**1. Software Application**

University has decided to set up a Lottery scheme to raise money for local charities and has started to develop a prototype **Lottery Web Application**. As a software security specialist, the university have asked you to help develop their prototype by **designing and implementing** some basic **functionality and security elements**.

**Lottery Web Application**

The university development team have already implemented a basic Java web application which they have provided for you for further development. The web application consists of the following elements:

* **Java web application**
* **Apache Tomcat Server**
* **MySqL Database**

Because the web application is a prototype, the university development team would like your final submitted program to run using **Docker** to enable them to test it before investing in the hardware and software needed to host the final product. A video showing how to set up and run the software is provided below.

**PROJECT JAVA CODE**

The university development team have uploaded the all the files you need to get started in a zip file for you [here](https://ncl.instructure.com/courses/24641/files/3277301/download?wrap=1). The folder *LotteryWebApp* is the Java Web Application project which can be opened and worked on in the IntelliJ IDEA.

Currently the *LotteryWebApp* project consists of the following:

* ***index.jsp -***An HTML/JSP home page with a basic form to input data.
* ***CreateAccount -***A Java servlet which:
  + executes when the home page form data is submitted
  + inserts data into a MySQL database table ***userAccounts***
  + outputs a 'success' message to a ***account.jsp***HTML/JSP page
  + outputs an 'error' message to the ***error.jsp***HTML/JSP page
* ***account.jsp -***An HTML/JSP page which
  + displays the 'success' from the ***CreateAccount*** servlet
  + contains a submit button to get all current data in the database
* ***UserLogin -***Another Java servlet which:
  + executes when the HTML page button is pressed on the page ***account.jsp***
  + queries a MySQL database table ***userAccounts*** for all its data
  + outputs the data retrieved from the MySQL database to an HTML/JSP page ***output.jsp***
* ***output.jsp -***An HTML/JSP page which displays the data retrieved from the database in a table and a link back to the home page ***index.jsp***
* ***error.jsp -***An HTML/JSP page which displays the 'error' from the ***CreateAccount*** servlet and a link back to the home page ***index.jsp***

**DOCKER CONTAINER IMAGES**

Inside the Coursework zip file you will find a README with instructions to get the web app running for different options. There are several options when developing:

* **Docker only:** Run tomcat server with web app deployed and MySQL database server in Docker
* **Localhost and Docker:**Run tomcat server with web app deployed on localhost (your computer) and MySQL database server in Docker
* **Localhost:** Run tomcat server with web app deployed and MySQL database server on localhost

I would recommend either of the bottom two options if you can when developing.

When making final submission, please make sure it is set up to run on **Docker only**. Instructions are in the README file (**I will also run through this in an online session**).

### ****Development Tasks****

You will be given a number of tasks to develop the Lottery Web Application some more to add both general functionality and security features.

#### **Task 0 - Initial Task**

* **Open** Java project in IntelliJ and familiarise yourself with its structure and code
* **Look** at the README and associated files and familiarise yourself with them
* **Set up** and test the web application you have been given
* You can use any development setup you like:
  + **Full Docker**- Apache Tomcat Server and MySQL Database Server both running on Docker
  + **Hybrid** - Local installation of Apache Tomcat Server running on own computer with MySQL Database Server running on Docker (my personal favourite)
  + **Full Local** - Local installation of Apache Tomcat Server and MySQL Database Server running on own computer

#### **Task 1 - Data Input**

1. **Redevelop** the home page **index.jsp**.
   1. **Create/extend** a registration form for new users which accepts first name, last name, email, phone number, username, and password.
      1. Implement appropriate client side input validation using pure JavaScript or the JavaScript library JQuery.
      2. Email must be a valid email address (look up standards on this).
      3. Phone number must be of the form xx-xxxx-xxxxxxx (e.g. 44-0191-1234567)
      4. Password must contain:
         * Between 8 and 15 characters
         * At least 1 uppercase and 1 lowercase character
         * At least 1 digit
   2. **Create/extend** a login form for existing users which accepts a username and password
      1. Implement appropriate client side input validation using pure JavaScript or the JavaScript library JQuery.
      2. Redevelop the Java servlet **UserLogin**to query the MySQL database table ***userAccounts*** to check if password and username match with an existing account.
         * If login is verified the web application displays a 'login success' message on the **account.jsp** page
         * If login is not verified the web application returns an 'login unsuccessful' message on the **error.jsp**page
2. **Create** a server side filter (Servlet Filter)**:**
   1. Ensure the following are not present in the data submitted via the registration form: <,  >,  !,  {,  } , insert, into, where, script, delete, input.
      1. If not present the web application displays a 'success' message on the **account.jsp** page
      2. If present the web application displays an 'error' message on the **error.jsp** page

