## DCUreg

# DUBLIN CITY UNIVERSITY

**SEMESTER 2** **SOLUTIONS** **2015/2016**

**MODULE:** EE417 – Web Application Development

### PROGRAMME(S):

|  |  |  |
| --- | --- | --- |
|  | MEN | MEng in Electronic Systems |
|  | MTC | MEng in Telecommunications Engineering |
|  | DME | B.Eng. in Digital Media Engineering |
|  | ECSAO | Study Abroad (Engineering & Computing) |
|  | ECSA | Study Abroad (Engineering & Computing) |
|  | ICE | BEng Info and Communications Engineering |
|  | MEQ | Masters Engineering Qualifier Course |
|  | IFPES | PG Int. Foundation Prog.(Elec. Systems) |
|  |  |  |

**YEAR OF STUDY:** 1,4,C,O,X

**EXAMINERS:**

|  |  |
| --- | --- |
| David Molloy | (Ext:8426) |
| Prof. Gerard Parr |  |
| Dr. Iain Phillips |  |
| Prof. Sakir Sezer |  |

**TIME ALLOWED:** 3 Hours

**INSTRUCTIONS:** Answer 4 questions. All questions carry equal marks.

**PLEASE DO NOT TURN OVER THIS PAGE UNTIL YOU ARE INSTRUCTED TO DO SO**

The use of programmable or text storing calculators is expressly forbidden.

Please note that where a candidate answers more than the required number of questions, the examiner will mark all questions attempted and then select the highest scoring ones.

***Requirements for this paper (Please mark (X) as appropriate)***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | |  | |  | |  | | *Log Tables*  *Graph Paper*  *Dictionaries*  *Statistical Tables*  *Bible* | |  | | --- | |  | |  | |  | |  | |  | | *Thermodynamic Tables*  *Actuarial Tables*  *MCQ Only – Do not publish*  *Attached Answer Sheet*  *Exam Paper to be returned with booklet* |

## QUESTION 1 (Solutions) [TOTAL MARKS: 25]

### Q 1(a) [22 Marks]

**Form.html**

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Registration Form</title>

<script>

**function** validate() {

**var** age = myForm.age.value;

**var** password = myForm.password.value;

**if** (!isInt(age)) {

alert("Age is not a valid integer - please fix before submitting!");

**return** **false**;

}

**else** **if** (password.length<=5) {

alert("Password must be at least 6 characters long");

**return** **false**;

}

**else** {

**return** **true**;

}

}

**function** isInt(value) {

**var** x = parseFloat(value);

**return** !isNaN(value) && (x | 0) === x;

}

</script>

</head>

<body>

<p><b>Please register for the application below:</b></p>

<form method=*"post"* name=*"myForm"* onsubmit="return validate();" action=*"/temp/Register"*>

<div>First name: <input type=*"text"* name=*"firstname"*/></div>

<div>Surname: <input type=*"text"* name=*"surname"*/></div>

<div>Address: <input type=*"text"* style="width:*400px*" name=*"address"*/></div>

<div>Email: <input type=*"text"* name=*"email"*/></div>

<div>Phone: <input type=*"text"* name=*"phone"*/></div>

<div>Age: <input type=*"text"* name=*"age"*/></div>

<div>Username: <input type=*"text"* name=*"username"*/></div>

<div>Password: <input type=*"password"* name=*"password"*/></div>

<input type=*"submit"* value=*"Register"* />

</form></body></html>

### *(4 Marks for core form, 6 Marks for Validation on either client or server)*

**User.java**

**package** temp;

**public** **class** User {

**private** String firstname;

**private** String surname;

**private** String address;

**private** **int** age;

**private** String email;

**private** String phone;

**private** String username;

**private** String password;

**public** User(String firstname, String surname, String address, **int** age,

String email, String phone, String username, String password) {

**super**();

**this**.firstname = firstname;

**this**.surname = surname;

**this**.address = address;

**this**.age = age;

**this**.email = email;

**this**.phone = phone;

**this**.username = username;

**this**.password = password;

}

**public** String getFirstname() {

**return** firstname;

}

**…. Etc…**

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

*(3 Marks for User Bean)*

**Register Servlet**

**package** temp;

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**import** javax.servlet.annotation.WebServlet;

@WebServlet("/Register")

**public** **class** Register **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** **void** doPost(HttpServletRequest req, HttpServletResponse res)

**throws** ServletException, IOException {

User user = **new** User(req.getParameter("firstname"), req.getParameter("surname"), req.getParameter("address"),

**new** Integer(req.getParameter("age")).intValue(), req.getParameter("email"), req.getParameter("phone"),

req.getParameter("username"),req.getParameter("password"));

