# Step 1. Configure the project in Eclipse.

* Refer: http://192.168.3.122:3690/svn/cresteldocs/Product/Mediation- CGF/SM\_Revamp\_7.0/7.0/Design/Development\_Guide

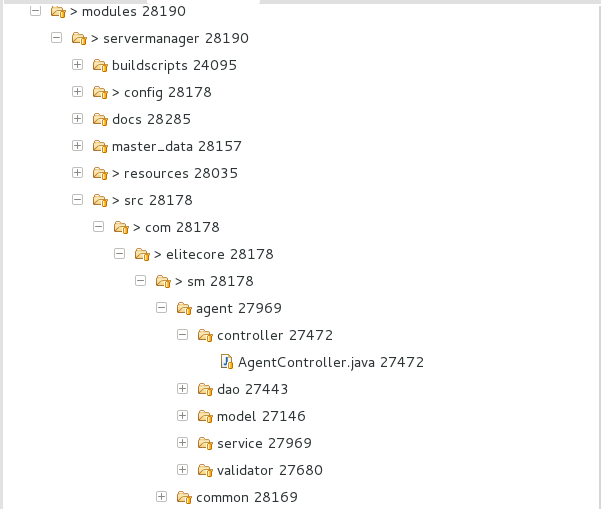
## 

* SVN Path: http://192.168.0.181:3690/svn/crestelmediation/projects/SM\_REVAMP/ServerManager/trunk

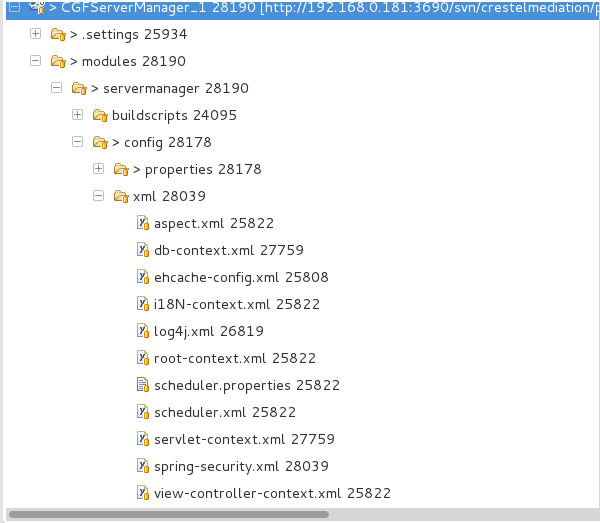
Note: Host Entry for crestelsvn

**192.168.0.181 crestelsvn**

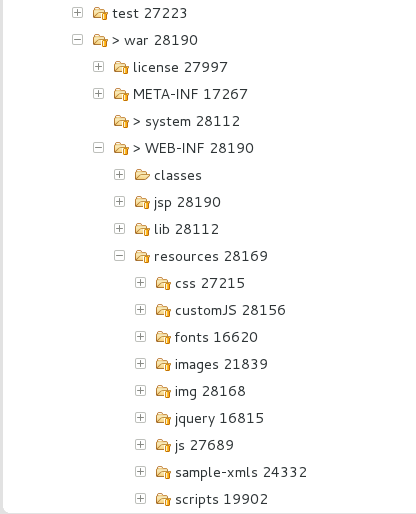
# Step 2. Project Structure



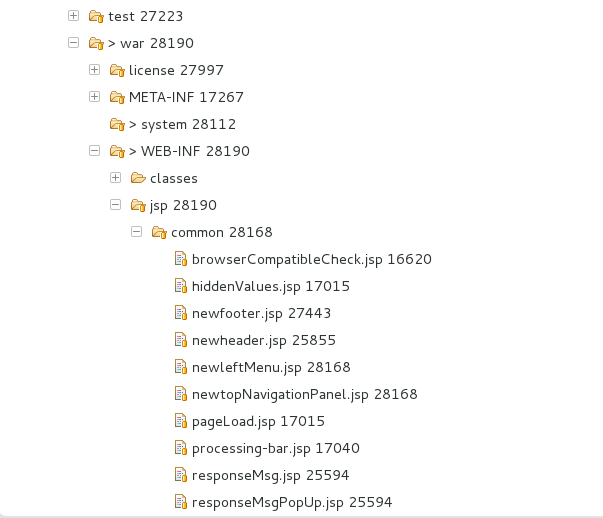
## Source (src) folder contains the Java Code. It have been sliced based on the modules (agent,common,composer,config,dashboard,device,server,serverinstance etc.)



