****

**Faculty of Computer Science**

**Dalhousie University**

**CSCI 5308**

**Quality Assurance**

**Project Report**

**FriendBook**

**Submitted By: (Group 1)**

**Suman Singh - B00797215 (sm644084@dal.ca)**

**Sudarshan Suresh Srikant - B00808452 (sd312736@dal.ca)**

**Yash Desai - B00810560 (**[**ys273139@dal.ca**](mailto:ys273139@dal.ca)**)**

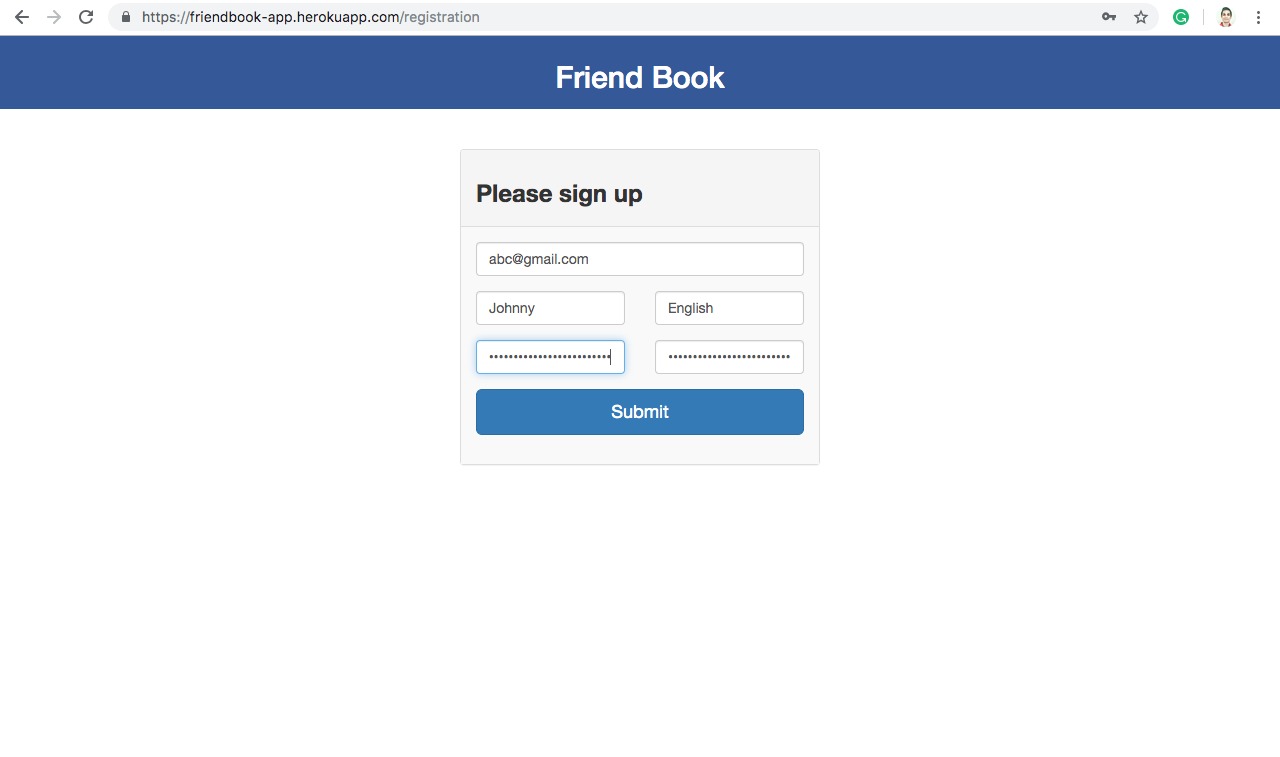
**Introduction**

FriendBook is a social networking platform where people all around the world can connect with each other. Our primary goal is to create a bridge between people and help them connect. The application contains features such as sending a friend request, posting text statuses, searching friends/users by applying different filters, comment on posts created by our friends. New users can also register on our website by clicking on the “Sign Up” button.

**Features of Friendbook:**

* **Registration**

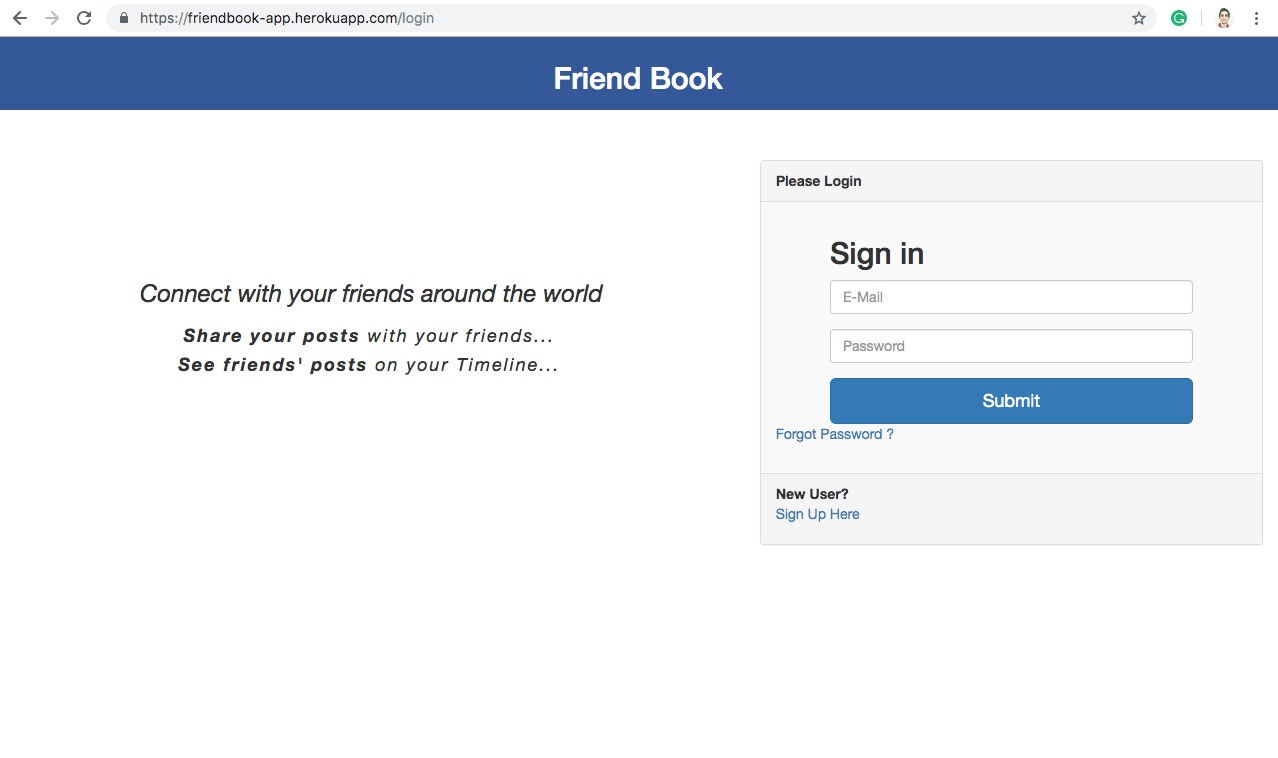
In order to use FriendBook, a user has to sign up first. They are supposed to provide basic contact information such as the first name, last name, email and password as shown in figure (1).



