DEGREE PLAN AUTOMATION

DELIVERABLE 3

Submitted by Team Shan

Sharanya Gottimukkula

Nanditha Bodanapu

Sri Harshini Vallabaneni

Aravind Thottempudi

CSCE 5430 - Software Engineering
Deliverable – 3

a. Requirements Destined for Development Phase –I

Functional Requirement	Development Phase	Start Date	End Date
FREQ-1	Phase 1	10/04/2018	10/05/2018
FREQ-2	Phase 1	10/06/2018	10/07/2018
FREQ-3	Phase 1	10/08/2018	10/11/2018
FREQ-5, FREQ-8	Phase 1	10/12/2018	10/16/2018
FREQ-6, FREQ-7, FREQ-9, FREQ-11	Phase 1	10/19/2018	10/31/2018

Implemented Requirements:
FREQ-1: Login (All Users), FREQ -2: Registration (All Users), FREQ-3: Manage users (Administrator), FREQ-5: Send Requests to professors and withdraw (Student), FREQ-8: Accept/decline requests (Professor), FREQ -6: Fill in the Degree Plan (Student), FREQ -7: Submit the Degree Plan to Professor for Signature (student)

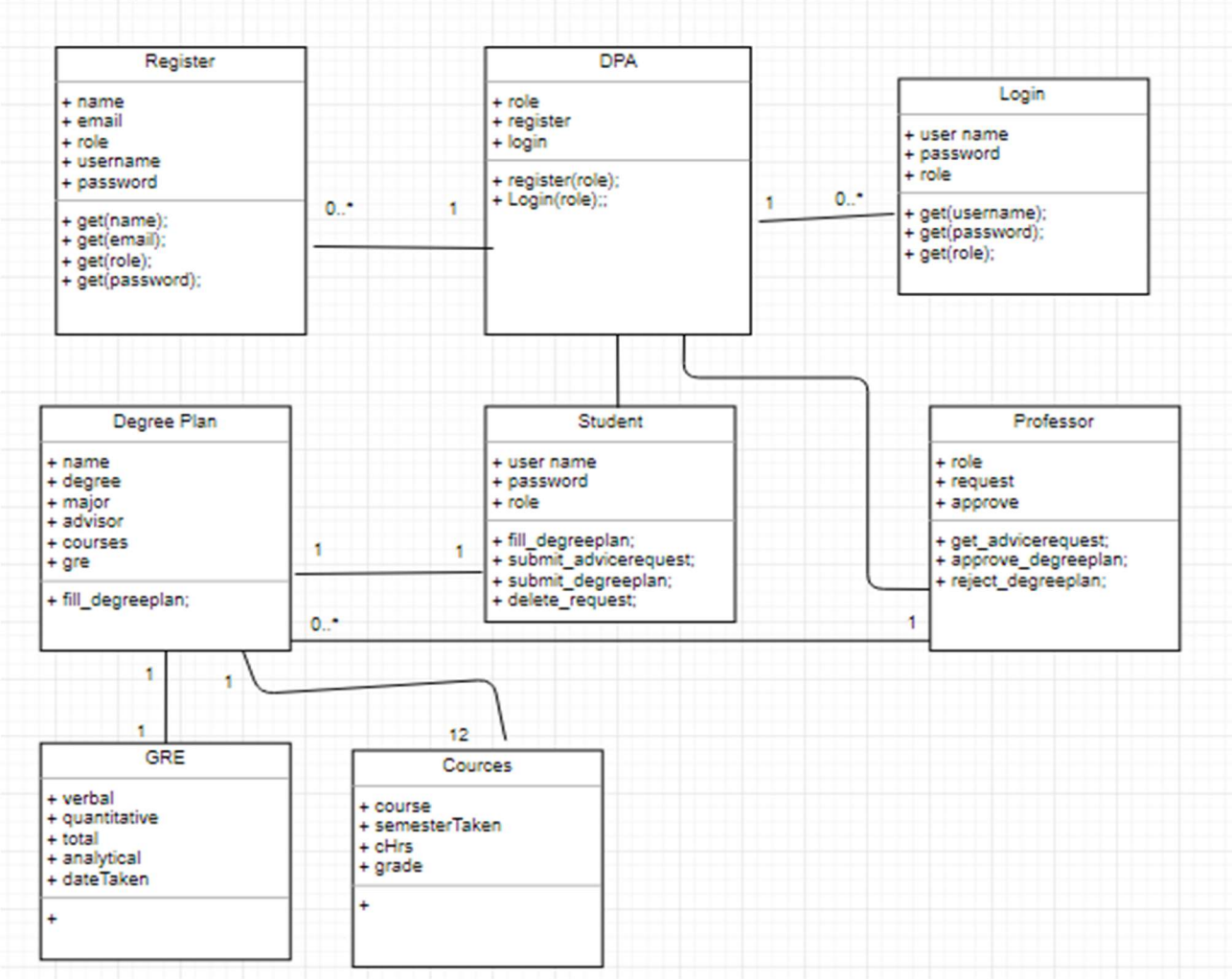
Unimplemented Requirements: FREQ-9, FREQ-11
As per the three-phase schedule specified in deliverable 2, we could not implement the requirements FREQ-9 and FREQ-11. The unexpected configuration issues while using Spring MVC and MYSQL workbench is the first reason for the delay and also all the team members could not get well with Spring MVC quickly, this increased the time taken by each resource to program certain functionality. These unexpected issues delayed the development of requirements destined for phase-I in the proposed schedule specified in deliverable 2. The requirements specified for phase-I in deliverable 2 where heavy when compared to phase-II and Phase-III, this improper planning is also a reason for not meeting the expected schedule all these forced the change in the plan and below is the updated plan.

Updated Plan for three Development Phases

Functional Requirement	Development Phase	Start Date	End Date
FREQ-1	Phase 1	10/08/2018	10/11/2018
FREQ-2	Phase 1	10/12/2018	10/15/2018
FREQ-3	Phase 1	10/16/2018	10/19/2018
FREQ-5, FREQ-8	Phase 1	10/20/2018	10/25/2018
FREQ-6, FREQ-7	Phase 1	10/26/2018	10/29/2018
FREQ-9, FREQ-11	Phase 2	10/30/2018	11//03/18
FREQ-13	Phase 2	11/04/2018	11/06/2018
FREQ-10	Phase 3	11/06/2018	11/08/2018
FREQ-12	Phase 3	11/09/2018	11/12/2018
FREQ-4	Phase 3	11/13/2018	11/14/2018
FREQ-14	Phase 3	11/15/2018	11/17/2018
FREQ-15	Phase 3	11/18/2018	11/20/2018

b. UML Diagrams

Class Diagram:



Use Case Text:

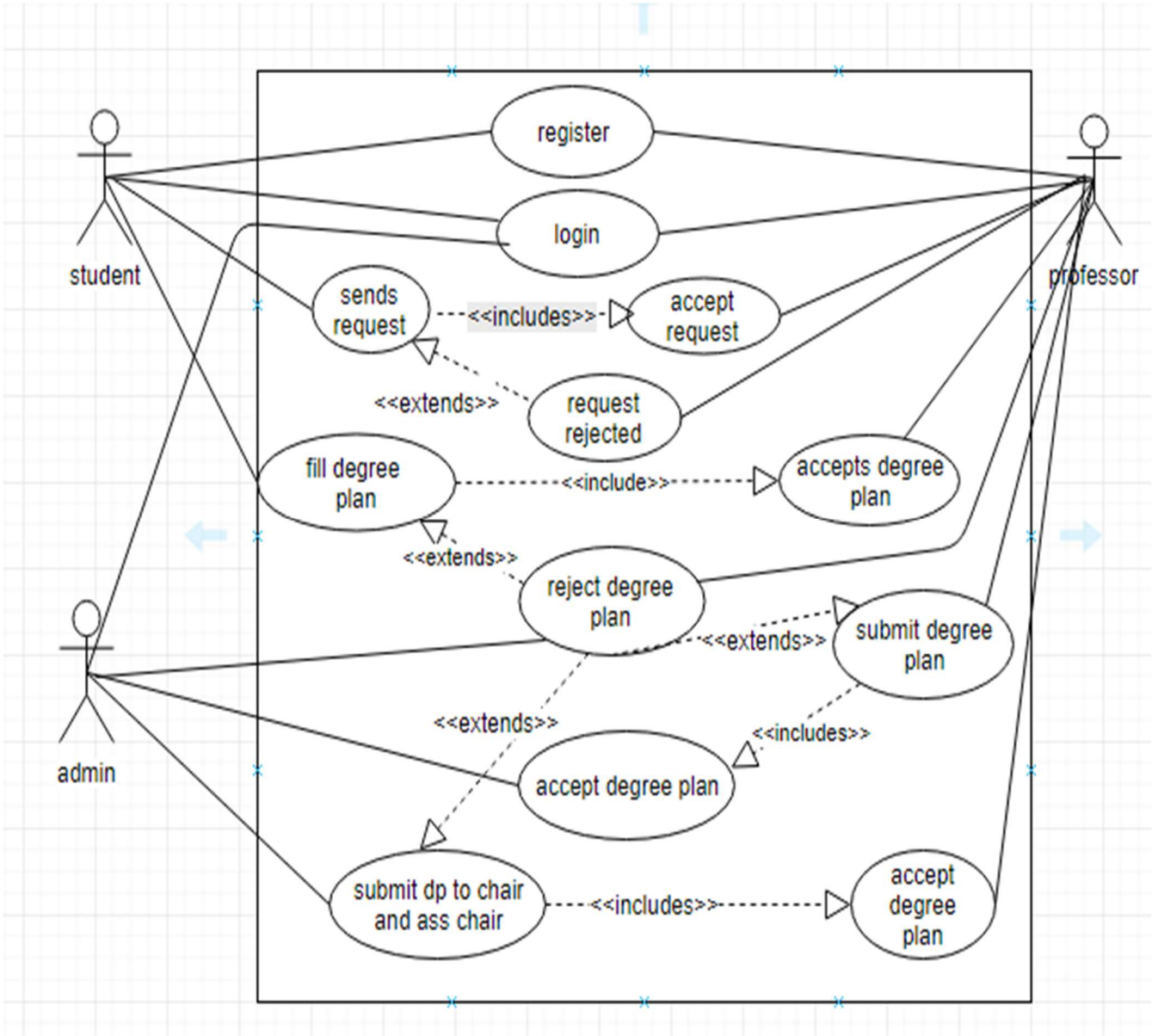
Main Success Scenario:

1: Student registers himself.
2: Student logs into the application.
3: Student sends advisory request form to professor.
4: Professor accepts the advisory request form from the student.
5: Student sends the degree plan form for approval to professor.
6: Professor approves the degree plan form.
7: Professor sends the approved degree plan form to Administrative specialist.
8: Administrative Specialist approves and forwards the approved degree plan form to Associate chair
9: The approved form from Associate Chair is sent to administrative specialist.
10: administrative specialist sends the degree plan approved by associate Chair to Chair.
11: The Chair approves the degree plan.
12: The Chair forwards the degree plan approved by him administrative specialist.

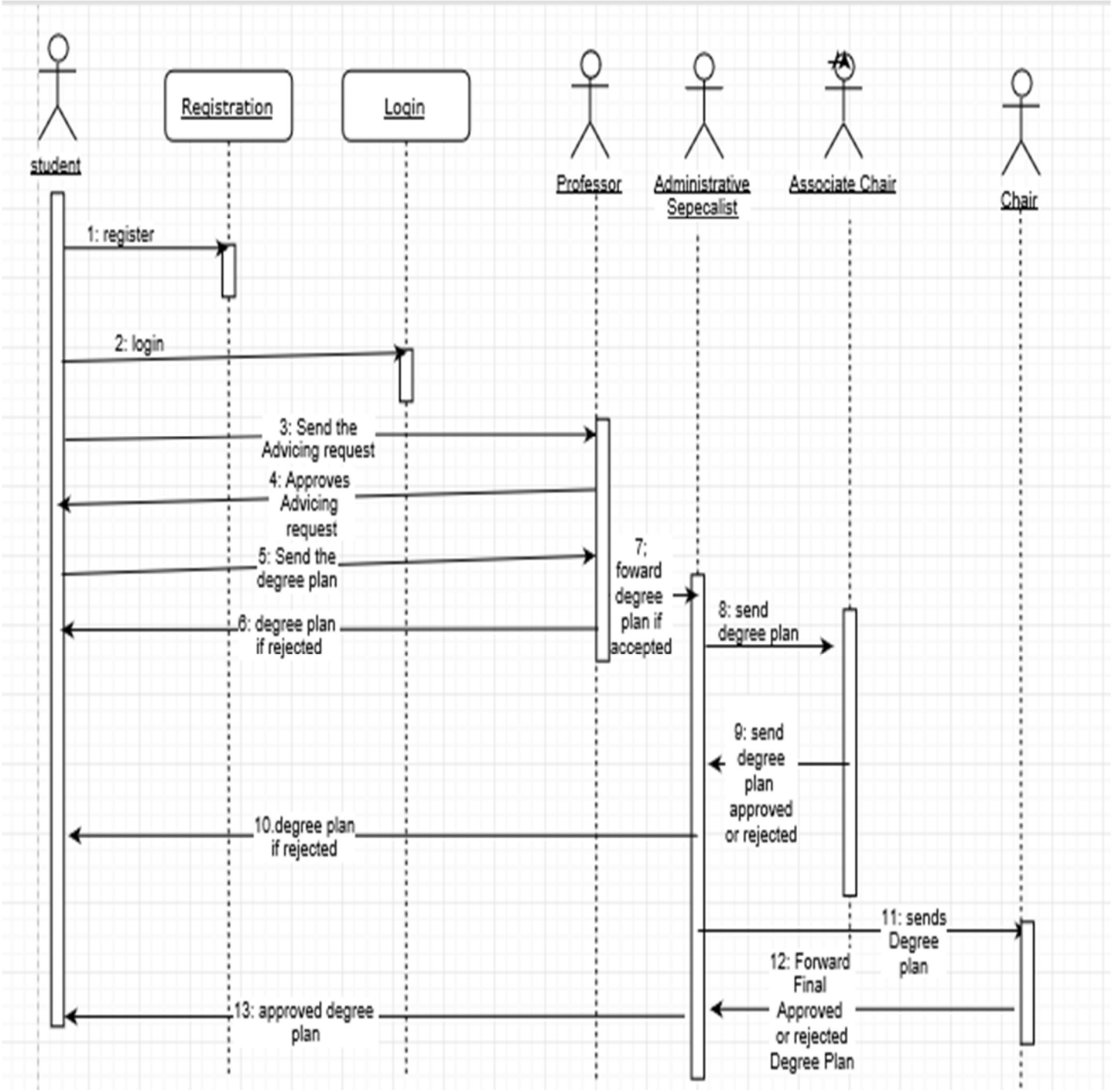
Extensions:

4a: Professor rejects the advisory request.
1. If professor rejects the advisor return to step 1
6a: Professor rejects the degree plan.
1. If professor rejects the degree plan, return to step 5.
9a: Administrative Specialist rejects degree plan.
1. If administrative rejects the degree plan go to step 5.
10a: Associate Chair has rejected the degree plan
1. If Associate Chair rejects the degree plan go to step 5.
11a: Chair has rejected the degree plan
1. If Chair rejects the degree plan go to step 5.