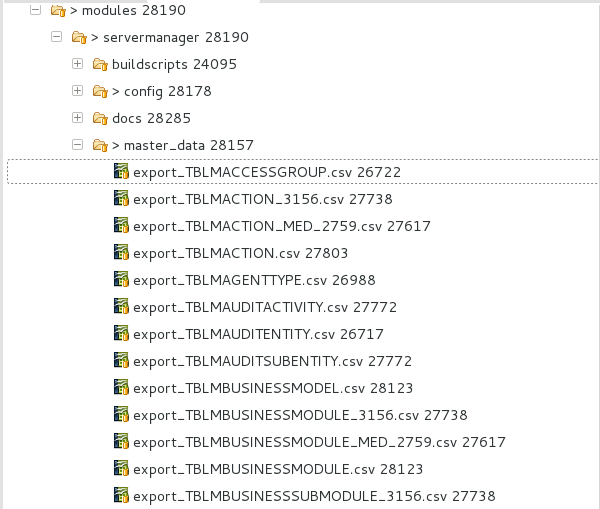
* **config** folder contains all the spring configuration:



* **resources** contains css, scripts, images, fonts that will be used in jsp pages.



* **jsp** contains jsp pages sliced based on modules.



* **masterdata** contains csv for master data to be inserted in db.

**Step 3. Module Creation**

*1) First of all identify the Menu, Sub-Menu (Tab) and user actions from Wireframe and then add entries of Module, Sub-Module and Action in export\_TBLMBUSINESSMODULE.csv, export\_TBLMBUSINESSSUBMODULE\_3156.csv and export\_TBLMACTION.csv respectively. Then fire the ANT build using liquibase\_build\_export.xml.*

1. Then give action rights to the Admin Staff. While development we can add data into export\_TBLTACCESSGROUPACTIONREL.csv. *Then fire the ANT build using liquibase\_build\_export.xml.*
2. To add any system paramter, add data in export\_TBLMSYSTEMPARAMETER.csv .*Then fire the ANT build using liquibase\_build\_export.xml.*
3. Create Model
4. Create interface DAO that will extend GenericDAO<T>
5. Create class DAOImpl that will extend GenericDAOImpl<T> and implement interface DAO (Step 5 DAO).
6. Create interface Service
7. Create class ServiceImpl that will implement Service (Step 7 Service).
8. Create class Controller that will extend BaseController.
9. Create class Validator (if any). If any regex needs to be added, then add that Regex to the export\_TBLMREGEX.csv and if any range validation, add it in export\_TBLMVALIDATIONRANGE.csv. *Then fire the ANT build using liquibase\_build\_export.xml.*

While using @Validated, the sequence of the BindingResult has to be next to the

@Validated Annotation parameter. Eg:-

@Validated @ModelAttribute(FormBeanConstants.ACCESS\_GROUP\_FORM\_BEAN) AccessGroup accessGroup ,

BindingResult result,

...,

...

1. Add the package name to register the Model for hibernate mapping in db- context.xml:

<beans:property name="packagesToScan" value="com.elitecore.sm.config.model, com.elitecore.sm.iam.model, com.elitecore.sm.systemparam.model" />

1. Add the package name for component scanning to register DAOImpl, ServiceImpl, Controller in servlet-context.xml:

<context:component-scan base-package="com.elitecore.sm.config.dao"/>

<context:component-scan base-package="com.elitecore.sm.config.service" />

<context:component-scan base- package="com.elitecore.sm.systemparam.controller" />

<context:component-scan base- package="com.elitecore.sm.systemparam.validator" />

1. Create View (jsp page).
   * While using jquery, we need to add the following statement..

<script src=*"js/jquery-ui.js"*></script>

* + To add header and footer,

<jsp:include page=*"../common/newheader.jsp"*></jsp:include>

<jsp:include page=*"../common/newfooter.jsp"*></jsp:include>

* + Use of Custom Tags:

1. If needed, create a new .tag file with your custom name at location

-/WEB-INF/tags

* + You can add attributes which will be pass from JSP.

<%@ attribute name="type" required="true" %>

<%@ attribute name="inputClassName" required="false" %>

* + You can use the different taglibs in your custom tag file.

