## Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Author</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joseph Nelson</td>
<td>4/28/13</td>
<td>Updated Diagrams, added highlevel UML</td>
</tr>
</tbody>
</table>
1. Introduction ........................................................................................................ 7
  1.1. Purpose ........................................................................................................ 7
  1.2. Scope .......................................................................................................... 7
  1.3. Definitions, Acronyms, Abbreviations ......................................................... 7
2. Architecture ..................................................................................................... 10
  2.1. Overview .................................................................................................... 10
  2.2. Docket Context Diagram ........................................................................ 10
  2.3. System Security Posture ............................................................................ 10
3. System Design ................................................................................................ 11
  3.1. Docket User Experience ............................................................................ 11
    3.1.1. High-Level Architecture ..................................................................... 11
    3.1.2. Request Handling .............................................................................. 12
    3.1.3. Docket UX CRUD Operations ............................................................... 14
      Figure 4 illustrates how the Docket UX services request to create, read,
      update or delete a Domain Object. While the Docket UX maintains its
      own model, any change that needs to be persisted will result in an HTTP
      request to the Docket DSS. After retrieving an object from the DSS,
      Docket UX stores the object it’s in-browser model. Any changes to the
      object will be stored locally until the user confirms they want to persist
      the object (e.g. by clicking a “Save” button) .............................................. 14
      .................................................................................................................. 14
    3.1.4. Security ................................................................................................ 15
    3.1.5. Object Model ........................................................................................ 16
    3.1.5.1. HTTP-Controller Mapping ............................................................... 17
      Table 2 lists the controllers that are applied when the browser receives
      a request for the given URI ........................................................................ 17
  3.2. Docket Decision Support System ............................................................... 17
    3.2.1. High-Level Architecture .................................................................... 17
      Figure 6 shows the Docket DSS architecture. Because the Docket
      Course Scheduling system was designed for modularity, the Docket
DSS system was designed and developed independently of the Docket UX.

Dependency Injection is a critical function performed by the Spring Web Services framework. As controllers are instantiated to handle request, Spring injects the objects required by a controller to perform its work. For example, a CREATE operation will require access to the backend database. To perform this operation, Spring injects a Data Access Object (DAO) corresponding to the object needed by the controller—the controller then calls methods of the DAO to complete the CREATE operation.
1. Introduction

1.1. Purpose

This document specifies the entire software architecture and design for the Docket Course Scheduling system. These design decisions directly relate to the functional and non-functional requirements as specified in the Docket Software Requirements Specification [reference?]

Docket is a decision support system that allows University administrators to effectively manage departmental resources, resolve scheduling conflicts, and reduce cost associated with scheduling academic courses. Docket’s primary goal is to identify scheduling conflicts and notify users in order to facilitate conflict resolution.

1.2. Scope

This document describes the software architecture and design for the Docket v1.0. The intended audience for this document includes designers, developers, and testers of the Docket software system.

1.3. Definitions, Acronyms, Abbreviations

Authentication The process of establishing and verifying the identity of a security principal
Authorization The process of verifying a security principal has the right to perform an action
Access Control List An Authorization mechanism that controls what security principals have access to what resources by specifying a list of authorized principals and associated permissions also ACL
Controller See Resource Request Handler
Cascading Style Sheet also CSS an markup language used to define the style (formatting) of an HTML document.
CSS See Cascading Style Sheet
Data Access Object A software component responsible for the retrieval, update, and deletion of domain objects
Data Binding a mechanism used to insure synchronization between a view and model. A change in either a view is automatically propagated to a model and vice versa
Data Confidentiality A security principle that ensures only authorized security principals can access security data
Data Integrity A security principle that ensures secured data is modified only by authorized security principals. Additionally, Data Integrity ensures that unauthorized data modification is easily detectable
**Decision Support System** A computer-based information system that supports business or organizational decision-making processes also DSS

**Dependency Injection** A style of object configuration in which an object’s fields and collaborators are set by an external entity.

**Docket UX** A decoupled web-based GUI for the Docket system also UX

**Domain Object** See Resource

**HibernateORM** a framework that provides mapping of Object Oriented data structures used in OOP programming to relational tables used in a RDMS

**Interactive Clients** Any client which requires a human operator

**Inversion of Control** A design style that emphasizes decoupling object creation from usage simplifying lifecycle management by delegating instantiation to an external entity

**JSON** JavaScript Object Notation, a lightweight text-based open standard for human-readable data exchange.

**Jersey** a Java-RS compliant framework that provides Domain Object data marshaling, URI processing, and mapping of URI to Controllers

**Non-Interactive Clients** Any client whose interaction with this system is automated or semi-automated and may not require a human operator

**Object-Relational Mapping Framework** A software component used to map an OOP object model to a relational model for storage in a relational database

**Persisting** The act of transferring an object from in-memory storage to permanent or persistent storage (i.e. database, file, etc.)

**Post-Data** When an end-user enters data into a web form, the HTTP client submits the data to the HTTP server via an HTTP POST method

**Principal** See Security Principal

**Resource** Any named information or collection thereof. Docket resources include Courses, Sections, Instructors, and Rooms also Domain Object

**Resource Request Handler** A software component that coordinates the retrieval and update of system resources also Controller

**Security Posture** Describes the planning and implementation approach to security. The Security Posture may describe hardware and software protections deployed to secure an asset

**Security Principal** A user, device, or system that can perform an action within the application

**SpringMVC** a Web Services framework that coordinates dependency injections, Email, and Security services

**Spring Security 3** a framework that provides Authentication and Authorization services for URL and Domain Object security

**URL** Uniform Resource Locator, the address of a World Wide Web page

**URI** Uniform Resource Identifier, the part of URL that uniquely identifies a resource

**Web Application Server** An application server that handles HTTP request and coordinates communication with application programs through any number of protocols
**Web Services Framework** A software component that facilitates development and integration for web services including facilities for IoC, Dependency Inject, Flexible Marshalling, and security.

**XML** Extensible Markup Language, a meta-language designed to enable easy data transformation, transportation, and storage.
2. Architecture

2.1. Overview

The system is comprised of two independent subsystems, the Docket UX and the Docket Decision Support System. The Docket Decision Support System exposes a RESTful API for consumption by interactive and non-interactive clients. The Docket UX is provided both as an example client-side application built on top the DSS and the primary interface for interactive (browser-based) clients. When deployed standalone, the Decision Support System supports automated (non-interactive) access through a simple REST API. Each subsystem is described in subsequent sections below.

