SANTA CLARA UNIVERSITY

FrugalInnovationLab

COEN 359 - Project Management Tool

Under the guidance of:

Dr. Rani Mikkilineni

Prepared by:

Daryl Rodrigues(SCU ID: W1113491)

Sachin Shinde (SCU ID: W1114443)

Date: September 2, 2015

Table of Contents

1. Description

The Frugal Innovation Lab at Santa Clara University provides opportunities for students and faculty to partner with industry clients and nonprofit organizations to develop new low-cost products and technologies for emerging markets.
We plan to design and develop a tool using Design Patterns that will help frugal lab manage all the projects that are in progress at the lab.

The project will consist of four main phases: analysis, design, implementation and testing:

Analysis: We plan to develop the use cases and identify analysis classes in this phase.
Design: For the design, we will develop different UML diagrams(class and sequence) to describe the design.
Implementation: The project will be implemented in Java using the object-oriented design and programming techniques and Design Patterns.

Stakeholders
The tool will be used primarily by the admin to create new profiles(students,faculty,client) or even to add or delete projects. Also other members who will be using the tool with limited access are faculty,students and client.

Problems addressed by our proposed solution
As mentioned in the previous section, our goal is to make the code more reusable and flexible. For example with the existing implementation, adding a new student or faculty requires a lot of changes and also results in violation of OCP.

Environment

- Operating System: Windows/linux/Mac
- Connection to Database(MySQL)

2. Skills and Technology requirements

This project requires knowledge in following areas -
Skills:

- Java Programming knowledge
- UML

- SQL for querying the database
- Database connectivity

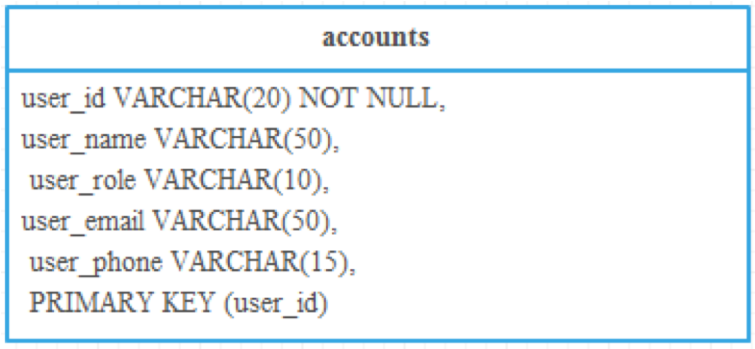
Technology:

- Netbeans as an IDE for implementing the java code and also to design the GUI.
- MYSQL for database.

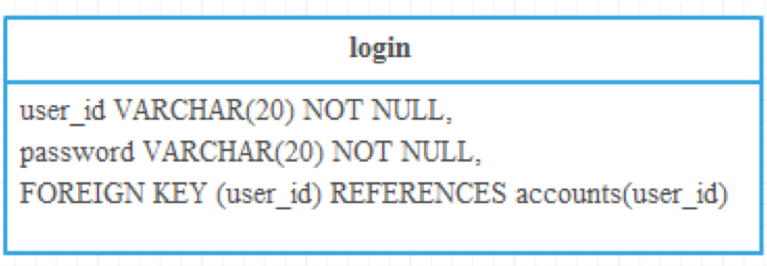
Application type

We plan to design and develop a **desktop** application.

3. Database Schema



user_id : This is an unique id assigned to every user using the system.Also its a primary key.
user_name: The name of user using the system.
user_role: There are 4 roles assigned to the user using the system. So the user can be a Student,Faculty,Admin and Partner/Client.
user_email : The email address of the user.
user_phone: The contact number of the user



The Login table consists the credentials required to log into the system.

user_id : This is an unique id assigned to every user. Also it's a foreign key referencing the primary key.
user_id in the Accounts table.
password: This is required to log into the system.

project
project_id VARCHAR(20) NOT NULL, project_name VARCHAR(50), project_description VARCHAR(50), client_id VARCHAR(20), start_date DATE, end_date DATE, faculty_id VARCHAR(20), PRIMARY KEY(project_id), FOREIGN KEY (client_id) REFERENCES accounts(user_id)

The Project table holds details regarding projects in Frugal Innovation Lab.

project_id: Unique ID assigned for each project.
project_name: The name of the project.
faculty_id: Unique ID of the faculty incharge for the particular project.
project_description: A small description on the project.
end_date: The date by which the project is expected to be completed.
starr_date: The date by which the project has started.
client_id : The unique ID associated with the client for a project. Also it's a foreign key referencing the primary key user_id in the Accounts table.

project_members
project_id VARCHAR(20), user_id VARCHAR(20), FOREIGN KEY (user_id) REFERENCES accounts(user_id), FOREIGN KEY (project_id) REFERENCES project(project_id)

project_id: Unique ID assigned for each project.
user_id : This is an unique id assigned to every user. Also its a foreign key referencing the primary key.

task
task_id VARCHAR(20) NOT NULL, task_description VARCHAR(50), project_id VARCHAR(20) NOT NULL, student_id VARCHAR(20), start_date DATE, end_date DATE, FOREIGN KEY (project_id) REFERENCES project(project_id), FOREIGN KEY (student_id) REFERENCES accounts(user_id)

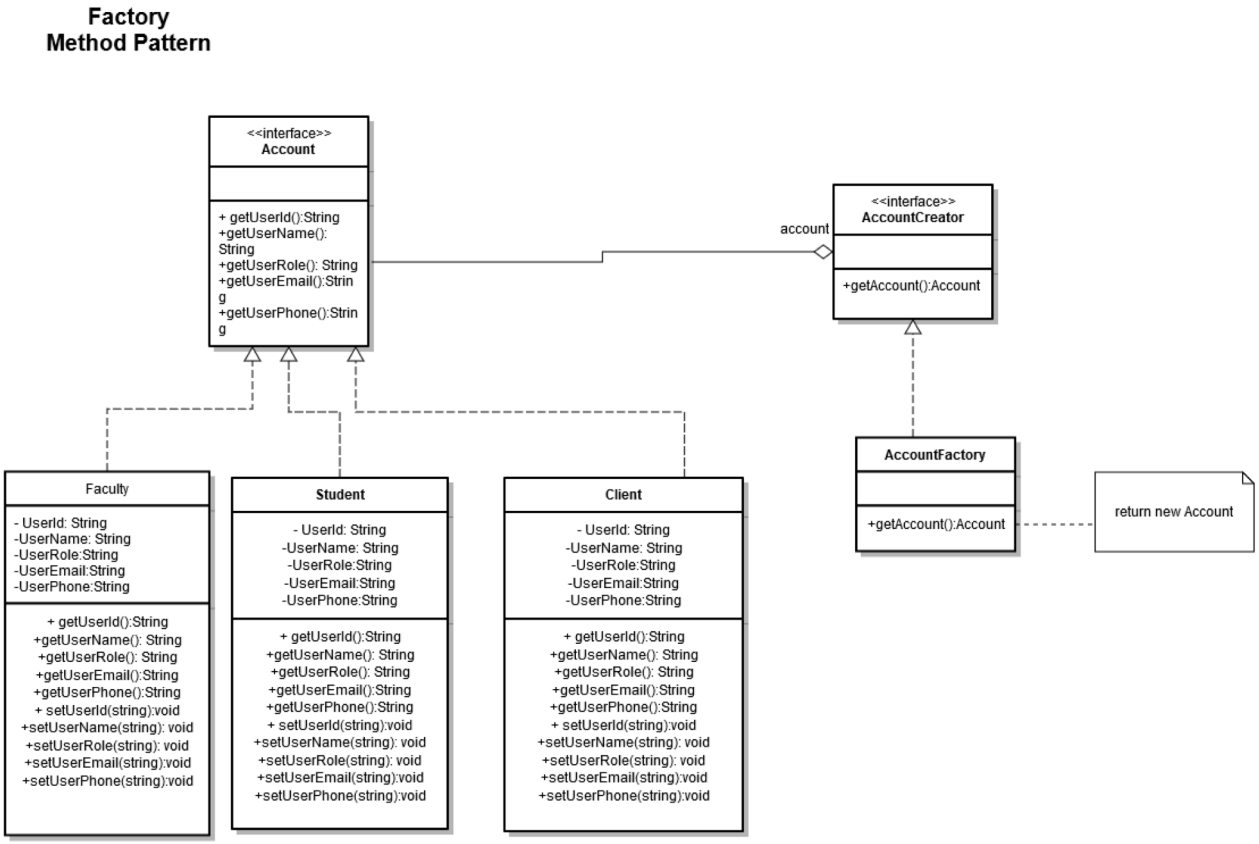
The Task table consists of all the details related to the individual tasks assigned to the project team members.
task_id : Unique ID assigned to ident
project_id:The ID of the project with which the task is associated.
student_id : This is an unique id assigned to every student.Also it's a foreign key referencing the primary key user_id in the Accounts table.
task_description: A small description for the task that is performed.
start_date: The date by which the task has to start. .
end_date: The date by which the task has to be completed.