<%@ taglib prefix="c" [uri="http://java.sun.com/jsp/jstl/core](http://java.sun.com/jsp/jstl/core)" %>

<%@taglib [uri="http://www.springframework.org/ta](http://www.springframework.org/tags)gs" prefix="spring"%>

1. In JSP,
   * Declare your tag at top:

<%@ taglib tagdir="/WEB-INF/tags" prefix="elitecore" %>

* + Usage:

<elitecore:inputHTML type="text" name="username" id="username" i18NCode="index.page.username" inputClassName="form-control"

></elitecore:inputHTML>

* Points to be kept in mind:

1. Don't use flush or clear in hibernate session.
2. If you want to trace spring web exception then in log4j.xml , change log level "DEBUG".

<logger name="org.springframework.web">

<!-- To get the spring web exceptions -->

<level value="DEBUG" />

<appender-ref ref="sm-spring" />

</logger>

3. While enctype="multipart/form-data" use then not able to get value of hidden parameters using request.getParameter("name"), it always return Null.

4. If the spring config executes the config initializer 2 time using thread:

* + localhost-startStop-1
  + http-bio-8080-exec-3

Then reason can be there will be some Bean Initializing Problem or Autowiring problem. So spring retries to initialize the config.

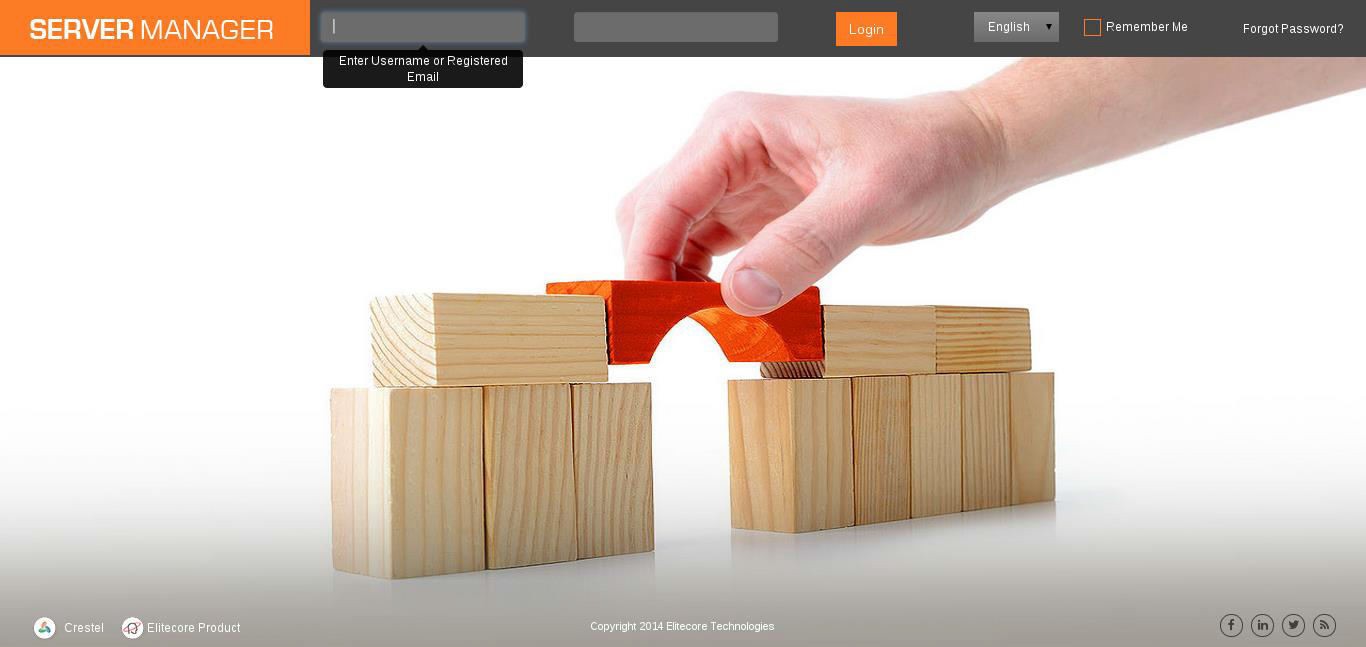
### 5. After Some attempts if fail to get value of hidden parameters and requests are discarded then may be problem is increase request size on every attempt.

check Http request information using mozilla firefox add-on named "HTTPFOX".

