Core Functionalities

Team Name: Dash

Project: Student Surplus

Functionalities:

1. **Registration**
    - Takes the user details, such as first name, last name, email, phone number and their address and stores the details into the database.
    - Throws an error if the email already exists and any required field is let null.
    - After successful completion redirects to profile details page.

user-registration-form.jsp:

```
<title>User Registration</title>


<link type="text/css" rel="stylesheet"
      href="${pageContext.request.contextPath}/resources/css/bootstrap.min.css">


</head>

<body>

      <div class="container col-md-6">
            <h3 style="color: blue;">New User Sign Up</h3>
            <hr />
      </div>
      <div class="container col-md-6">
            <form action="registration" method="POST" name="registration"
                  class="needs-validation" novalidate="novalidate"
                  enctype="multipart/form-data">
                  <div class="form-row">
                        <div class="col-md-4 mb-3">
                              <label for="firstName">First name</label>
                              <spring:bind path="user.first_name">
                                    <input type="text" name="${status.expression}"
                                          value="${status.value}" class="form-control"
id="firstName"
                                          placeholder="First Name" required="required" />
                                    <div class="valid-feedback">Looks good!</div>
                              </spring:bind>
                        </div>
```

RegistrationController.java:

- **Takes the values from the model objects and sends it to the service and model classes**

```
	@RequestMapping(value = "/success")
	public String success(@ModelAttribute("user") User user,
@ModelAttribute("logincred") LoginCredential login,
			ModelMap model) {
		User getUser = registrationService.getUser(3);
		LoginCredential getLogin = registrationService.getLoginId(1);
		Address getAddress = registrationService.getAddress(1);
		model.addAttribute("user", getUser);
		model.addAttribute("logincred", getLogin);
		model.addAttribute("address", getAddress);
		return "success";
	}

	@RequestMapping(value = "/registration", method = RequestMethod.POST)
	public String submit(@ModelAttribute("user") User user,
@ModelAttribute("logincred") LoginCredential login,
			@ModelAttribute("address") Address address, HttpServletRequest request)
{

		boolean result = registrationService.add(user);
		if (result == false) {
			request.setAttribute("Msg", "Email already exists!");
			return "user-registration-form";
		} else if (result == true) {
			login.setLogin_id(user.getEmail());
			login.setUid(user.getUser_id());
			registrationService.addLogin_Cred(login);

			address.setUser_id(user.getUser_id());
			registrationService.addAddress(address);

			// saving profile picture
			if (user.getPicture() != null && !user.getPicture().isEmpty()) {
				fileUploadHelper.uploadPicture(request, user.getPicture(),
user.getUser_id());
				return "user-profile";
			}
		}
		return "user-profile";
	}
```

RegistartionServiceImpl.java:

- Is used for both registration and for updating the details of the user after registration takes the input from controller classes and sends it to the Dao classes

```
@Autowired
private RegistrationDAO registrationDAO;

@Transactional
public boolean addLogin_Cred(LoginCredential login) {
	registrationDAO.addLogin_Cred(login);
	return true;
}

@Transactional
public void editLogin_Cred(LoginCredential login) {
	registrationDAO.editLogin_Cred(login);

}

@Transactional
public void deleteLogin_Cred(int credential_id) {
	registrationDAO.deleteLogin_Cred(credential_id);

}

@Transactional
public LoginCredential getLoginId(int credential_id) {
	// TODO Auto-generated method stub
	return registrationDAO.getLoginId(credential_id);
}


@Transactional
public boolean add(User user) {
	boolean result = registrationDAO.add(user);
	return result;

	}

@Transactional
public boolean addAddress(Address address) {
	registrationDAO.addAddress(address);
	return true;
}

@Transactional
public Address editAddress(Address address) {
	return registrationDAO.editAddress(address);

	}
```

RegistrationDaoImpl.java:

- Consists of the actual database queries that are used to store and add, delete and retrieve the data from the database, also returns a value to the service classes to verify the query execution status.

```
	@Override
	public User getUser(int user_id) {

		return (User) session.getCurrentSession().get(User.class, user_id);

	}

	@Override
	public User getByEmail(String email) {
		Query query = session.getCurrentSession().createQuery("from User where email= :email");
		query.setParameter("email", email);
		return (User) query.getSingleResult();

	}

	@Override
	public boolean addLogin_Cred(LoginCredential login) {
		String hashedPassword = DigestUtils.sha256Hex(login.getPassword());
		login.setPassword(hashedPassword);
		session.getCurrentSession().save(login);
		return true;
	}

	@Override
	public void editLogin_Cred(LoginCredential login) {
		session.getCurrentSession().update(login);

	}

	@Override
	public void deleteLogin_Cred(int credential_id) {
		session.getCurrentSession().delete(getLoginId(credential_id));

	}
```

2. Login

- Takes the username and password from the user.
- Verifies them by checking them if they're valid by getting the values from the database.
- Encrypts the user given password into SHA 256 to compare it with the encrypted value that's in the database.
- Returns an error if validation fails.
- Redirects to home page if validation is successful.