2.2. Docket Context Diagram

Figure 1 shows the unified Docket Course Scheduling system context. Interactive user interaction with the product is through Docket UX, a client-side web application. DSS is hosted on a web application server that also serves Docket UX content. Once an interactive user navigates to the Docket homepage, all required JavaScript, CSS, and static content are downloaded and loaded in the user’s browser.

![Docket Context Diagram]

2.3. System Security Posture

Security is a key focus of the Docket Course Scheduling system design. Security protections are deployed at nearly every level of the technology stack to ensure data confidentiality and system integrity. Table 1 summarizes the Docket Application security posture. Please the respective sections for design-level details.

<table>
<thead>
<tr>
<th>Security Principle</th>
<th>Category</th>
<th>Provider</th>
<th>Section</th>
</tr>
</thead>
</table>

The table above provides...
<table>
<thead>
<tr>
<th>Authentication</th>
<th>Application – DSS - Logon</th>
<th>Spring Security 3</th>
<th>3.2.4.1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Application – DSS - Web</td>
<td>Spring Security 3</td>
<td>3.2.4.2.1</td>
</tr>
<tr>
<td>Authorization</td>
<td>Application – DSS - Objects</td>
<td>Spring Security 3</td>
<td>3.2.4.2.2</td>
</tr>
<tr>
<td>Data Confidentiality</td>
<td>Network</td>
<td>SSL Encryption</td>
<td>3.2.4.3.1</td>
</tr>
<tr>
<td>Data Integrity</td>
<td>Network</td>
<td>SSL Encryption</td>
<td>3.2.4.4.1</td>
</tr>
</tbody>
</table>

**Table 1 System security summary**

Application level security refers to software protections implemented within the application or provided by third-party packages that protect application assets. Application level security is deployed to protect a specific subsystem or component. Network level security refers to security protections that are deployed to protect data sent over a network connection.

**3. System Design**

**3.1. Docket User Experience**

The Docket User Experience, or Docket UX, is an in-browser JavaScript-based web application and the primary graphical interface for interactive clients. The following sections describe the Docket UX Architecture and Design.

**3.1.1. High-Level Architecture**

*Figure 2* depicts the high-level Docket UX architecture. Because the Docket Course Scheduling system was designed for modularity, the Docket UX system was designed and developed independently of the Docket DSS.
Docket UX leverages several technologies to provide a seamless user experience and ease of software maintenance/development. AngularJS is a JavaScript framework that provides Data Binding, Dependency Injection, and URI-to-Controller mapping, among other services that make up the core of Docket UX functionality. Additionally, The Docket UX leverages Bootstrap, an open source front-end framework used to generate the CSS code responsible for the look and feel of the Docket UX. Finally, the jQuery JavaScript library is used to simplify XML- and JSON-intensive access and manipulation when working with the Docket Model.

Note that the Docket UX Core and the Docket UX Presentation functional components are depicted separately for reference and discussion purposes. In practice, all objects in both the Docket UX Core and the Docket UX Presentation layers exist within the AngularJS framework.

3.1.2. Request Handling

Figure 3 is a generic illustration of how functional components in the Docket UX work together in response to a user-initiated request.
Figure 3 Docket UX Request Handling
3.1.3. **Docket UX CRUD Operations**

**Figure 4** illustrates how the Docket UX services request to create, read, update or delete a Domain Object. While the Docket UX maintains its own model, any change that needs to be persisted will result in an HTTP request to the Docket DSS. After retrieving an object from the DSS, Docket UX stores the object it’s in-browser model. Any changes to the object will be stored locally until the user confirms they want to persist the object (e.g. by clicking a “Save” button).

---

**Figure 4 CRUD Operations in the UX**
3.1.4. Security  

3.1.4.1. Logon Handling  

Docket UX delegates login responsibilities to Docket DSS. The sequence diagram in Figure show how Docket UX and Docket DSS work together to process a login request.

Figure 5 DSS Login Request Handling
Objects within the Docket UX exists within the AngularJS framework. As a result, controllers in the Docket UX aren’t objects that have (or need) data members or methods. In the AngularJS framework, both the controller and views in the MVC pattern share the same variable known as `$scope`. Consequently, controllers function as modules for grouping code used to intialize or modify the `$scope`. A complete discussion of the `$scope` variable is outside the scope of this document, however,
a more detailed discussion may be found in the References section of this document.

### 3.1.5.1. HTTP-Controller Mapping

**Table 2** lists the controllers that are applied when the browser receives a request for the given URI.

<table>
<thead>
<tr>
<th>URI</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>/schedule</td>
<td>ScheduleController</td>
</tr>
<tr>
<td>/courses</td>
<td>CoursesController</td>
</tr>
<tr>
<td>/room</td>
<td>RoomsController</td>
</tr>
<tr>
<td>/instructors</td>
<td>InstructorController</td>
</tr>
<tr>
<td>/sections</td>
<td>SectionController</td>
</tr>
</tbody>
</table>

**Table 2 HTTP-to-Controller Mapping**

### 3.2. Docket Decision Support System

#### 3.2.1. High-Level Architecture

**Figure 6** shows the Docket DSS architecture. Because the Docket Course Scheduling system was designed for modularity, the Docket DSS system was designed and developed independently of the Docket UX.
**Figure 6 Docket DSS Architecture**
Dependency Injection is a critical function performed by the Spring Web Services framework. As controllers are instantiated to handle request, Spring injects the objects required by a controller to perform its work. For example, a CREATE operation will require access to the backend database. To perform this operation, Spring injects a Data Access Object (DAO) corresponding to the object needed by the controller—the controller then calls methods of the DAO to complete the CREATE operation.

### 3.2.2. Request Handling

The following sections describe how various components of the Docket DSS work together to service a request. For brevity, Section 3.2.2.1 describes request handling by Jersey and SpringWS. Section 3.2.2.2 describes request handling by SpringWS and Hibernate. Operationally, the system performs both sequences to service a single request.

#### 3.2.2.1. Jersey and SpringWS

The sequence diagram in **Figure 7** illustrates how the RESTful API framework (Jersey) and Web Services framework (SpringWS) work together to process an HTTP Request. This sequence is representative of a typical request. As a result, this sequence would be valid for any controller. Similarly, a controller may require more than one DAO object. As a result SpringWS may inject more than one DAO and the sequence would still apply.
Figure 7 DSS Request handling
3.2.2.2. SpringWS and Hibernate Request Handling

The sequence diagram in Figure 9 illustrates how Controller, DAOs, Spring, Hibernate, and Database components work together to persist and retrieve objects.