**Figure 1 Registration Page**

* **Login**

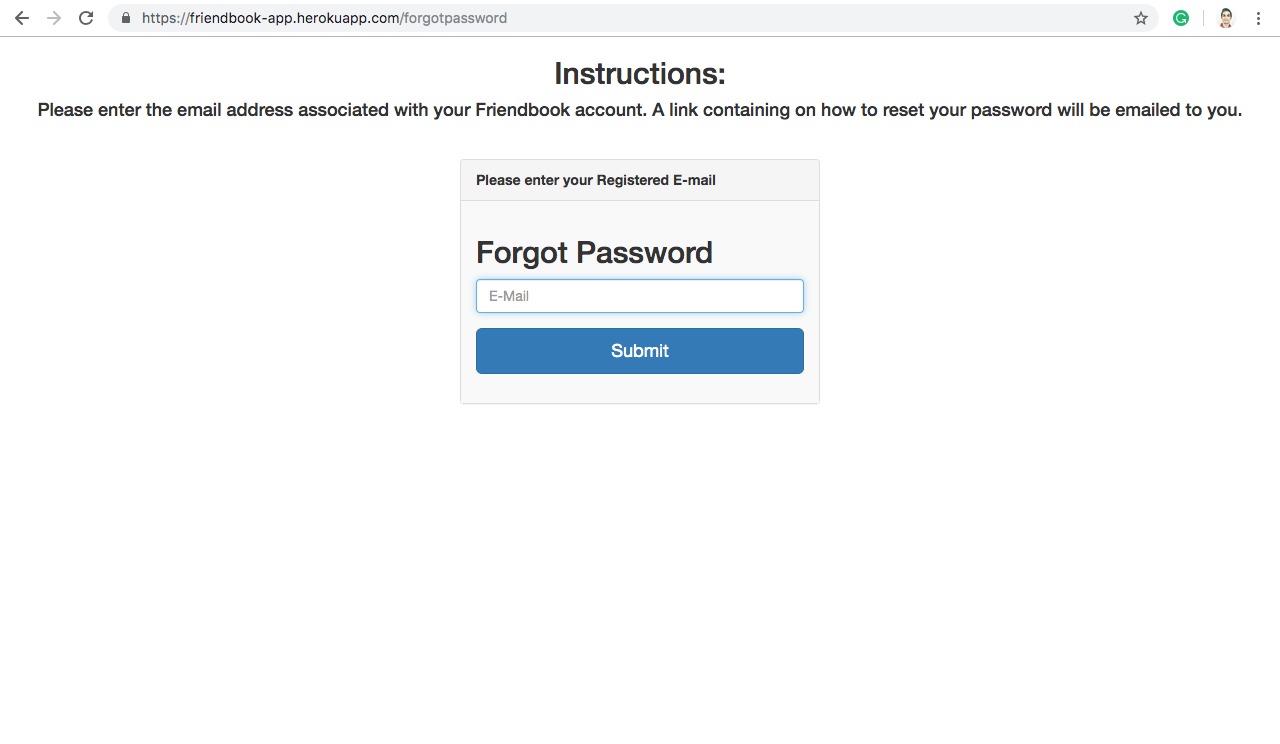
After registration, the user has to log into his/her account by providing valid login credentials as shown in figure (2).



**Figure 2 Login Page**

* **Forgot Password**

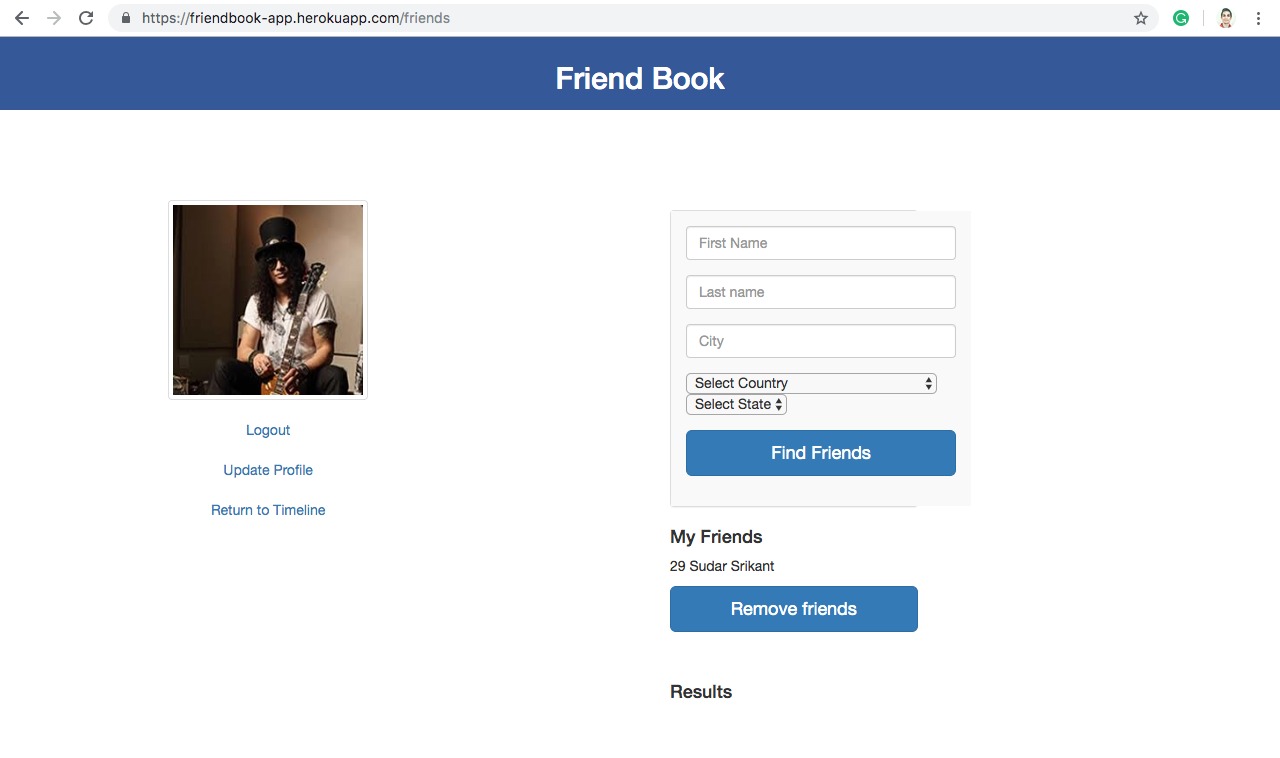
Forgot Password will let the user recover their password. We are sending a forgot password link to their respective email address.