LoginController.java:

- Takes the input from the login page and forwards it to the login service, and gets a result back to check whether the login validation was successful

```
@Autowired
	private LoginService loginService;

	@RequestMapping(value = "/home", method = RequestMethod.GET)
	public ModelAndView showHome(ModelAndView model) {
		model.setViewName("home");
		return model;
	}

	@RequestMapping(value = "/login", method = RequestMethod.GET)
	public ModelAndView login(ModelAndView model) {
		LoginCredential lc = new LoginCredential();
		model.addObject("login",lc);
		model.setViewName("login");
		return model;
	}
	@RequestMapping(value = "/login_do", method = RequestMethod.POST)
	public ModelAndView submit(@ModelAttribute("login") LoginCredential login,
ModelAndView model, HttpSession session) {
		Boolean t =  loginService.findUser(login.getLogin_id(), login.getPassword());
		if (t) {
			session.setAttribute("login_id", login.getLogin_id());
			session.setMaxInactiveInterval(30000);
			model.setViewName("redirect:/home");
		}
		else {
			session.setAttribute("errorMessage", "Invalid Username or Password");
			model.setViewName("redirect:/login");
		}

		return model;
	}
```

LoginServiceImpl.java

- Implements the login service and encrypts the data entered by the user into SHA256 before comparing it with the values in the database, since we are encrypting the values before storing it into the database.

```
@Autowired
SessionFactory sessionFactory;

@Autowired
LoginDAO loginDao;

public LoginServiceImpl() {
	loginDao = new LoginDAOImpl();
}

@Transactional
@Override
public boolean findUser(String email, String Password) {

	LoginCredential lc = loginDao.getLoginCredential(email);

	if (lc.getCid() == -1) {
		System.out.println("no result");
		return false;
	}
	else {
		Password = DigestUtils.sha256Hex(Password);
		if (Password.equals(lc.getPassword())) {
			return true;
		}
		else {
			System.out.println("this is where it goes wrong");
			return false;
		}
	}

}
```

LoginDaoImpl.java

- Gets the values from the database and stores it into the model class to retrieve it later in the service class.
- Returns true or false to validate the details in the service class.

```
@Repository
public class LoginDAOImpl implements LoginDAO{

	@Autowired
	private SessionFactory sessionFactory;

	@Override
	public LoginCredential getLoginCredential(String email) {
		try {
			@SuppressWarnings("rawtypes")
			Query query=sessionFactory.getCurrentSession().createQuery("from
LoginCredential where login_id='" + email + "'");
			/*System.out.println(((query.getResultList().get(0))).toString());
			System.out.println(((query.getSingleResult()))).toString());*/
			return (LoginCredential) query.getSingleResult();
		} catch (Exception e) {
			LoginCredential l = new LoginCredential();
			l.setCid(-1);
			return l;
		}
```

Login.jsp

- Provides the front-end view of the login page to the user and returns model objects to the controller class.

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" />
<link type="text/css" rel="stylesheet"
      href="${pageContext.request.contextPath}/resources/css/global.css">
<script
      src="https://stackpath.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
      type="text/javascript"></script>
<script
      src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"
      type="text/javascript"></script>

<title>Login Page</title>

</head>
<style>
</style>
<body class="container login-container">
      <form:form method="POST" modelAttribute="login" action="login_do">
            <div class="loginpageform">
                  <p class="error" id="error"></p>
                  <h3 class="txtcnt">Sign in to your account</h3>
                  <br />
                  <form:input type="text" class="form-control logininputfield"
                        path="login_id" placeholder="Email" />
                  <br />
                  <form:input type="password" class="form-control logininputfield"
                        id="pwd" path="password" placeholder="Password" />
                  <br /> <a href="/unt.cse.studentsurplus/registration" class="rgtbtn">
                        Register Here </a>
                  <button class="btn btn-success signin" value="submit"
type="submit">Sign
                        In</button>
            </div>
            <div>
          <h3 style="color:red;">${errorMessage}</h3>
            </div>
      </form:form>
      </body>
<script>
</script>
</html>
```

3. Profile Update:

- Gets the data from the database using the session details after the user is logged into his/her account.
- Provides a feature where user can update his/her details.
- After getting the values the database is updated with those values.
- Profile page is re updated with current user details.

RegistrationController.java

- It also takes the model objects from the jsp page and makes class and method calls to update the data of the user in the database, the below are the controller methods for different jsp files that are used for updating user details.