4. Detailed Class Diagrams with Code Segments

1. Factory Method Pattern

Applicability

Our intention was to define an interface for creating an account object by giving the subclass the power to decide which class to instantiate. The idea behind doing this was that it would give us flexibility for future use. Also using factory method pattern, we have taken care of OCP principle.
The creation of an object is encapsulated in the factory method getAccount(). The newly created object is referred through a common interface.



```
private void createAccountButtonActionPerformed(java.awt.event.ActionEvent evt)
{
AccountFactory accountFactory = new AccountFactory(userId.getText().trim(), userName.getText().trim(),  userRoleCombo.getSelectedItem().toString(),userEmail.getText().trim(), userPhone.getText().trim());
Account account = accountFactory.getAccount(userRoleCombo.getSelectedItem().toString());
AdminActions adminActions = new AdminActions();
adminActions.createAccount(account, password.getText().trim());
}
```

2. Builder Pattern – Project

Applicability

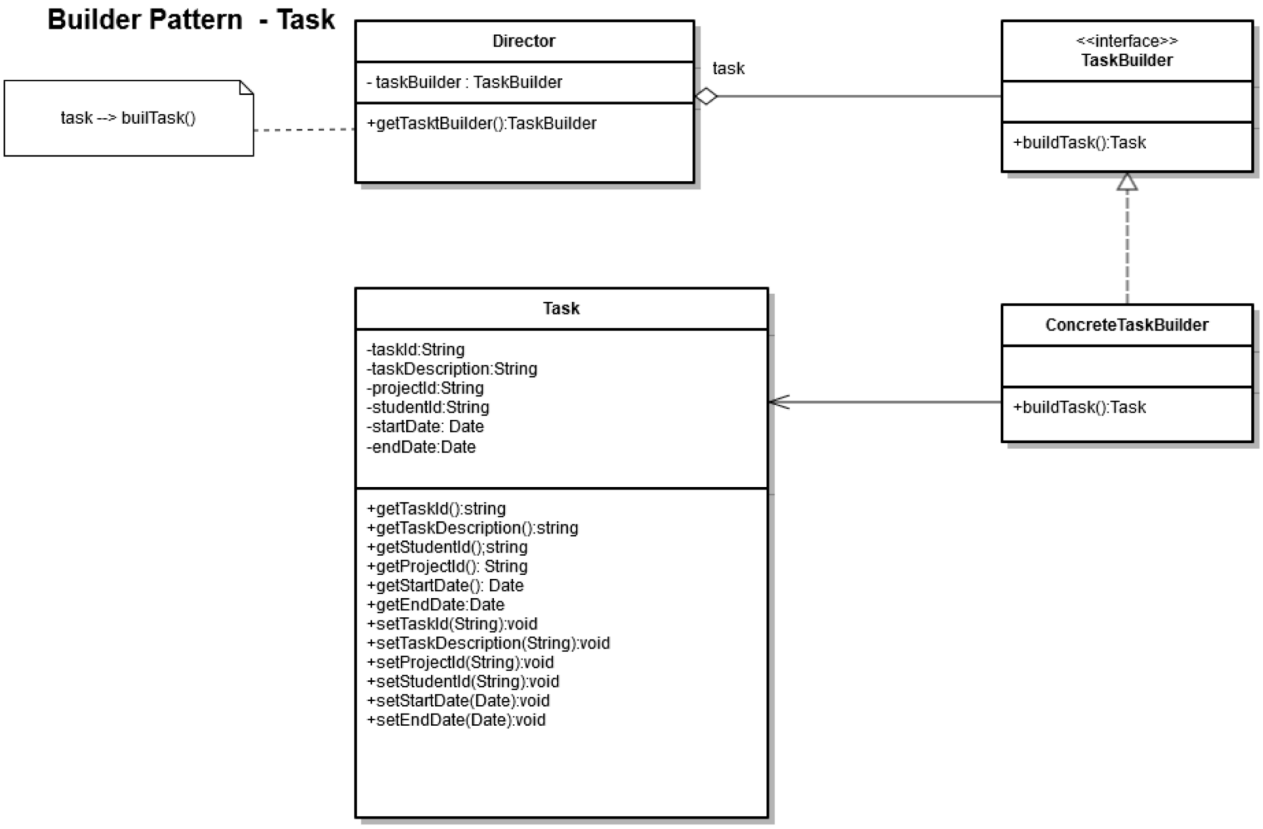
This pattern separates the process of construction of complex object,Project from its representation so that the same construction process can be used to create different representations. Helps in incremental building of complex- structured object. We have used this pattern for construction of complex objects which is project in our system.

```
private void registerProjectActionPerformed(java.awt.event.ActionEvent evt)
{       Director director = new Director();
        ProjectBuilder  projectBuilder = director.getProjectBuilder();
        Project project = projectBuilder.buildProject();
        project.setProjectId(projectId.getText().trim());
        project.setProjectName(projectName.getText().trim());
        project.setProjectDescription(projectDescription.getText().trim());
        project.setClientId(clientId.getText().trim());
        project.setStartDate(startDate.getDate());
        project.setEndDate(endDate.getDate());

        facultyActions.registerProject(project);
}
```

3. **Builder Pattern – Task**

Applicability

This pattern separates the process of construction of complex object,Task from its representation so that the same construction process can be used to create different representations. Helps in incremental building of complex- structured object. We have used this pattern for construction of complex objects which is task in our system.



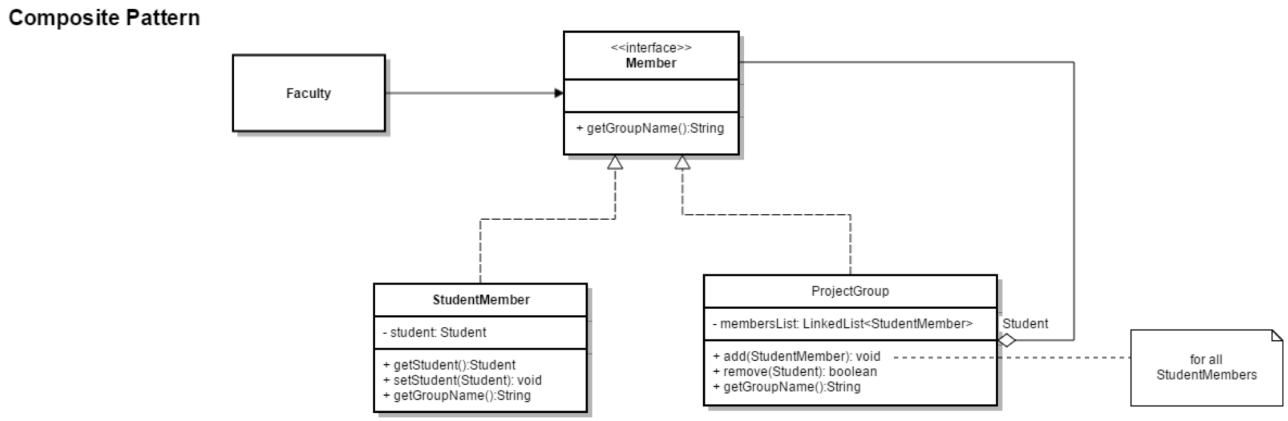

```
private void createTaskButtonActionPerformed(java.awt.event.ActionEvent evt) {
        TaskDirector director = new TaskDirector();
        TaskBuilder taskBuilder = director.getTaskBuilder();
        Task task = taskBuilder.buildTask();
        task.setProjectId(projectId.getText());
        task.setTaskDescription(taskDescription.getText());
        task.setTaskId(taskId.getText());
        task.setStudentId(studentId.getSelectedItem().toString());
```

```
        task.setStartDate(startDate.getDate());
        task.setEndDate(endDate.getDate());
         facultyActions.registerTask(task);
}
```

4. Composite Pattern

Applicability

We have used this pattern to represent part-whole hierarchy for the objects. ProjectGroup is a composite. We have implemented this pattern using **Type-Safety**.




```
public class ProjectGroup implements Member{
  public LinkedList<StudentMember> membersList = new LinkedList<StudentMember>();

  // To add a new student as a member to the project group
  public void add(StudentMember studentMember){
     membersList.add(studentMember);
  }
```