Figure 8 Persisting objects with SpringWS and Hibernate
3.2.3. Security

The Docket UX delegates security concerns to the Docket DSS. As a result, protecting DSS components is important to providing strong overall security and system integrity. To provide Authentication and Authorization, the DSS leverages Spring Security 3 as its primary security provider. Other security protections are implemented by Table 3 indicates the Spring Security 3 packages used to provide Application level security to the Docket DSS.

<table>
<thead>
<tr>
<th>Security Principle</th>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>*core.jar</td>
<td>Provides the core implementation of Spring Security 3</td>
</tr>
<tr>
<td>Configuration</td>
<td>*config.jar</td>
<td>Used by Spring Security to parse the security namespace for configuration purposes</td>
</tr>
<tr>
<td>Authorization</td>
<td>*web.jar</td>
<td>Used by Spring Security to enforce access-control restrictions on URLs</td>
</tr>
<tr>
<td>Authentication</td>
<td>*cas-client.jar</td>
<td>Supports authentication to CAS single-sign on server. Not implemented in the Docket v1.0 implementation</td>
</tr>
<tr>
<td>Authorization</td>
<td>*acl.jar</td>
<td>Supports applying security policies to specific domain object instances</td>
</tr>
</tbody>
</table>

Table 3 Spring Security Packages

Note: The asterisks of Table 3 appear in the package name for brevity. In practice, the complete package name would include “spring-security-*package name*.jar” with *package name* being the name specified in column 2 of Table 3. For example, the complete package name for Spring Security 3 core would be spring-security-core.jar.

The following sections provide brief discussions of basic security principles and how they are implemented in the Docket DSS.

3.2.3.1. Authentication

3.2.3.1.1. Logon

When a client request authentication, the DSS checks the user-supplied authentication cookie against a set of existing cookies representing authenticated users. If the supplied cookie is not in the “remembered cookies” set, the DSS request logon credentials from the client. The DSS then queries the backend database using the username as the key to retrieve a cryptographically hashed value representing the password. The DSS then performs the one-way cryptographic has of the supplied password and compares the
value against the hash retrieved from the database. If both values match, the user is authenticated and the DSS responds with an HTTP 200 and a session token which must be passed as a post-parameter in future requests.

**Figure 10** shows a conceptual view of this process. In practice, however, SpringWS delegates authentication by allowing Spring Security 3 to intercept request to URI that have been annotated with Spring Security 3 special annotations. For a more detailed discussion on SpringWS and Spring Security 3, please see the “Spring Security Framework – Introduction” reference in the in References section of this document.

![Figure 9 Logon Request Handling](image)
3.2.3.2. Authorization

3.2.3.2.1. URL-Based Access Control

The Docket DSS utilizes Spring Security’s Expression-based Access Control. Operationally, Expression-based Access Control results in restrictions to the URLs a client may access and to HTTP methods the client may request.

Figure 11 shows a URL-based access control scenario based on authorization token.

![URL-Based Access Control Diagram]

Spring Expressions are predefined by the DSS administrator in configuration XML (see “Expression-Based Access Control” in the References section of this document) and take effect when the WAR is redeployed. By default all users are allowed to access the base URL. However, by default, only authenticated users are allowed to use the HTTP GET method with the DSS. Additional restrictions are set on remaining HTTP methods. Table 4 summarizes the default HTTP method access level by user type.

<table>
<thead>
<tr>
<th>User Type</th>
<th>Role</th>
<th>HTTP Method</th>
<th>Default Access Level</th>
</tr>
</thead>
</table>

Figure 10 Spring Security 3 URL-based authorization
### Domain Object Access Control

Domain object access control is deployed within the DSS to ensure that a given security principal is authorized to work with a given domain object in a given way. The DSS leverages Spring Security 3 to enforce domain object security in the same way URL security is enforced.

### Data Confidentiality

#### TLS/SSL Encryption

Docket DSS enforces negotiation of HTTPS for all communication with clients. As a result, all communication is protected by TLS/SSL encryption as it transits the network. Connections for which HTTPS negotiation fails are disconnected immediately. Data assets include username, password, authorization tokens, and data associated with courses, instructors, sections, and locations. Docket DSS clients can be confident that data assets are only viewable to authorized parties.

### Data Integrity

#### TLS/SSL Encryption

Docket DSS enforces negotiation of HTTPS for all communication with clients. As a result, all communication is protected by TLS/SSL encryption as it transits the network.
Connections for which HTTPs negotiation fails are disconnected immediately. TLS/SSL encryption insures that if data is modified in transit, both the sender and receiver can detect the change in the data.

### 3.2.2. Object Model

The following sections provide a description for all Classes used in the Docket DSS. Please refer to the “Docket DSS UML” diagram in the appendix for further details on hierarchies and relationships. Figure 11 is a representative example of the relationship between controllers, DAOs, and model objects. Additionally Figure 11 shows the relationship between the DAO and the Hibernate SessionFactory Object that interacts with the backend database.
Figure 11 Representative example of Docket Controller DAO pattern

Please see the “Docket DSS UML” diagram in the Appendix of this document for a complete depiction of the class hierarchies and their relationships. The following sections describe the Docket DSS classes in turn.

3.2.2.1. Domain Object Model

The following sections describe the Classes Docket DSS uses to represent logical entities in the scheduling domain space. Please refer to the accompanying UML diagram for a complete description of object relationships and hierarchies.
3.2.2.1.1. **Course Class**

The Course Class represents a single course in the system. The Course Class represents a course that is available for scheduling. DSS clients add new courses to the system by posting a JSON or XML object with appropriate HTTP Content-Type header.

**JSON Transformation**

The JSON handler produces the following output

```json
{
    "id": 1,
    "departmentCode": "CS",
    "courseNumber": 200
}
```

**XML Transformation**

The XML Handler produces the following output

```xml
<
```
3.2.2.1.2. **ScheduledSection Class**

The ScheduledSection represents an existing Course has been assigned at least a TimeBlock.