```
@RequestMapping("/myProfile")
	public String showUserProfile(@ModelAttribute("address") Address address,
@ModelAttribute("user") User user,
			ModelMap model, HttpSession session) {

		String email = (String) session.getAttribute("login_id");
		System.out.println(email);
		user = registrationService.getByEmail(email);
		address = registrationService.getAddressofUser(user.getUser_id());
		model.addAttribute("address", address);
		model.addAttribute("user", user);
		return "user-profile";
	}

	@PostMapping("/updateUserInfo")
	public String callUpdateUserInfoForm(@ModelAttribute("user") User user1, Model
model, HttpSession session) {
		String email = (String) session.getAttribute("login_id");
		User user = registrationService.getByEmail(email);
		user1.setUser_id(user.getUser_id());
		user1 = registrationService.edit(user1);
		Address address = registrationService.getAddressofUser(user1.getUser_id());
		model.addAttribute("address", address);
		model.addAttribute("user", user);
		return "user-profile";
	}

	@PostMapping("/updateAddress")
	public String callUpdateAddressInfoForm(@ModelAttribute("user") User user,
			@ModelAttribute("address") Address address, Model model, HttpSession
session) {
		String email = (String) session.getAttribute("login_id");
		user = registrationService.getByEmail(email);
		address.setUser_id(user.getUser_id());
		address = registrationService.editAddress(address);
		model.addAttribute("address", address);
		model.addAttribute("user", user);
		return "user-profile";
	}
```

RegistrationDaoImpl:

- This class updates the address details of the user using the database queries and return a result whether the database transaction was successful or not.

```
	@Override
	public Address editAddress(Address address) {
		Query query = session.getCurrentSession().createQuery(
				"UPDATE Address SET street=:street, apartment= :apartment, city=
:city, zip_code = :zipcode, state= :state  WHERE user_id= :user_id");
		query.setParameter("street", address.getStreet());
		query.setParameter("apartment", address.getApartment());
		query.setParameter("city", address.getCity());
		query.setParameter("zipcode", address.getzip_code());
		query.setParameter("state", address.getState());
		query.setParameter("user_id", address.getUser_id());
		try {
			int result = query.executeUpdate();
		} catch (Exception e) {
			System.out.println("hello");
		}
		return address;

	}

	@Override
	public void deleteAddress(int address_id) {
		session.getCurrentSession().delete(getLoginId(address_id));

	}

	@Override
	public Address getAddress(int address_id) {
		return (Address) session.getCurrentSession().get(Address.class, address_id);
	}

	@Override
	public void getLogin_id(String email, String Password) {

	}

	@Override
	public Address getAddressofUser(int userId) {

		Session currentSession = session.getCurrentSession();
		Query qry = currentSession.createQuery("from Address where user_id=:userId");
		qry.setParameter("userId", userId);
		return (Address) qry.getSingleResult();

	}
```

User.java:

- A model class to store the objects and then retrieve them efficiently.

```
private MultipartFile picture;
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name="user_id", nullable = false)
	public int getUser_id() {
		return user_id;
	}

	public void setUser_id(int user_id) {
		this.user_id = user_id;
	}

	public String getFirst_name() {
		return first_name;
	}

	public void setFirst_name(String first_name) {
		this.first_name = first_name;
	}

	public String getMiddle_name() {
		return middle_name;
	}

	public void setMiddle_name(String middle_name) {
		this.middle_name = middle_name;
	}

	public String getLast_name() {
		return last_name;
	}
```

RegistrationControllerTest.java

- Tests the code using Unit Testing.

```
    void happyPath_when() {
          when(registrationService.add(any(User.class))).thenReturn(true);

    when(registrationService.addLogin_Cred(any(LoginCredential.class))).thenReturn(true
);
          when(registrationService.addAddress(any(Address.class))).thenReturn(true);
    }

    void happyPath_verify() {
          verify(registrationService).add(any(User.class));
          verify(registrationService).addLogin_Cred(any(LoginCredential.class));
          verify(registrationService).addAddress(any(Address.class));
    }

    @Test
    public void testSubmit_happyPath() {
          user.setPicture(null);

          happyPath_when();

          String result = underTest.submit(user, login, address, request, session);

          happyPath_verify();

          assertEquals("user-profile", result);
    }

    @Test
    public void testSubmit_pictureIsEmpty() {
          user.setPicture(picture);

          happyPath_when();
          when(picture.isEmpty()).thenReturn(true);

          String result = underTest.submit(user, login, address, request, session);

          happyPath_verify();
          verify(picture).isEmpty();

          assertEquals("user-profile", result);
    }
```