**Figure 3 Forgot Password Page**

* **Friends**

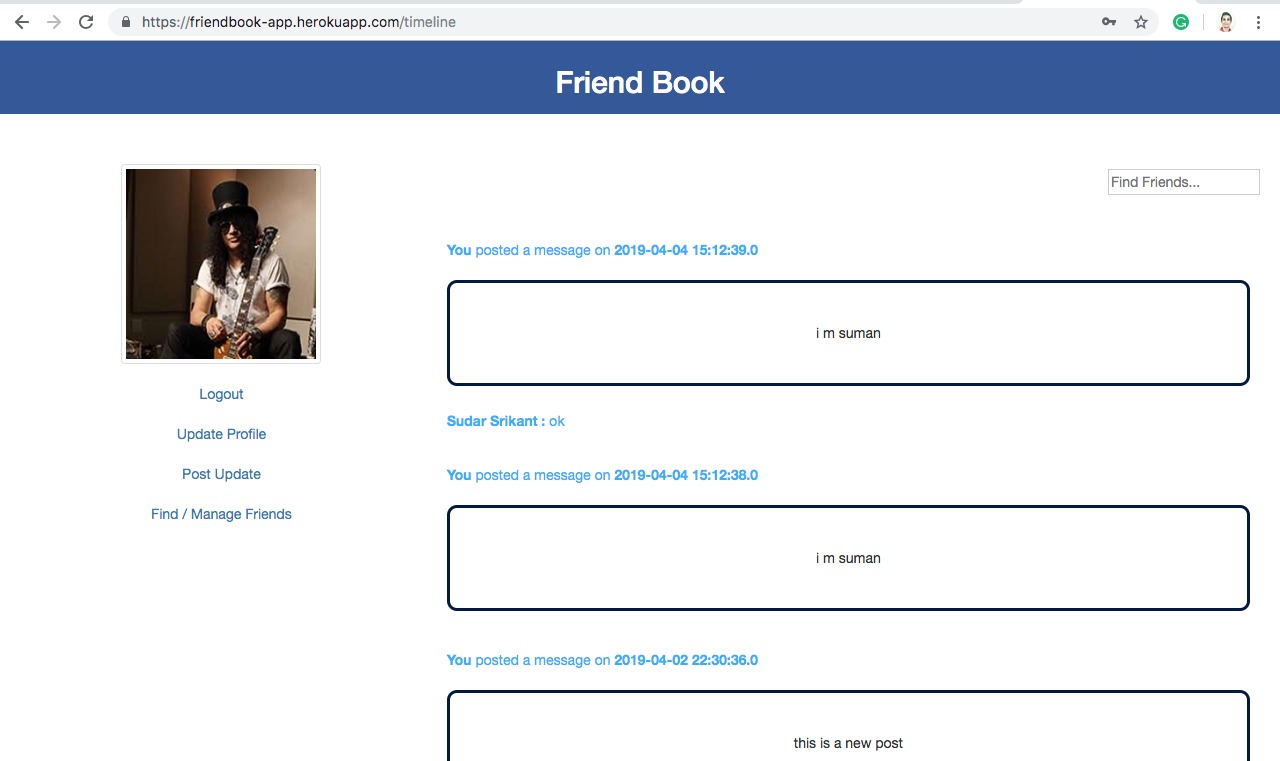
This feature will allow users to find other FriendBook users by providing different search filters such as search by first name, last name, city, etc.