**JSON Transformation**

The JSON Handler produces the following output

```json
{
    "course": {
        "id": "long",
        "departmentCode": "string",
        "courseNumber": "int"
    },
    "id": "long",
    "instructor": {
        "id": "long",
        "lastName": "string"
    }
}
```
The XML Handler produces the following output

```
<scheduledSection>
  <course>
    <courseNumber>200</courseNumber>
    <departmentCode>CS</departmentCode>
  </course>
  <instructor>
    <firstName>Dario</firstName>
    <lastName>Salvucci</lastName>
  </instructor>
  <location>
    <capacity>0</capacity>
    <id>1</id>
  </location>
  <timeBlock>
    <endTime>
      <hour>0</hour>
      <minute>0</minute>
    </endTime>
    <startTime>
      <hour>0</hour>
      <minute>0</minute>
    </startTime>
  </timeBlock>
</scheduledSection>
```
3.2.2.1.3. Instructor Class

The Instructor Class represents a single instructor in the system

**JSON Transformation**

The JSON Handler produces the following output

```
{
    "id": 1,
    "firstName": "Dario",
    "lastName": "Salvucci"
}
```

**XML Transformation**

The XML Handler produces the following output

```
<instructor>
    <!-- XML output here -->
</instructor>
```
<firstName>string</firstName>
<lastName>string</lastName>
</instructor>
3.2.2.1.4. **Room Class**

The Room Class represents a single location in the system. The Room Class enforces real-world capacity constraints for Rooms, allowing the Suggestion algorithms to suggest an appropriate number of sections based on Room capacity.

**JSON Transformation**

The JSON Handler produces the following output:

```json
{
   "id": 1,
   "capacity": 0,
   "name": null
}
```

**XML Transformation**

The XML Handler produces the following output:

```
<room>
</room>
```
3.2.2.2. **TimeBlock Class**

The TimeBlock Class represents a contiguous block of time in the system. DSS users may add blocks of time independent of adding other resources (i.e. Instructors, Rooms, Course, and Scheduled Sections).

**JSON Transformation**

The JSON Handler produces the following output

```json
{
  "id": 1,
  "startTime": {
    "hour": 0,
    "minute": 0
  }
}
```
XML Transformation

The XML Handler produces the following output

```xml
<timeblock>
  <endTime>
    <hour>0</hour>
    <minute>0</minute>
  </endTime>
  <startTime>
    <hour>0</hour>
    <minute>0</minute>
  </startTime>
</timeblock>
```

3.2.2.3. Time Class

```
<<java Class>>

Time
doctor.model

- hour: int
- minute: int

- Time()
- Time(int, int)
- getHour(): int
- setHour(int): void
- getMinute(): int
- setMinute(int): void
- compareTo(Time): int
- hashCode(): int
- equals(Object): boolean

Figure 17 Time Class
The Time Class represents a discreet unit of time, specified in 24-hour format. The *hour* member may be in range [0, 23], inclusive. Likewise, minutes may be in range [0, 59], inclusive.
**JSON Transformation**

The JSON Handler produces the following output

```json
{
    "hour": 0,
    "minute": 0
}
```

**XML Transformation**

The XML Handler produces the following output

```xml
<time>
    <hour>0</hour>
    <minute>0</minute>
</time>
```
Data Access Objects (DAO) forms a hierarchy of objects that enables persisting and retrieving Domain Objects in the database. DAOs provide create, read, update, and delete operations in addition to custom querying of the database. Each DAO is responsible for a single Domain Object. The Dependency Injection component of the Web Services Framework injects the ORM SessionFactory object into DAOs to provide access to the connection to the database.

3.2.2.4.1. AbstractDAO Class

The Abstract DAO Class represents the contract that enforces subclass hierarchy among the DAO classes.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>persistEntry</td>
<td>Persists an object</td>
</tr>
<tr>
<td>retrieveObjectById</td>
<td>Obtain a list of persisted objects</td>
</tr>
<tr>
<td>retrieveObjectsByClass</td>
<td>Retrieve existing object by ID</td>
</tr>
<tr>
<td>updateEntity:</td>
<td>Delete object by ID</td>
</tr>
<tr>
<td>deleteEntity:</td>
<td>Update existing object</td>
</tr>
</tbody>
</table>
3.2.2.4.2. CourseDAO Class

The Course DAO Class provides create, read, update, and delete operations for the Course Object.

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCourse</td>
<td>Persists a Course</td>
</tr>
<tr>
<td>listCourses</td>
<td>Obtain a list of persisted Courses</td>
</tr>
<tr>
<td>getCourseById</td>
<td>Retrieve existing Course by ID</td>
</tr>
<tr>
<td>deleteCourse</td>
<td>Delete Course by ID</td>
</tr>
<tr>
<td>updateCourse</td>
<td>Update existing Course</td>
</tr>
</tbody>
</table>

**Figure 19 CourseDAO**