User Case Diagram



Sequence Diagram:



c. Test Cases (Unit Tests)

This is test case tests validateUser() method of loginDaoImpl that is it checks if the validateUser() method is getting the expected login i.e the correct login list from the database if the credentials are valid. This test takes the valid username, password and role as input and expects the output of the validateUser() method to be the exact matching login list. Assert compares expected login list with the login list returned by validateUser() and if it is same, this test call will pass.[2]

```
@Test
	public void testValidateUserForPopulatedList() {
		mockLoginList = new ArrayList<Login>();
		mockLoginList.add(new Login());
		when(mockJDBCTemplate.query(Matchers.anyString(),
		Matchers.any(LoginRowMapper.class)))
		.thenReturn(mockLoginList);
		LoginDaoImpl mockObj = new LoginDaoImpl();
		List<Login> actualLoginList = mockObj.validateUser("user",
		"password", "role");
		Assert.assertEquals(mockLoginList, actualLoginList);
	}
```

This is test case tests validateUser() method of loginDaoImpl class that is it checks if the validateUser() method is getting null list from the database if the credentials are invalid. This test takes the blank list (invalid credentials) as input and compares it with the login list returned by validateUser() and if it null that is same the input null list, this test will pass. [2]

```
@Test
	public void testValidateUserForBlankList() {
		mockLoginList = new ArrayList<Login>();
		when(mockJDBCTemplate.query(Matchers.anyString(),
		Matchers.any(LoginRowMapper.class)))
		.thenReturn(new ArrayList<Login>());
		LoginDaoImpl mockObj = new LoginDaoImpl();
		List<Login> actualLoginList = mockObj.validateUser("user",
		"password", "role");
		Assert.assertEquals(mockLoginList, actualLoginList);
	}
```

This test case checks if the user insertUserDetails() method of RegisterDaoImpl class is inserting the user into the database if all the details are valid. This test takes a Register list that is a valid user details list (mock list) as input and expects the output to be 1 and asserts it with 1. If it is equal the test will pass. [2]

```
@Test
	public void testValidateUpdatePositive() {
		mockresult=1;
	when(mockJDBCTemplate.update(Matchers.anyString(),objCap.capture() ))
		.thenReturn(mockresult);
		Register register=new Register();
		RegisterDaoImpl mockobj=new RegisterDaoImpl();
		int actualresult=mockobj.insertUserDetails(register);
		Assert.assertEquals(actualresult, mockresult);
		}
```

This test case checks the hashString() method called by login and register service classes. This test takes password and hashing algorithm as input and assert compares the hashed password returned by hashString() method against the expected hashed value of it is same the test case will pass. [1]

```
@Test
	public void testHashString() throws NoSuchAlgorithmException,
	UnsupportedEncodingException {
		Hashing hashTest = new Hashing();
		String hashedValue = hashTest.hashString("Test@432", "MD5");
		assertEquals("d232704062b0fea5c8d5b869cddef2a5", hashedValue);
	}
```

This test case checks if the length of the hashed password is eual to the expected hash length or not. It calls the hashString() method by duplicate password and hashing algorithm and checks if the length of the hashed password returned by the method is equal to the actual length 32. If it is same the test case will pass[1]

```
@Test
	public void testHashCodeLength() throws NoSuchAlgorithmException,
	UnsupportedEncodingException {
		Hashing hashTest = new Hashing();
		String hashedValue = hashTest.hashString("Test@432", "MD5");
		int length = hashedValue.length();
		assertEquals(32, length);
	}
```

This test checks if the convertByteArrayToHexString() method is returning the string data type. This test case calls the convertByteArrayToHexString()by passing in the hashed bytes as input and compares class of the output returned by method against java.lang.string if it is same this test will pass[1]

```
@Test
	public void testGetClass() {
		Hashing hashTest = new Hashing();
		byte[] hashedbytes = {43, -122, 76, -83, 14, -34, -103, 44, 47, -
		79, 57, 84, -48, 99, -116, -87};
		Class c =
		(hashTest.convertByteArrayToHexString(hashedbytes)).getClass();
		assertEquals("java.lang.String",
		(hashTest.convertByteArrayToHexString(hashedbytes)).getClass().ge
		tName());
	}
```

This test tests the sendAdvisorRequest()method of RequestDaoImpl class. It class the method sendAdvisorRequest() by passing in a mock request (genuine request) and expects the return value to be 1. It asserts the mock result 1 with the actual result and if both are same the test will pass. [2]

```
@Test
	public void testSendAdvisorRequest() {
		int mockresult=1;
	when(mockJDBCTemplate.update(Matchers.anyString(),objCap.capture() ))
		.thenReturn(mockresult);
		Request request=new Request();
		RequestDaoImpl mockobj=new RequestDaoImpl();
		int actualresult=mockobj.sendAdvisorRequest(request);
		Assert.assertEquals(actualresult, mockresult);
	}
```

This test case tests the getSentRequests() method of the RequestDaooImpl class. It sends username as input to the method getSentRequests(and expects the output to be the mockSentRequestList if the actualSentRequestList returned by getSentRequests() matches the mockSentRequestList the test case will pass. [2]

```
@Test
	public void testGetSentRequests() {

	List mockSentRequestList = new ArrayList<Request>();
	mockSentRequestList.add(new Request());
		when(mockJDBCTemplate.query(Matchers.anyString(),
		Matchers.any(ReceivedRequestRowMapper.class)))
		.thenReturn(mockSentRequestList);
		RequestDaoImpl mockobj=new RequestDaoImpl();
		List<Request> actualSentRequestList =
		mockobj.getSentRequests("userName");
		Assert.assertEquals(mockSentRequestList, actualSentRequestList);

	}
```

This test case tests the getacceptedRequests() method of the RequestDaooImpl class. It sends username as input to the method getSentRequests(and expects the output to be the mockRequestList if the actualRequestList returned by getacceptedRequests() matches the mockRequestList the test case will pass[2]

```
@Test
public void testGetacceptedRequests() {

	List mockRequestList = new ArrayList<Request>();
	mockRequestList.add(new Request());
		when(mockJDBCTemplate.query(Matchers.anyString(),
		Matchers.any(RequestRowMapper.class)))
		.thenReturn(mockRequestList);
		RequestDaoImpl mockobj=new RequestDaoImpl();

		List<Request> actualRequestList =
		mockobj.getacceptedRequests("userName");
		Assert.assertEquals(mockRequestList, actualRequestList);

}
```

This test tests the getOptionalCourses() method of the degreePlanService class it tests if the method is returning the correct list of optional courses. It mocks the getOptionalCourses() method of degreePlanDao and returns the mockCourseList of size 2 to the getOptionalCourses() method and asserts the length of the list returned by getOptionalCourses() to be 2. If it is same the method getOptionalCourses() is returning correct list and the test case will succeed. [2]

```
@Test
public void testGetOptionalCourses() {
		List<String> mockCourseList = Arrays.asList("CSCE 5350 SE",
		"CSCE5450 CS");
	   when(degreePlanDao.getOptionalCourses()).thenReturn(mockCourseList);
		List<String> result = new ArrayList<String>();
		result = degreePlanService.getOptionalCourses();
		assertEquals(2, result.size());
}
```

This test tests the submitDegreePlan() method of the degreePlanService class it tests if the method is returning properly saving the degree plan in the database and properly submitting it to the major professor. This test mocks the submitDegreePlan() method of degreePlanDao with any input and expects the output to be 1 that is the it expects the degree plan to be successfully inserted into the database. Assert compares 1 with the result and if it is true the test case will pass. [2]

```
@Test
public void testSubmitDegreePlan() {
            when(degreePlanDao.submitDegreePlan(any(),
            anyString())).thenReturn(1);
            int result = degreePlanService.submitDegreePlan(any(), ());
            assertEquals(1, result);
}
```

This case tests the split() method of the RequestServiceImpl class. It sends the known string string details as input to the split method and expects the length of the returned array to be 2, if it is same then the test will succeed. It is asserting the expected length of array to be 2 because in real application it is expected to return only two values in array.

```
@Test
public void testSplit() {
            RequestService requestService = new RequestServiceImpl();
            String details = "{'professorEmail':['BryantBarett@unt.edu'],
            'userName':['sharanya']}";
            String[] result = requestService.split(details);
            assertEquals(2, result.length);
}
```

This test case tests the getGroupDCourses() method of the degreePlanService class. It Mocks the getCoreCourses() method of degreePlanDao and returns a mocklist to the getGroupDCourses() method. getGroupDCourses() filters the GroupDcourses from the mock list and returns to the result. The asserts compares the result.size() with 1 because the mock list has only one GroupD course with major CS. If the result is equal to one the test case succeeds. [2]

```
@Test
      public void testGetGroupDCourses() {
            Course c1 = new Course();c1.setCourseName("CSCE
            5500");c1.setGroupId("A");c1.setMajor("CS");
            Course c2 = new Course();c2.setCourseName("CSCE
            5400");c2.setGroupId("B");c2.setMajor("CE");
            Course c3 = new Course();c3.setCourseName("CSCE
            5600");c3.setGroupId("D");c3.setMajor("CS");
            Course c4 = new Course();c4.setCourseName("CSCE
            5200");c4.setGroupId("D");c4.setMajor("CE");
            List<Course> mockList = new ArrayList<Course>();

      mockList.add(c1);mockList.add(c2);mockList.add(c3);mockList.add(c4);
            when(degreePlanDao.getCoreCourses()).thenReturn(mockList);
            List<Course> result = degreePlanService.getGroupDCourses("CS");
            assertEquals(1, result.size());
      }
```

This test case tests the deleteRequest() method of the RequestServiceImpl class. It Mocks the deleteRequest() method of RequestDao class and returns a result of integer value where 0, indicates that no records were deleted and 1 indicates that the record is deleted. The asserts compares the result with 1 because the method returns 1 on successful deletion. If the result is equal to one the test case succeeds.

```
@Test
public void testDeleteRequest() {
	when(requestdao.deleteRequest(anyString(), anyString())).thenReturn(1);
	int result = requestService.deleteRequest(anyString(), anyString());
	assertEquals(1, result);
}
```

This test case tests the checkCredentials() method of the LoginServiceImpl class. It Mocks the validateUser() method of LoginDao class and returns a mockloginlist to the loginServiceImpl. LoginServiceImpl performs its service on the mocklist and returns the result and this test expects the result list size to be 1 and compares it to be 1. If both are one then the test case succeeds.

```
@Test
public void testLoginServiceImpl() {
	mockLoginList = new ArrayList<Login>();
	mockLoginList.add(new Login());
	when(logindao.validateUser(anyString(), anyString(),
	anyString())).thenReturn(mockLoginList);
	List<Login> result = new ArrayList<Login>();
	result = loginService.checkCredentials("sharanya",
	"d232704062b0fea5c8d5b869cddef2a5", "student");
	assertEquals(1, result.size());
}
```

This test case tests the getGroupCCourses() method of the degreePlanService class. It Mocks the getCoreCourses() method of degreePlanDao and returns a mocklist to the getGroupCCourses() method. getGroupCCourses() filters the GroupCcourses from the mock list and returns to the result. The asserts compares the result.size() with 1 because the mock list has only one GroupC course with major CS. If the result is equal to one the test case succeeds.

```
@Test
public void testGetGroupCCourses() {
		Course c1 = new Course();c1.setCourseName("CSCE
		5500");c1.setGroupId("A");c1.setMajor("CS");
		Course c2 = new Course();c2.setCourseName("CSCE
		5400");c2.setGroupId("B");c2.setMajor("CE");
		Course c3 = new Course();c3.setCourseName("CSCE
		5600");c3.setGroupId("C");c3.setMajor("CS");
		Course c4 = new Course();c4.setCourseName("CSCE
		5200");c4.setGroupId("C");c4.setMajor("CE");
		List<Course> mockList = new ArrayList<Course>();

	mockList.add(c1);mockList.add(c2);mockList.add(c3);mockList.add(c4);
		when(degreePlanDao.getCoreCourses()).thenReturn(mockList);
		List<Course> result = degreePlanService.getGroupCCourses("CE");
		assertEquals(1, result.size());
}
```

This test tests the getStaffDirectory() method of the retrieveStaffDaoImpl class. It mocks the jdbc template and returns the mockprofessorList to the getStaffDirectory which returns actualProfList. The assert compares both and if equal the test case succeeds.

```
@Test
	public void testGetStaffDirectory() {

	List<Register> mockProfList = new ArrayList<Register>();
	mockProfList.add(new Register());
		when(mockJDBCTemplate.query(Matchers.anyString(),
		Matchers.any(ProfessorRowMapper.class)))
		.thenReturn(mockProfList );

		RetrieveStaffDaoImpl mockrsdi = new RetrieveStaffDaoImpl();
		List<Register> actualprofList=mockrsdi.getStaffDirectory();
		Assert.assertEquals(mockProfList, actualprofList);

	}
```

This test tests the getMyStudents() method of the retrieveUsersDaoImpl class. It mocks the jdbc template and returns the mockStudentList to the getMyStudents() which returns actualStudentList. The assert compares both and if equal the test case succeeds.

```
@Test
	public void testGetmyStudentsForPopulatedList() {

		mockMyStudentList = new ArrayList<Request>();
		mockMyStudentList.add(new Request());
		when(mockJDBCTemplate.query(Matchers.anyString(),
		Matchers.any(MyStudentRowMapper.class)))
		.thenReturn(mockMyStudentList);
		RetrieveUsersDaoImpl mockObj = new RetrieveUsersDaoImpl();
		List<Request> actualStudentList = mockObj.getMyStudents("uName");
		Assert.assertEquals(mockMyStudentList, actualStudentList);

	}
```

This test tests the getMyAdvisors() method of the retrieveUsersDaoImpl class. It mocks the jdbc template and returns the mockMyAdvisorRequestList to the getMyStudents() which returns actualMyAdvisorRequestList. The assert compares both and if equal the test case succeeds.

```
@Test
public void testGetMyAdvisorsForPopulatedList() {
		mockMyAdvisorRequestList = new ArrayList<Request>();
		mockMyAdvisorRequestList.add(new Request());
		when(mockJDBCTemplate.query(Matchers.anyString(),
		Matchers.any(MyAdvisorRowMapper.class)))
		.thenReturn(mockMyAdvisorRequestList);

		RetrieveUsersDaoImpl mockObj = new RetrieveUsersDaoImpl();
		List<Request> actualMyAdvisorRequestList =
		mockObj.getMyAdvisors("userName");
		Assert.assertEquals(mockMyAdvisorRequestList,
		actualMyAdvisorRequestList);
```

d. Team Member Contribution

Name of the Member	Components Developed	Overall Contribution(%)
Sharanya Gottimukkula	FREQ-5, FPREQ-8 Designed HTML pages for showing the professor list, student list, sent requests, received requests and accepted requests. Created Database tables required to store the request related data. Involved in writing RequestController, RequestServiceImpl and RequestDaoImpl classes which handles all the requests between HTML and database	25 %
Nanditha Bodanapu	FREQ-6, FREQ-7 Designed HTML degree plan form. Created Database tables required to store the degree plan related data. Involved in writing DegreePlanController, DegreePlanServiceImpl and DegreePlanDaoImpl classes which handles all the degree plan between HTML and database	25 %
Aravind Swamy Thottempudi	FREQ-1, FREQ-2 Designed HTML pages for login and registration. Created Database tables required to store the users and allow them to login. Involved in writing LoginController, RegisterController, RegisterServiceImpl and LoginServiceImpl classes which handles all the login and register requests	25 %
Sri Harshini Vallabhaneni	FREQ-3 Designed HTML page for administrator and his functionality manage users Created Database tables required to store the admin managed users. Involved in writing AdminController, Student Controller, RetrieveStaffImpl and RetrieveUsersService classes which handles all the admin functionalities and retrieve user functionalities	25 %

e. User Manual

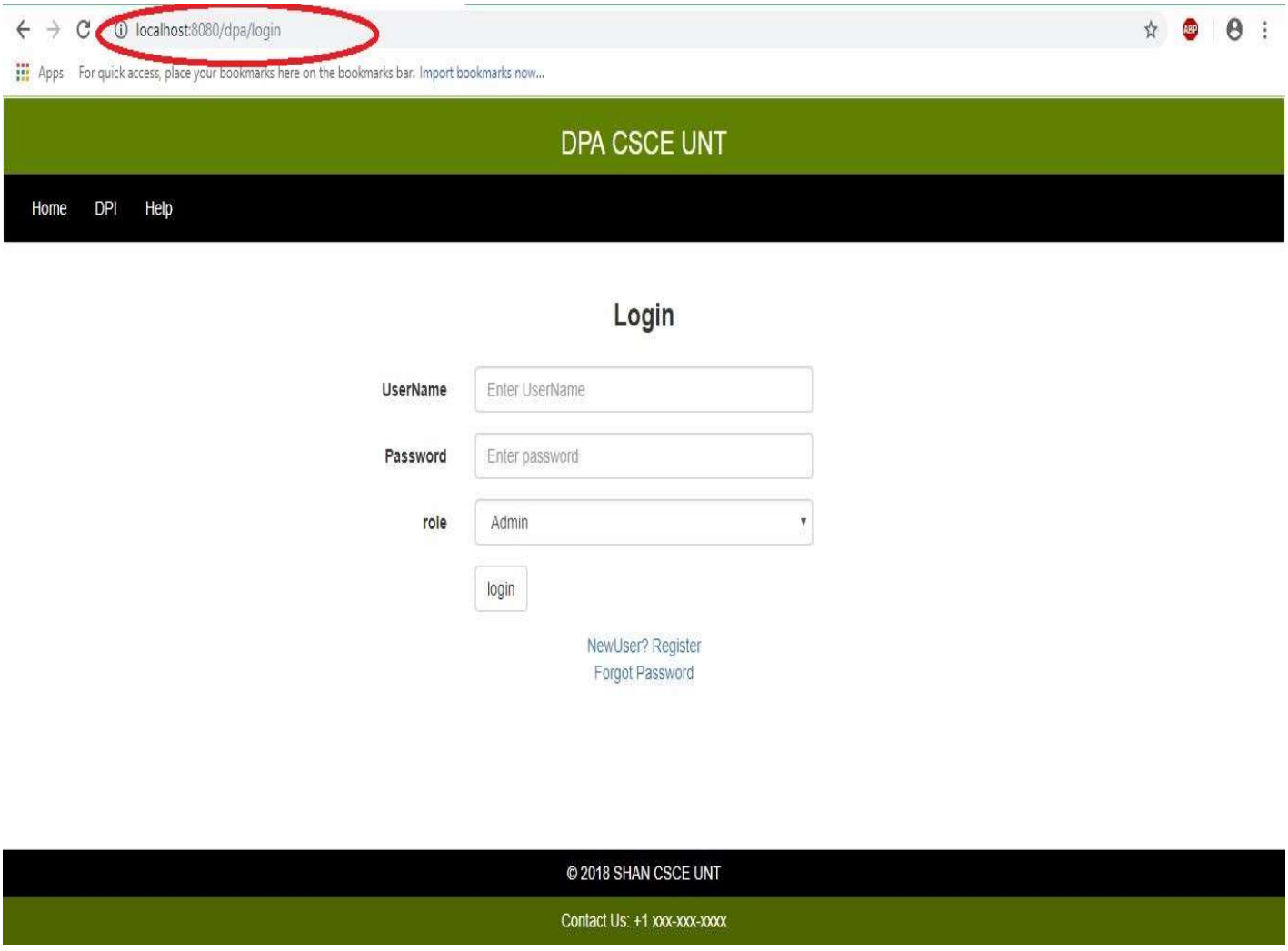
Degree Plan Automation System:

This is web application. In order to use this system, user needs to install the below software:

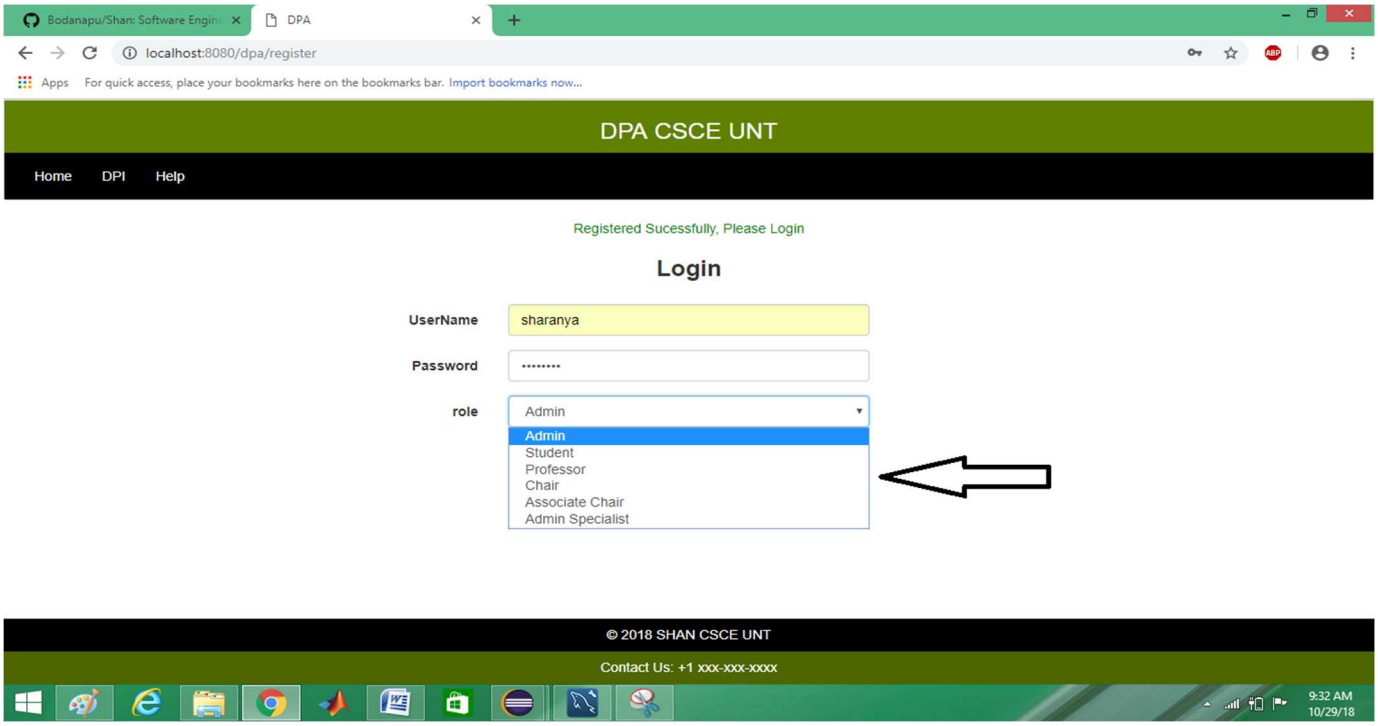
Eclipse
MYSQL workbench and server
Google chrome

Type the URL " localhost:8080/dpa/login" in the browser and hit enter to access the main page of the DPA system.

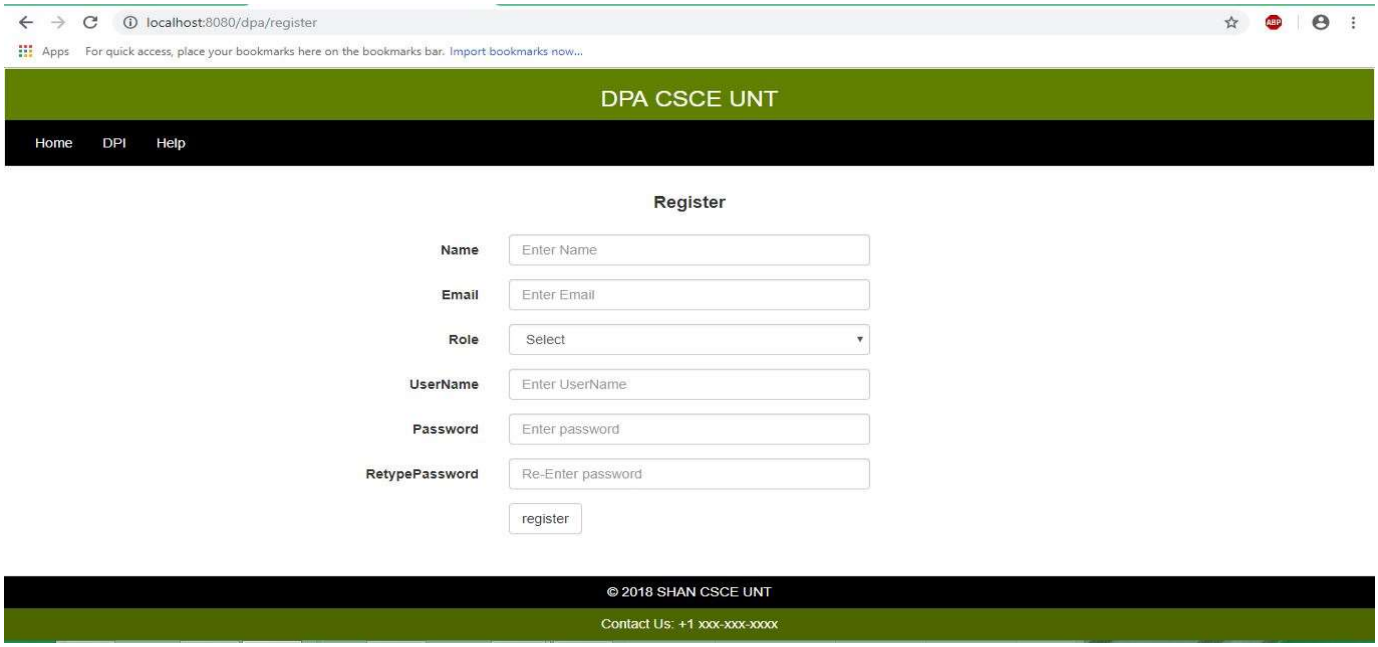
This page has login fields and a link to new user registration page and forgot password page.



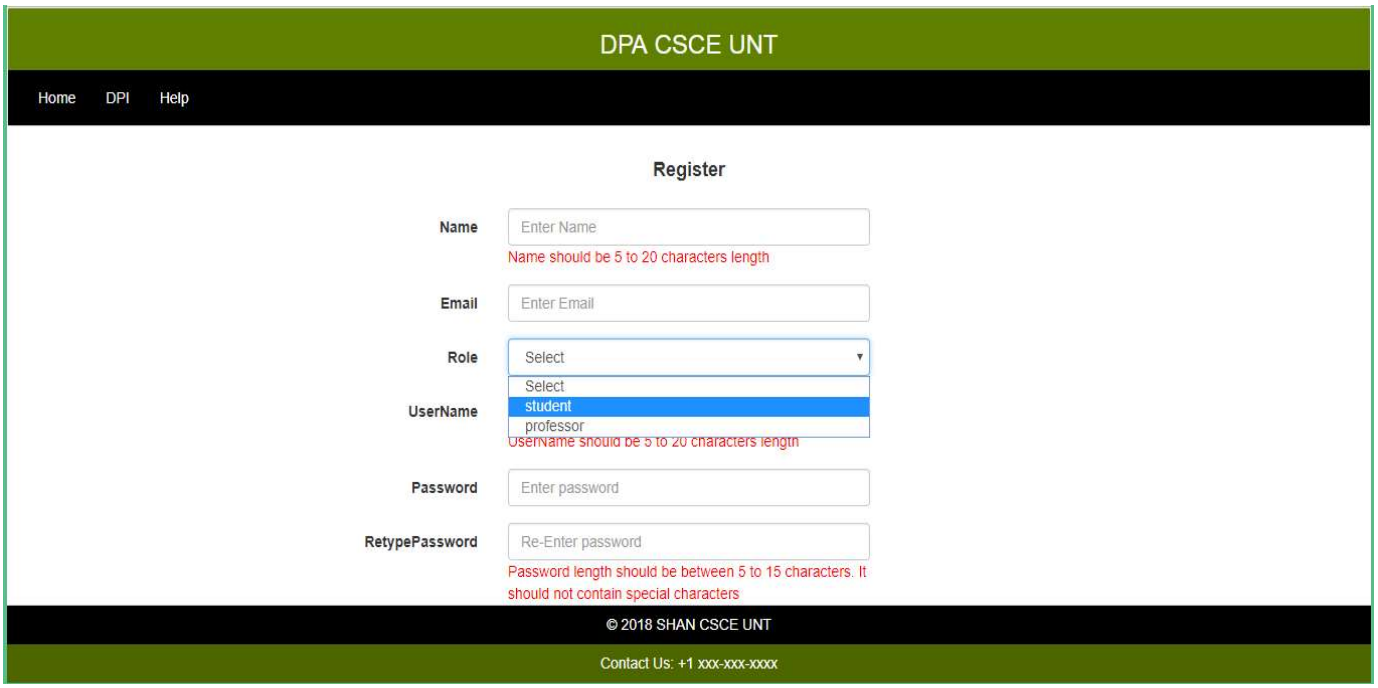
The login fields allow admin, student, professor, chair, associate chair and admin specialist to login by typing in their username, password and choosing appropriate role from the dropdown field, which has options as in screenshot below. All the three fields are required. The username and password fields are validated and will not allow special characters.



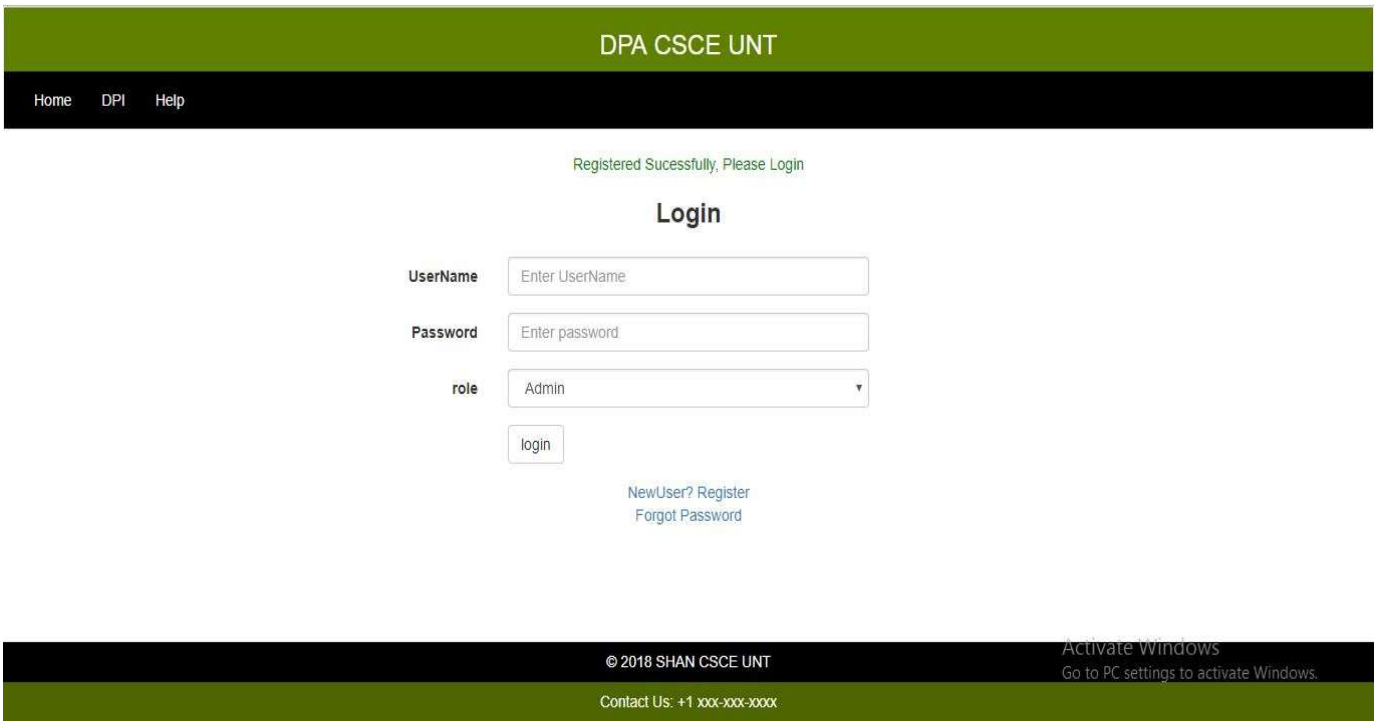
By clicking on the “NewUser?Register” link on the main page of the website user will be redirected to registration page



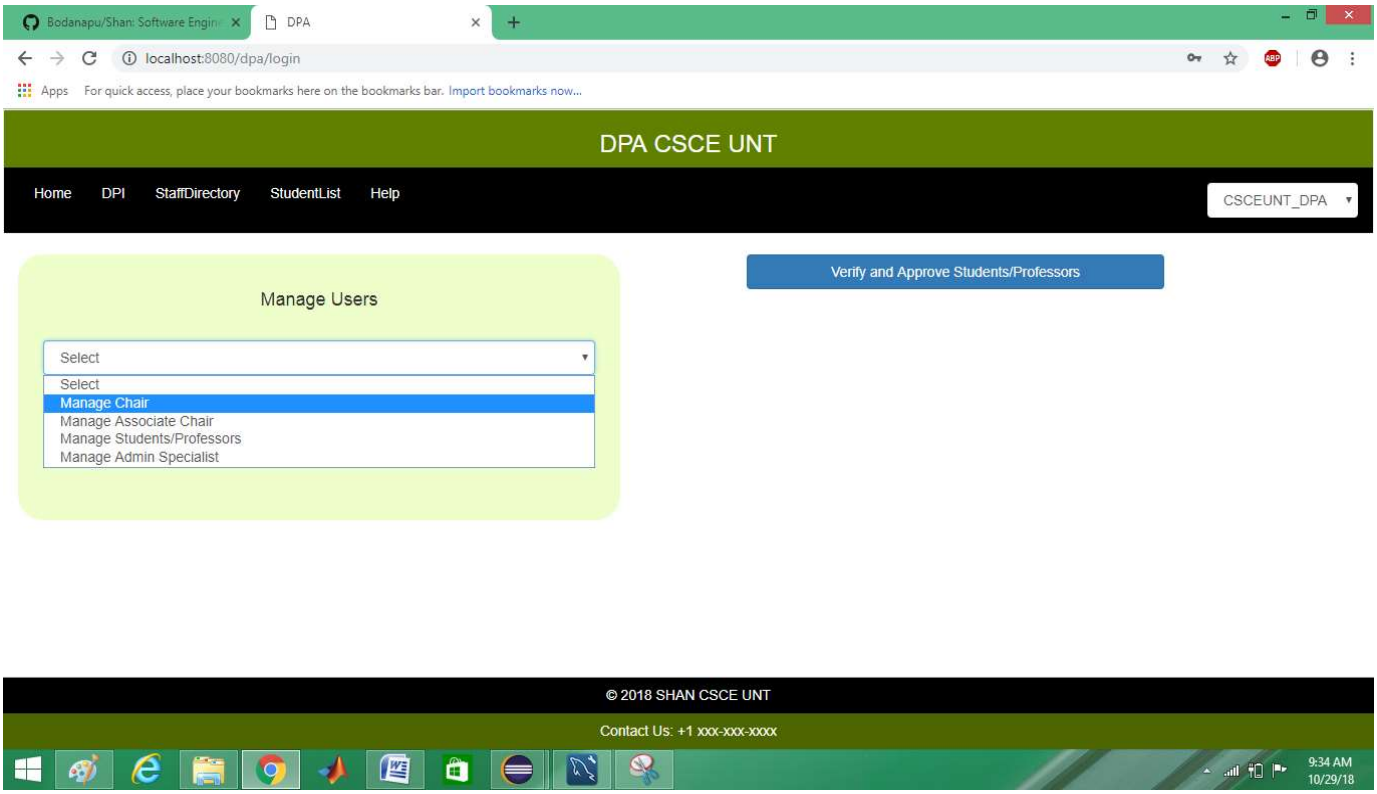
This registration page allows only students and professors to register. Role dropdown has only two roles student and professor. All the fields are mandatory and have validations, which are notified to the user on focus.