**Figure 4 Find Friends Page**

* **Timeline**

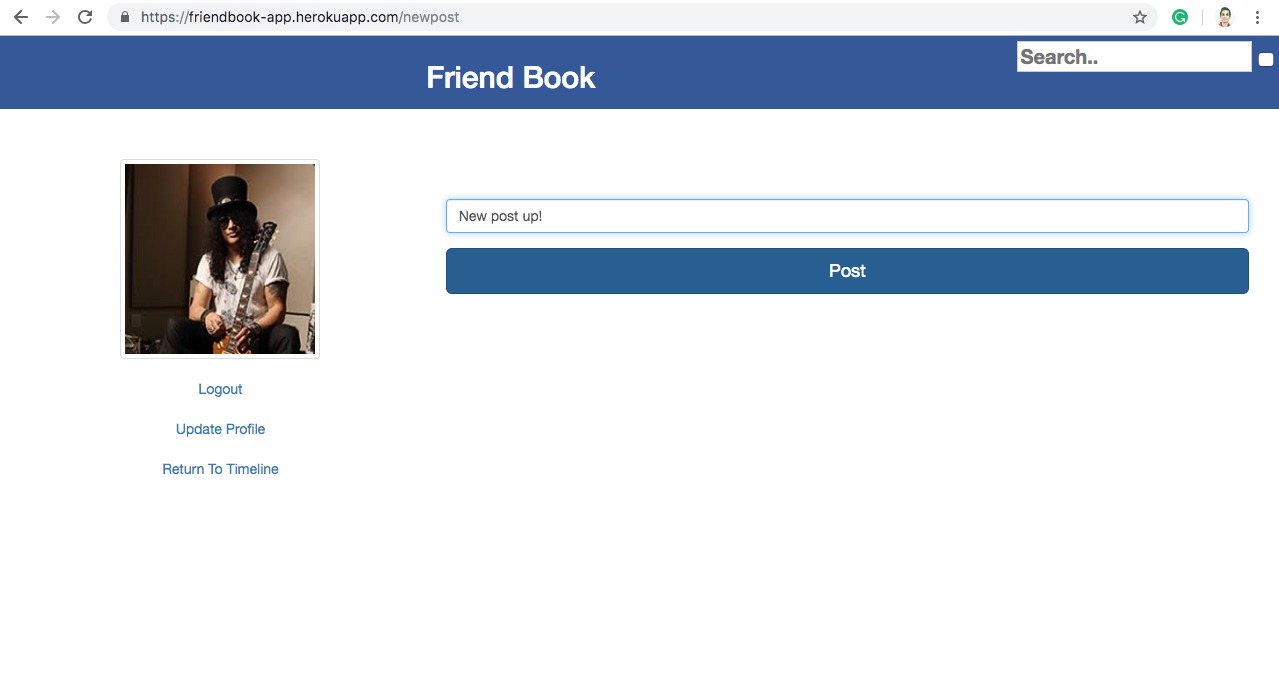
After setting up the account, the user is directed to the timeline page. In the timeline page, a user can see his/her friend’s posts and make a comment on that post. Moreover, the user themselves can post on their timeline and this post would be reflected in their friends’ timeline.



**Figure 5 Timeline Page**

* **New Post**

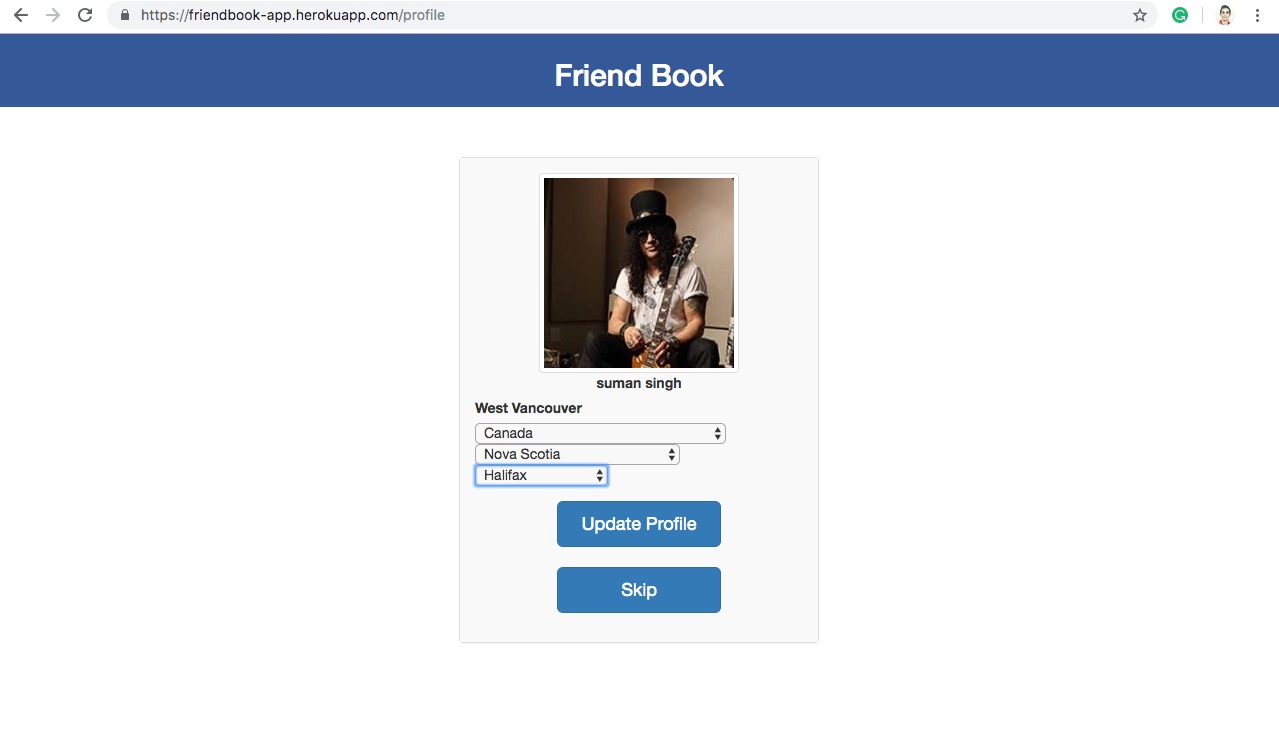
This feature will let a user to post a text status on his/her timeline and let his/her friends know that the user has posted a text status by displaying the post on their respective timelines.



**Figure 6 New Post Page**

* **Profile**

The profile feature lets users to update their profile picture, change city, province or country.



**Figure 7 Profile Page**

**Technologies and Tools Used**

**Server Side Technology:**

JAVA

**Framework:**

Spring Boot

**Client Side Technology:**

Hyper Text Markup Language (HTML)

Cascading Style Sheet (CSS)

Java Server Pages (JSP)

