that relates to an individual's physical or mental health or the provision of or payment for health care
PII	Personally identifiable information: Individually identifiable health information
RESTful	One type of Internet service interface, typically program-to-program, over which information is exchanged, using the HTTP protocols.
SOP	Standard operating procedure: A detailed document precisely describing the performance of a protocol
UI	User interface: The Web-based graphical interface that enables various

Term	Definition
	groups to enter and retrieve data from the CDR-Lite
XML	Extensible Markup Language: A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is defined in the XML 1.0 Specification produced by the W3C and in several other related specifications, all free open standards.

APPENDIX B. CDR-LITE DESIGN APPROVAL

The undersigned acknowledge that they have reviewed the CDR-Lite Design Document and agree with the information presented therein. Changes to this document will be coordinated with and approved by the undersigned or their designated representatives.

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____