Upon successful registration the user will be redirected to the login page with the success message

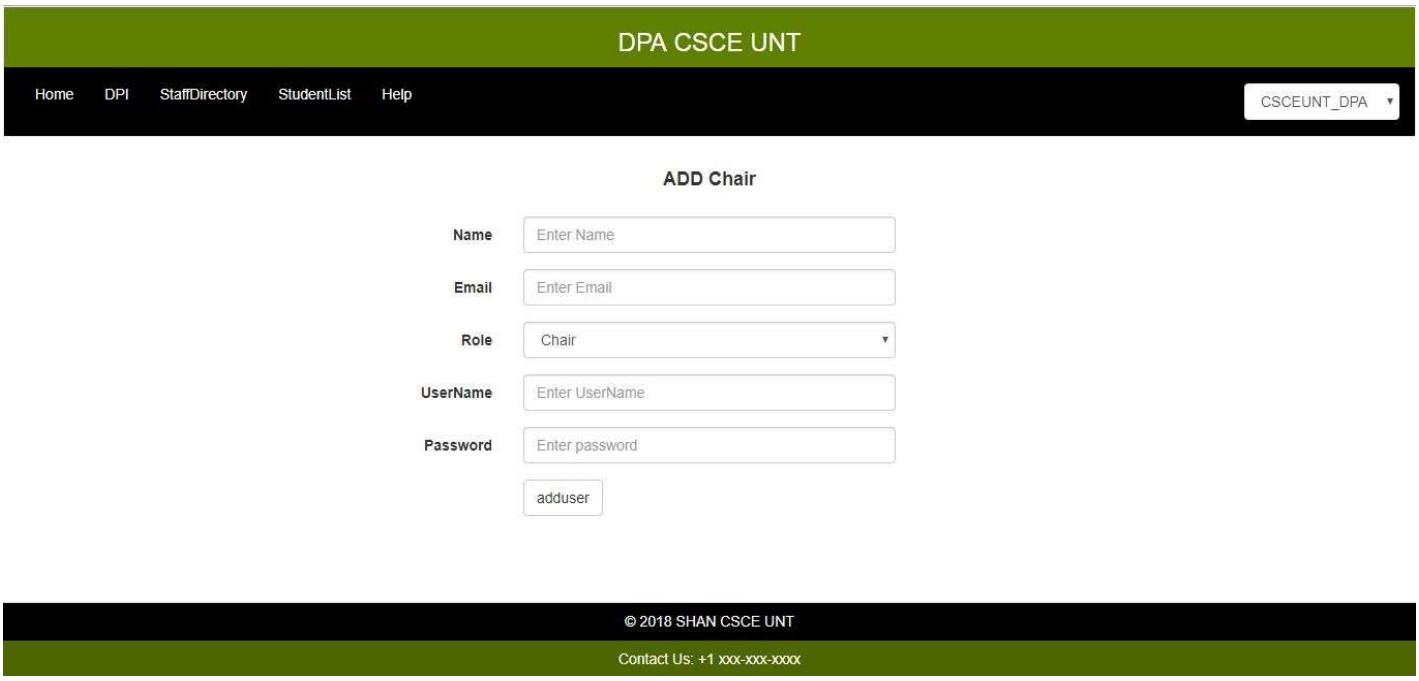


User login as Admin: If you login with role as admin, you will be directed to admin home page where you has options to manage chair, associate chair, students/professors and admin specialist.

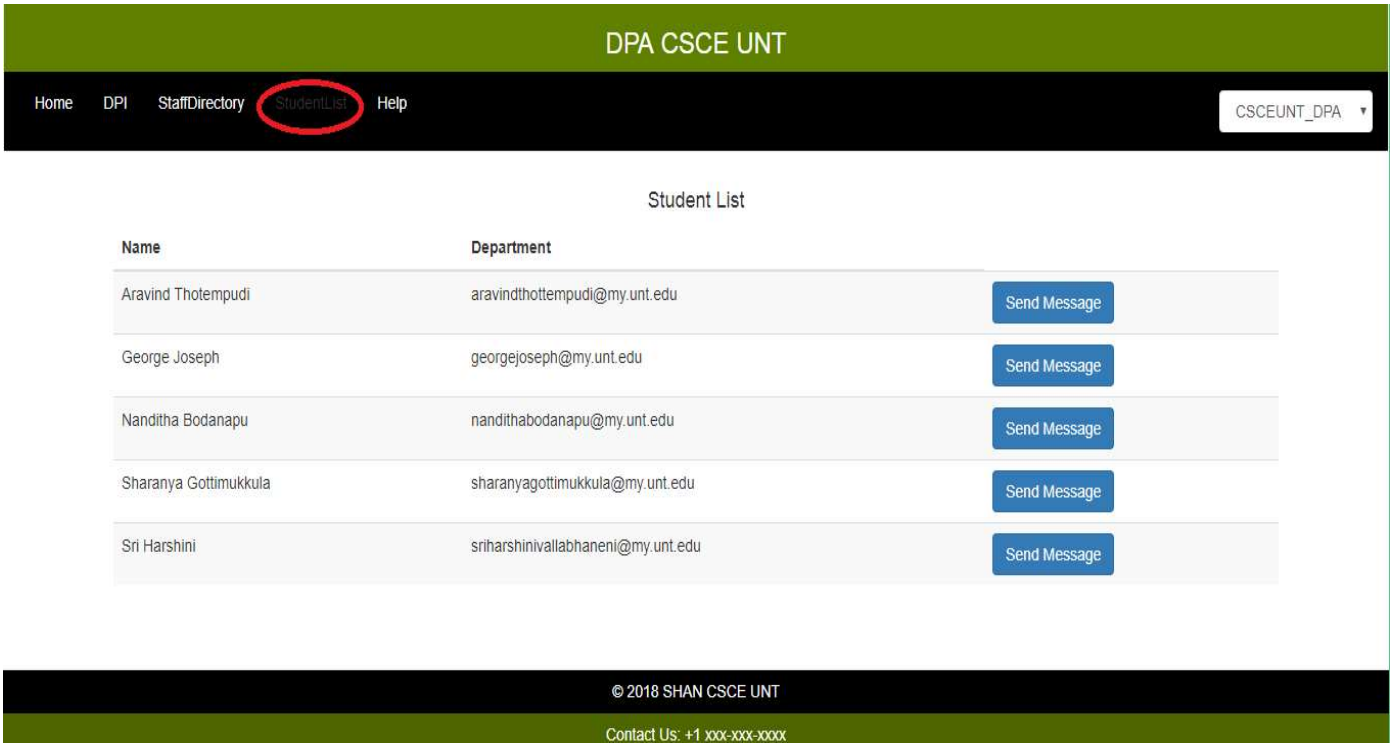


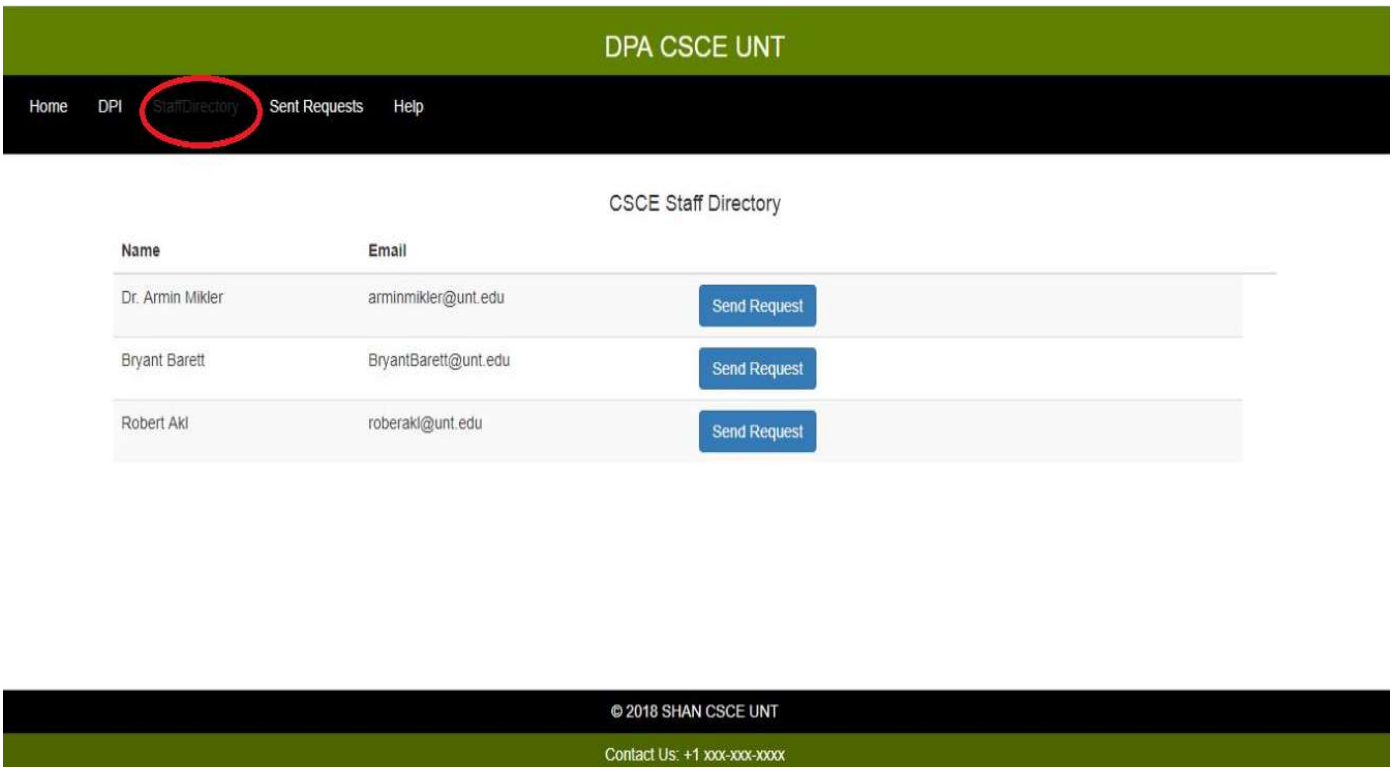
By clicking on manage chair you get options to add, delete or update chair. By clicking on manage associate chair you get options to add, delete or update associate chair. By clicking on manage professors/students you get options to delete or update professors/students. By clicking on manage admin specialist you get options to add, delete or update admin specialist.

By clicking on ADD chair you will be directed to a add chair page where you can fill in the chair details and add him/her.

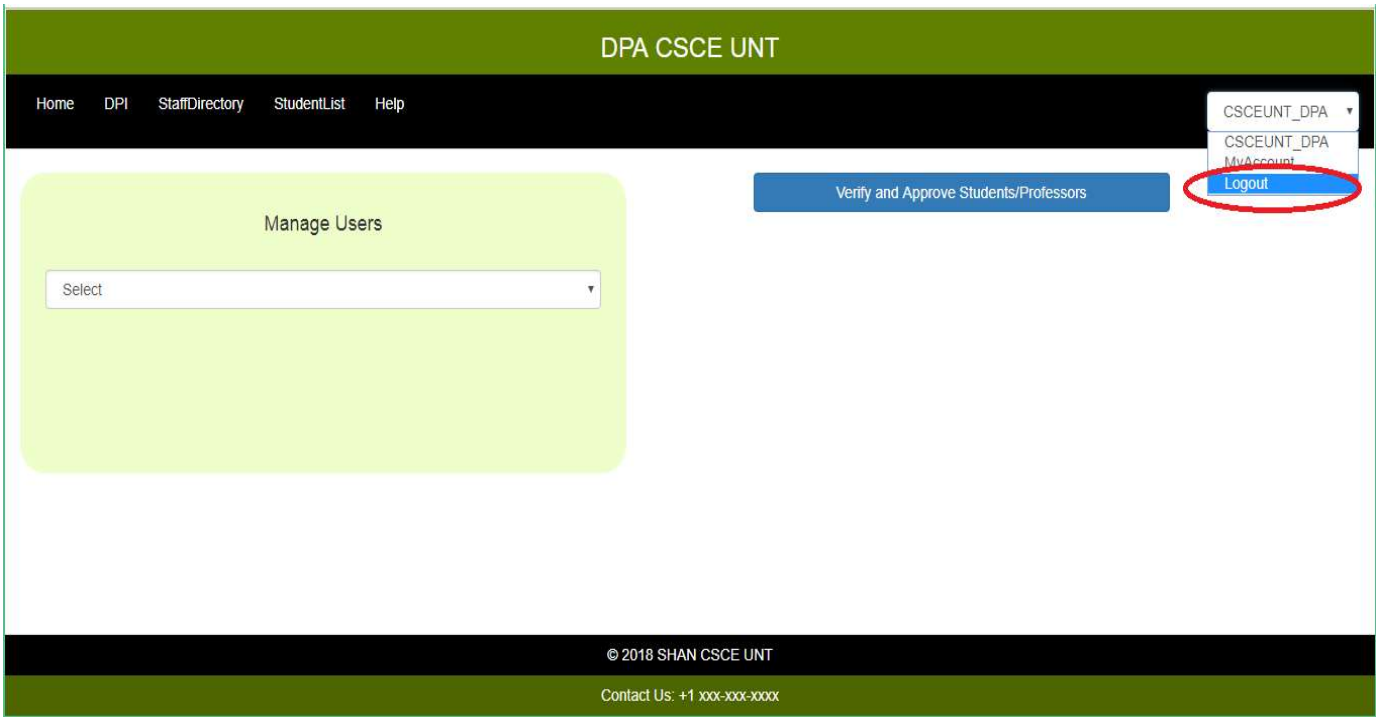


Admin can also view the student list and staff directory by clicking on the links “StaffDirectory” and “StudentList” options on the menu bar.

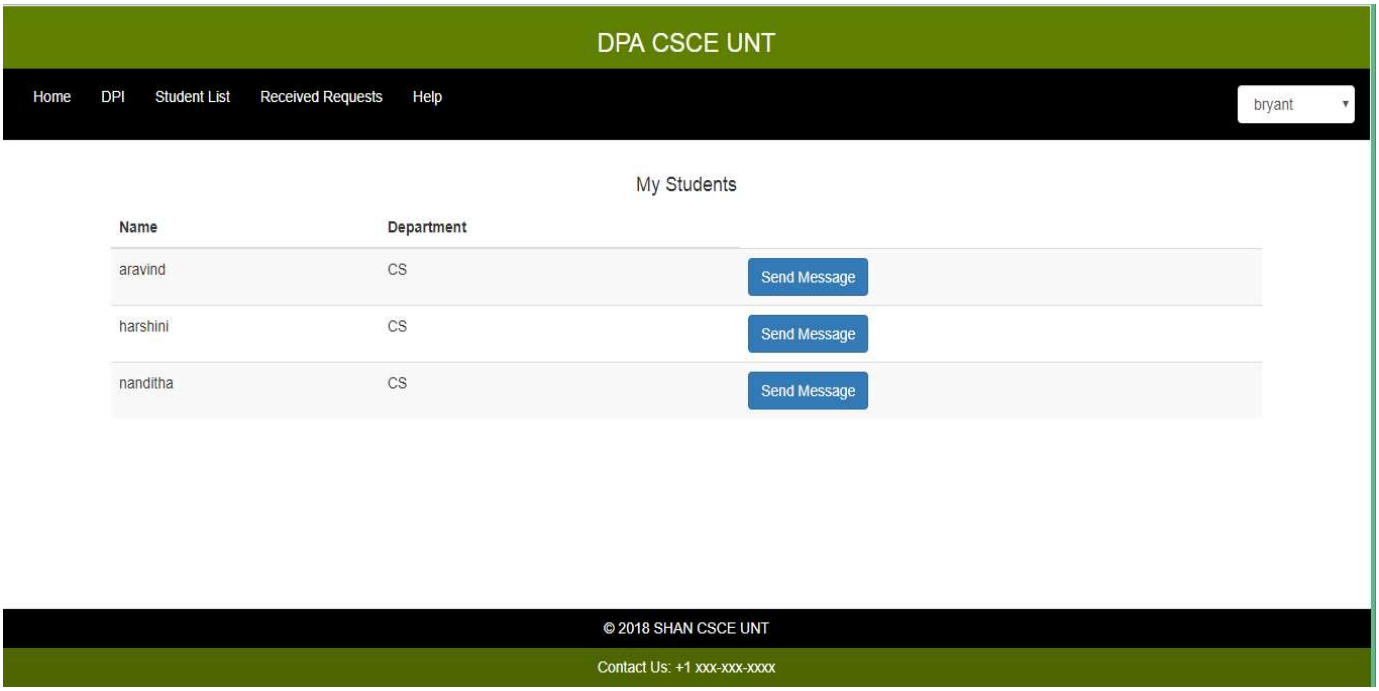




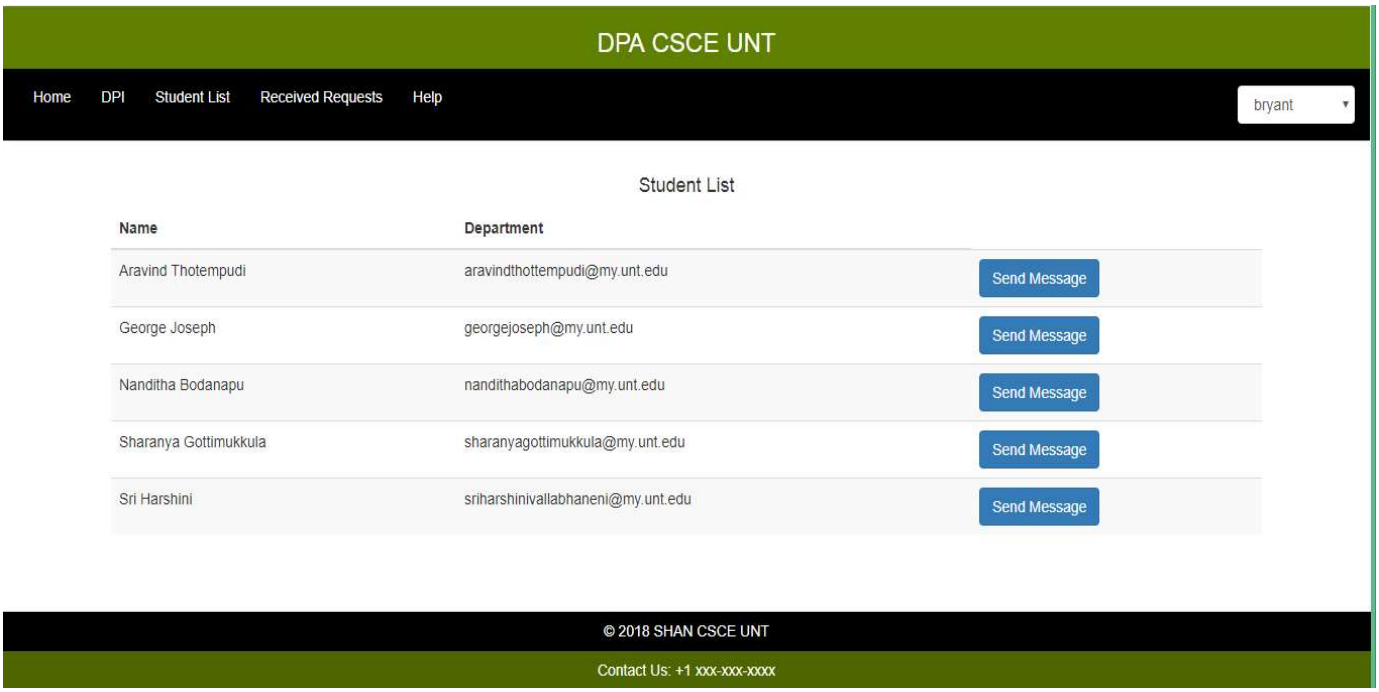
You can logout by clicking the button logout present in the dropdown with your username display.