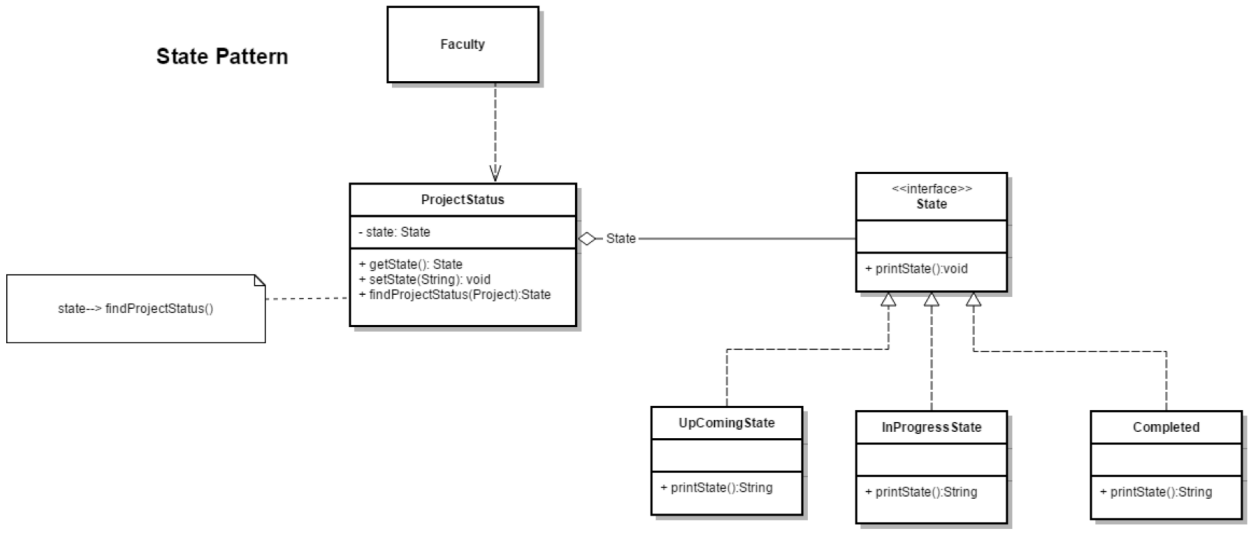
```
  // To remove a student from the project group
  public boolean remove(Student Student){
     StudentMember studentMember = null;
     for(int i=0;i<membersList.size();i++){
        studentMember = membersList.get(i);
        if(Student.getUserId().equalsIgnoreCase(studentMember.getStudent().getUserId())){
           membersList.remove(i);
           return true;
        }
     }
     return false;
  }}

public class FacultyActions {
private ProjectGroup projectGroup = null;
…
          // To add project member to the group
            public void addMembers(StudentMember studentMember){
               projectGroup.add(studentMember);
            }

            // To remove project member from the group
            public boolean removeMembers(Student student){
               return projectGroup.remove(student);
            }
}
facultyActions.addMembers(member);
```

5. State Pattern

Applicability

- The projects behavior depends on its state and it changes its behavior at run time depending on that state.
- *Localizes state-specific behavior and partitions behavior for different states.* The state pattern puts all behaviors associated with a particular state of project into one object. Because all state

specific code lives in a state subclass, new states and transitions can be added by defining new subclasses.




```
private void populateProjects(){
        ProjectStatus projectStatus = new ProjectStatus();
        State  state = projectStatus.findProjectStatus(project);
}
```

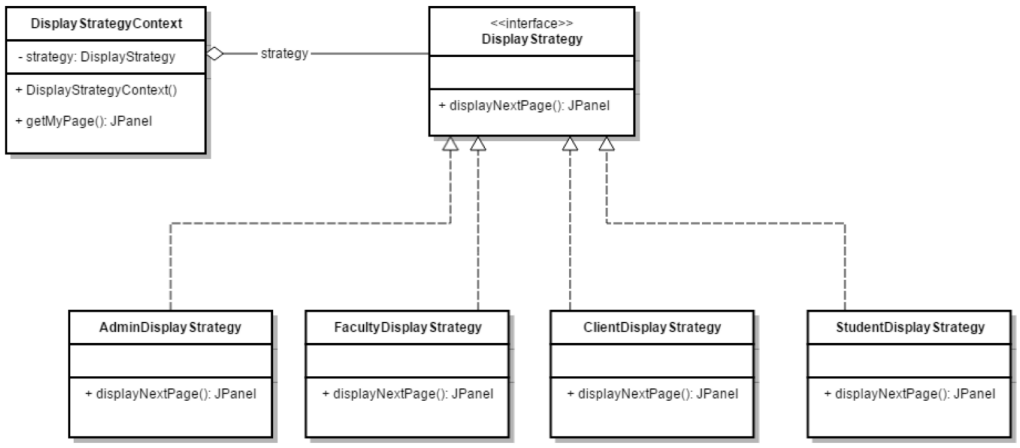
6. Strategy Pattern

Applicability

We used strategy pattern for limited view implementation. DisplayStrategy and DisplayContext interact to implement the chosen algorithm for displaying a UI. This pattern is applicable as there are choices of implementations available.

This pattern provides an efficient way to eliminate conditional statements by adding a separate class for each behavior.

Strategy Pattern - Limited view implementation



DisplayStrategyContext displayStrategyContext = null;
DisplayStrategyContex=new DisplayStrategyContext(ProjectManagementGlobalSession.user_id);
ProjectManagementGlobalSession.centralPanel.add(displayStrategyContext.getMyPage());
ProjectManagementGlobalSession.centralPanel.updateUI();

7. Security pattern

Applicability

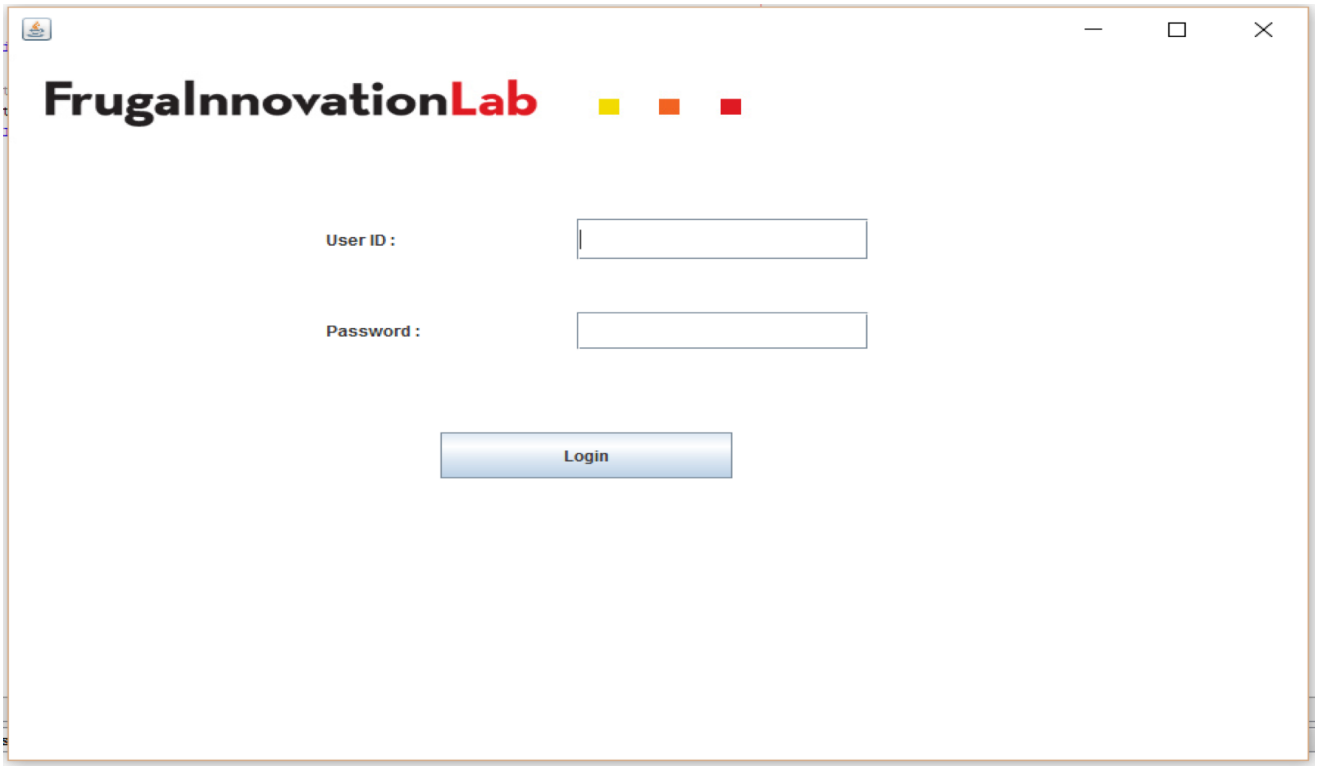
- **Single Access point** - Login screen is one and only allowed path into the system. The information entered by the user is verified and a session is created on successful login.