# Step 4. Flow between Browser and Controller

* 1. User enters the URL:  *http://localhost:9191/CGFServerManager/ http://localhost:9191/CGFServerManager/welcome*

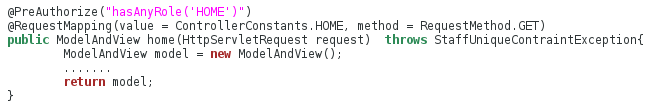
It will invoke the HomeController method, where the */* or */welcome* url is mapped. And user will be redirected to the Login Page.



* 1. User enters credentials and on Submit the request is sent to the *j\_spring\_security\_check* which is part of Spring Security. Internally it invokes the *LimitLoginAuthenticationServiceImpl#authenticate* to authenticating the user credentials from *SpringLoginServiceImpl#loadUserByUsername*.

If username is available in database, User model of Spring is returned duly filled with Authorities user has. So that ACL is handed over to Spring Security rather managing at our end. After successful authentication user will be redirected to the

***/home*** (*HomeController#home*).



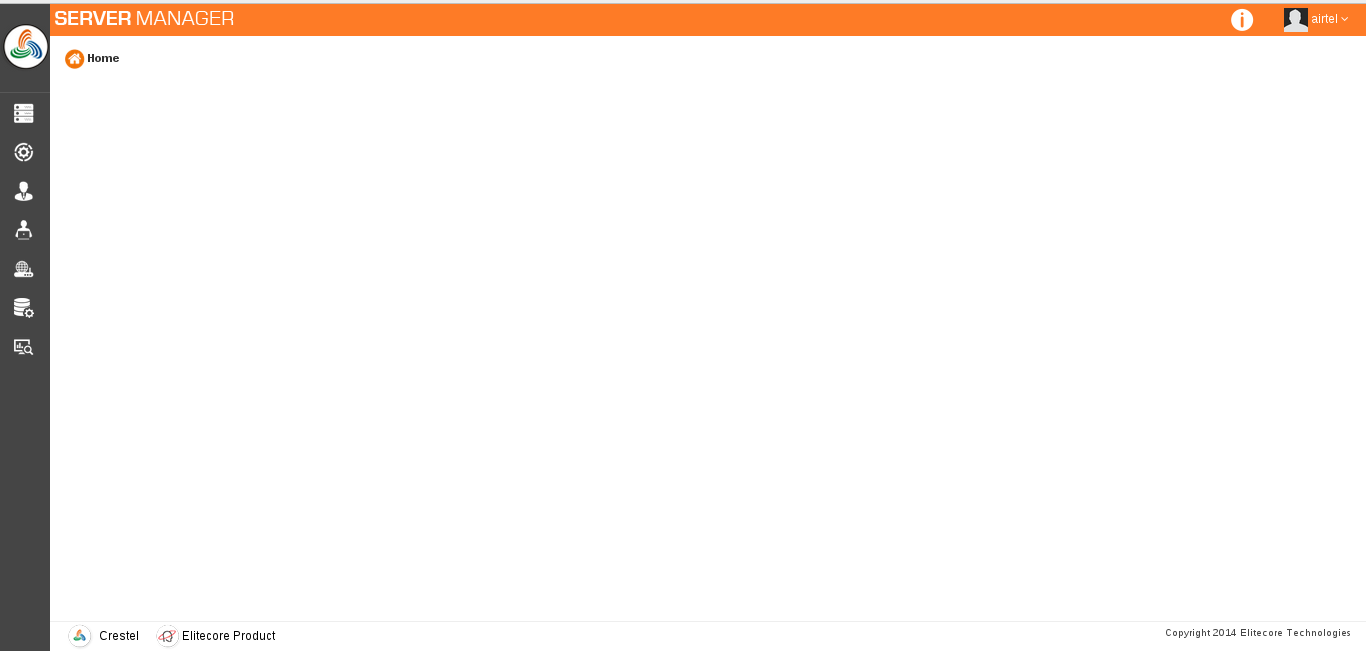
*@PreAuthorize* – will check that logged-in user has authority to call this function or not. If not, then it will throw AccesDeniedException which will redirect user to error/403 (*ExceptionHandlingController#handleAccessDeniedException*)

*@RequestMapping* – value is the url which we want to make this method call and method depict by which http method can this method be called.