```java
<<Java Class>>
CourseDAO

docket.dao

- sessionFactory: SessionFactory

- CourseDAO()
- addCourse(Course): void
- listCourses(): List<Course>
- getCourseById(long): Course
- deleteCourse(long): void
- updateCourse(Course): void
```
3.2.2.4.3. *ScheduledSectionDAO Class*

The ScheduledSection DAO class provides create, read, update, and delete operations for the ScheduledSection Object.

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addSection</td>
<td>Persists a ScheduledSection</td>
</tr>
<tr>
<td>listSections</td>
<td>Obtain a list of persisted ScheduledSections</td>
</tr>
<tr>
<td>getSectionById</td>
<td>Retrieve existing ScheduledSection by ID</td>
</tr>
<tr>
<td>deleteSection</td>
<td>Delete ScheduledSection by ID</td>
</tr>
<tr>
<td>updateSection</td>
<td>Update existing ScheduledSection</td>
</tr>
</tbody>
</table>

3.2.2.4.4. *InstructorDAO Class*

The Instructor DAO class provides create, read, update, and delete operations for the Instructor Object.
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addInstructor</td>
<td>Persists an Instructor</td>
</tr>
<tr>
<td>listInstructors</td>
<td>Obtain a list of persisted Instructors</td>
</tr>
<tr>
<td>getInstructorById</td>
<td>Retrieve existing Instructor by ID</td>
</tr>
<tr>
<td>deleteInstructor</td>
<td>Delete Instructor by ID</td>
</tr>
<tr>
<td>updateInstructor</td>
<td>Update existing Instructor</td>
</tr>
</tbody>
</table>

#### 3.2.2.4.5. RoomDAO Class

![RoomDAO Class](source)

The Room DAO class provides create, read, update, and delete operations for the Course Object.

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addRoom</td>
<td>Persists a Room</td>
</tr>
<tr>
<td>listRooms</td>
<td>Obtain a list of persisted Rooms</td>
</tr>
<tr>
<td>getRoomById</td>
<td>Retrieve existing Room by ID</td>
</tr>
<tr>
<td>deleteRoom</td>
<td>Delete Room by ID</td>
</tr>
<tr>
<td>updateRoom</td>
<td>Update existing Room</td>
</tr>
</tbody>
</table>
3.2.2.4.6. **TimeBlockDAO Class**

![TimeBlockDAO Class](image)

The TimeBlock DAO class provides create, read, update, and delete operations for the Course Object.

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addTimeBlock</td>
<td>Persists a TimeBlock</td>
</tr>
<tr>
<td>listTimeBlocks</td>
<td>Obtain a list of persisted TimeBlocks</td>
</tr>
<tr>
<td>getTimeBlockById</td>
<td>Retrieve existing TimeBlock by ID</td>
</tr>
<tr>
<td>deleteTimeBlock</td>
<td>Delete TimeBlock by ID</td>
</tr>
<tr>
<td>updateTimeBlock</td>
<td>Update existing TimeBlock</td>
</tr>
</tbody>
</table>

3.2.2.5. **Docket Controllers**

Resource request handlers control access to DSS resources (e.g. Courses, Instructors, etc.). The RESTful API Framework selects the appropriate Resource Request Handler based on the unique URI in an HTTP request. In general, a Resource Request Handler method corresponds to exactly one URI. The Post-parameters for a given HTTP request are converted by the Web Service Framework into formal parameters for methods of a given Resource Request Handler.

3.2.2.5.1. **Specifying Input Format**

DSS clients may update existing resources by supplying parameters in the HTTP message body in either JSON or XML format. DSS clients must then set the HTTP Content-type header based on either JSON or XML. **Table 5** summarizes valid Content-Type values.
### 3.2.2.5.2. Specifying Output Format

DSS clients may specify the content type for HTTP responses by changing the Accept HTTP header field. This causes Docket DSS to format the result of HTTP methods as either JSON or XML. **Table 6** summarizes valid Accept header values.

<table>
<thead>
<tr>
<th>Accept String</th>
<th>Message Body Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/xml</td>
<td>XML</td>
</tr>
<tr>
<td>application/json</td>
<td>JSON</td>
</tr>
</tbody>
</table>

**Table 6 Valid Accept header values**

### 3.2.2.5.3. CourseController Class

![Figure 24 Course Controller Class](image)

\[
\text{docket.controller.CourseController}
\]

- `CourseDAO courseDAO`
- `UrlInfo urlInfo`  
  - `Course.CourseList allCourses()`  
  - `Response createCourse(String departmentCode, int courseId)`  
  - `Course getCourseById(long id)`  
  - `void updateCourse(long id, Course course)`  
  - `void updateCourse(long id, String departmentCode, Integer courseId)`  
  - `void deleteCourse(long id)`

**Figure 24 Course Controller Class**

Responsible for handling requested mapped to /courses/ URI. This object uses the CourseDAO object to create, read, update, and delete stored objects in the object database.

### HTTP-Controller Method Mapping

**Table 7** lists the HTTP Method to Docket Controller mapping. Details on return values are listed in subsequent sections.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>URI</th>
<th>Controller Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/courses</td>
<td>allCourses()</td>
</tr>
<tr>
<td>POST</td>
<td>/courses</td>
<td>createCourse(post-</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>URL Path</td>
<td>Function Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>GET</td>
<td>/courses/id</td>
<td>getCourseById(id)</td>
</tr>
<tr>
<td>PUT</td>
<td>/courses/id</td>
<td>updateCourse(id, Course object)</td>
</tr>
<tr>
<td>POST</td>
<td>/courses/id</td>
<td>updateCourse(id, post-parameters)</td>
</tr>
<tr>
<td>DELETE</td>
<td>/courses/id</td>
<td>deleteCourse(id)</td>