- **Authentication and Authorization** - Credential check is performed every time a user tries to log into the system ensuring authentication and authorization is implemented using limited view.
- **Roles** - Different roles are created for the system(admin,student,faculty and client) depending on various responsibilities assigned to each user.
- **Session** - We have created a session variable in order to localize information in a multi user environment.
- **Limited view** - Our system allows users to only see what they have access to.

```
// Session
ProjectManagementGlobalSession.user_id = this.userId;
ProjectManagementGlobalSession.loggedInUser = user;

// role
ProjectManagementGlobalSession.user_role = this.userRole;

// limited view
DisplayStrategyContext displayStrategyContext = null;
DisplayStrategyContexT = new DisplayStrategyContext(ProjectManagementGlobalSession.user_id);
ProjectManagementGlobalSession.centralPanel.add(displayStrategyContext.getMyPage());
ProjectManagementGlobalSession.centralPanel.updateUI();

// Authentication
Authentication authentication = new Authentication(userId.getText(),new
String(password.getPassword()), this);
authentication.authenticate();
```

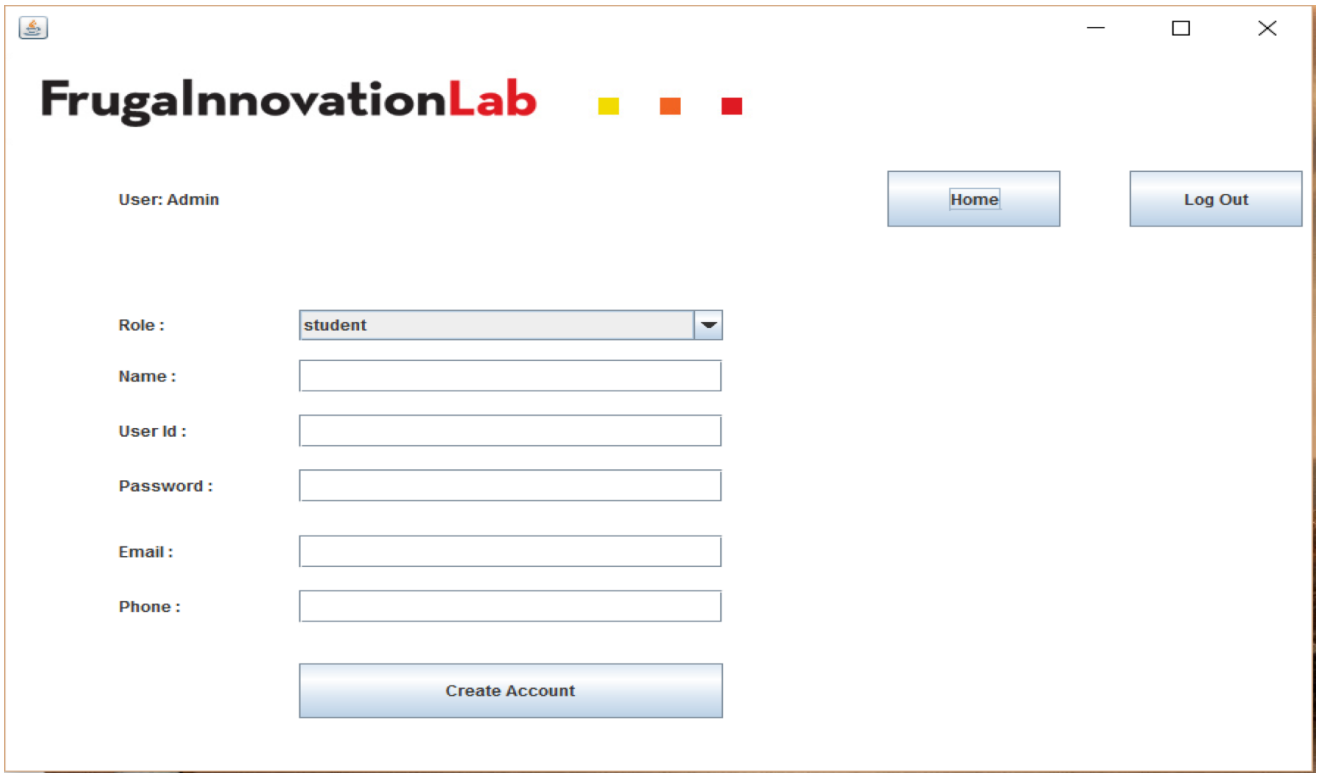
Project GUI screenshots

1.Login screen

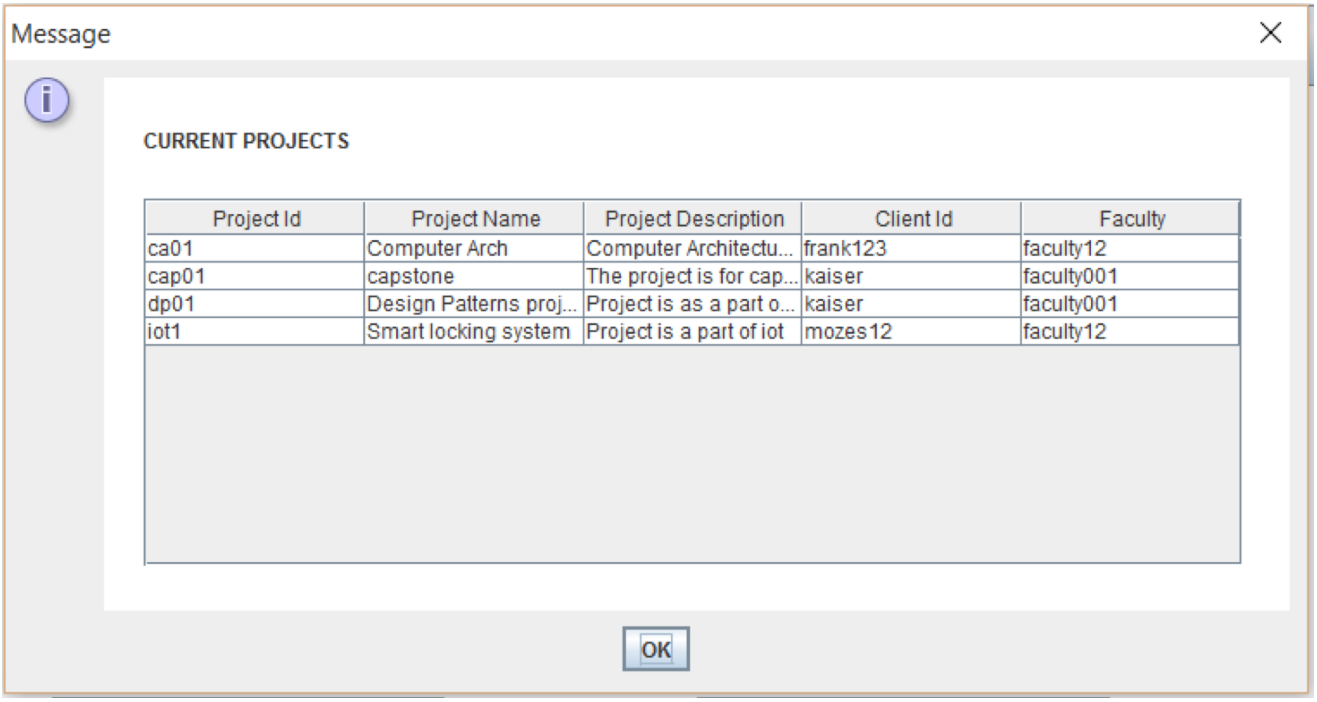


2.Admin Home page

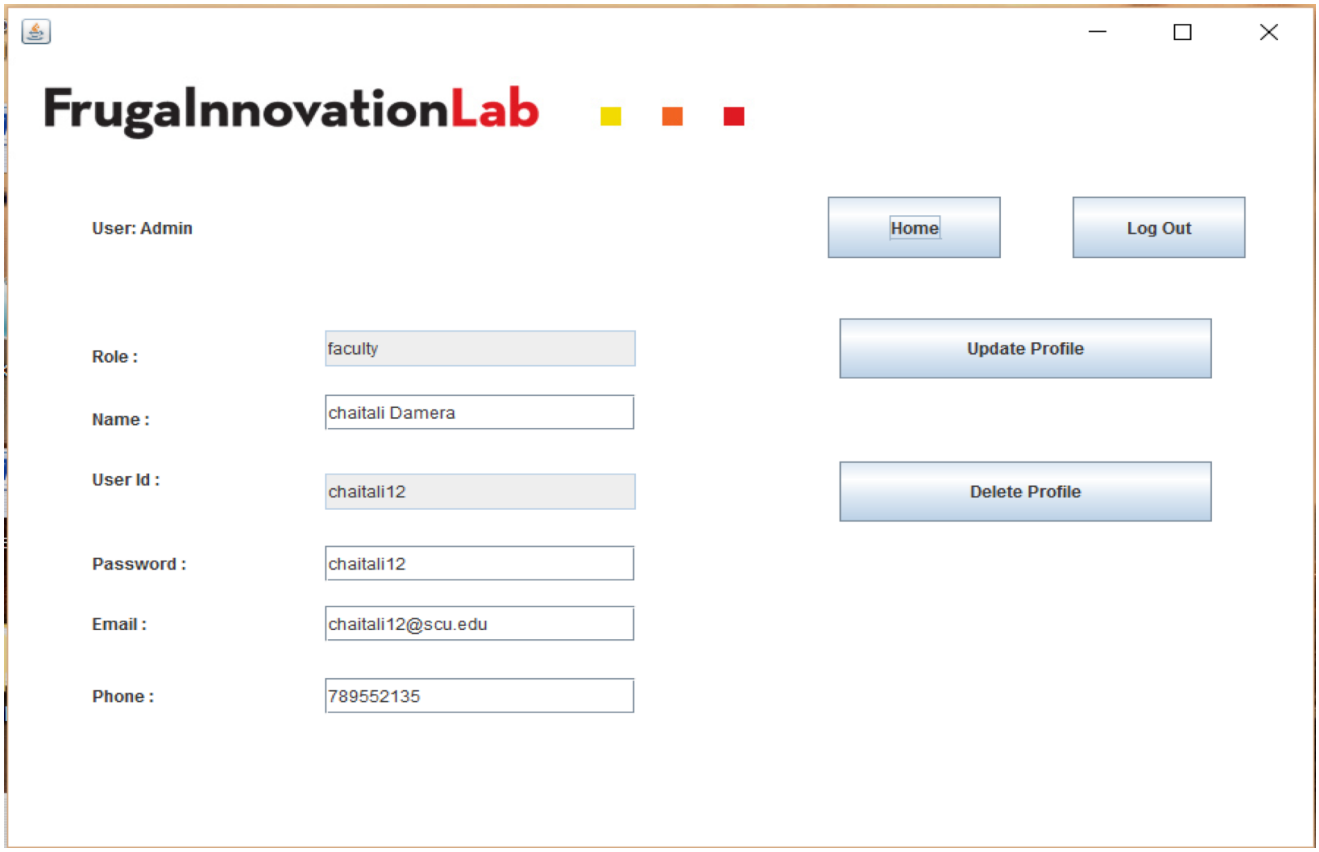
3. Create new account