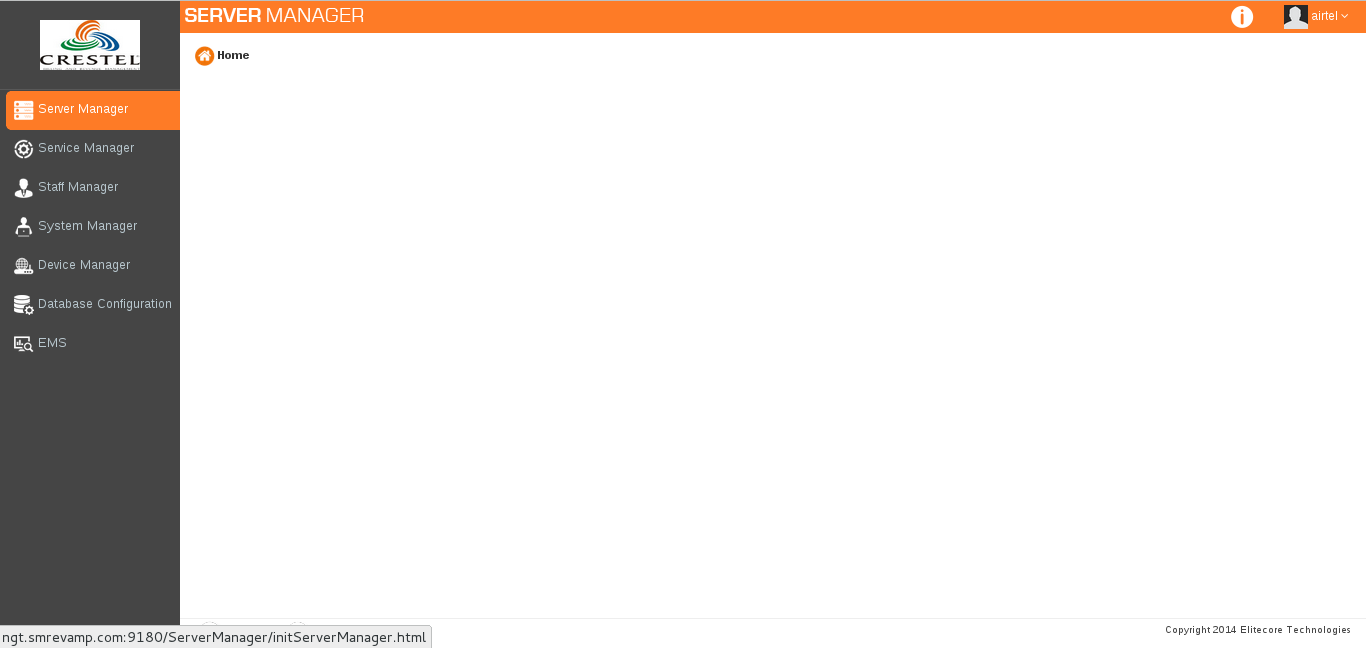
*ModelAndView* – this will have:

* + - View Name (eg: for login.jsp, view name would be *login*). model.setViewName(ViewNameConstants.***HOME\_PAGE***);
* Set of objects in form of key-value model.addObject("username", springUserBean.getUsername());