HttpSession session = req.getSession(**true**);

session.setAttribute("theUser", user);

res.sendRedirect("Secure");

}

}

*(6 Marks for Registration Servlet/JSP)*

### Secure Servlet

**package** temp;

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**import** javax.servlet.annotation.WebServlet;

@WebServlet("/Secure")

**public** **class** Secure **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res)

**throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter out = res.getWriter();

HttpSession session = req.getSession(**true**);

User theUser = session.getAttribute(“theUser”);

**if** (theUser!=**null**) {

out.println("<html><head><title>Secure Page</title></head>");

out.println("<body><h1>Successful Access</h1>”);

out.println(“Welcome “ + theUser.getFirstname() + “</body></html>");

} **else** {

out.println("<html><head><title>Denied</title></head>");

out.println("<body><h1>Denied</h1></body></html>");

}

out.close();

}

}

*(3 Marks for Secure Servlet/JSP)*

### Q 1(b) [3 Marks]

XHTML has a number of advantages over HTML. While it is less forgiving, it has a more rigid structure which still allows the developer to utilise the full functionality of HTML and CSS. By using XHTML we can ensure that the DOM model is preserved, which has a number of advantages for us if we are interacting with and modifying the DOM through JavaScript code or through Ajax. As we are also standards complaint using XHTML, we are more likely to have browser compatible code.

## [End of Solutions of Question 1]

## QUESTION 2 (Solutions) [TOTAL MARKS: 25]

### Q 2(a) [12 Marks]

1. In the diagram, we can see two subnetworks: Public Subnet 1 is a public subnetwork and needs to allow routing to the internet. For this reason, it should be allocated ‘Routing Table B’. Routing Table B has a route 0.0.0.0/0 to the internet gateway, which facilitates traffic to/from the internet. On the other hand, ‘Private Subnet 1’ is a private network and resources should not have direct access to the internet. In this scenario, we only want traffic locally within our VPC so ‘Routing Table A’ applies here. *(4 Marks)*
2. The application server is unable to perform updates, as it is located in a Private Subnet and has no direct access to the internet. In order to facilitate communication, we should deploy a NAT (Network Address Translation) Gateway in the Public Subnet 1. We could then change the routing table to route external traffic to that gateway allowing access to the internet. The application server could then patch as required. *(4 Marks)*
3. Again, because the application server is located in a private subnetwork, there is no route to SSH in from outside. Even the NAT Gateway would only let requests for traffic \*out\* from the instance. Best practice would dictate that we should use a Bastion server. We would have one minimal server in a public subnet which could be accessed from a small set of IP addresses (of developers/administrators). Once the developer SSH connects to the bastion server, they can then SSH to the other servers (assuming they have appropriate security keys etc.) *(4 Marks)*

### Q 2(b) [7 Marks]

As the question asks that we use the base diagram, we will only address the rules for what is shown in the diagram (and will not add rules for NAT/Bastion servers as discussed in Part 2(a).

**AppSecurityGroup**

***Inbound***

* ELBSecurityGroup -> TCP -> 80 (HTTP)

***Outbound***

* 0.0.0.0/0 -> TCP

**DBSecurityGroup**

***Inbound***

* AppSecurityGroup -> TCP -> 1521 (Oracle)

***Outbound***

* 0.0.0.0/0 -> TCP

**ELBSecurityGroup**

***Inbound***

* 0.0.0.0/0 -> TCP -> 80 (HTTP)
* 0.0.0.0/0 -> TCP -> 443 (HTTPS)

***Outbound***

* AppSecurityGroup -> TCP -> 80 (HTTP)

### Q 2(c) [4 Marks]

[Discussion of two of horizontal scaling, vertical scaling or elastic scaling from the course material]

### Q 2(d) [2 Marks]

The Presentation-Application-Data (PAD) architecture is important when developing web applications as if we stick to the PAD architecture we will have clean separation between our core tiers. For example, if we later wished to change our front end of our application away from HTML to an new interface we could more easily manage this. Likewise, by keeping our data tier independent, we can easier move database provider or even data storage type.