**Database:**

MySQL

**Continuous Integration Tool:**

Jenkins

**Build Automation Tool:**

Maven

**Version Control:**

Github

**Ticket Controlling Tool:**

Trello

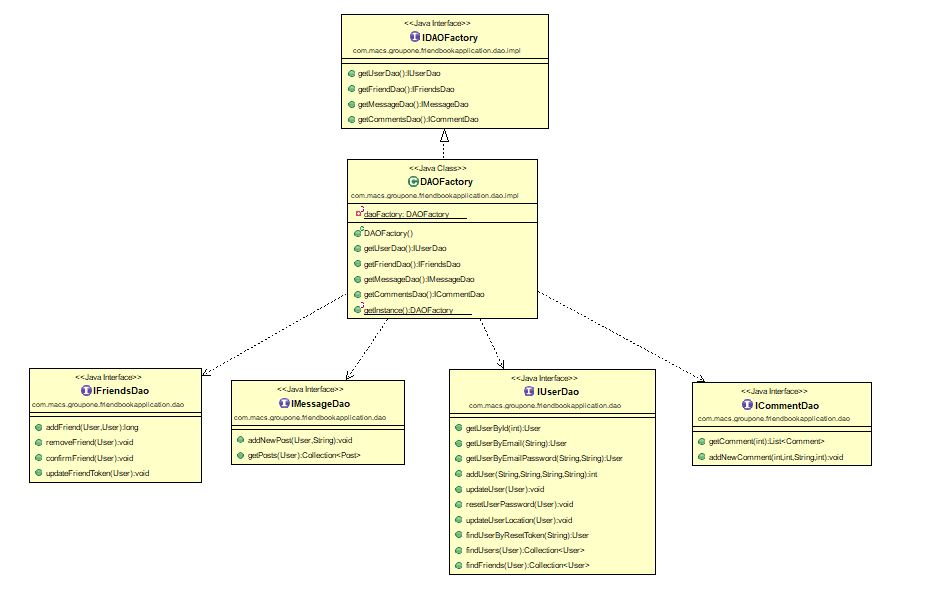
**Cloud Server Deployment:**

Heroku

**Design Pattern Implementation**

**Factory Pattern**

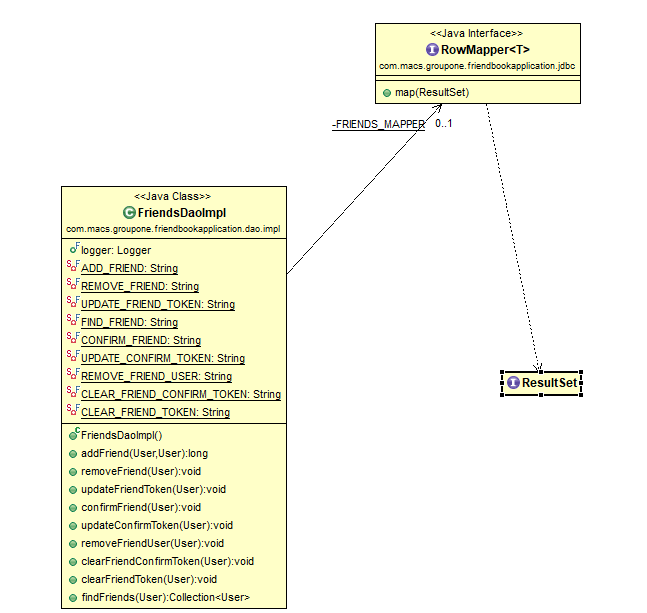
We have implemented Factory pattern to hide the creation logic of an object from the client. This factory comes under a creational pattern. As shown in the figure (8), we have created a factory class called “DAOFactory.java” and “ServiceFactory.java” which implements IDAOFactory and IServiceFactory interface.



**Figure 8 Factory Design Pattern[1]**

**Bridge Pattern**

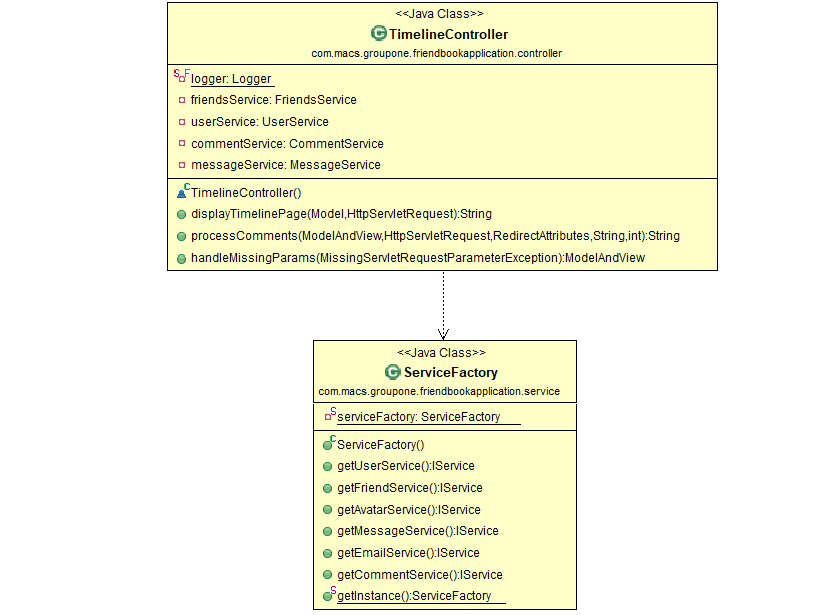
Bridge pattern is used to decouple an abstraction from its implementation so that if they require to alter in future, they can alter without affecting each other. This pattern comes under structural pattern. In our project, as shown in the figure (9), we have used Bridge pattern to map result set objects to the custom objects using Row mapper interface which acts as a bridge between result set object and custom object.