</tr>
</tbody>
</table>

**Table 7 HTTP Method Mapping**
**Method Description**

**GET /courses/[id]**
Return a json or xml representation of a course given a course's unique identifier.

**PUT /courses/[id]**
Update the representation of a course given its unique identifier.

**POST /courses/[id]**
Post takes two post-parameters departmentCode=string, and courseNumber=#. This updates the course who identifier matches [id] replacing the current value of the post-parameters with the newly provided parameters.

**DELETE /courses/[id]**
Hard removes the course with identifier [id] from the database.

**GET /courses**
Returns a list of all courses currently in the database. There is currently no data paging.

**POST /courses**
Post takes two post-parameters departmentCode=string, and courseNumber=#. This creates a new course object in the database. The newly created resource's URI is returned in the HTML response.

### 3.2.2.5.4. ScheduledSectionController Class

<table>
<thead>
<tr>
<th>Controller Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScheduledSectionDAO sectionDAO</td>
</tr>
<tr>
<td>CourseDAO courseDAO</td>
</tr>
<tr>
<td>InstructorDAO instructorDAO</td>
</tr>
<tr>
<td>RoomDAO roomDAO</td>
</tr>
<tr>
<td>TimeBlockDAO timeBlockDAO</td>
</tr>
<tr>
<td>UriInfo uriInfo</td>
</tr>
</tbody>
</table>

**Figure 25 Scheduled Section Controller Class**
Responsible for handling requested mapped to /sections/ URI. Objects maps requests to the underlying ScheduledSectionDAO object for retrieval, insertion, and updating the stored object in the back-end database.

### HTTP-Controller Method Mapping

Table 8 lists the HTTP Method to Docket Controller mapping. Details on return values are listed in subsequent sections.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>URI</th>
<th>Controller Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/sections</td>
<td>allSections()</td>
</tr>
<tr>
<td>POST</td>
<td>/sections</td>
<td>createSection(post-parameters)</td>
</tr>
<tr>
<td>GET</td>
<td>/sections/id</td>
<td>getSectionById(id)</td>
</tr>
<tr>
<td>PUT</td>
<td>/sections/id</td>
<td>updateSection(id, ScheduledSection object)</td>
</tr>
<tr>
<td>POST</td>
<td>/sections/id</td>
<td>updateSection(id, post-parameters)</td>
</tr>
<tr>
<td>DELETE</td>
<td>/sections/id</td>
<td>deleteSection(id)</td>
</tr>
</tbody>
</table>

**Table 8 HTTP Method Mapping**

**Method Description**

**GET /sections/[id]**

Return a json or xml representation of a section given a section's unique identifier.

**PUT /sections/[id]**

Update the representation of a section given its unique identifier.

**POST /sections/[id]**

Post takes four post-parameters courseId=#, timeblockId=#, instructorId=#, and roomId=#. This updates the section who identifier matches [id] replacing the current value of the post-parameters with the newly provided parameters.

**DELETE /sections/[id]**

Hard removes the section with identifier [id] from the database.
**GET /sections**
Returns a list of all sections currently in the database. There is currently no data paging.

**POST /sections**
Post takes four post-parameters courseId=#, timeblockId=#, instructorId=#, and roomId=#. This creates a new section object in the database. The newly created resource's URI is returned in the HTML response.
### 3.2.2.5.5. InstructorController Class

![Figure 26 Instructor Controller Class](image)

Responsible for handling requested mapped to /instructors/ URI. Objects maps requests to the underlying InstructorDAO object for retrieval, insertion, and updating the stored object in the back-end database.

**HTTP-Controller Method Mapping**

*Table 9* below lists the HTTP Method to Docket Controller mapping. Details on return values are listed in subsequent sections.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>URI</th>
<th>Controller Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/instructors</td>
<td>allInstructors()</td>
</tr>
<tr>
<td>POST</td>
<td>/instructors</td>
<td>createInstructor(post-parameters)</td>
</tr>
<tr>
<td>GET</td>
<td>/instructors/id</td>
<td>getInstructorById(id)</td>
</tr>
<tr>
<td>PUT</td>
<td>/instructors/id</td>
<td>updateInstructor(id, Instructor object)</td>
</tr>
<tr>
<td>POST</td>
<td>/instructors/id</td>
<td>updateInstructor(id, post-parameters)</td>
</tr>
<tr>
<td>DELETE</td>
<td>/instructors/id</td>
<td>deleteInstructor(id)</td>
</tr>
</tbody>
</table>

*Table 9 HTTP Method Mapping*
Method Description

GET /instructors/[id]
Return a json or xml representation of an instructor given an instructor's unique identifier.

PUT /instructors/[id]
Update the representation of an instructor given its unique identifier.

POST /instructors/[id]
Post takes two post-parameters firstName=string, and lastName=string. This updates the instructor who identifier matches [id] replacing the current value of the post-parameters with the newly provided parameters.

DELETE /instructors/[id]
Hard removes the instructor with identifier [id] from the database.

GET /instructors
Returns a list of all instructors currently in the database. There is currently no data paging.

POST /instructors
Post takes two post-parameters firstName=string, and lastName=string. This creates a new instructor object in the database. The newly created resource's URI is returned in the HTML response.

3.2.2.5.6. RoomController Class