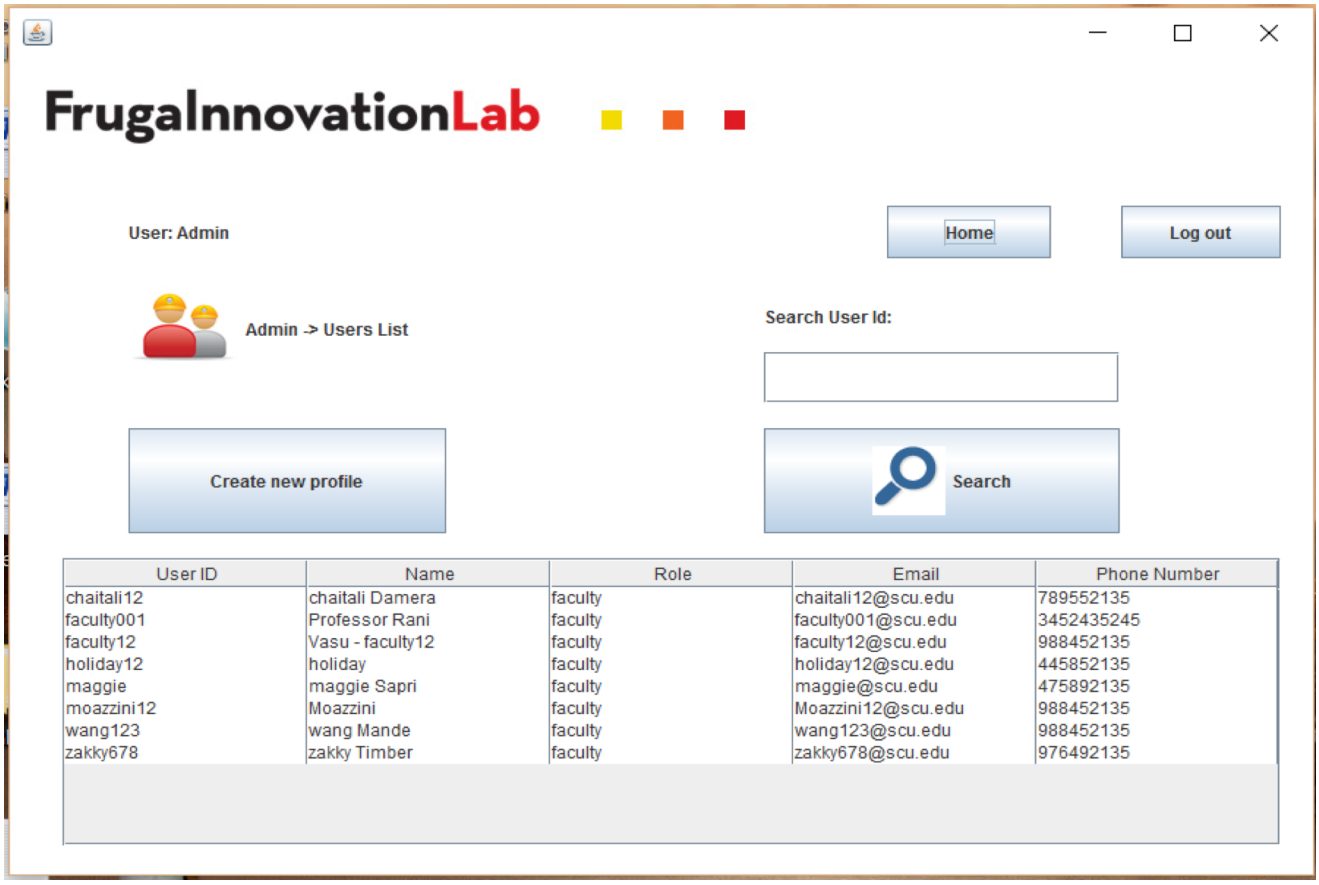
4. **Admin projects view**



5. Update account

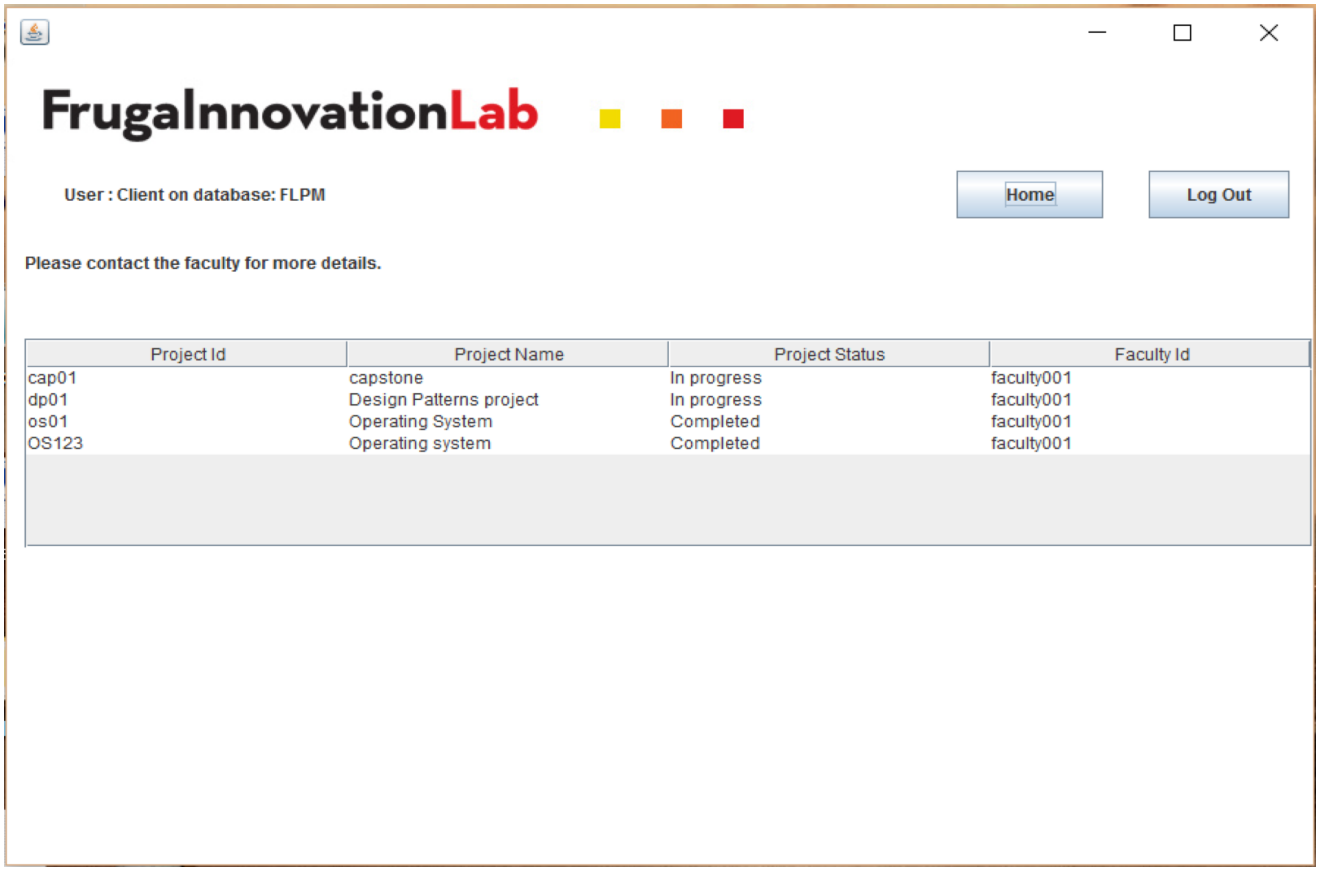


6. Existing accounts list (faculty)

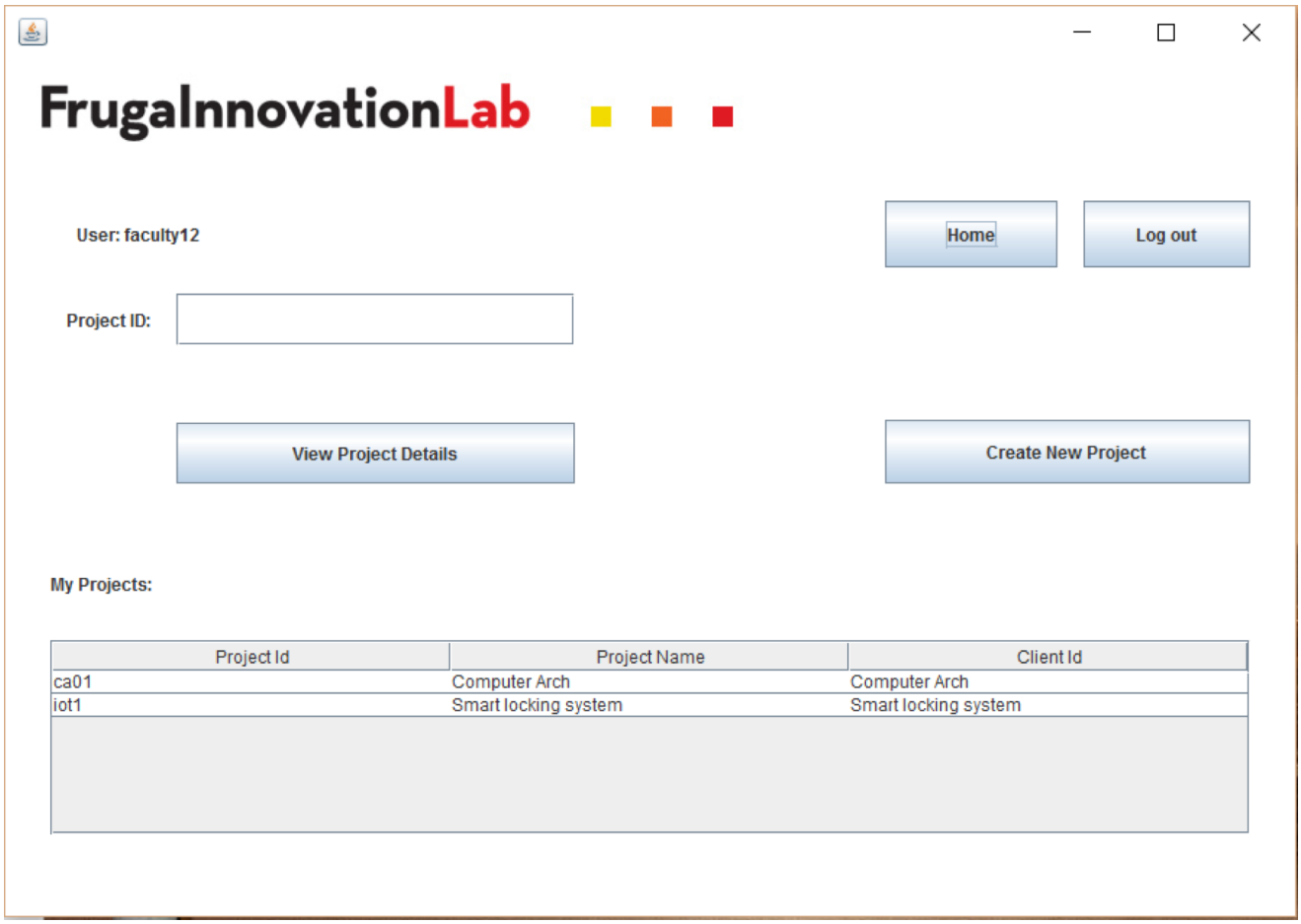


User ID	Name	Role	Email	Phone Number
chaitali12	chaitali Damera	faculty	chaitali12@scu.edu	789552135
faculty001	Professor Rani	faculty	faculty001@scu.edu	3452435245
faculty12	Vasu - faculty12	faculty	faculty12@scu.edu	988452135
holiday12	holiday	faculty	holiday12@scu.edu	445852135
maggie	maggie Sapri	faculty	maggie@scu.edu	475892135
moazzini12	Moazzini	faculty	Moazzini12@scu.edu	988452135
wang123	wang Mande	faculty	wang123@scu.edu	988452135
zakky678	zakky Timber	faculty	zakky678@scu.edu	976492135