**Figure 9 Bridge Design Pattern[1]**

**Singleton Pattern**

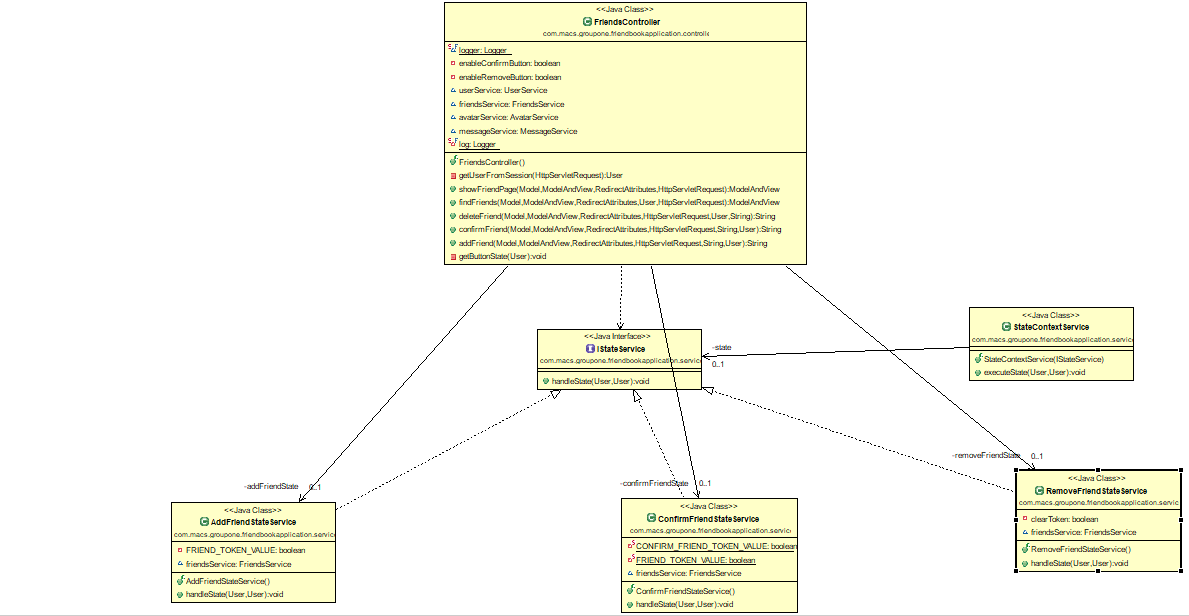
We have used Singleton Pattern to restrict the instantiation of the class to make sure that there will only be a single instance of the class throughout the project. As shown in the figure [10], in ServiceFactory.java class we have created a getInstance() method which returns the object of Service Factory which is used to get the instance of different classes throughout the project.



**Figure 10 Singleton Design Pattern[1]**

**State Pattern**

In our project we have created a context class called “StateContextService.java” which implements the IStateService interface. We have created three state classes as shown in the figure (11) which implements IStateService interface. Also, a factory is used to create new states whenever the client calls a particular state. State pattern comes under Behavioral pattern as the class behavior changes based on its state.

****

**Figure 11 State Design Pattern[1]**

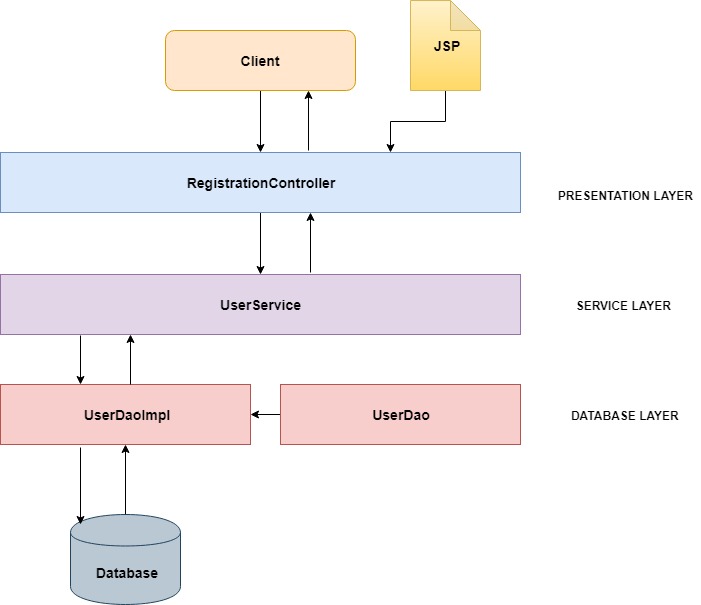
**Separation of Layers**

We have used MVC (Model-View-Controller) architecture for the separation of Presentation, Business and Data layer. It is used to segregate the application’s process.

1. Model – The model represents the structure of the data along with all the JAVA entities. It carries and maintains the application data.
2. View – The view forms the User Interface of the application. It depicts the visualization of the model data. In our project, our JSP pages form the view of the application.
3. Controller – The controller controls the data flow and handles the user request. It renders the view with the model data. It acts between the Model and the View.

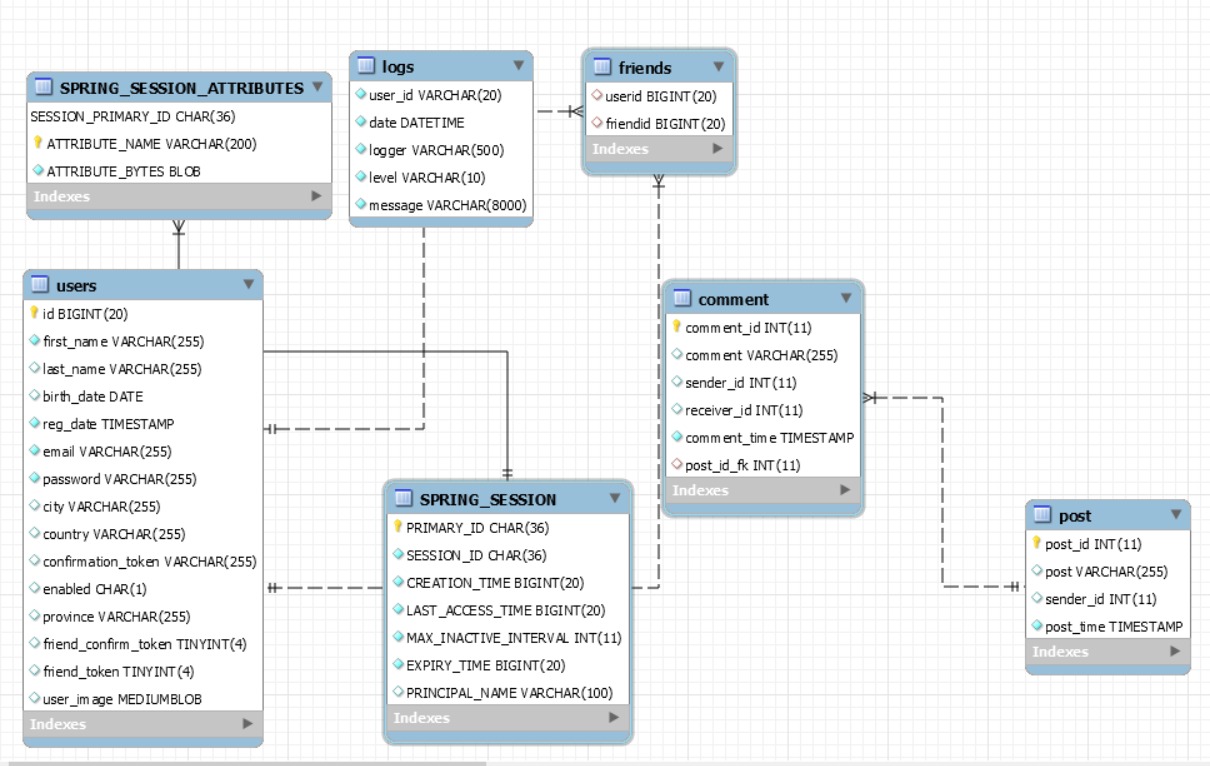
Figure (12) shows the Spring MVC architecture of Friendbook. The following figure shows the MVC architecture for Registration Module.