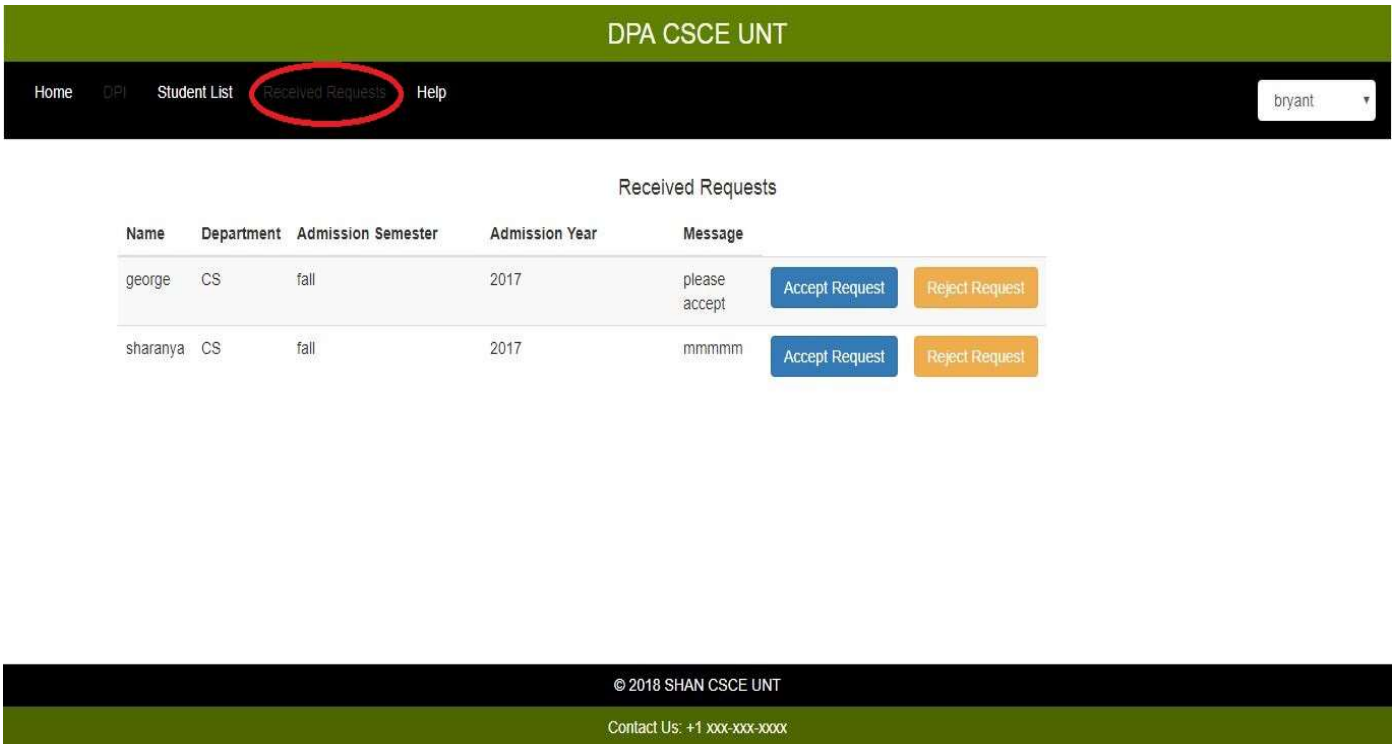
User login as Professor: If you login, choosing role as a professor, chair or associate chair you will be directed to a professor home page. Professor home displays list of students, whose request has been accepted by you.



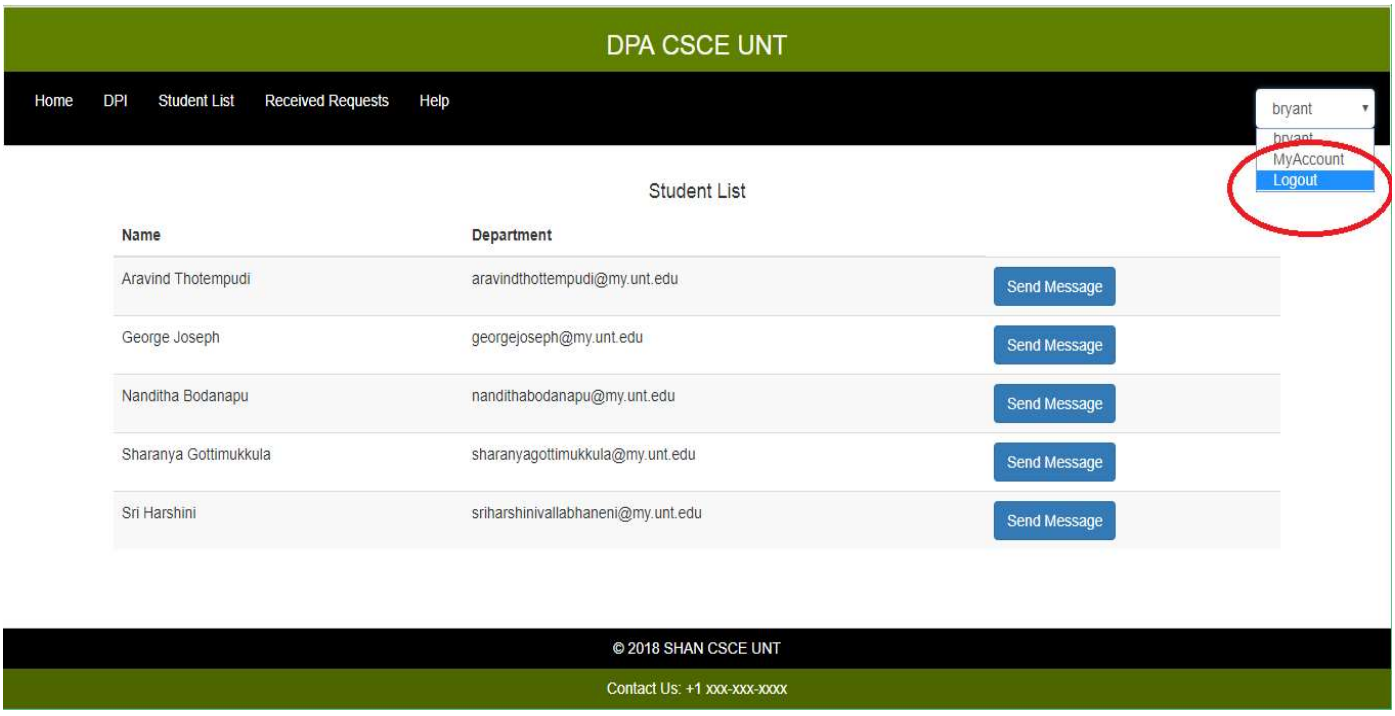
You have options to view Student List and received requests by clicking on the options “StudentList” and “ReceivedRequests” on the navigation bar at the top.



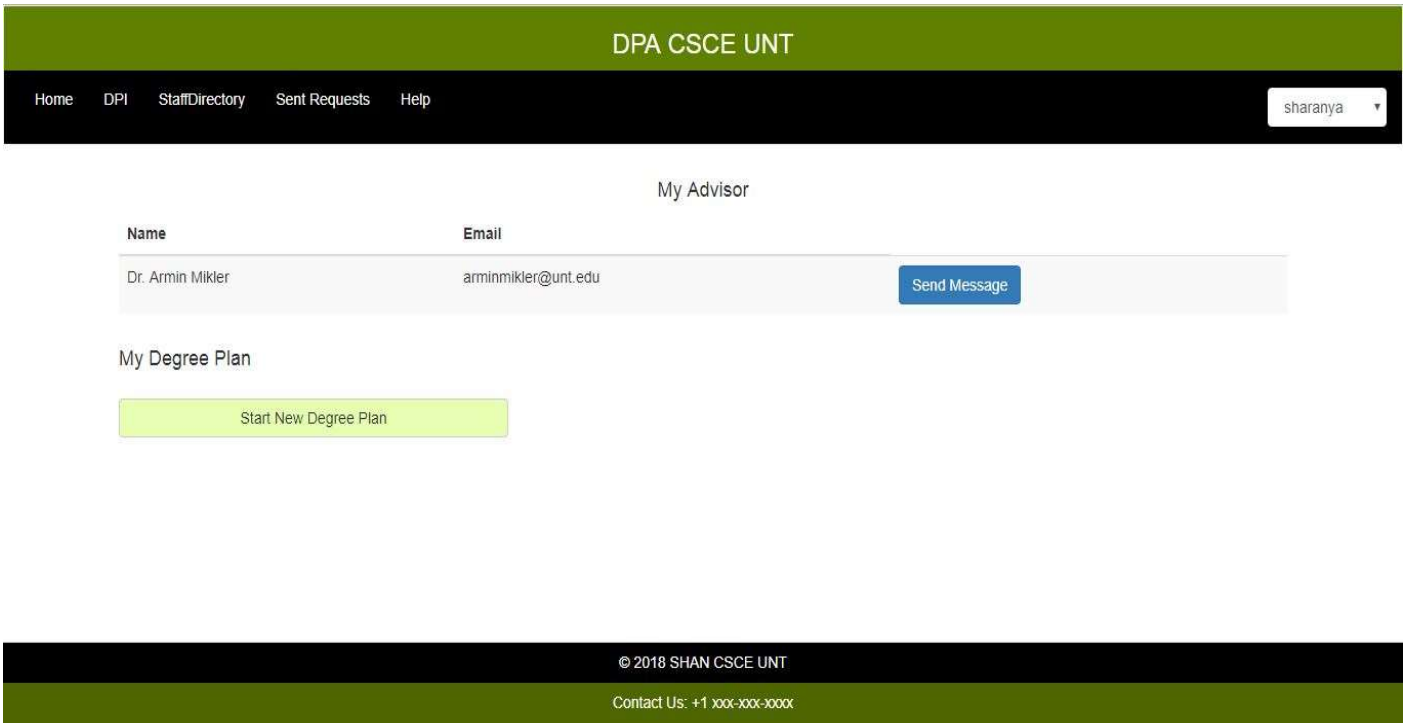
You can accept the request by clicking on the Accept Request button and reject the request by clicking on the Reject Request button



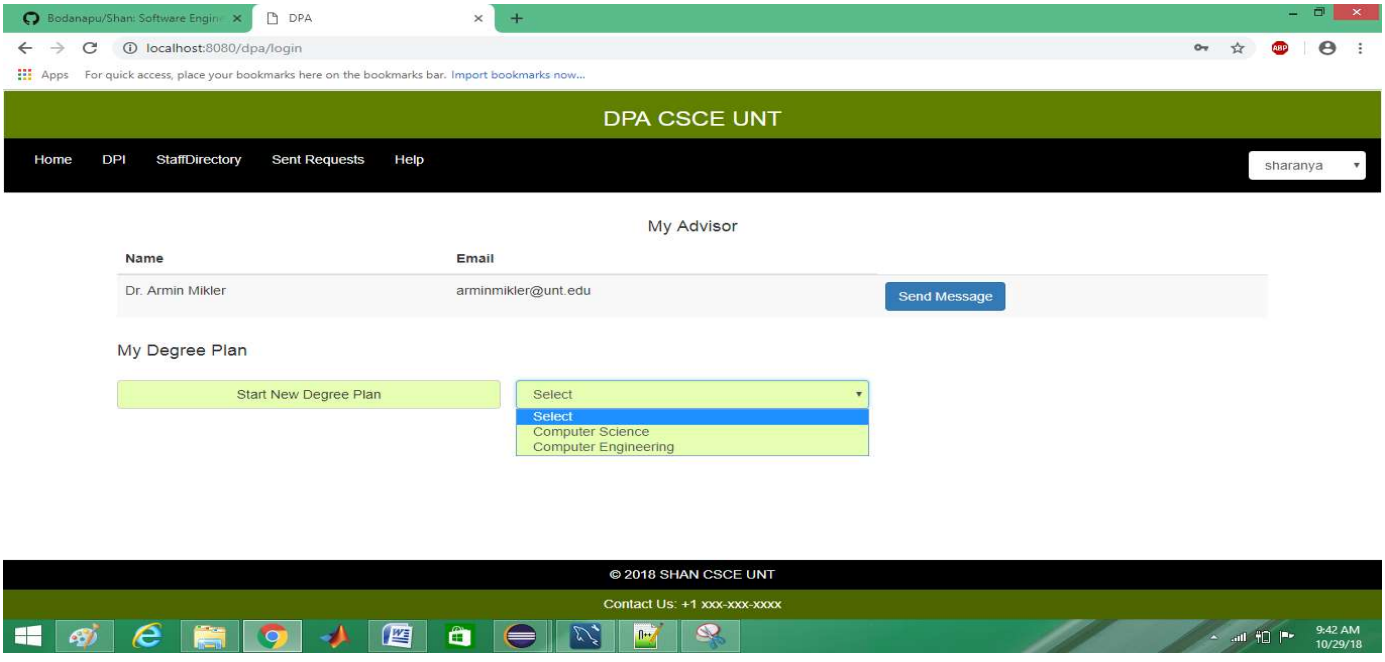
You can logout by clicking the button logout present in the dropdown with your username display.



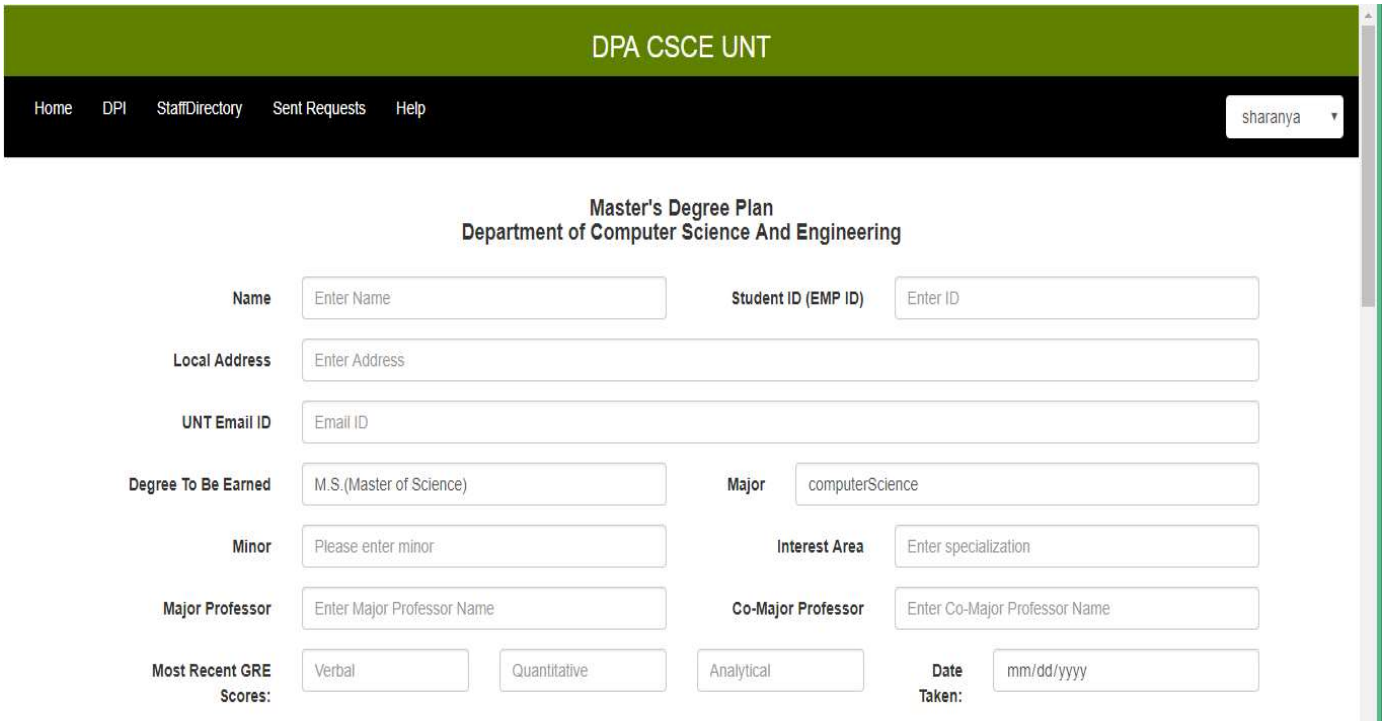
User login as Student: If you login by choosing the role as student, you will be directed to the student home. Student home displays the major professor name and email and also has an option to start new degree plan.