## [End of Solutions of Question 2]

## QUESTION 3 (Solutions) [TOTAL MARKS: 25]

### Q 3(a) [11 Marks]

i)

### create table orders (

### OrderID INTEGER,

### CustomerID INTEGER,

### EmployeeID INTEGER,

### OrderDate VARCHAR(20),

### RequiredDate VARCHAR(20),

### ShippedDate VARCHAR(20),

### ShipVia VARCHAR(100),

### Freight VARCHAR(100),

### ShipName VARCHAR(100),

### ShipAddress VARCHAR(100),

### ShipCity VARCHAR(40),

### ShipRegion VARCHAR(60),

### ShipPostalCode VARCHAR(20),

### ShipCountry VARCHAR(100),

### PRIMARY KEY(OrderID),

CONSTRAINT CNST1,

FOREIGN KEY (CustomerID) REFERENCES CUSTOMERS

ON DELETE CASCADE

CONSTRAINT CNST2,

FOREIGN KEY (OrderID) REFERENCES ORDER\_DETAIL

ON DELETE CASCADE

### )

*(5 Marks)*

### ii) Order\_Detail has primary key ‘OrderID’ combined with ‘ProductID’. Each of these is in turn a foreign key to Order and Product correspondingly.

Product has primary key ‘ProductID’. Customer has primary key ‘CustomerID’

### *(3 Marks)*

### iii) Select \* from customer c, order o where o.customerid=c.customerid

### (3 Marks)

### Q 3(b) [5 Marks]

**Select \* from product where ProductName=’value’;**

Because ProductName is not a primary key, the table is not optimised for searching quickly against this column. As the number of rows increase, the query could start to perform poorly. To handle this read request, we can add an index.

***Create index productname\_index on Product(ProductName);***

### Q 3(c) [4 Marks]

* **RESTRICT / NO ACTION** - this delete rule prevents you from deleting a row from the parent table if the row has any children. The SQL2 standard somewhat confusingly, actually calls the RESTRICT rule, NO ACTION.
* **CASCADE**- tells the DBMS that when a parent row is deleted, all of its child rows should also be automatically deleted from the child table.
* **SET NULL** - tells the DBMS that when a parent row is deleted, the foreign key values in all of its child rows should automatically be set to NULL (unknown).
* **SET DEFAULT** - tells the DBMS that when a parent row is deleted, the foreign key values in all of its child rows should automatically be set to the default value for that particular column.

### Q 3(d) [5 Marks]

The web service client proxy is written so that it appears as though the code is being executed locally and looks very much like the creation of a local constructor. However, in reality what is happening is that XML is being used for communication between the local client and the remote web service. This communication is happening probably using SOAP which handles the sending and receiving of these XML messages.

Hence, in this scenario, the CalculatorProxy.sum() request actually creates an XML message with the parameters to be passed over HTTP to the web service. The remote web service performs the calculation, returns the result in XML and the client parses this XML to get the result.

## [End of Solutions of Question 3]

## QUESTION 4 (Solutions) [TOTAL MARKS: 25]

### Q 4(a) [5 Marks]

**package** temp;

**public** **class** VowelCheck {

**public** **static** **void** main(String[] args) {

**if** (args.length!=1) {

System.***err***.println("Invalid number of arguments!");

}

**else** System.***out***.println(*checkVowel*(args[0]));

}

**private** **static** String checkVowel(String s) {

**if**((s.contains("a"))||(s.contains("e"))||(s.contains("i"))||(s.contains("o"))||(s.contains("u"))) {

**return** "This string contains a vowel!";

} **else** {

**return** "This string does not contain a vowel!";

}

}

### }

### Q 4(b) [8 Marks]

**import** org.junit.Test;

**import** **static** org.junit.Assert.assertEquals;

**public** **class** MyTest {

@Test

**public** **void** testNormalVowel() {

System.***out***.println("MyTest: testNormalVowel()");

assertEquals("This string contains a vowel!",VowelCheck.*checkVowel*("Hello"));

}

@Test

**public** **void** testNormalNoVowel() {

System.***out***.println("MyTest: testNormalNoVowel()");

assertEquals("This string does not contain a vowel!",VowelCheck.*checkVowel*("Myth"));

}

@Test

**public** **void** testNull() {

System.***out***.println("MyTest: testNull()");

assertEquals("Invalid",VowelCheck.*checkVowel*(**null**));

}

@Test

**public** **void** testCapitals() {

System.***out***.println("MyTest: testCapitals()");

assertEquals("This string contains a vowel!",VowelCheck.*checkVowel*("HELLO"));

}

### }

### Q 4(c) [5 Marks]

**public** **class** VowelCheck {

**public** **static** **void** main(String[] args) {

**if** (args.length!=1) {

System.***err***.println("Invalid number of arguments!");

}

**else** System.***out***.println(*checkVowel*(args[0]));

}

**public** **static** String checkVowel(String s) {

**if** (s==**null**) {

**return** "Invalid";

}

s = s.toLowerCase();

**if**((s.contains("a"))||(s.contains("e"))||(s.contains("i"))||(s.contains("o"))||(s.contains("u"))) {

**return** "This string contains a vowel!";

} **else** {

**return** "This string does not contain a vowel!";

}

}

}

### Q 4(d) [5 Marks]

Model 2 or Model-View-Controller (MVC), solves many of the inherent problems with the original Model 1 design by providing a clear separation of application responsibilities. In the MVC, a central servlet, known as the Controller, receives all requests for the application. The Controller then processes the request and works with the Model to prepare any data needed by the View (usually a JSP) and forwards the data to a JSP. The JSP then uses the data prepared by the Controller to generate a response to the browser. In this architecture, the business and presentation code accommodates multiple interfaces to an application and provides excellent reuse

of code.

