Glossary:

Model Manager: the back-end desktop based application used for creating project models by defining entities, properties and user permissions. The project and entity collection is then exposed to the front-end traceability tool for creating instances and linking it up. This is implemented using Swing framework and RestFul webservice for persisting the project collections in MongoDB.

Local Repository: A temporary folder is used for storing the project collection and entity collections. This design choice was adopted to avoid multiple hit to the database which aims at improving the system performance.

ModelClass package:

1. Project Collection.java- composes Projects.java
2. Projects.java- composes Model.java
3. Model.java- composes Entities.java
4. Entities.java- depends on Property.java (association)

Scenario-1 (Login)

Concern: Confidentiality, Non-repudiation

Model manager should be made available to those who are authorized to access them. Only the project managers, admin and group leads should be given access to the desktop based model manager tool.

Components Realized: LoginComponent, LoginService

File Name: Login.java
File Description: This screen is used to authenticate the users signing into model manager. A frame is displayed to capture the username and password from the user and is validated against the set of authorized users present in the system.
Function present: authenticateUserAction()
Mapped to element: LoginService

Description:

Function name	authenticateUserAction
Function trigger Point	User provides his username and password and hits the “login” button– actionListener triggers the actionPerformed event
Function parameters	void
Function return type	void
Functions description	The function takes the entered values of username and password from the user, It invokes a service LoginService and displays an alert box with a message of “Login Success” or “Login Failure”. If login successful it proceeds to the menu screen.

Function pseudo code	// Display fields for entering username and password
	
	authenticateUserAction ()
	{
	String userName = userNameField.getText();
	String password = pwd.getText();
	
	//call the login service by passing username and password.

	CloseableHttpClient httpClient = HttpClients.createDefault();
	HttpPost postRequest = new HttpPost("http://sase-server3.ssn.net:8080/TraceabilityServicesProvider-0.0.1-SNAPSHOT/Model/**LoginService**");
	
	postRequest.setEntity(username+” ”+ password);
	response = httpClient.execute(postRequest);
	
	if (response.equals("Success")
	Load "ProjectManager.java"
	else
	JOptionPane.showMessageDialog("Login unsuccessful")
	}

Sample User Collection:

```
{"users":{

        "employees": [{
        "employeeName" : "Arul Dhana Saam Prakash",
        "employeeId" : "SSK002",
        "password" : "aruldhana",
        "organizationRole" : "Developer",
        "projects" : [ {
         "projectIndexId" : "chem01",
         "projectRole" : ["Developer"]
          },
          {
         "projectIndexId" : "chem02",
         "projectRole" : ["Developer"]
          }]
         },

         {
        "employeeName" : "Aparanna Raghuraman",
        "employeeId" : "SSK005",
        "password" : "aparannar",
        "organizationRole" : "Developer",
        "projects" : [ {
         "projectIndexId" : "chem01",
         "projectRole" : ["Developer"]
          } ]
         },

        {
        "employeeName" : "Dinesh PD",
        "employeeId" : "SSK003",
        "password" : "dineshpd",
        "organizationRole" : "Developer",
        "projects" : [ {
         "projectIndexId" : "chem01",
         "projectRole" : ["Grouplead"]
          },
                {
              "projectIndexId" : "chem02",
              "projectRole" : ["Developer"]
                }]
              },

            {
             "employeeName" : "Sriram J",
             "employeeId" : "SSK007",
             "password" : "sriramj",
             "organizationRole" : "Developer",
             "projects" : [ {
              "projectIndexId" : "chem02",
              "projectRole" : ["Grouplead"]
                } ]
              }
]}}
```

File Name: ProjectManager.java, ProjectListService.java

File Description: This screen is used to obtain the project name from the authenticated user who has now signed into the model manager tool. A frame is displayed to capture the project name from the list of project names for that particular user in the dropdown box. If the user wishes to create a new project, then new project collection will be saved in the mongoDB by invoking "SaveProjectService" and also the collection is created in the local repository.