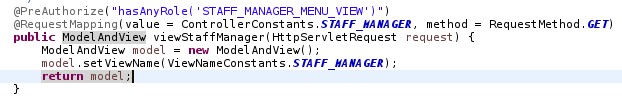
# Home Page View



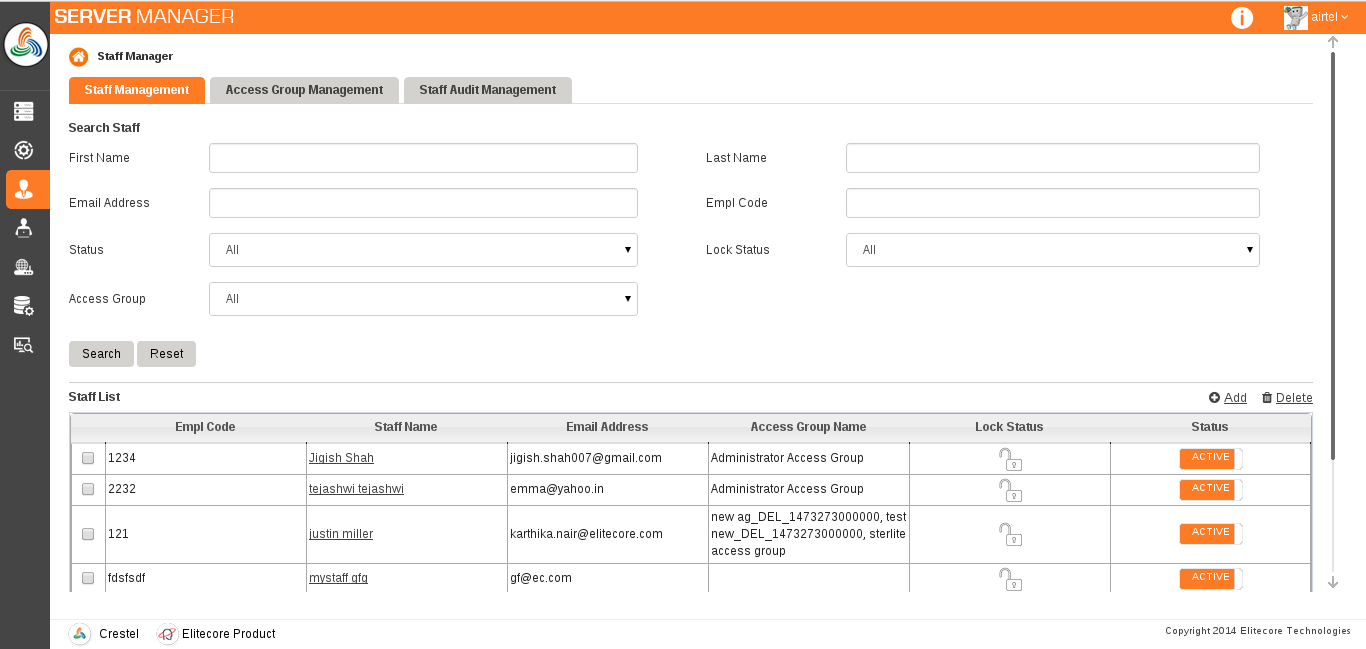
**Left Menu View**



4.3) When user clicks “Staff Manager” from left menu, the /staffManager (StaffController#viewStaffManager) will be invoked:

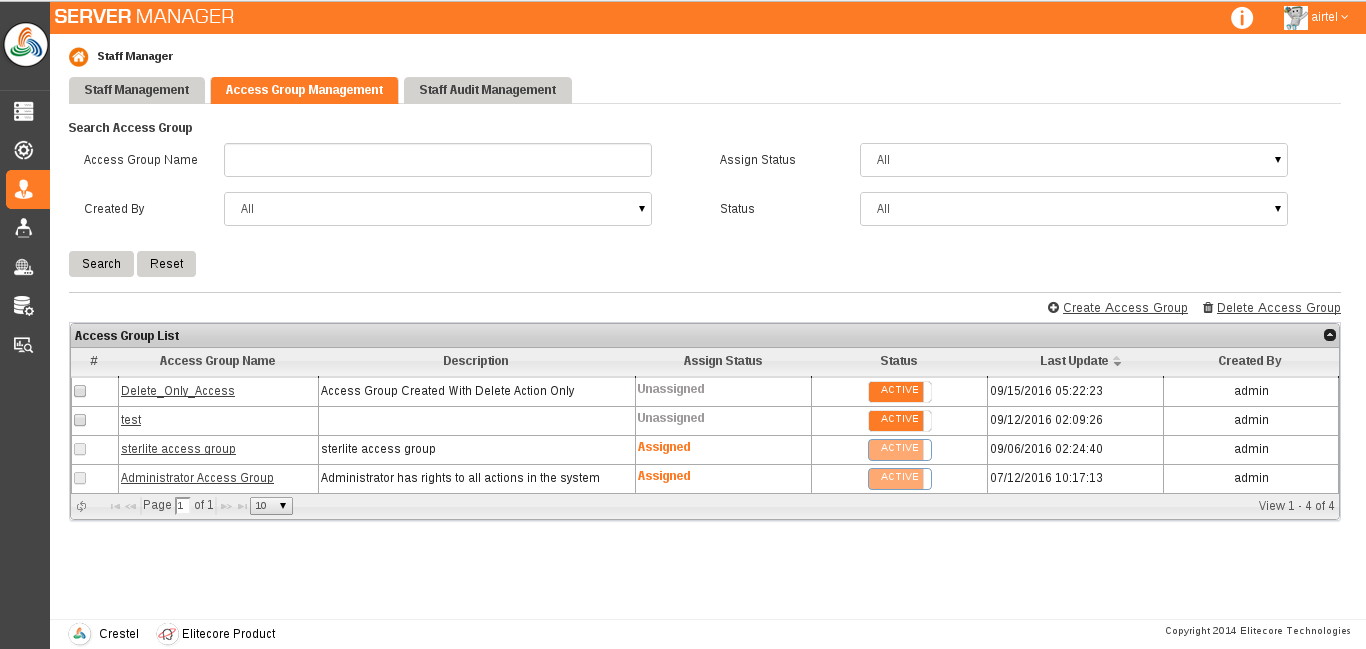


This will redirect user to view iam/staffManager.