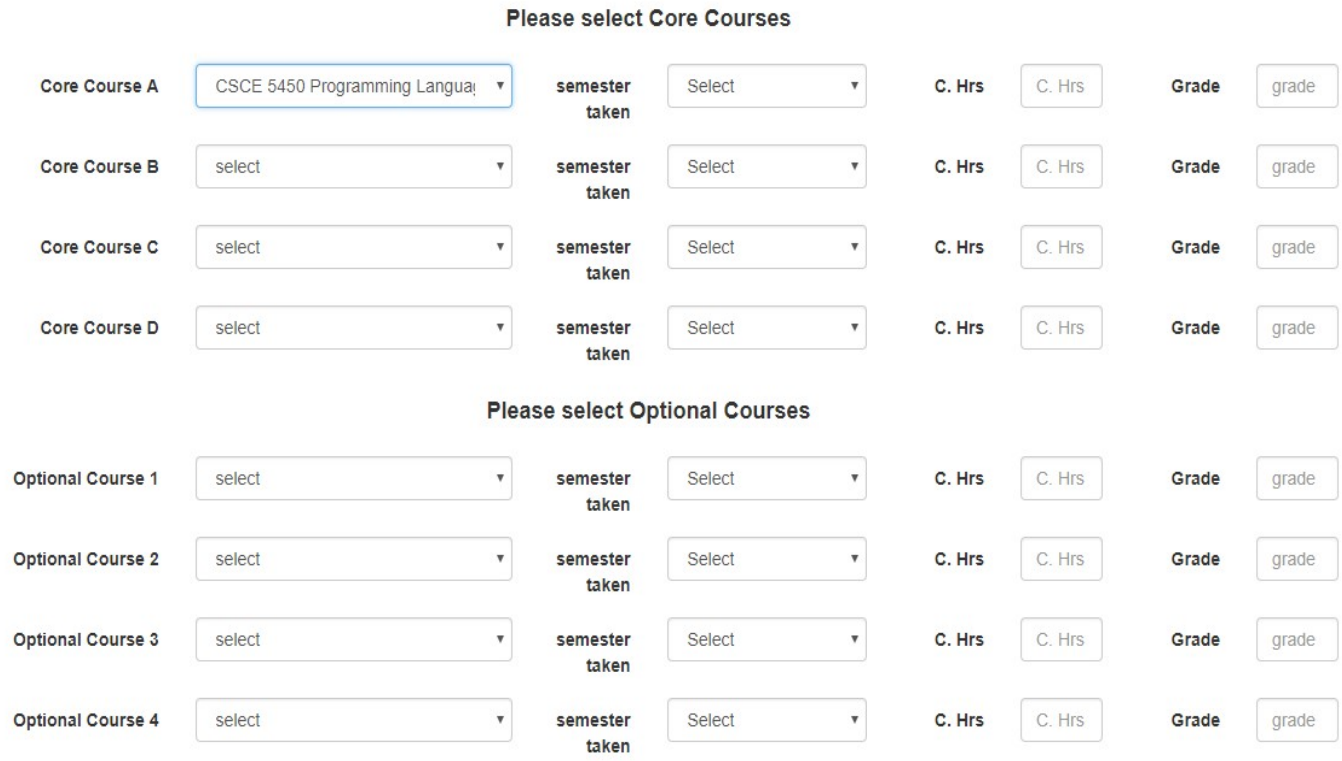


After clicking the start degree plan button you can choose the major for which you want to file the degree plan. Depending on the major option you choose you will be getting the dropdown courses related to that major.

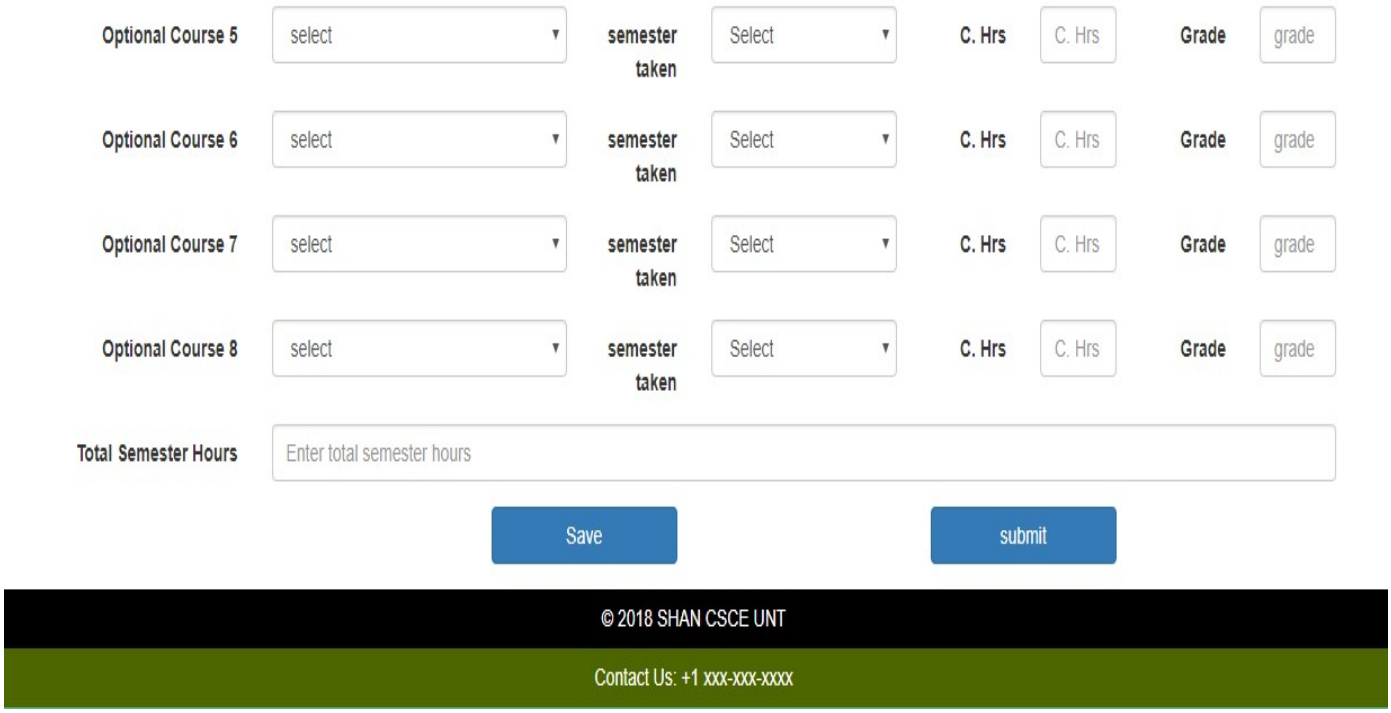


After choosing the major the degree plan form as in below screenshots will be displayed. The major will be auto populated as per your choice of major.



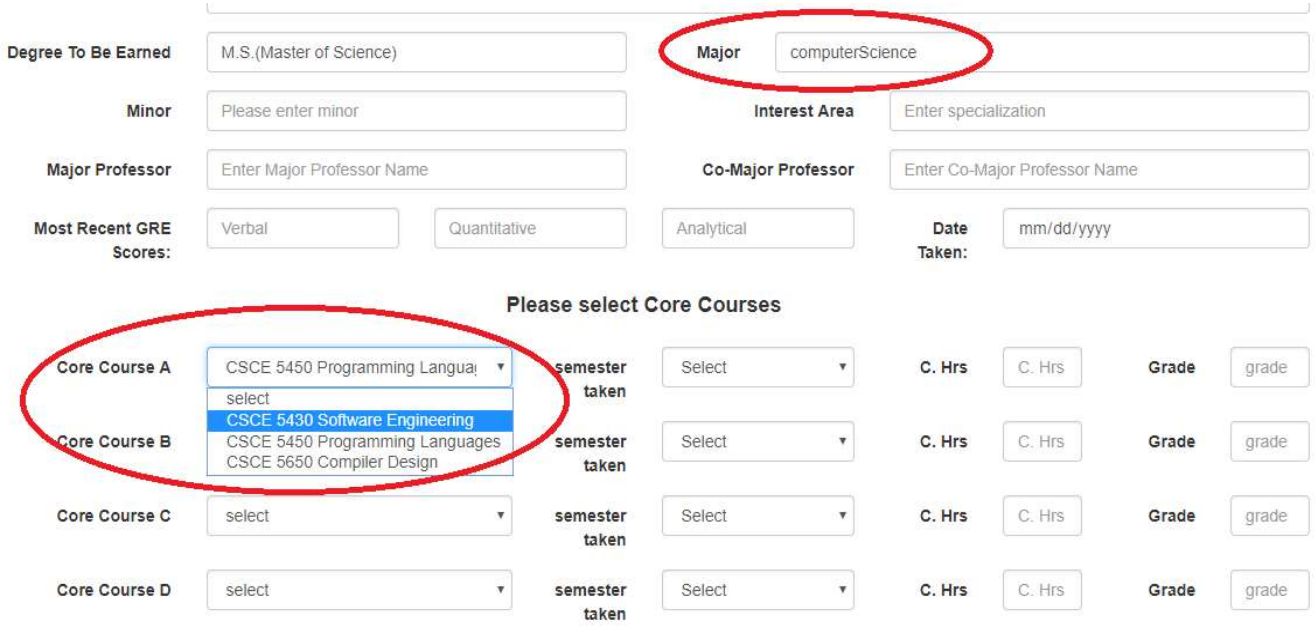


You have options to save the filled degree plan or to submit the degree plan. Submit also saves the degree plan for you and sends it to your major professor

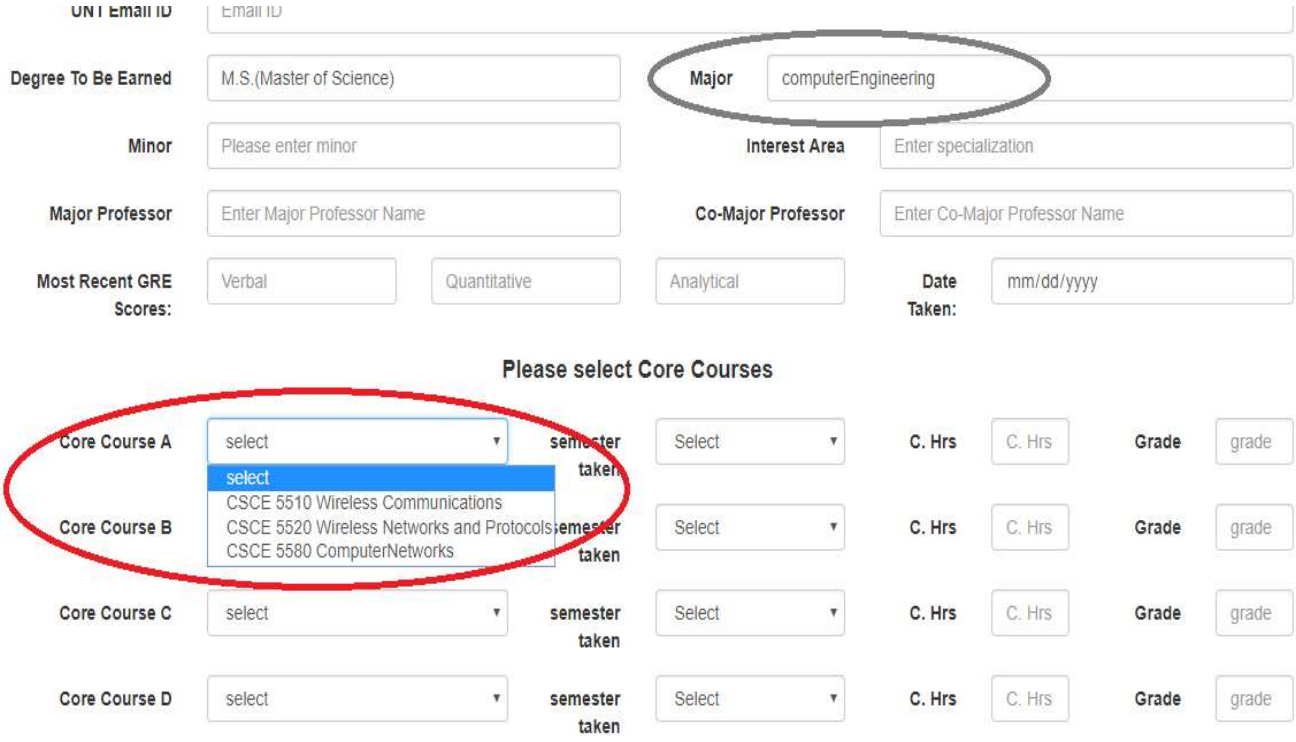


The core course options you get when you choose Computer Science are different from what you get when you choose Computer Engineering.

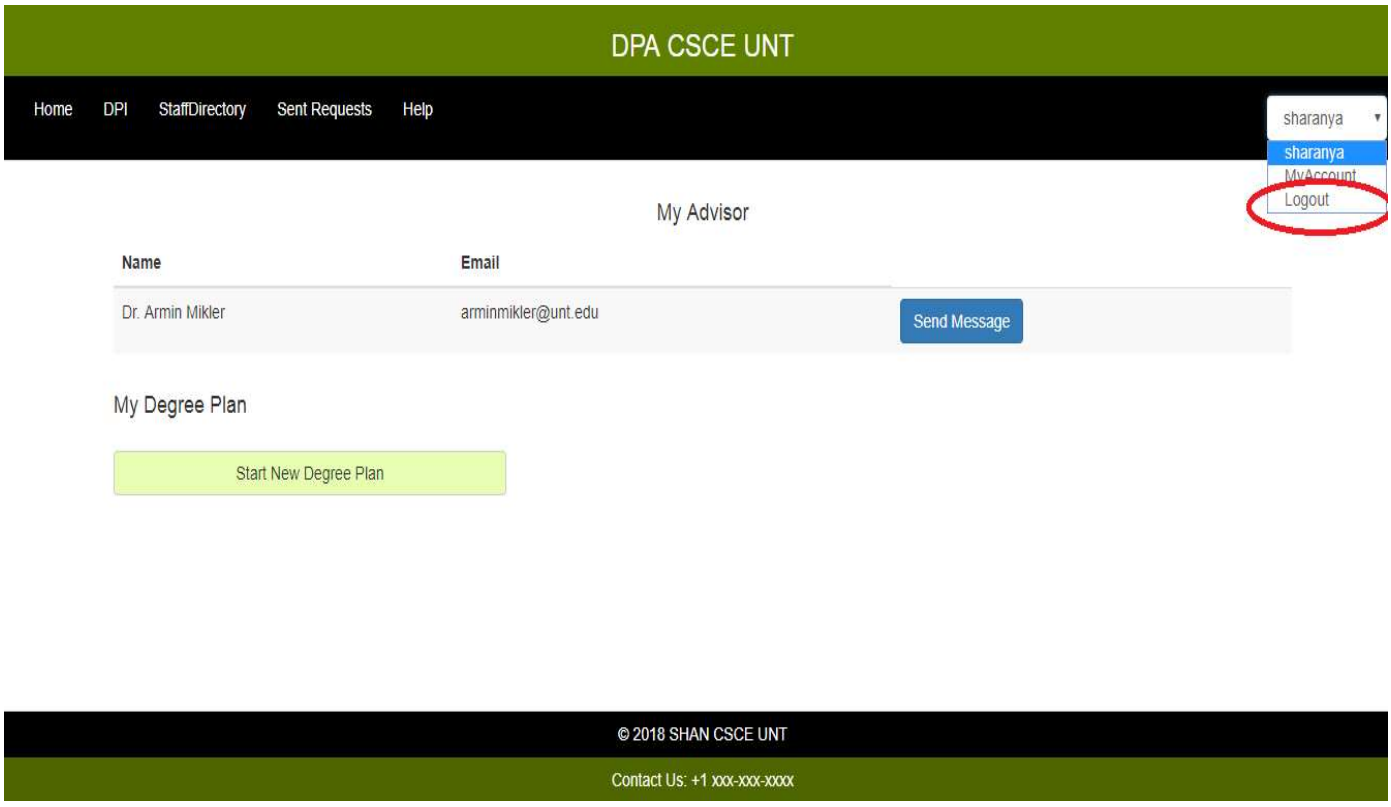
Core course list when Computer Science is chosen