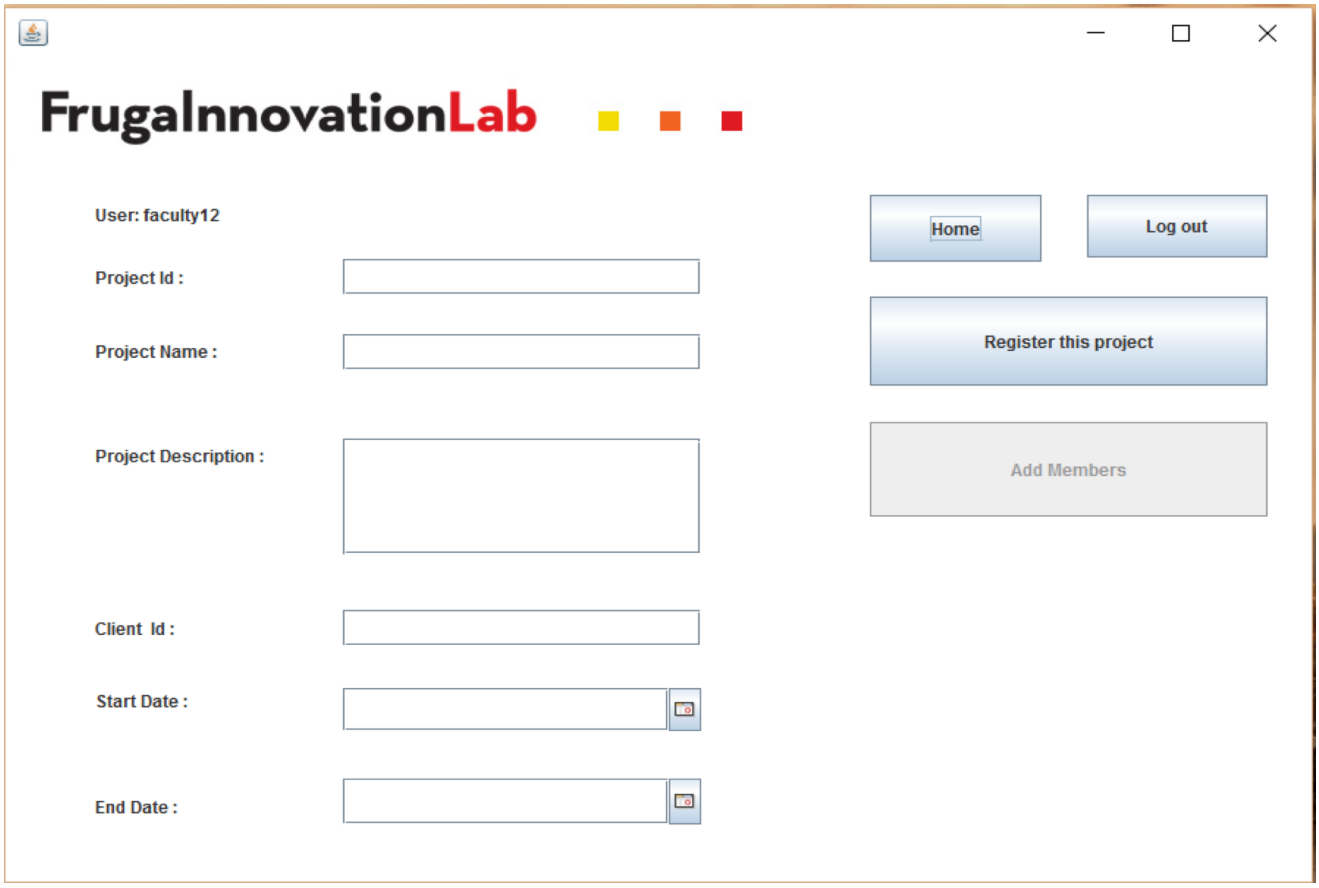
7. Client Home page



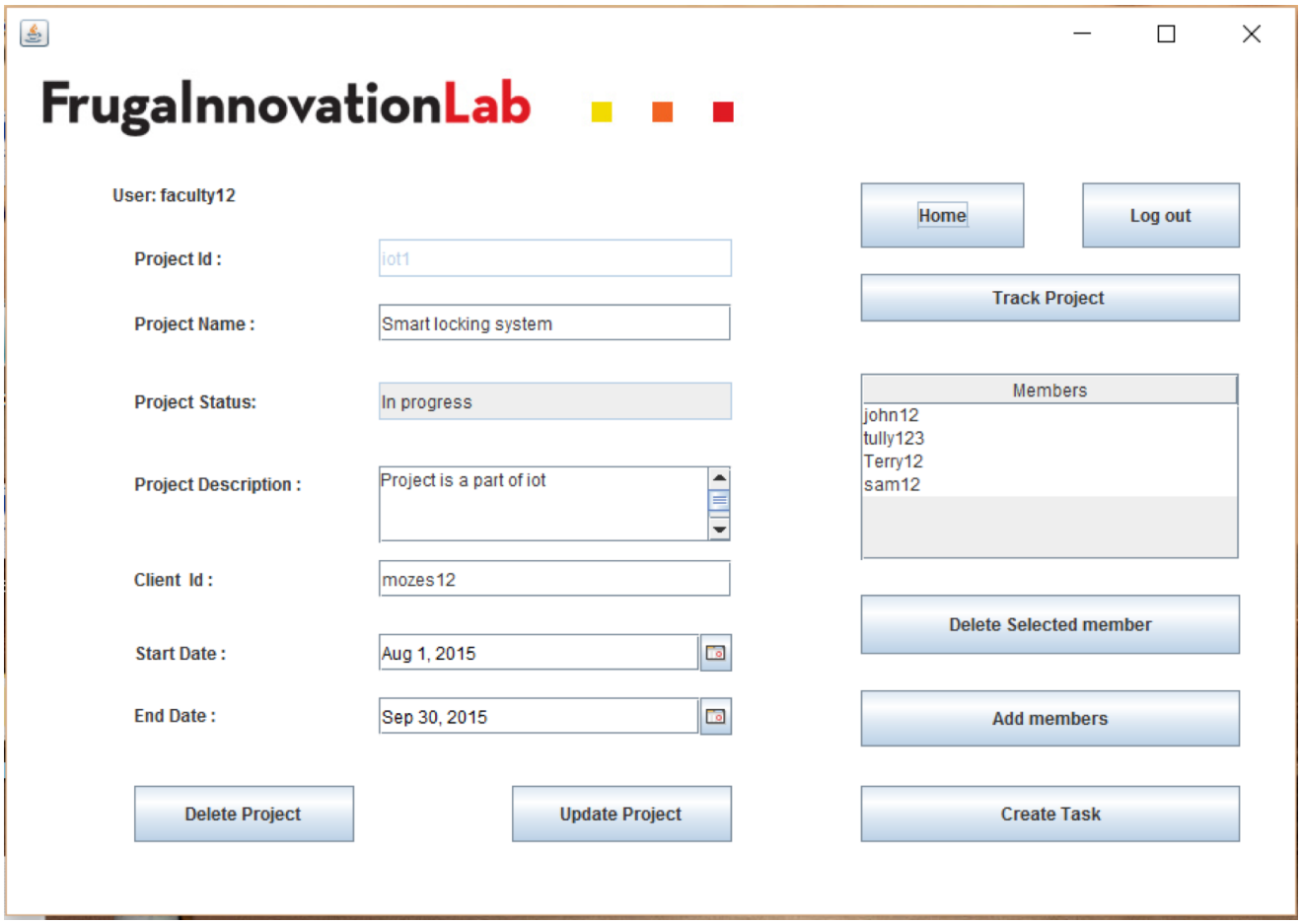
8. Faculty Home page



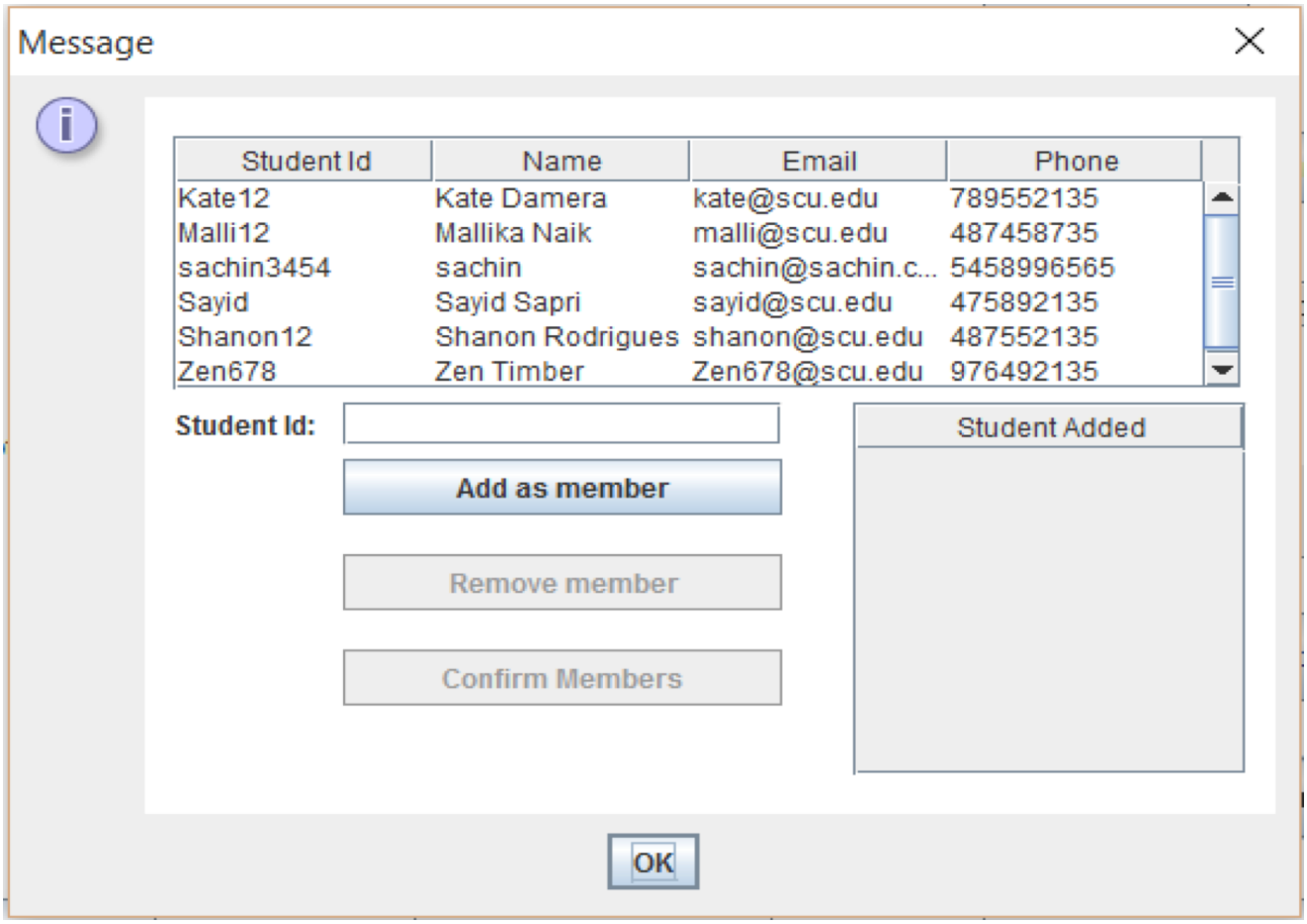
9. New project



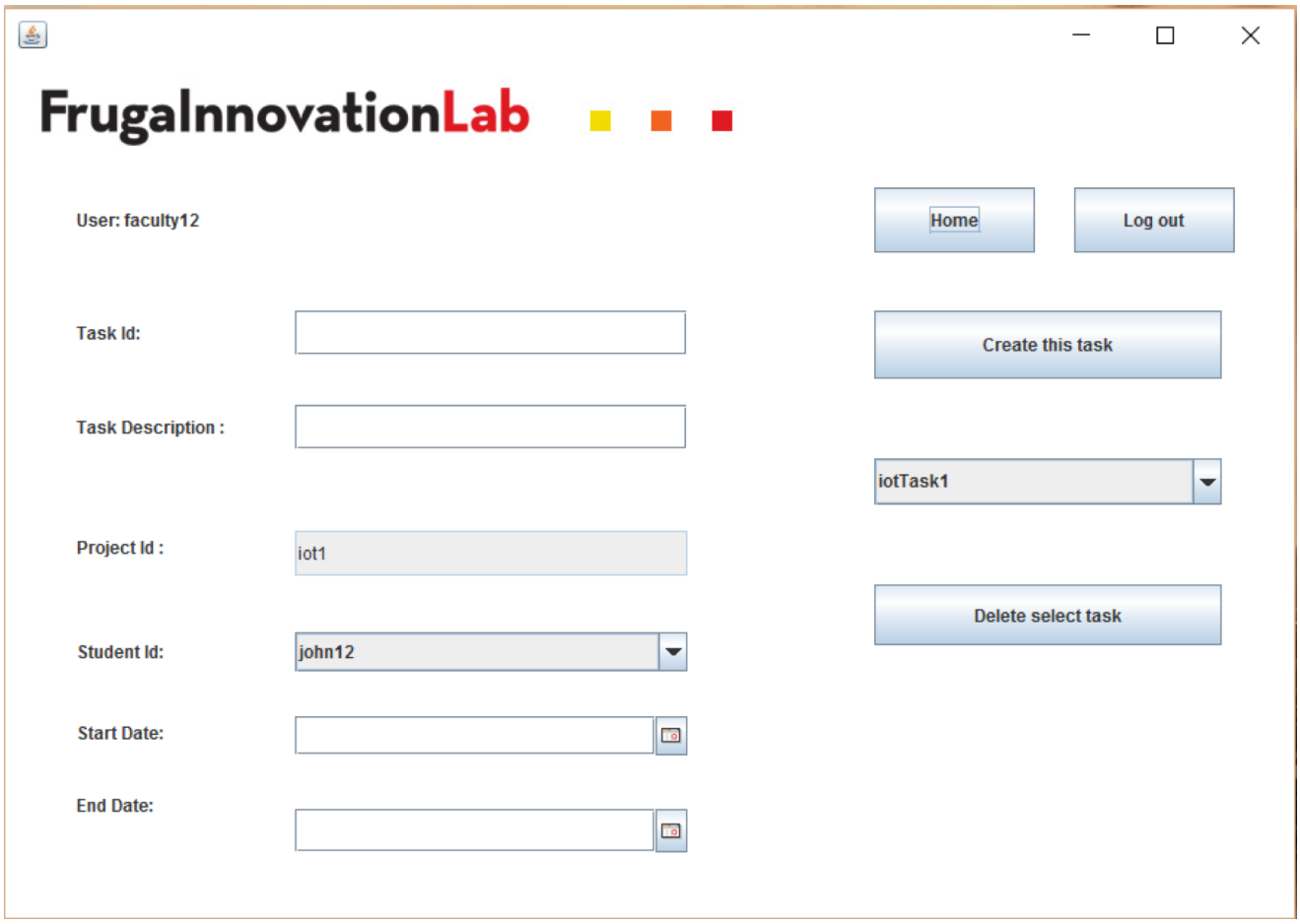