# Step 5. Flow between Controller-Service-Dao

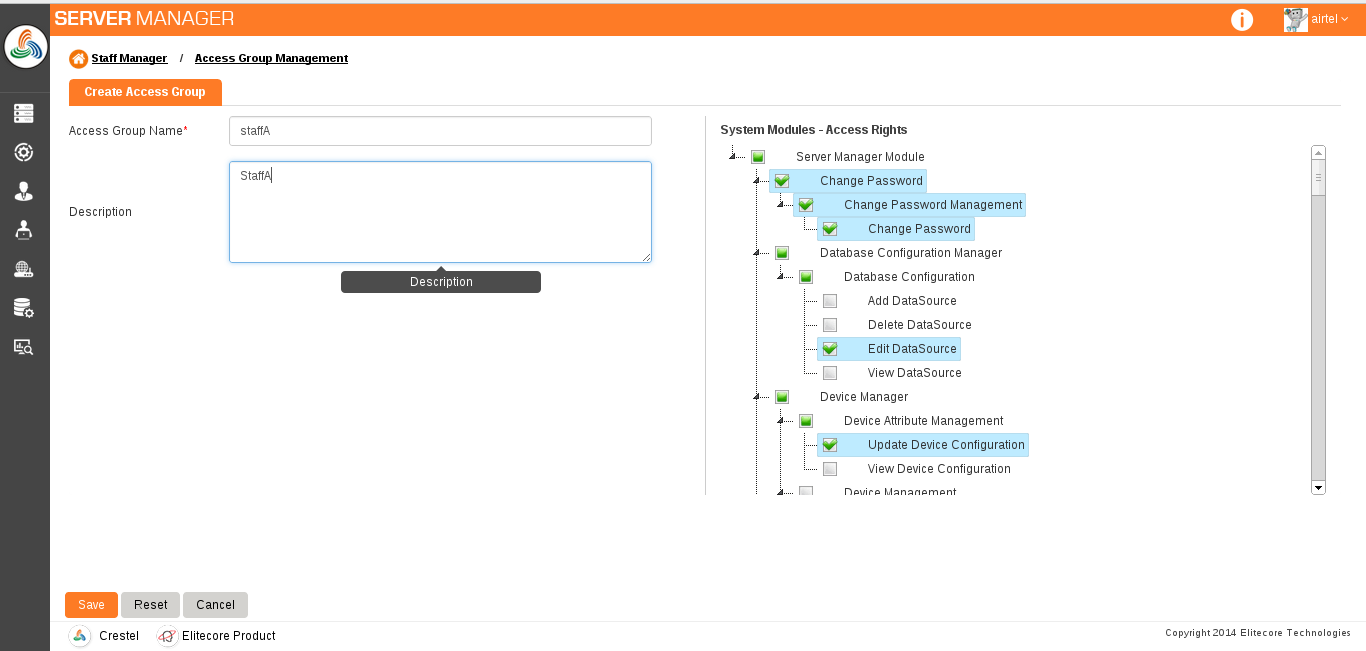
* 1. User clicks “Access Group Management” Tab



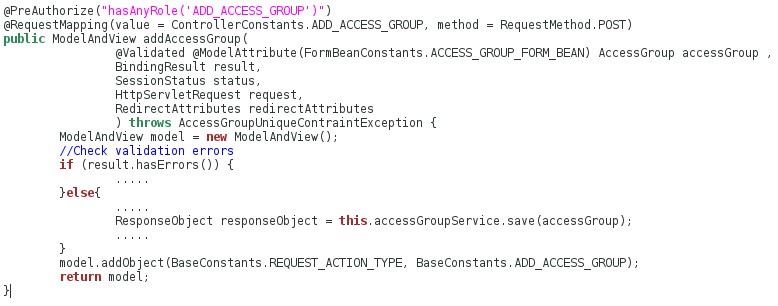
* 1. User clicks on “Create Access Group”. This will invoke /initAddAccessGroup (AccessGroupController#initAddAccessGroup)



Here we set the AccessGroup Object to map to the addAccessGroup.jsp page form.