Core course list when Computer Engineering is chosen

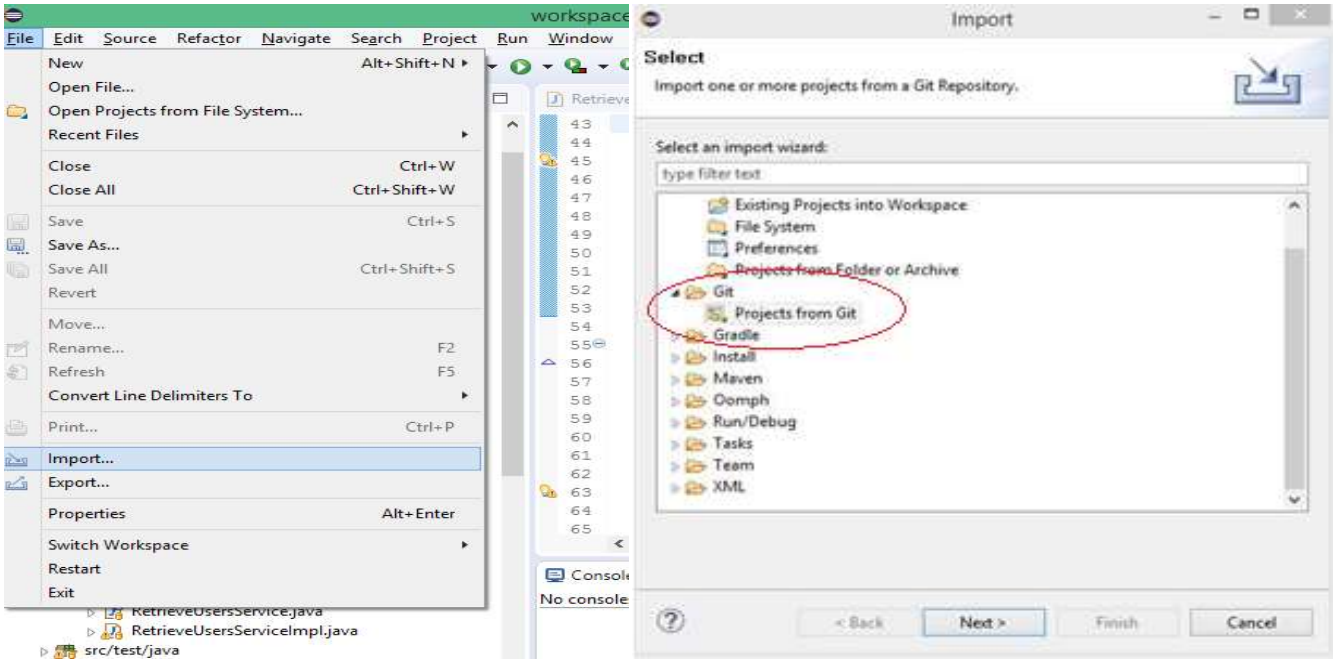


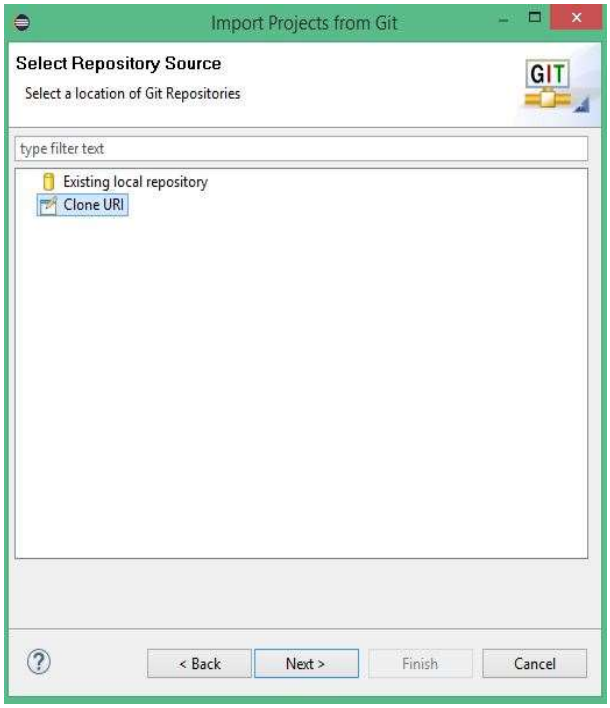
You can logout by clicking the button logout present in the dropdown with your username display.

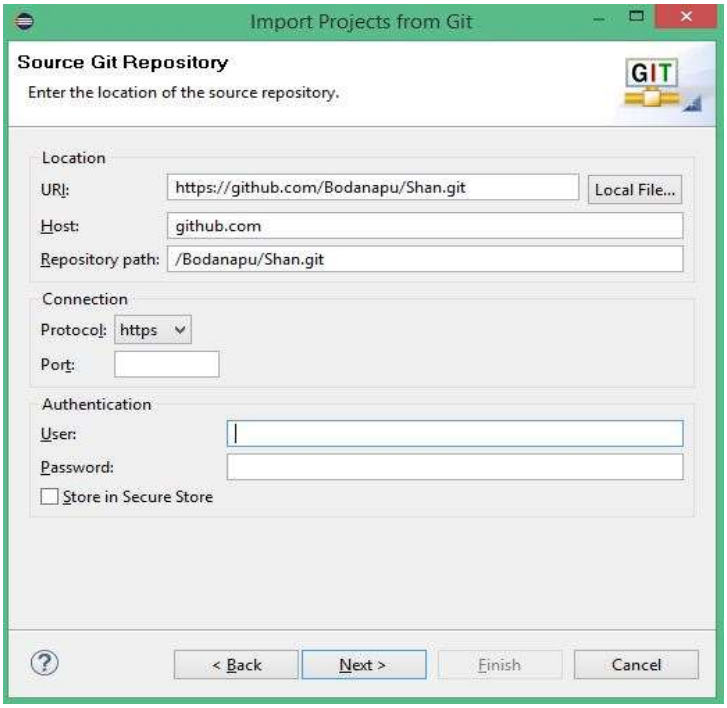


f. Instructions to compile and run both program and Test Cases

Software Required:
Eclipse
MYSQL workbench and server
Import the code in to eclipse[3]:
Open the eclipse workspace and import the code from GIT repository by clicking:
File -> import->Git->Projects from Git->clone URI



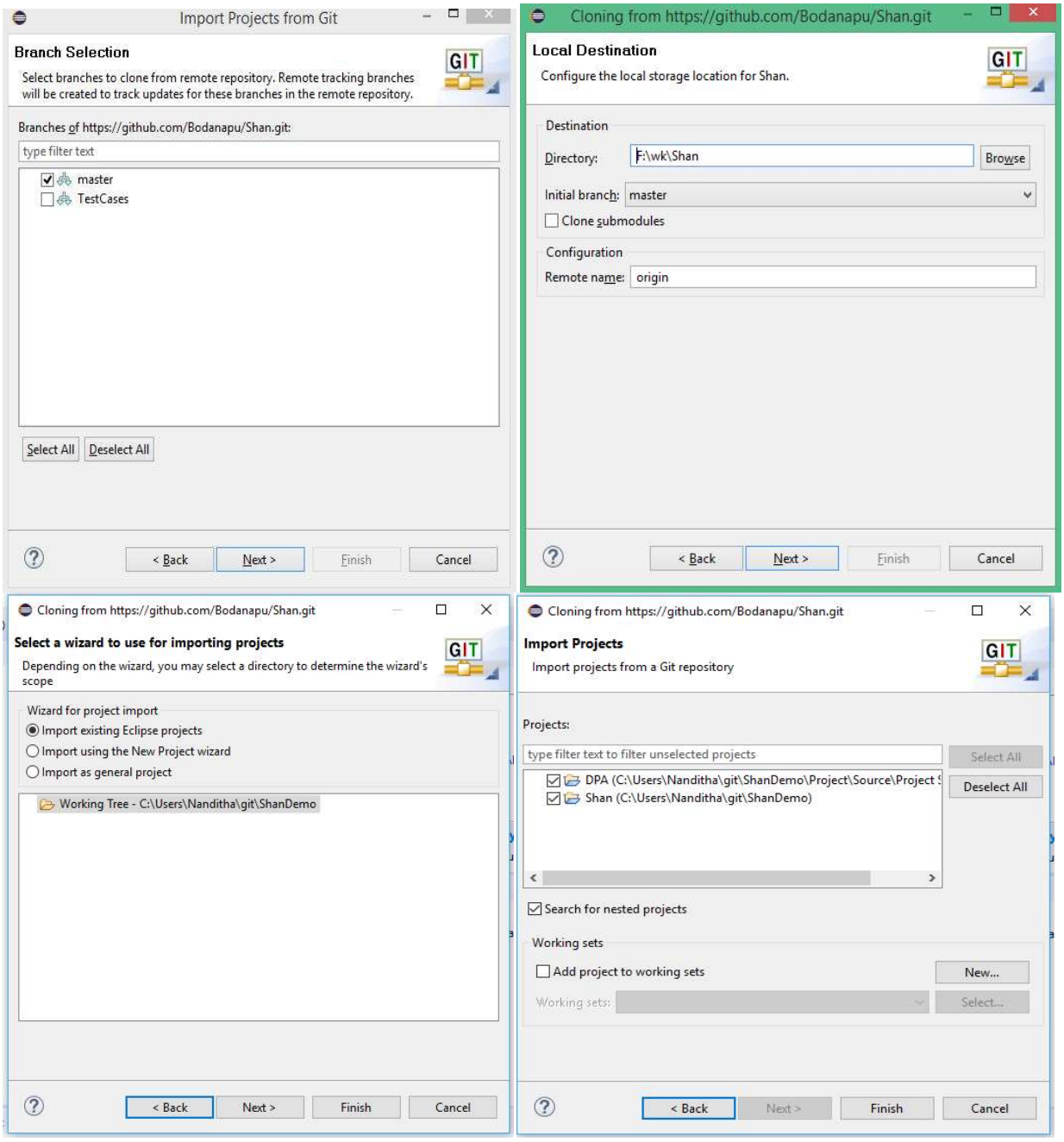




Type in the URI: https://github.com/Bodanapu/Shan.git and enter your github account details, username and password and click next. You can then select the branch "master" and click next. You will have choose a folder to store the project and import it a as import existing eclipse project and then click next, you will see two choices to clone 1. DPA 2. SHAN choose both and click fetch. You will now find the DPA maven project with the entire SHAN repository structure in your eclipse project explorer.

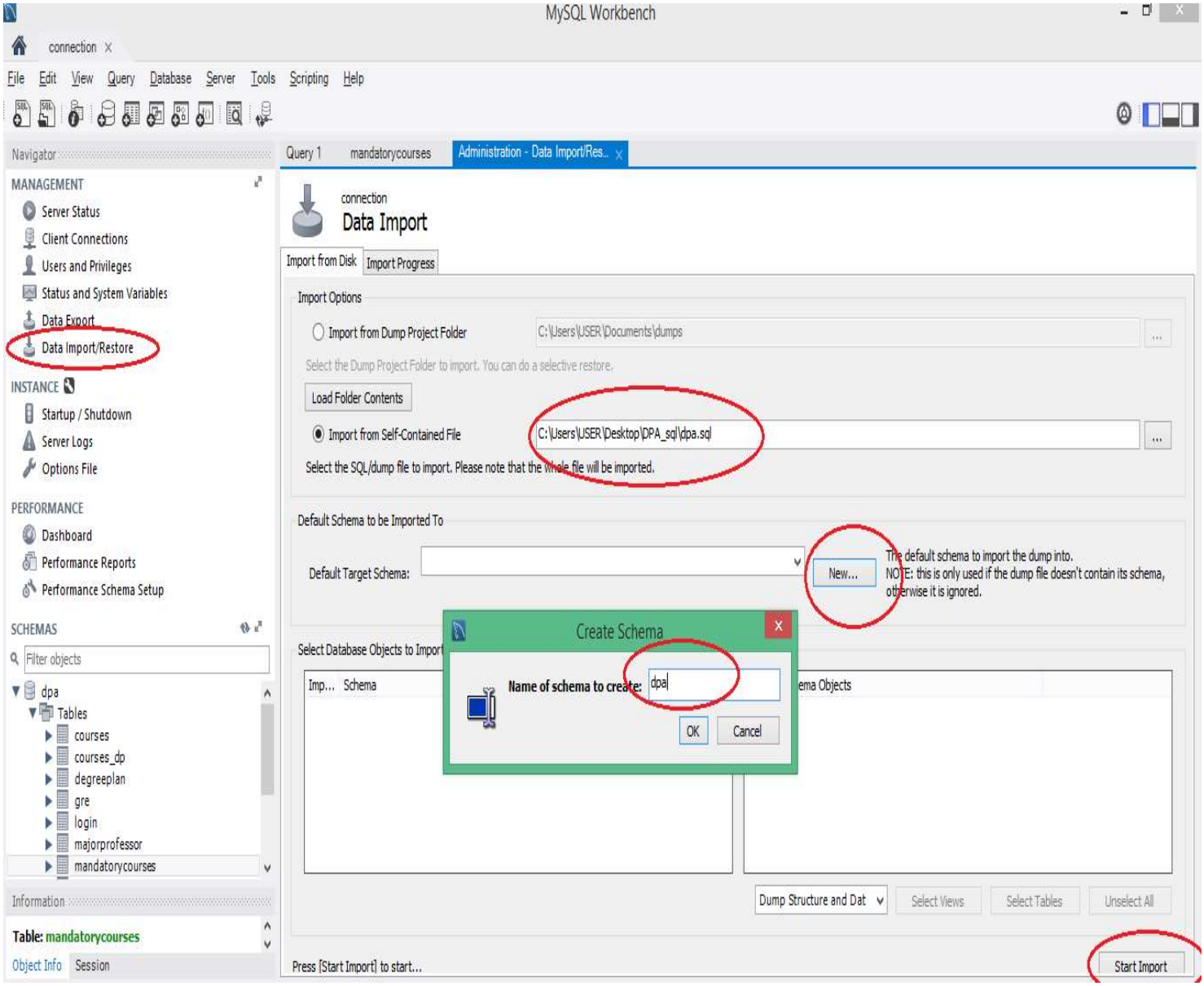
Build path to jdk 1.8:

Right click on the DPA project -> Bulid Path -> Configure Build path -> Add Library -> JRE System Library -> Alternate JRE -> browse to jdk1.8.0_152 location on your system and click finish -> apply and close.



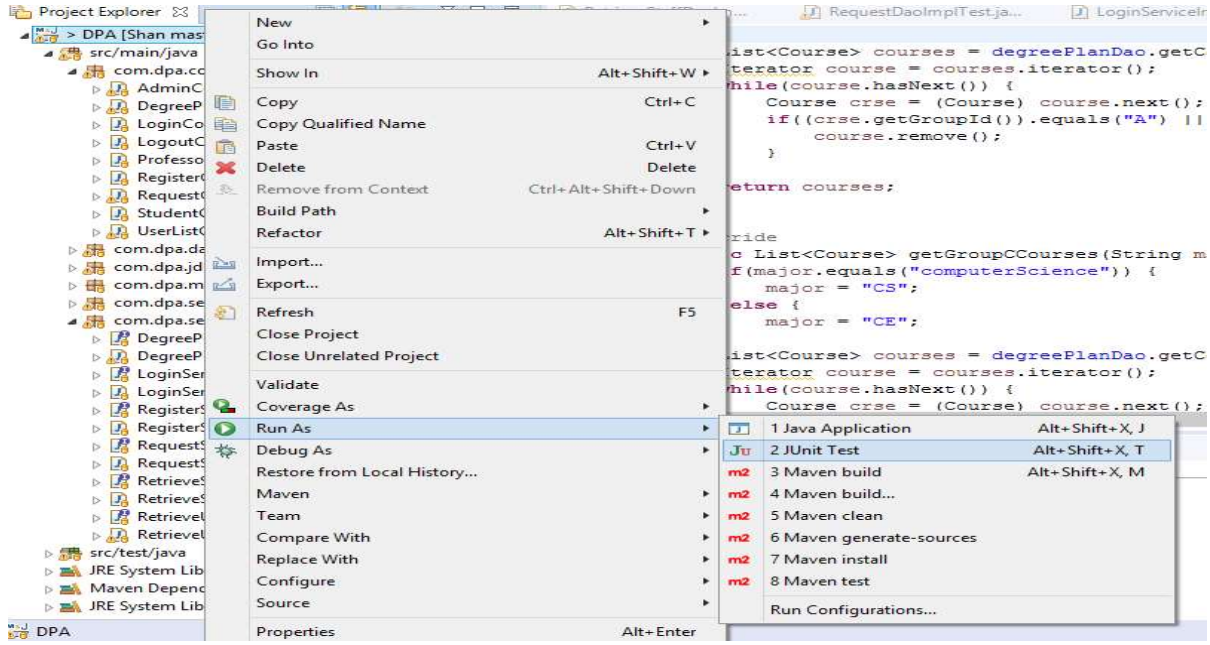
Import dpa.sql file into workbench (Database Import) [4]:

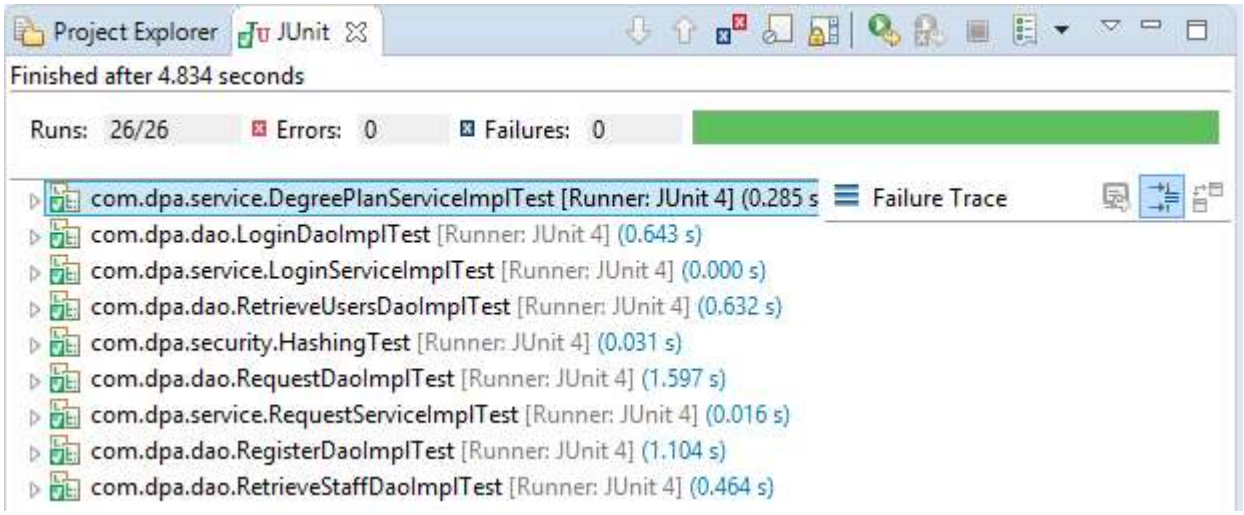
You will find a dpa.sql file in the repository path "Shan/Project/Source/ProjectSrc/Phase-I/Src/Main/Db/". Start the SQL server and open the MYSQL workbench and create a connection using username and password as root then click import and choose the self contained dpa.sql , create a new schema "dpa" and start import. After import you will have the dpa database created will all the tables in the workbench.