Figure (13) shows the Database Architecture with different entities and their relationship.



**Figure 12 Friendbook MVC Architecture**

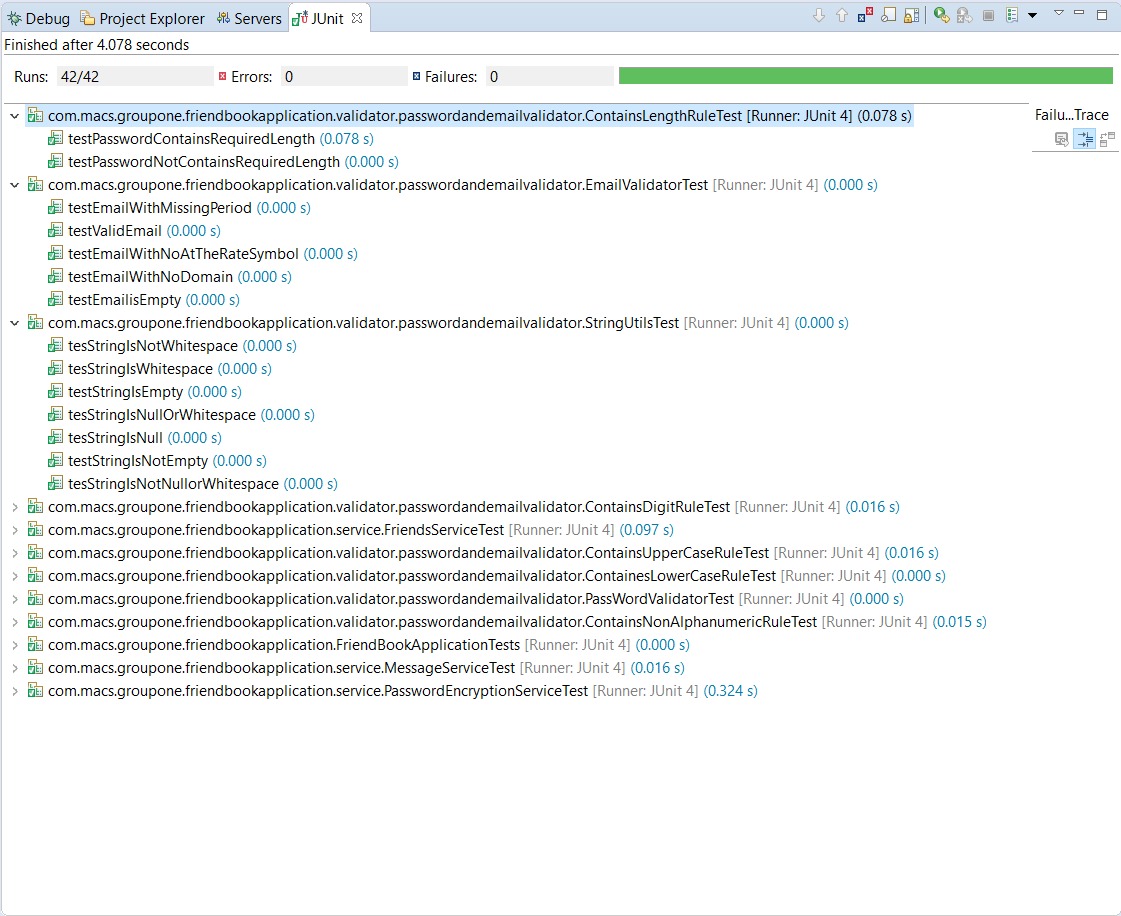
**Database Schema**



**Figure 13 Friendbook Database Schema**

**Testing**

We have used Junit for writing test cases as shown in the figure (14). We have wrote around 42 test cases to test the logical functionality of the app and all the test cases are passing.

****

**Figure 14 Test Cases**

**Naming/Spacing Convention**

Naming and Spacing is an integral part of the programming world. It plays a vital role for any programmer to understand the nomenclature of the program. It increases the user readability of the code which allows users to understand code more quickly.

For the spacing in our project, we have used tabs instead of space as tabs give more user readability and make the code look even.

For the naming convention, we have used “CamelCase” naming convention style for good maintenance and readability of code. CamelCase is of two types “LowerCamelCase” and “UpperCamelCase.” In UpperCamelCase first letter of every word is capitalized, while in LowerCamelCase first letter of the word begins with lowercase and rest all with uppercase.

Following table shows the list of naming and coding conventions we have used in our project.

|  |  |  |
| --- | --- | --- |
| **Identifier Type** | **Rules for Naming** | **Example** |
| Packages | We have declared all our packages in lowercase | com.macs.groupone.friendbookapplication  com.macs.groupone.friendbookapplication.controller |
| Classes | We have declared all our classes in Upper Camel Case | class FriendsService  class UserService |
| Interfaces | For interface, we followed with the naming convention “IInterfaceName”. | interface IService  interface IServiceFactory |
| Methods | For methods we have followed with the Lower camel case nomenclature | void removeFriend();  void confirmFriend();  void removeFriendUser(); |
| Variables | For declaration of variables we have used Lower Camel Case naming convention. | private String passwordConfirm;  private String confirmationToken;  private String lastName; |
| Constants | We have declared all our constants in an Upper Case. | public static final string ADD\_FRIEND = ”{call addFriend(?,?)}”; |

**Refactoring**

Refactoring is changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is an essential part of the software development world. Refactoring make the code easy to understand, more readable, maintainable and it reduces the cost. The following are the refactoring techniques we have used to remove the bad smell from the code.