* 1. User fills details and click on Save. It will invoke addAccessGroup (AccessGroupController#addAccessGroup).

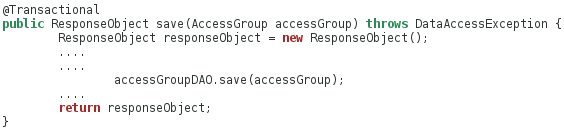


AccessGroup - Model will automatically be set by the spring.

@Validated - Will call the AccessGroupValidator#validate for validation.

BindingResult - This parameter should always be next to the @Validated parameter. This will contain errors that are set by Validator class.

* 1. In above code of addAccessGroup: this.accessGroupService.save(accessGroup); will invoke the AccessGroupServiceImpl#save

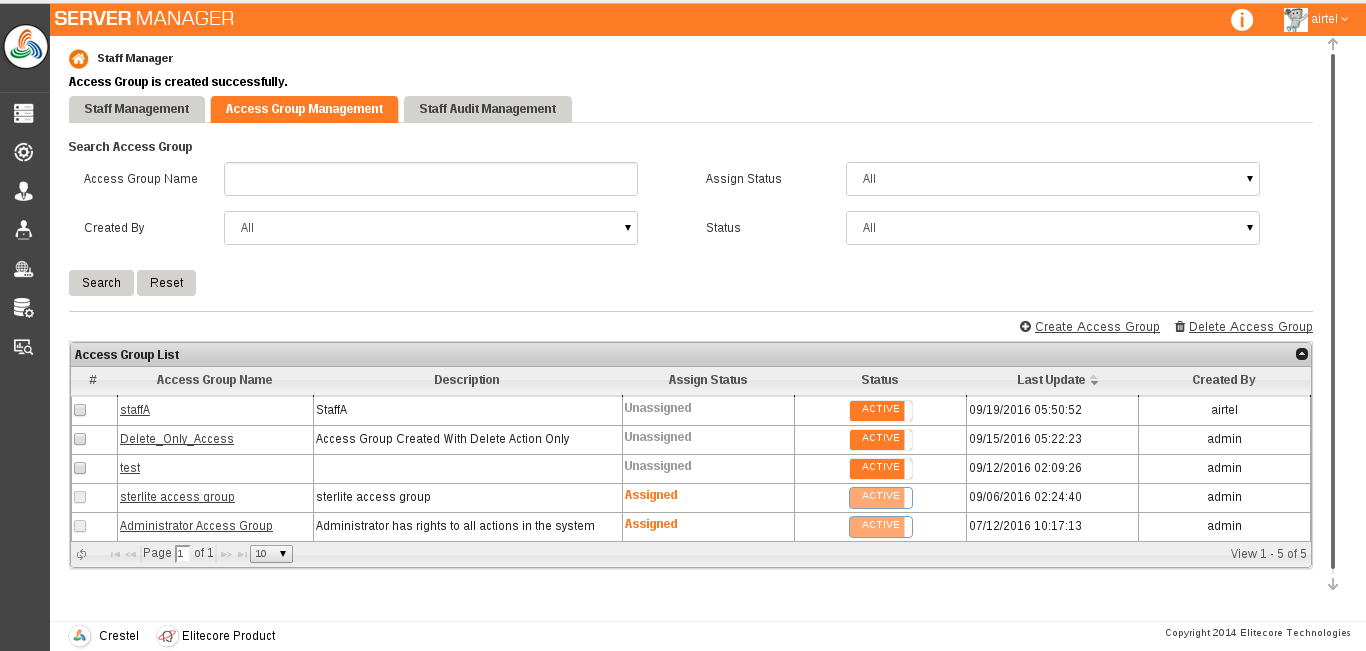


* 1. In above code of save: accessGroupDAO.save(accessGroup);

will invoke the AccessGroupDAOImpl->GenericDAOImpl<T>#save



Common method that can be used for all models are kept under GenericDAOImpl<T>.



So, out flow depicts:

* + 1. **View** – UI part that will be displayed to user.
    2. **Controller** – Mapping the request url to forward request and validate the data.
    3. **Service** – Business Logic and transaction management of hibernate.

4) **Dao** – Interaction with database.

\*\*\*\*\*\*\*