Functions present: retrieveProjectsAssociatedToTheUser(userNameString), projectSelected(projectName)

Mapped to element: ProjectListService, SaveProjectService

Description:

Function name	retrieveProjectsAssociatedToTheUser
Function trigger Point	User gets authenticated by providing his/her username and password and the "login" is successful– GET Method is used to fetch the project list for the user.
Function parameters	userName
Function return type	void
Functions description	The function takes the entered values of username from the login screen, it invokes a service retrieveProjectsAssociatedToTheUser and displays the list of projects under that particular user in the dropdown box. (ProjectManager.java frame is linked with Login frame for this purpose, when the user is authenticated login frame closes and project manager frame would become the forefront)

Function pseudo code	retrieveProjectsAssociatedToTheUser (userNameString)
	{
	String userName = userNameString;
	
	//call the project list service by passing username.
	
	CloseableHttpClient httpClient = HttpClients.createDefault();

	HttpPost postRequest = new HttpPost("http://sase-server3.ssn.net:8080/TraceabilityServicesProvider-0.0.1-SNAPSHOT/Model/ **ProjectListService** ");
	
	postRequest.setEntity(username);
	response = httpClient.execute(postRequest);
	
	// display the project list in the dropdown box
	}

Sample Project Collection:

```
{

        "projects": [{
        "_id" : "chem01",
        "projectName" : "Simulated Distillaries",
        "description" : "This project is for simulating the working of distillaries",
        "instanceCollectionIndexId" : "chem01Instances",
        "modelCollectionIndexId" :"chem01Model",
        "roleCollectionIndexId" : "chem01Roles"

       },
       {
        "_id" : "chem02",
        "projectName" : "Simulated Boiler Plants",
        "description" : "This project is for simulating the working of boiler plants",
        "instanceCollectionIndexId" : "chem02Instances",
        "modelCollectionIndexId" :"chem02Model",
        "roleCollectionIndexId" : "chem02Roles"

       }
]}
```

Note: Project Collection is a child collection of User Collection where key is a foreign key to EmployeeID present in User Collection.

Description:

Function name	projectSelected
Function trigger Point	User either selects one project from the list provided or wishes to create a new project to work with.
Function parameters	projectName
Function return type	void
Functions description	The function takes the projectName value from the project manager frame, it invokes a service "**FetchModelService**" and the project model for the selected project is obtained from the datastore. Otherwise, if the user has provided a new project which is not in the dropdown box, "**SaveProjectService"** creates the new project collection in the database and the local repository.

Function pseudo code	projectSelected (projectName)
	{
	String projectName = projectNameField.getText();
	
	// if the project name given is chosen from the dropbox, then invoke FetchModelService to get the project model collection.
	
	CloseableHttpClient httpClient = HttpClients.createDefault();
	HttpPost postRequest = new HttpPost("http://sase-server3.ssn.net:8080/TraceabilityServicesProvider-0.0.1-SNAPSHOT/Model/ **FetchModelService** ");
	
	postRequest.setEntity(projectName);
	response = httpClient.execute(postRequest);
	
	// if it is a new project then perform validation and then save the project collection in database and also in the working dir temporary folder.
	
	response=validateProjectName(projectName);
	
	if(response.equals("Success")
	HttpPost postRequest = new HttpPost("http://sase-server3.ssn.net:8080/TraceabilityServicesProvider-0.0.1-SNAPSHOT/Model/ **SaveProjectService** ");
	else
	JOptionPane.showMessageDialog("Project name already exists")
	
	}

Design choice: "CreatedBy" field, in project model collection to capture who is the owner of the project model.

Scenario-2

Model manager X defines an entity for a project. He associates the person responsible to manage the entity (aka Group Lead). Once he has done modelling the entity he would commit the project model for group lead to create properties for the entity created. Similarly, as per the process followed by the project teams he defines a couple of more entities and assign leads for each of them.

Components Realized: Entity Manager, Properties Manager, Users Manager, RetrieveStandardEntities, MenuComponent, SaveModelService

File Name: EntityManager.java, ModelClass package

File Description: This frame is used to create entity as per the project model, there are no service calls since the entity collection Is saved in the local repository until commit.