Model View Controller

**Browser**

**Database**

**Controller**

Request Processing

Data Validation

**Model**

Business Logic

Data Manipulat.

**View**

Response Generation

### Q 4(e) [2 Marks]

def usersFromIreland = User.findAllByCountry(“Ireland”);

## [End of Solutions of Question 4]

## QUESTION 5 (Solutions) [TOTAL MARKS: 25]

### Q 5(a) – Bean Part [5 Marks]

### Q 5(b) – Annotations Part [5 Marks]

Can represent any entity in the world: a bank account, a customer, a booking, a rental car etc.. For the ease of these solutions, I’m simply going to use a bank account.

**package** edu.ee.beans;

**import** java.util.Date;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table (name="Accounts")

**public** **class** Account {

**private** **int** id;

**private** **boolean** active;

**private** String accountType;

**private** String currency;

**private** **float** balance;

**private** Date creationDate;

**public** Account(**boolean** active, String accountType, String currency,

**float** balance, Date creationDate) {

**super**();

//this.id = id;

**this**.active = active;

**this**.accountType = accountType;

**this**.currency = currency;

**this**.balance = balance;

**this**.creationDate = creationDate;

}

**public** Account() { **super**(); }

@Id

@GeneratedValue

**public** **int** getId() { **return** id; }

**public** **void** setId(**int** id) { **this**.id = id; }

**public** **boolean** isActive() { **return** active; }

**public** **void** setActive(**boolean** active) { **this**.active = active; }

**public** String getAccountType() { **return** accountType; }

**public** **void** setAccountType(String accountType) {

**this**.accountType = accountType;

}

**public** String getCurrency() { **return** currency; }

**public** **void** setCurrency(String currency) { **this**.currency = currency; }

**public** **float** getBalance() { **return** balance; }

**public** **void** setBalance(**float** balance) { **this**.balance = balance; }

**public** Date getCreationDate() { **return** creationDate; }

**public** **void** setCreationDate(Date creationDate) {

**this**.creationDate = creationDate;

}

}

### Q 5(c) [4 Marks]

Handwritten SQL:

* Rewrite all 'CREATE TABLE' statements and any other data definition statements we require
* Download and set up the JDBC library jar file for the new database
* Rewrite potentially every method which contains SQL in any code files
* Debug and test SQL

### Hibernate:

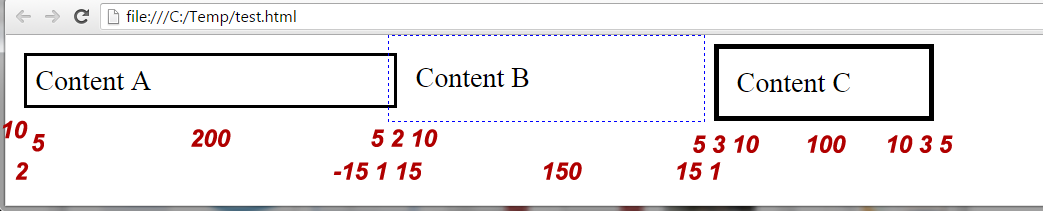
* Download and set up the JDBC library jar file for the new database
* Edit some configuration lines in order to move to the new database

### Q 5(d) [4 Marks]

1. The primary use of DOM within Ajax is that the model allows us to explicitly specify objects in web pages which we wish to target for "in-page replacement".
2. The XMLHttpRequest is the core of the Ajax model - without it the model would not exist\*. The XMLHttpRequest JavaScript object is the enabling technology which is used to exchange data asynchronously with the web server. In short, XMLHttpRequest lets us use JavaScript to make a request to the server and process the response without blocking the user.

### Q 5(e) [7 Marks]

i)



ii) Only the first box will fit if the browser size is 300px. The other floating boxes will move to the line below. However, these other boxes are smaller than the first box, so we only need to calculate the first box to work out the content width.

10 + 2 + 5 + 200 + 5 + 2 + 10 = 234px

## [End of Solutions of Question 5]

## [END OF SOLUTIONS]