```java
docket.controller.RoomController

ALL  ►  ◄  ....►  ....◄  ►  ◄  ....►  ◄  ....►  ◄  ... ►
- RoomDAO roomDAO
- UriInfo uriInfo
- Room.RoomList allRooms()
- Response createRoom(String name, int capacity)
- Room getRoomById(long id)
- void updateRoom(long id, Room room)
- void updateRoom(long id, String name, Integer capacity)
- void deleteRoom(long id)
```

Figure 27 Room Controller Class
HTTP-Controller Method Mapping

Table 10 below lists the HTTP Method to Docket Controller mapping. Details on return values are listed in subsequent sections.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>URI</th>
<th>Controller Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/rooms</td>
<td>allRooms()</td>
</tr>
<tr>
<td>POST</td>
<td>/rooms</td>
<td>createRoom(post-parameters)</td>
</tr>
<tr>
<td>GET</td>
<td>/rooms/id</td>
<td>getCourseById(id)</td>
</tr>
<tr>
<td>PUT</td>
<td>/rooms/id</td>
<td>updateRoom(id, Room object)</td>
</tr>
<tr>
<td>POST</td>
<td>/rooms/id</td>
<td>updateRoom(id, post-parameters)</td>
</tr>
<tr>
<td>DELETE</td>
<td>/rooms/id</td>
<td>deleteRoom(id)</td>
</tr>
</tbody>
</table>

Table 10 HTTP Method Mapping

Method Description

GET /rooms/[id]
Return a json or xml representation of a room given a room's unique identifier.

PUT /rooms/[id]
Update the representation of a room given its unique identifier.

POST /rooms/[id]
Post takes two post-parameters name=string, and capacity=#. This updates the room who identifier matches [id] replacing the current value of the post-parameters with the newly provided parameters.

DELETE /rooms/[id]
Hard removes the room with identifier [id] from the database.

GET /rooms
Returns a list of all rooms currently in the database. There is currently no data paging.

POST /rooms
Post takes two post-parameters name=string, and capacity=#. This creates a new instructor object in the database. The newly created resource's URI is returned in the HTML response.

3.2.2.5.7. **TimeBlockController Class**
### Figure 28 Timeblock Controller Class

Responsible for handling requested mapped to `/timeblock/` URI. Objects maps requests to the underlying TimeblockDAO object for retrieval, insertion, and updating the stored object in the back-end database.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeBlockDAO timeBlockDAO</td>
<td></td>
</tr>
<tr>
<td>UriInfo uriInfo</td>
<td></td>
</tr>
<tr>
<td>TimeBlock.TimeBlockList allTimeBlocks()</td>
<td></td>
</tr>
<tr>
<td>Response createTimeBlock(int startHour, int startMinute, int endHour, int endMinute, List&lt;String&gt; days)</td>
<td></td>
</tr>
<tr>
<td>TimeBlock getTimeBlockById(long id)</td>
<td></td>
</tr>
<tr>
<td>void updateTimeBlock(long id, TimeBlock block)</td>
<td></td>
</tr>
<tr>
<td>void updateTimeBlock(long id, Time startTime, Time endTime, Set&lt;Day&gt; days)</td>
<td></td>
</tr>
<tr>
<td>void deleteTimeBlock(long id)</td>
<td></td>
</tr>
</tbody>
</table>
HTTP-Controller Method Mapping

Table 11 lists the HTTP Method to Docket Controller mapping. Details on return values are listed in subsequent sections.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>URI</th>
<th>Controller Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/timeblock</td>
<td>allTimeBlocks()</td>
</tr>
<tr>
<td>POST</td>
<td>/timeblock</td>
<td>createTimeBlock(post-parameters)</td>
</tr>
<tr>
<td>GET</td>
<td>/timeblock/id</td>
<td>getTimeBlockById(id)</td>
</tr>
<tr>
<td>PUT</td>
<td>/timeblock/id</td>
<td>updateTimeBlock(id, TimeBlock object)</td>
</tr>
<tr>
<td>POST</td>
<td>/timeblock/id</td>
<td>updateTimeBlock(id, post-parameters)</td>
</tr>
<tr>
<td>DELETE</td>
<td>/timeblock/id</td>
<td>deleteTimeBlock(id)</td>
</tr>
</tbody>
</table>

Table 11 HTTP Method Mapping

Method Description

GET /timeblock/[id]
   Return a json or xml representation of a timeblock given a room's unique identifier.

PUT /timeblock/[id]
   Update the representation of a timeblock given its unique identifier.

POST /timeblock/[id]
   Post takes five post-parameters startHour=# between 0 – 23, startMinute=# between 0 -59, endHour=# between 0 -23, endMinute=# between 0 – 59, and days=List[String] of days offered. This updates the timeblock who's identifier matches [id] replacing the current value of the post-parameters with the newly provided parameters.

DELETE /timeblock/[id]
   Hard removes the timeblock with identifier [id] from the database.

GET /timeblock
   Returns a list of all timeblocks currently in the database. There is currently no data paging.

POST /timeblock
   Post takes five post-parameters startHour=# between 0 – 23,
startMinute=# between 0 -59, endHour=# between 0 -23,
endMinute=# between 0 - 59, and days=List[String] of days
offered. The newly created resource's URI is returned in the HTML
response
3.2.2.6. Notification

Notifications are a consumable entry in DSS. If a user is subscribed to a course, changes to said course trigger by other users result in a new notification stored in DSS. Each notification is associated to a user by a user’s unique identify.

3.2.2.6.1. Notification Process

A front end application that connects to DSS can poll the DSS notification service periodically to check for notifications associated to the current authentication token. All notifications are a single consumable message. Retrieval of notifications results in their deletion on DSS’ service. Any notification left in DSS at midnight trigger DSS’ sending of all user notifications to the email address stored for said user.

The sequence diagram in Figure 27 shows how Docket DSS works with the JavaMailSender component of SpringWS to send notifications to users.
3.2.2.6.2. **Mailer Class**

The Mailer class is responsible for sending notifications to a user’s email address. This happens whenever a notification in DSS is not retrieved via polling by midnight. A list of notifications and email address are handed to the mailer class for transmission. The messages are turned into an email body and assigned a subject. The mailer throws a MessagingException in the event of a failure.

3.2.2.7. **Subscription**

A service that enables users to subscribe to courses that they are interested in and get notifications when something changes in that course. Users have a profile that has courses that they are subscribed to in the DSS where an event listener checks for changes in those courses. Any changes to a subscribed course trigger a notification action.

3.2.2.7.1. **Subscription Process**

A front-end subscribes users to a course through the DSS services. This is a separate process from polling for notifications. A front-end also unsubscribes a user from a course through the DSS services.

3.2.2.8. **Exceptions**

3.2.2.8.1. **BadRequestException**

The BadRequestException Class is an exception object thrown by Resource Request Handlers in response to request whose post-