10. Update existing project



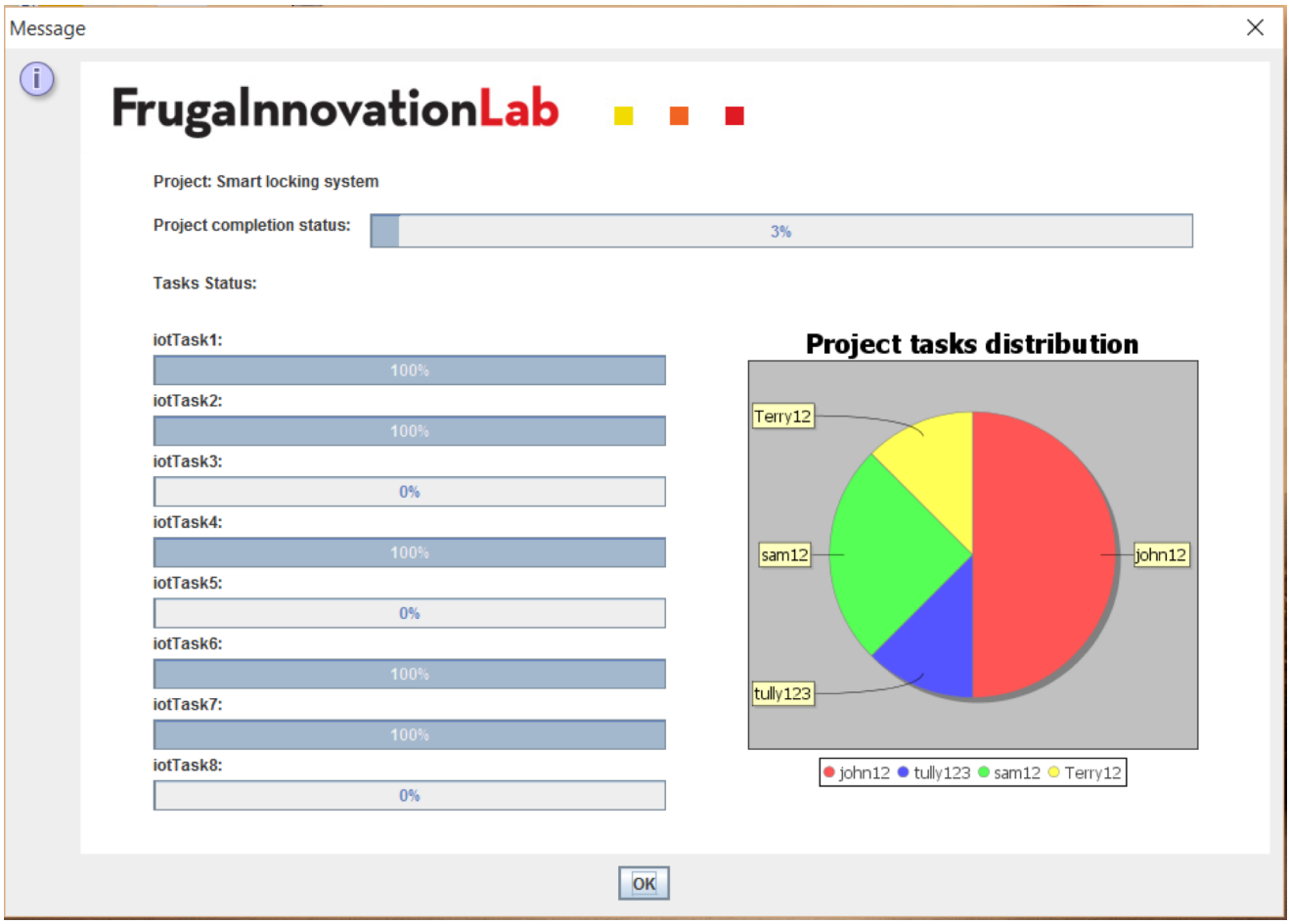
11. Add members to project



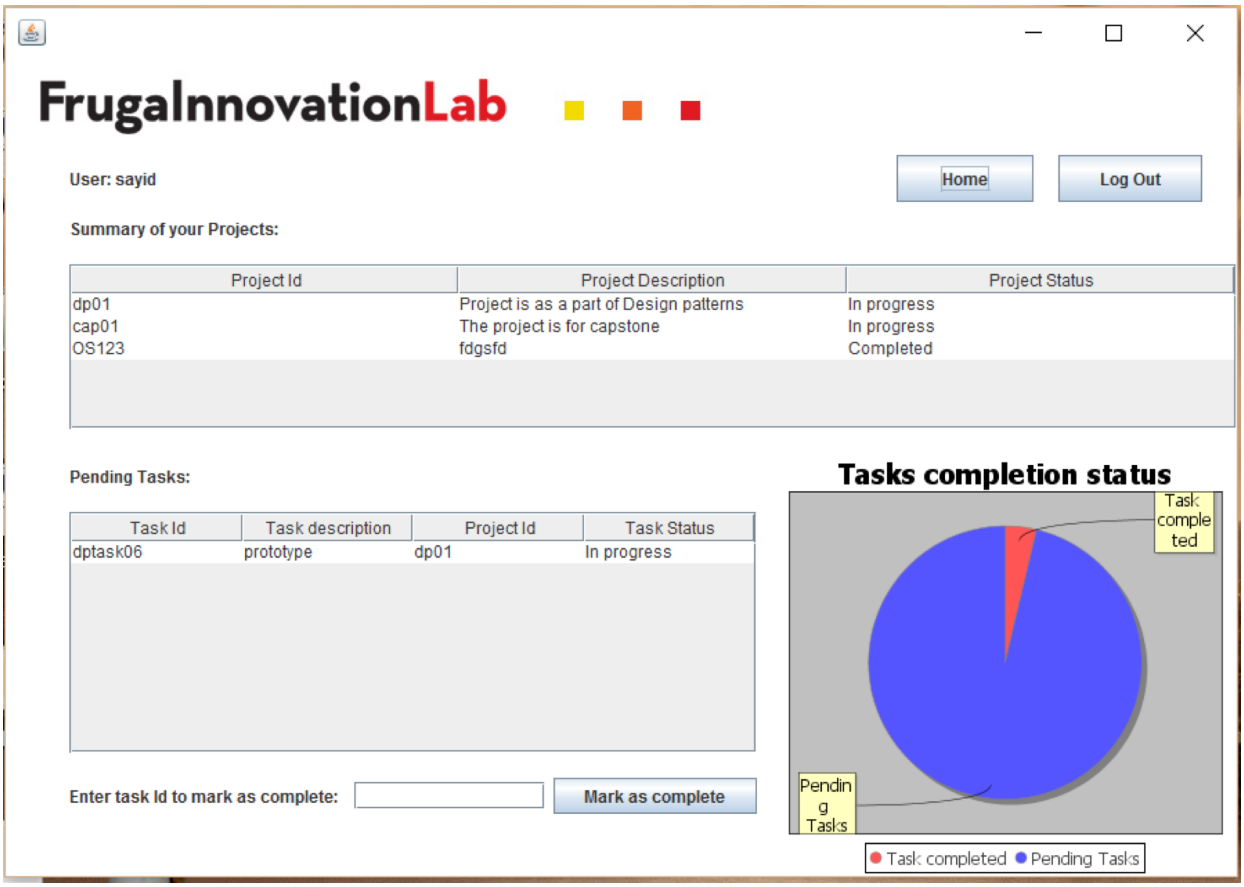
12. Create new task



13. Track project progress



14. Student home page



15. Sample message dialogue