Functions present: saveEntity(entityName,associatedStdEntity), retrieveStandardEntity, deassociateStandardEntity(entityName)

Mapped to element: RetrieveStandardEntitiesService

Function name	saveEntity

Function trigger Point	User creates the entity for the project.
Function parameters	entityName associatedStdEntity
Function return type	void
Functions description	The function takes the projectName value from the project manager frame, it invokes a service "**FetchModelService**" and the project model for the selected project is obtained from the datastore. Otherwise, if the user has provided a new project which is not in the dropdown box, "**SaveProjectService"** creates the new project collection in the database and the local repository.
Function pseudo code	see below

```
saveEntity (entityName, associatedStdEntity) {
                ObjectMapper mapper
                EntityModel node
                PropertiesModel p1
                /* create the entity node object, by setting the entityName and
                  associated properties from the user input through form */
                workingDir = System.getProperty("user.dir");

                /* check if the entity already exists in the local repository (working directory) */

                File f = new File (workingDir + "\\temp\\" + entityName + ".json");
                f.getParentFile().mkdirs();
                if (f.exists()) {
                        /* throw a pop up "entity already created" */
                } else {
                        /* create entity json in local repository */
                        mapper.defaultPrettyPrintingWriter().writeValue(new File(workingDir
+ "\\temp\\" + entityName + ".json"), node);
                       /* entity created successfully */
               }
```

Key Design Decisions:

- EntitiesList, Timestamp, change log
- The associatedStdEntity will be NULL if the manager does not associate to standard entity.
- The standard entities collection should have a key value pair saying it is standard.
- All entities should have a isDelete key value pair for safe deletion.

Standard Entity Collection: This is the standard CMMI Level 5 properties collection which is a static collection stored in mongo and is retrieved using "**RetrieveStandardEntitiesService**" whenever model manager is creating an entity for the project model. This was brought in to have a uniform definition for all the entities and to maintain consistent in the entity definitions. Users can also choose to custom add properties for every entity.

Feature	Standard Properties	Design Element	Length	Values	Mandatory
	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
Business Actor	Priority	dropdown	3	Low, Medium, High	No
	ID	textbox	10	Integer	Yes
Business Use Case	Name	textbox	20	String	Yes

	Description	textarea	50	String	Yes
	Priority	dropdown	3	Low, Medium, High	No
	Effort	textbox	5	Integer	No
	Release	textbox	15	String	No
	Document	textbox	20	String	No
Feature	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Priority	dropdown	3	Low, Medium, High	No
	Difficulty	dropdown	3	Low, Medium, High	No
	Release	textbox	15	String	No
	Status	dropdown	2	Approved, Pending	No
Project	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
Stakeholder Needs	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Requested By	textbox	20	String	No
	Release	textbox	15	String	No
	Priority	dropdown	3	Low, Medium, High	No
Supplementary Spec	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Arch Significance	dropdown	3	Low, Moderate, Critical	No
	Difficulty	dropdown	3	Easy, Medium, Complex	No
	Priority	dropdown	3	Low, Medium, High	No
System Actor	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Priority	dropdown	3	Low, Medium, High	No
System Use Case	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Document	textbox	20	String	No
	Iteration	textbox	3	Integer	No
	Effort	textbox	5	Integer	No
	Priority	dropdown	3	Low, Medium, High	No
	Difficulty	dropdown	3	Easy, Medium, Complex	No
Analysis	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Model File	textbox	20	String	No
	Iteration	textbox	3	Integer	No
	Designer	textbox	20	String	No

	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Model File	textbox	20	String	No
	Iteration	textbox	3	Integer	No
Design	Designer	textbox	20	String	No
	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Iteration	textbox	3	Integer	No
	Data File	textbox	20	String	No
Test Case	Script File	textbox	20	String	No
	ID	textbox	10	Integer	Yes
	Name	textbox	20	String	Yes
	Description	textarea	50	String	Yes
	Document	textbox	20	String	No
	Iteration	textbox	3	Integer	No
	Effort	textbox	5	Integer	No
	Status	dropdown	2	Approved, Pending	No
Test Suite	Priority	dropdown	3	Low, Medium, High	No

Scenario-3 (50 simultaneous users) Model manager creates property A and B. Group lead creates property C & D

1. **Concurrent model update:**
    - To prevent inconsistencies due to concurrent manipulations to project models, Audit Manager component emerged.
    - Notifies users about changes to a model.
    - Notification occurs every 5 minutes, and project manager can choose to sync up.
    - Otherwise, when commit happens the entity document is locked in mongoDB and both documents are compared and merged before saving the model.