parameters specify illegal values. Please see “Resource Request Handlers” section of this document for more information on valid and invalid values.
3.2.2.9. Application Configuration

3.2.2.9.1. Application Class

The Application Class is the main application entry point. This class handles instantiating the web server and running it on this host machine. The Application Class reads the SYSTEM.HOST and the SYSTEM.PORT properties to determine the base URL and port name to initialize the application with, respectively.

3.2.2.10. Swagger

Swagger is a framework used to visualize the Docket DSS for documentation purposes. Swagger is an annotation-driven framework that exposes the Docket DSS API in a web interface. You can see the complete API by navigating to http://kex.cs.drexel.edu/docket/dev/api

3.2.2.10.1. ApiListingResource Class

This call defines the URI location of the Swagger API portion of the application. This class also defines the payload type that is sent. Docket strictly returns a JSON payload of the API commands. This
JSON is then processed by a Swagger instance and translated into viewable HTML.
3.2.2.10.2. **ApiOriginFilter Class**

![ApiOriginFilter Class](image)

ApiOriginFilter is hooked into all Swagger URI requests. A ServletRequest is passed to doFilter which appends Access-Control-Allow-* headers to the ServletResponse. This particular filter adds Access-Control-Allow-Origin: "*", Access-Control-Allow-Methods: "GET,POST,DELETE,PUT", and Access-Control-Allow-Headers: "Content-Type". These additional headers are digested by a Swagger instance which further modifies the generated HTML output.

3.2.2.10.3. **ServletBootstrapper Class**

![ServletBootstrapper Class](image)

The ServletBootstrapper class is the servlet object for Docket's configuration to make use of the Swagger annotations.

4. Requirements Traceability Matrix

<table>
<thead>
<tr>
<th>SRS</th>
<th>SDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4.2.1.1</td>
<td>3.2.5.9.1</td>
</tr>
<tr>
<td>R4.2.1.2</td>
<td>3.2.5.9.1</td>
</tr>
<tr>
<td>R4.2.1.3</td>
<td>3.1</td>
</tr>
<tr>
<td>R4.2.2.1</td>
<td>3.2.5.9.1</td>
</tr>
<tr>
<td>R4.2.2.2</td>
<td>3.2.5.9.1</td>
</tr>
<tr>
<td>R4.2.2.3</td>
<td>3.2.5.9.1</td>
</tr>
<tr>
<td>R4.2.2.4</td>
<td>3.2.5.9.1</td>
</tr>
<tr>
<td>Section</td>
<td>Reference</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>R4.2.2.5</td>
<td>3.2.5.9.1</td>
</tr>
<tr>
<td>R4.3.1</td>
<td>3.1.4</td>
</tr>
<tr>
<td>R4.3.1.1</td>
<td>3.1.4</td>
</tr>
<tr>
<td>R4.3.1.2</td>
<td>3.1.4</td>
</tr>
<tr>
<td>R4.3.2</td>
<td>3.1.4</td>
</tr>
<tr>
<td>R4.3.3</td>
<td>3.1.4</td>
</tr>
<tr>
<td>R4.3.4</td>
<td>3.1.4</td>
</tr>
<tr>
<td>R4.3.5</td>
<td>3.1.4</td>
</tr>
<tr>
<td>R4.3.6</td>
<td>3.1.5.6</td>
</tr>
<tr>
<td>R4.3.7</td>
<td>3.1.5.6</td>
</tr>
<tr>
<td>R4.3.8</td>
<td>3.1.5.6</td>
</tr>
<tr>
<td>R4.4.1</td>
<td>3.2.5.1.1</td>
</tr>
<tr>
<td>R4.4.1.1</td>
<td>3.2.5.1.1</td>
</tr>
<tr>
<td>R4.4.1.2</td>
<td>3.2.5.1.1</td>
</tr>
<tr>
<td>R4.4.1.3</td>
<td>3.2.5.1.1</td>
</tr>
<tr>
<td>R4.4.1.4</td>
<td>3.2.5.1.1</td>
</tr>
<tr>
<td>R4.4.1.5</td>
<td>3.2.5.1.1</td>
</tr>
<tr>
<td>R4.4.1.6</td>
<td>3.2.5.1.1</td>
</tr>
<tr>
<td>R4.4.2</td>
<td>3.2.5.4.2, 3.2.5.5.1</td>
</tr>
<tr>
<td>R4.5.1</td>
<td>3.2.5.1.3</td>
</tr>
<tr>
<td>R4.5.1.1</td>
<td>3.2.5.1.3</td>
</tr>
<tr>
<td>R4.5.1.2</td>
<td>3.2.5.1.3</td>
</tr>
<tr>
<td>R4.5.1.3</td>
<td>3.2.5.1.3</td>
</tr>
<tr>
<td>R4.5.2</td>
<td>3.2.5.4.4, 3.2.5.5.3</td>
</tr>
<tr>
<td>R4.6.1</td>
<td>3.2.5.1.4</td>
</tr>
<tr>
<td>R4.6.1.1</td>
<td>3.2.5.1.4</td>
</tr>
<tr>
<td>R4.6.1.2</td>
<td>3.2.5.1.4</td>
</tr>
<tr>
<td>R4.6.1.3</td>
<td>3.2.5.1.4</td>
</tr>
<tr>
<td>R4.6.1.4</td>
<td>3.2.5.1.4</td>
</tr>
<tr>
<td>R4.6.2</td>
<td>3.2.5.4.5, 3.2.5.5.4</td>
</tr>
<tr>
<td>R4.6.3</td>
<td>3.2.5.4.5, 3.2.5.5.4</td>
</tr>
<tr>
<td>R4.7.1</td>
<td>3.2.5.1.2</td>
</tr>
<tr>
<td>R4.7.1.1</td>
<td>3.2.5.1.2</td>
</tr>
<tr>
<td>R4.7.1.2</td>
<td>3.2.5.1.2</td>
</tr>
<tr>
<td>R4.7.1.3</td>
<td>3.2.5.1.2</td>
</tr>
<tr>
<td>R4.7.1.4</td>
<td>3.2.5.1.2</td>
</tr>
<tr>
<td>R4.7.1.5</td>
<td>3.2.5.1.2</td>
</tr>
<tr>
<td>R4.7.1.6</td>
<td>3.2.5.1.2</td>
</tr>
<tr>
<td>R4.7.1.7</td>
<td>3.2.5.1.2</td>
</tr>
<tr>
<td>R4.7.2</td>
<td>3.2.5.4.3, 3.2.5.5.2, 3.2.5.8.1</td>
</tr>
<tr>
<td>R4.7.2.1</td>
<td>3.2.5.4.3, 3.2.5.5.2, 3.2.5.8.1</td>
</tr>
<tr>
<td>R4.7.2.2</td>
<td>3.2.5.4.3, 3.2.5.5.2, 3.2.5.8.1</td>
</tr>
<tr>
<td>R4.7.2.3</td>
<td>3.2.5.4.3, 3.2.5.5.2, 3.2.5.8.1</td>
</tr>
<tr>
<td>R4.7.3</td>
<td>3.2.5.4.3, 3.2.5.5.2</td>
</tr>
<tr>
<td>R4.8.1</td>
<td>3.1.5.3, 3.1.5.4, 3.1.5.5, 3.1.5.7</td>
</tr>
<tr>
<td>R4.8.2</td>
<td>3.1.5.3, 3.1.5.4, 3.1.5.5, 3.1.5.7</td>
</tr>
<tr>
<td>R4.8.2.1</td>
<td>3.1.5.3, 3.1.5.4, 3.1.5.5, 3.1.5.7</td>
</tr>
<tr>
<td>R4.8.2.2</td>
<td>3.1.5.3, 3.1.5.4, 3.1.5.5, 3.1.5.7</td>
</tr>
<tr>
<td>R4.8.2.3</td>
<td>3.1.5.3, 3.1.5.4, 3.1.5.5, 3.1.5.7</td>
</tr>
<tr>
<td>R4.8.3</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.3.1</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.3.2</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.3.3</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.3.4</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.3.4.1</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.3.4.2</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.3.4.3</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.3.4.4</td>
<td>3.1.5.5</td>
</tr>
<tr>
<td>R4.8.4</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.8.4.1</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.8.4.2</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.8.4.2</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.8.4.3</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.8.4.3.1</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.8.4.3.2</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.8.4.3.3</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.8.4.3.4</td>
<td>3.1.5.7</td>
</tr>
<tr>
<td>R4.9.1</td>
<td>3.2.5.7</td>
</tr>
<tr>
<td>R4.9.2</td>
<td>3.2.5.6, 3.2.5.7</td>
</tr>
<tr>
<td>R4.9.3</td>
<td>3.2.5.7</td>
</tr>
<tr>
<td>N5.1.1</td>
<td>3.2.2, 3.2.3.2</td>
</tr>
<tr>
<td>N5.1.1.1</td>
<td>3.2.2, 3.2.3.2</td>
</tr>
<tr>
<td>N5.1.1.2</td>
<td>3.2.2, 3.2.3.2</td>
</tr>
<tr>
<td>N5.1.1.3</td>
<td>3.2.2, 3.2.3.2</td>
</tr>
<tr>
<td>N5.1.1.4</td>
<td>3.2.2, 3.2.3.2</td>
</tr>
<tr>
<td>N5.1.2</td>
<td>3.2.3.1, 3.2.5.5</td>
</tr>
<tr>
<td>N5.1.2.1</td>
<td>3.2.3.1, 3.2.5.5</td>
</tr>
<tr>
<td>N5.1.2.2</td>
<td>3.2.3.1, 3.2.5.5</td>
</tr>
<tr>
<td>N5.2.1</td>
<td>3.1.1</td>
</tr>
</tbody>
</table>
5. References

Spring


"Spring Framework - Chapter 22 Email." SpringSource.org. Spring, Retrieved February 1 2013, from http://static.springsource.org/spring/docs/2.0.8/reference/mail.html

Spring Security


REST


Hibernate ORM


Unobtrusive JavaScript

**AngularJS**


6. Appendix