Procedure to Run the Test Cases:

Once you find the maven project DPA in your eclipse project explorer right click on the DPA and select the option Run As -> JunitTest, the test cases will run and show the test cases passed and failed both.





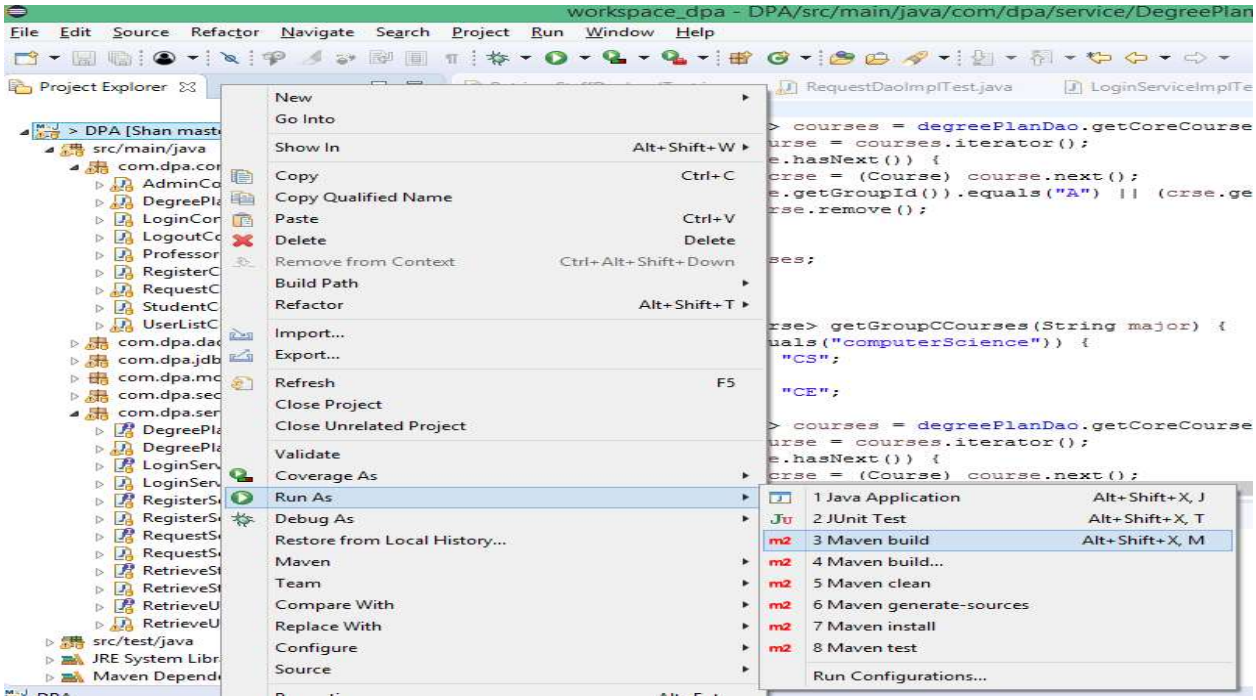
Procedure to Run the Program and check using web browser:

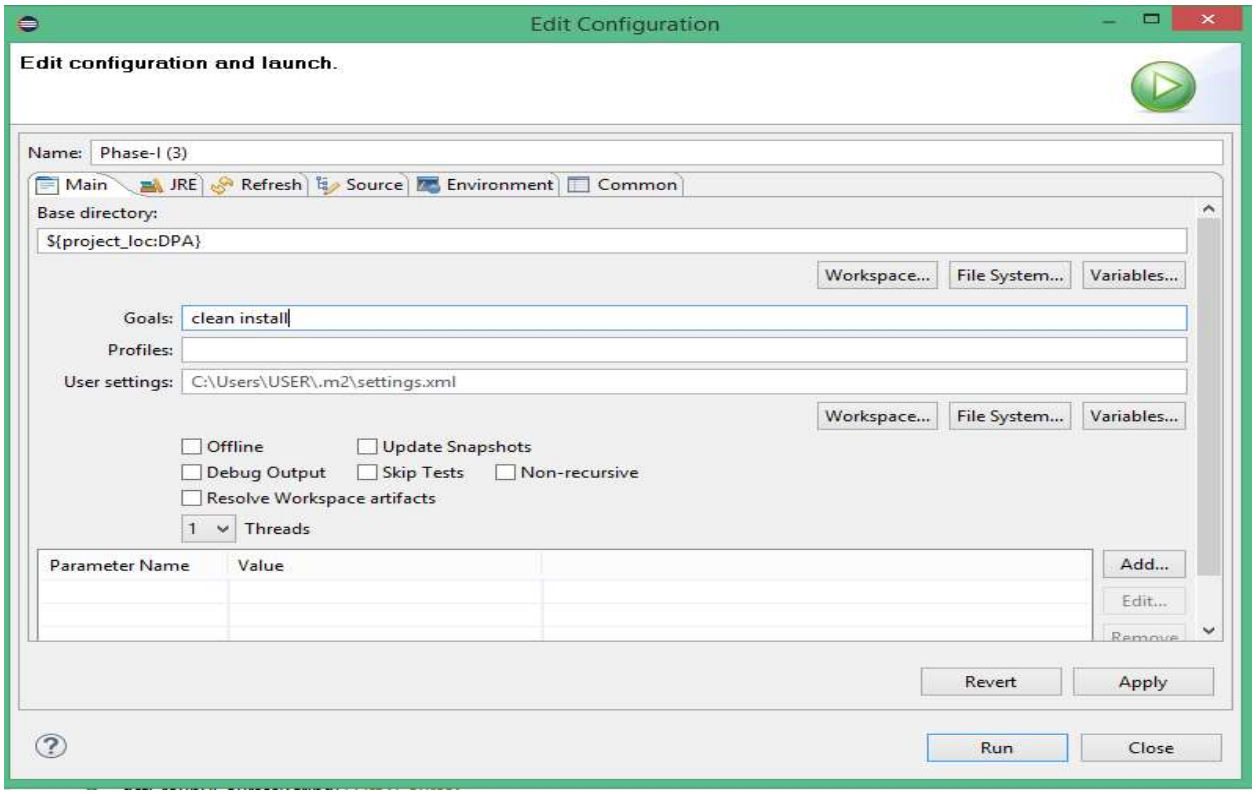
Right click on the maven project DPA in the eclipse project explorer and choose Run As -> Maven build. When you are first time running the project the Maven build asks you to set the goal. Type “clean install” in the goals field and click run.

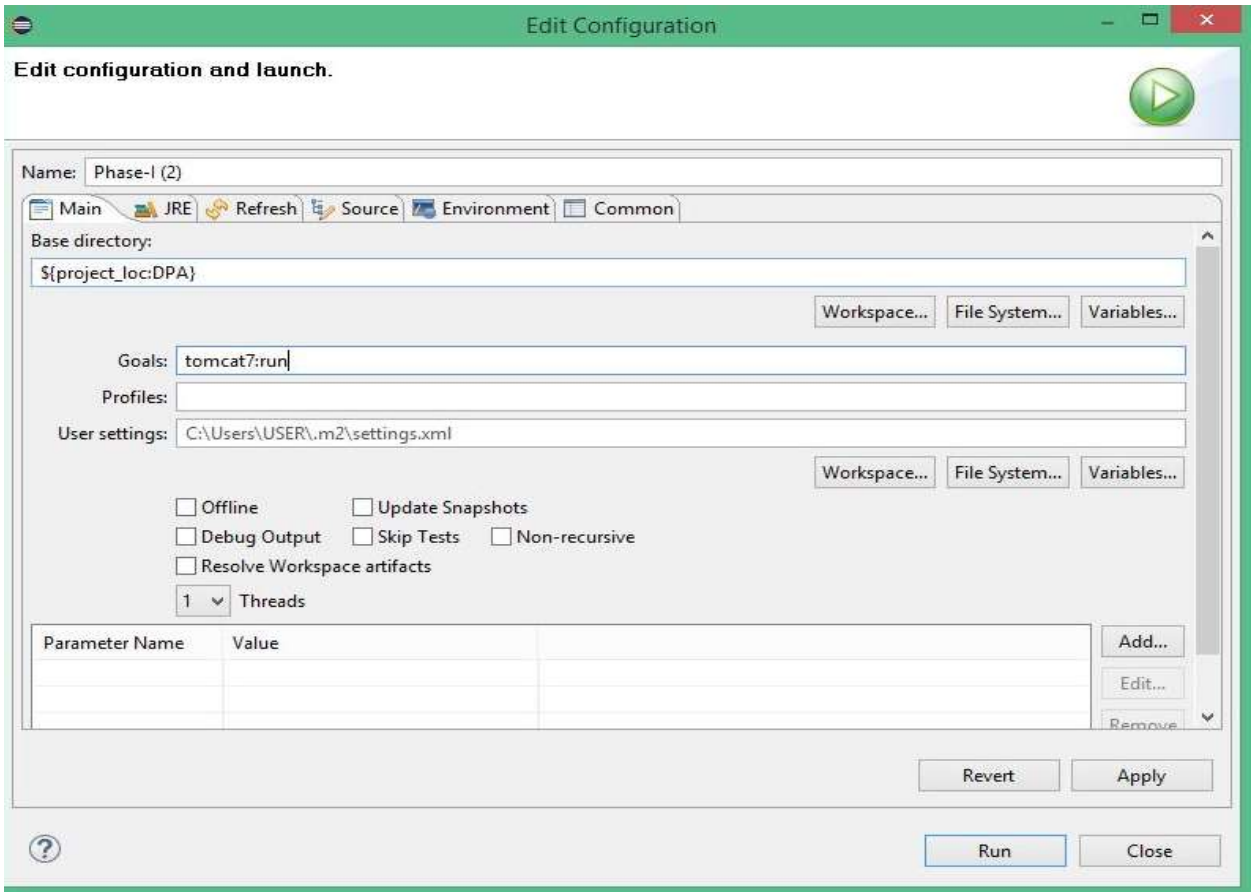
The maven will download and build all the dependencies of the project present in the “pom.xml” file.

Then click Run As -> Maven build and set the goal as “tomcat7:run” this runs the code using tomcat 7. Tomcat 7 is auto downloaded using the plugin present in the “pom.xml” file.

The project will now compile and run the code and gives a URL “localhost:8080/dpa/login” . Type URL this in the chrome browser (Google Chrome) and you can access the website.



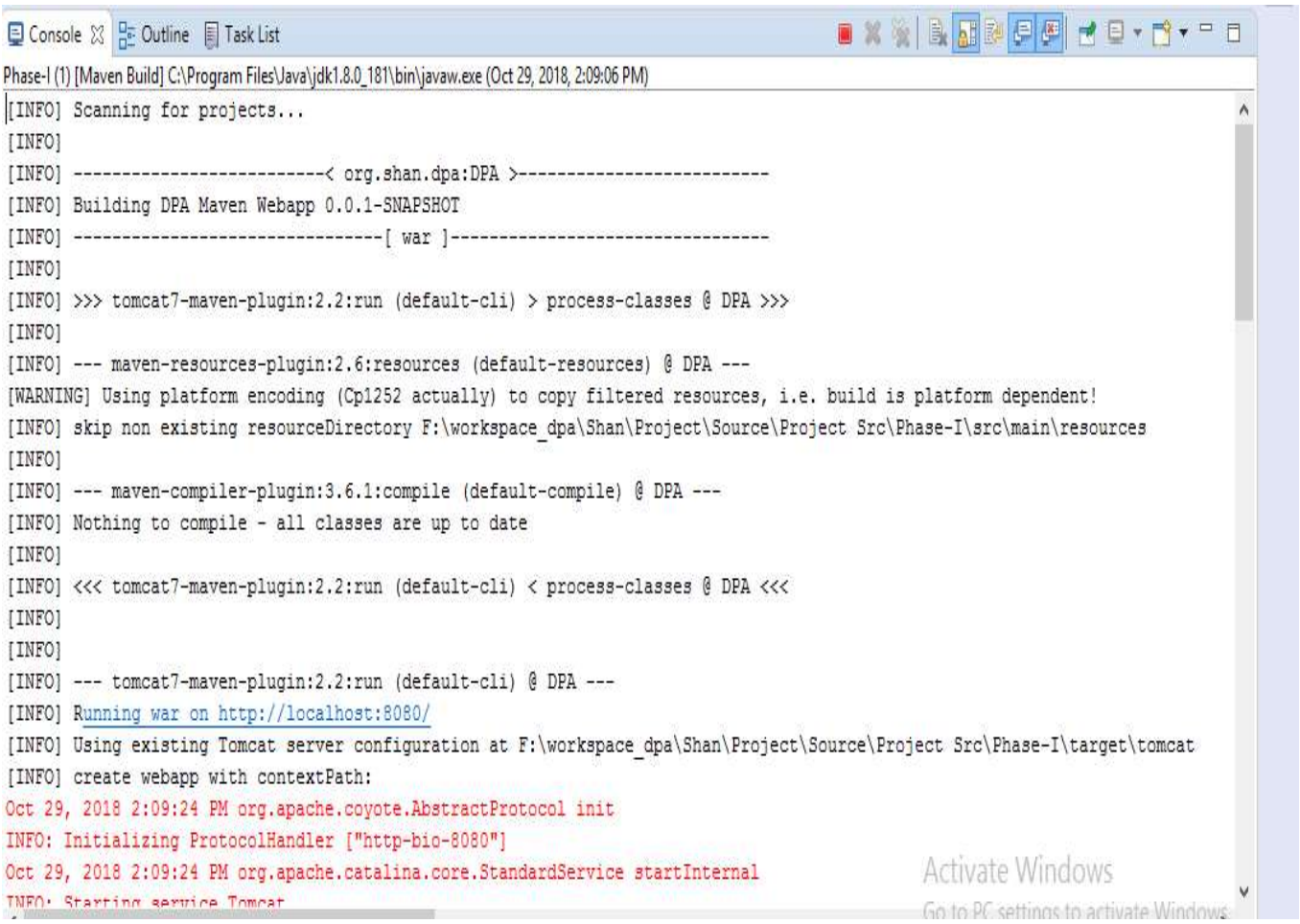





```
Results :

Tests run: 26, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ DPA ---
[INFO] Packaging webapp
[INFO] Assembling webapp [DPA] in [F:\workspace_dpa\Shan\Project\Source\Project Src\
[INFO] Processing war project
[INFO] Copying webapp resources [F:\workspace_dpa\Shan\Project\Source\Project Src\Pr
[INFO] Webapp assembled in [2348 msecs]
[INFO] Building war: F:\workspace_dpa\Shan\Project\Source\Project Src\Phase-I\target
[INFO] WEB-INF\web.xml already added, skipping
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ DPA ---
[INFO] Installing F:\workspace_dpa\Shan\Project\Source\Project Src\Phase-I\target\DP
[INFO] Installing F:\workspace_dpa\Shan\Project\Source\Project Src\Phase-I\pom.xml t
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 31.012 s
[INFO] Finished at: 2018-10-29T14:14:53-05:00
[INFO] ------------------------------------------------------------------------
```



g. Feedback Received During Peer Review Section and actions taken

During the peer review, Mad Magicians Team gave us feedback on the sample screens and the repeated requirements for login and registration modules for each user. They also spotted one of the administrative specialist's functionality that was not clear enough for them to understand.
Based on the feedback we removed the login and registration as a requirement for each user and made them unique and same single requirement for all the users by including the role field in the login and registration pages.
There was a mistake in the sample screen where student name was inserted in the admin page, we corrected it in the second version of deliverable-2
The Degree Plan Information update requirement was not clearly cited with all details administrative specialist can update. We just documented that requirement as "administrative specialist will update mandatory courses" but she can update all the course information and courses. So changed the requirement description to "administrative specialist is capable of updating all the degree plan information which includes updating all the CSCE department courses as well as the change in forms or course requirements of the department".
The above is the feedback we received and the changes we made according to the feedback.

Reference:

[1]. https://www.youtube.com/watch?v=o5k9NOR9lrI
[2]. https://www.youtube.com/watch?v=d2KwvXQgQx4
[3]. https://stackoverflow.com/questions/6760115/importing-a-github-project-into-eclipse
[4]. https://www.linode.com/docs/databases/mysql/deploy-mysql-workbench-for-database-administration/