2. **Comparison:**
    - Check for the isDelete key, if yes, entity/property/link already deleted.
    - Or check for the timestamp
        - If timestamp is same, invoke saveModelService() to save the model
        - Else invoke fetchModelService() and add the change logs from changelog file stored in the local repository. After merging, when save is successful, copy the changelog to changeLogCollection permanently for non-repudiation purpose.

3. **Non-repudiation:**
    - All entities should have a isDelete key value pair for safe deletion.
    - A change log and timestamp for every manipulation is recorded for notifying the changes to all the other users.

Scenario-4

Group Lead X adds a new property for an entity, on the other hand model manager is working with the old copy of the entity. The new changes are not present in model manager's desktop. Under this circumstance, model manager would get notified about the latest update that keeps happening to the project model every 5 minutes and has an option to sync up with the latest model.

Components Realized: Audit Component

File Name: NotificationPane.java

File Description: This frame is used to print the change log details; the pane refreshes every 5 minutes to keep the manager and group lead informed about all the changes being done to the project model. They can choose to select the "Sync" button action method to fetch the most recent model and merge change log in the local repository.

Functions present: refreshNotificationPane()

Mapped to element: FetchChangeLogService

Description:

Function name	refreshNotificationPane
Function trigger Point	Changes being done to the project model by all model manager users.
Function parameters	void
Function return type	void
Functions description	The function is used for refreshing the notification pane present in the right-hand side corner of the MenuComponent.

Function pseudo code	refreshNotificationPane() {
	/* Set Timer for 5 minutes and then invoke the notify() function*/
	Timer timer = new Timer(0, new ActionListener() {
	  @Override
	  public void actionPerformed(ActionEvent e) {
	    notify();
	  }
	});
	// delay for 5 minutes and start the timer
	}
	/* Perform this until the session is valid */
	}
	
	notify(){
	/* read the change log collection from mongodb and display it in the notification pane. */
	
	// invoke "**FetchChangeLogService**" method and obtain the change log details from mongo
	}

File Name: LinkManager.java
File Description: This frame is present for users to define linkages between the entities. For example: Requirement ←→ Design.

Function present: saveLinkBetweenEntity(fromEntity, toEntity)
Mapped to element: none

Function name	saveLinkBetweenEntity
Function trigger Point	When entities are defined by the model manager, and he wants to provide linkages between them (based on the process followed by project teams) or wants to edit the linkage.
Function parameters	fromEntity toEntity
Function return type	void
Functions description	This is used for defining the linkages between the entities and save them in the local repository

Function pseudo code	saveLinkBetweenEntity(fromEntity, toEntity){
	
	// Update the "**LinkedTo**" field in the entity collection
	
	// Remove the entity name from the linked to array present in the entity collection- incase the user wants to delete or edit the linkages.
	}

Key Decision Choices:

S.No	**Design Choices**
1	Local and global change log, this is required when two or more group leads work on their individual entities and want to commit to the database collection without modifying anny data previously saved by model manager and other leads. Therefore, with change log it will be possible to capture who has done the manipulations
2	Lock the data collection while saving project related details. This is to avoid any overwriting of data when the same document is being accessed by multiple users.
3	JSON structure would be created for each entity so that performance can be improved and at the same time simultaneous data access can be achieved by having a check on incorrect data manipulations. Also, 16MB is the maximum length a JSON document can have so we have smaller chunks for example collection created for each entity.
4	CreatedBy and Timestamp entries have to be added to every json collection that is being created.