#### **Task 2 - Data Creation & Storage**

1. **Extend** and expand the **CreateAccount** servlet.
   1. **Create** a hash of the user's password.
   2. **Update** the accounts database - userAccounts table (e.g. in any docker data.sql files), to be able to store the hashed password, and replace **Pwd varchar(50) NOT NULL,** with **Pwd varchar(200) NOT NULL,**
   3. **Insert** all registration data into the accounts database - userAccounts table.
   4. **Set** user information first name, last name, username, email, and hashed password as session object attributes
      * [**See below:**A Note about HTTPSessions and HttpServletRequests]
2. **Redevelop** ***accounts.jsp*** page
   1. **Get** user information attributes (**EXCEPT PASSWORD**) from the session object and insert them so they are visible to the user near top of page below 'login success' message
   2. **Create**below the user information a form with attribute action=AddUserNumbers that:
      * allows the user to enter 6 integers between 0 and 60 inclusive.
      * contains a button for the user to click and which calls a Javascript function which populates the form with 6 randomly secure integers between 0 and 60 inclusive (i.e similar to a lucky dip)
      * contains a submit button
   3. **Create**a simple form with attribute action=GetUserNumbers which contains a submit button named Get Draws
3. **Create** a new Java Servlet called ***AddUserNumbers***.
   1. **Modify** the doPost method so it:
      * **Creates** a single String from the received 6 integers (e.g. "1,2,3,4,5,6")
      * **Creates** an Encryption keyPair object and adds it as an attribute to the session object.
        + [**See below:**A Note about HTTPSessions and HttpServletRequests]
        + [**Note:** keyPair only needs to stored for a single session (for marking purposes)]
      * **Encrypts** the single String containing the 6 numbers
      * **Amends** the encrypted String to a text (.txt) file using the first 20 characters of the user's hashed password as the filename (remember the password should have been set as an attribute of the request object)
        + [**Note:** the filename may not be unique of course but you can comment on that in your lab report]
        + [**Note:** text files only need to be stored for a single session (for marking purposes)]
4. **Create** a new Java Servlet called ***GetUserNumbers***.
   1. **Modify** the doPost method so it:
      * **Reads**all the encrypted Strings from a text (.txt) file whose file name is the first 20 characters of the user's hashed password (remember the password should have been set as an attribute of the session object)
      * **Decrypts** each String read from file and adds them to an Array
      * **Adds** Array as an HttpServletRequest request attribute with the name "draws" and dispatches request to ***account.jsp*** page
        + [**Tip:**request.getRequestDispatcher("/account.jsp")]
5. **Expand**and extend ***accounts.jsp*** page once more:
   1. **Add** an empty paragraph tag <p></p> and embed jsp code (e.g. inside jsp tags <% java code here %>) in between tags that prints out content of draws attribute from request object.
      * [T**ip:** you can use loops inside jsp tags *<% %>*]
6. **Update** the ***UserLogin*** servlet to:
   1. **Remove** all pre-existing attributes in the session object (this is a very crude and basic method for logging out/removing any trace of a previous user on same browser)
      * [**Tip:** the session methods *getAttributeNames()*, *removeAttribute()* and the *Enumeration* Interface may help with this. Alternatively you could look at the session *invalidate()* method to remove everything but remember ***you may want to keep hold of the encryption keypair object***]
   2. **Query** the MySQL database table userAccounts to check if password and username match with an existing account (remember existing accounts should now have a password that is hashed).

#### **Task 3 - Data Access**

1. **Limit**user logins to 3
   * [**Note:** it is enough just to disable the form which can be reactivated by refreshing the page]
2. **Create** an admin home page (**admin\_home.jsp**) inside a folder ***admin***
   * Add functionality to view all users account details (**EXCEPT PASSWORD**)
3. **Create** two roles admin and public and implement a RBAC strategy in your web application
   * A user with a public role will be directed to their account page (***accounts.jsp)***as before and should not have access to **admin\_home.jsp**
   * A user with admin role will be directed to **admin\_home.jsp**and should not have access to user accounts page (***accounts.jsp)***
     + [**Tip:**Think about how you would extend and expand your web application to include Role Based Access Control (RBAC)]
     + [**Note:** You can assume a user is automatically permitted to register as either role]
4. **Create** a winning random lottery draw (6 numbers) and store it somewhere appropriate (e.g. the database)
   * [**Note:** This can be manually created for tested purpose so you can create a match with a known existing user draw]
5. **Extend** and expand the ***accounts.jsp***page
   * Add a button which invokes a check on the stored winning lottery draw against the user's current lottery draws for a match and output a message on the page if there is a match and the user wins.
   * If it still exists you can remove the button to get all data from the database (the **output.jsp** page can also be removed as it's not really needed any more)
6. **Ensure** appropriate access modifiers are in place in your Java code.

#### **Task 4 - Data Transit**

1. Implement a Secure Socket Layer scheme into your web application