1. **Preserve Whole Object**

In the findUsers method of UserService.java initially, we were passing a list of arguments to find a user. So, instead of a long parameter list, we are passing the whole object.

1. **Form Template Method**

For all the DaoImpl classes to reduce all the redundant code, we have created a separate class called “RowMapper.java” that will map all the user objects with the result set.

1. **Replace Conditional with Polymorphism**

Instead of a switch statement for password validation rules, we have created an interface and implement it to the individual password validation rule classes.

1. **Extract Method**

We have reduced the amount of redundant code by extracting them into a single method and accessing it wherever required.

**Technical Debt**

We could have applied the observer design pattern in one of our features called the “New Post” which lets users notify whenever someone post a text status.

**Member’s Contribution**

|  |  |
| --- | --- |
| **Member Name** | **List of Classes and Stored Procedures** |
| **Suman Singh**  **Suman Singh** | **FriendBookApplication.java**  **ApplicationConfigPropertyConfigurator.java**  **ValidationPropertyConfigurator.java**  **Constants.java**  **ForgetPasswordController.java**  **LoginController.java**  **ProfileController.java**  **AbstractDao.java**  **UserDao.java**  **DAOFactory.java**  **IDAOFactory.java**  **UserDaoImpl.java**  **DatabaseAccessException.java**  **DatabaseConnectionFailure.java**  **DatabaseOperationException.java**  **JdbcManager.java**  **JdbcManagerImpl.java**  **RowMapper.java**  **AvatarService.java**  **IService.java**  **IServiceFactory.java**  **PasswordEncryptionService.java**  **ServiceFactory.java**  **UserService.java**  **ForgetPasswordValidator.java**  **FormValidatorFactory.java**  **IFormValidatorFactory.java**  **LoginValidator.java**  **RegistrationValidator.java**  **ResetPasswordValidator.java**  **ValidationCode.java**  **ContainesLowerCaseRule.java**  **ContainsDigitRule.java**  **ContainsLengthRule.java**  **ContainsNonAlphanumericRule.java**  **ContainsUpperCaseRule.java**  **EmailValidator.java**  **PassWordValidator.java**  **PasswordRule.java**  **StringUtils.java**  **PasswordEncryptionServiceTest.java**  **ContainesLowerCaseRuleTest.java**  **ContainsDigitRuleTest.java**  **ContainsLengthRuleTest.java**  **ContainsUpperCaseRuleTest.java**  **EmailValidatorTest.java**  **PassWordValidatorTest.java**  **StringUtilsTest.java**  **Stored Procedures:**  **findFriend(int)**  **findUserByResetToken(int)**  **getMessageByUserID(int)**  **getUserByEmail(varchar)**  **getUserByEmailPassword(varchar,varchar)**  **getUserById(int)**  **removePost(int)**  **resetUserPassword(varchar,varchar,varchar,char)**  **updateUser(varchar, varchar, char)**  **updateUserImage(blob,varchar)**  **updateUserLocation(varchar, varchar,varchar,varchar)** |
| **Sudarshan Suresh Srikant** | **FriendsController.java**  **NewPostController.java**  **FriendsDao.java**  **FriendsDaoImpl.java**  **Friend.java**  **Post.java**  **User.java**  **AddFriendStateService.java**  **ConfirmFriendStateService.java**  **FriendsService.java**  **IStateService.java**  **RemoveFriendStateService.java**  **StateContextService.java**  **StateFactoryService.java**  **FriendsServiceTest.java**  **IMockUserFactory.java**  **MockUserFactory.java**  **Stored Procedures:**  **addFriend(int,int)**  **addNewPost(int,int)**  **addNewComment(int, int varchar, int)**  **clearFriendConfirmToken(int)**  **clearFriendToken(int)**  **findFriends(int)**  **removeFriend(int)**  **getPostCreator(int)**  **getUserList(varchar,varchar,varchar,varchar,varchar)**  **removeFriendUser(int)**  **updateConfirmToken(int)**  **updateFriendToken(int)** |
| **Yash Desai** | **RegistrationController.java**  **TimelineController.java**  **CommentDaoImpl.java**  **MessageDaoImpl.java**  **CommentService.java**  **EmailService.java**  **MessageService.java**  **MessageServiceTest.java**  **Stored Procedures:**  **getComment(int)**  **getMessage(int)** |

|  |  |
| --- | --- |
| **Member Name** | **List of Presentation Layer File** |
| **Suman Singh** | **forgotPassword.jsp**  **login.jsp**  **registration.jsp**  **resetpassword.jsp**  **profile.jsp** |
| **Sudarshan Suresh Srikant** | **friends.jsp**  **newpost.jsp** |
| **Yash Desai** | **timeline.jsp** |

**References:**

[1]"ObjectAid UML Explorer", Objectaid.com, 2019. [Online]. Available: https://www.objectaid.com. [Accessed: 12- Apr- 2019]. [2]"JDBC - Exceptions Handling (SQLException Methods)", www.tutorialspoint.com, 2019. [Online]. Available: https://www.tutorialspoint.com/jdbc/jdbc-exceptions.htm. [Accessed: 07- Apr- 2019].

[3]S. enforcing, "Simple Java password rule enforcing", Code Review Stack Exchange, 2019. [Online]. Available: https://codereview.stackexchange.com/questions/146721/simple-java-password-rule-enforcing/146763. [Accessed: 07- Apr- 2019].

[4]"Util.java", Javapractices.com, 2019. [Online]. Available: http://www.javapractices.com/apps/movies/javadoc/src-html/hirondelle/movies/util/Util.html. [Accessed: 07- Apr- 2019].

[5]Cdn.dal.ca, 2019. [Online]. Available: https://cdn.dal.ca/content/dam/dalhousie/images/dept/communicationsandmarketing/01%20DAL%20FullMark-Blk.jpg.lt\_412f83be03abff99eefef11c3f1ec3a4.res/01%20DAL%20FullMark-Blk.jpg. [Accessed: 07- Apr- 2019].

[6]E. Paraschiv, "Getting Started with Forms in Spring MVC | Baeldung", Baeldung, 2019. [Online]. Available: https://www.baeldung.com/spring-mvc-form-tutorial. [Accessed: 07- Apr- 2019].