## DCUreg

# DUBLIN CITY UNIVERSITY

**SEMESTER 2** **SOLUTIONS** **2016/2017**

**MODULE:** EE417 – Web Application Development

### PROGRAMME(S):

|  |  |  |
| --- | --- | --- |
|  | ECE | BEng Electronic & Computer Engineering |
|  | MEN | MEng in Electronic Systems |
|  | MTC | MEng in Telecommunications Engineering |
|  | ECSAO | Study Abroad (Engineering & Computing) |
|  | ECSA | Study Abroad (Engineering & Computing) |
|  | MEQ | Masters Engineering Qualifier Course |
|  | ICE | BEng Info and Communications Engineering |
|  |  |  |

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

**EXAMINER(S):**

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

**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.

***There are no additional requirements for this paper.***

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

### Q 1(a) [15 Marks]

i)

create table tcustomers (

ID integer,

SURNAME varchar2(80) not null,

FIRSTNAME varchar2(80) not null,

ADDRESS varchar2(200),

PHONE varchar2(50),

PRIMARY KEY(ID)

)

create table ttour (

CODE integer,

TYPE varchar2(30),

DESTINATION varchar2(50),

DEPARTUREDATE varchar2(50),

DURATION INTEGER NOT NULL,

PRIMARY KEY(CODE)

)

create table tbooking (

BOOKINGID INTEGER,

CUSTOMERID INTEGER,

TOURCODE INTEGER,

NUMBERPEOPLE INTEGER,

PRIMARY KEY(BOOKINGID),

CONSTRAINT CNST1

FOREIGN KEY (CUSTOMERID) REFERENCES TCUSTOMERS

ON DELETE SET NULL,

CONSTRAINT CNST2

FOREIGN KEY (TOURCODE) REFERENCES TTOUR

ON DELETE SET NULL

)

(8 Marks)

ii)

select c.firstname, c.surname from tcustomers c, ttour t, tbooking b

where c.id=b.customerid and t.code=b.tourcode

and t.destination='Austria'

(2 Marks)

iii)

select type, avg(duration) from ttour group by type

(2 Marks)

iv)

select t.code, b.numberpeople from ttour t, tbooking b

where t.code=b.tourcode

(2 Marks)

v)

create index tourdescindex1 on ttour(destination)

(1 Mark)

### Q 1(b) [2 Marks]

JDBC drivers allow our java applications to “talk” to SQL databases. They facilitate this by providing an implementation of the standard JDBC API and perform all of the underlying work to perform these JDBC operations natively on each database. Because this interface is ultimately a native interface, separate JDBC drivers are needed for each database vendor database.

### Q 1(c) [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 1(d) [4 Marks]

### PaaS: Platform as a Service. With PaaS, consumers are provided with computing platforms which typically includes the operating systems, programming language execution environment, database, web application server, load balancers and so on. This provides an instant environment for the deployment of web applications and allows developers to focus on the core business logic of the system (writing the application code) rather than spending considerable time on managing the "platform".

1. IaaS: Infrastructure as a Service: IaaS is the foundation block of cloud computing. Rather than purchasing or renting space in a costly datacenter, cloud consumers rent space in a virtual data center from a cloud provider. With IaaS the "raw materials" are provided and consumers will pay for only the resources they consume. These resources include CPU cores, memory, hard disk storage and bandwidth. Consumers are provided with the capability of modifying the resources available to them, typically through interfaces provided by the cloud provider.

## [End of Solutions of Question 1]

## 

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

### Q 2(a) [17 Marks]

<html>

<head>

<title>Session Tracking</title>

</head>

<body>

<form method="post" action="Stage1Servlet" name="form1">

<br/>Country: <select name="country"><option>Ireland</option><option>England</option><option>Scotland</option><option>Wales</option></select>

<br/>Name: <input type="text" name="fullname">

<br/><br/>

<input type="submit" value="Submit Form to First Stage"/>

</form>

</body>

### </html>

(4 Marks)

**package** com.ee417;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

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

**import** javax.servlet.http.Cookie;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

@WebServlet("/Stage1Servlet")

**public** **class** Stage1Servlet **extends** HttpServlet {

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

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String country = request.getParameter("country");

String name = request.getParameter("fullname");

// First let's use HTTPSession to Store this information

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

session.setAttribute("name", name);

session.setAttribute("country", country);

// Next let's use persistent cookies to store the information

Cookie c1 = **new** Cookie("name", name);

Cookie c2 = **new** Cookie("country", country);

response.addCookie(c1);

response.addCookie(c2);

// Lastly let's use hidden form fields to pass the information

out.println("<form method=\"post\" action=\"Stage2Servlet\">");

out.println("<input type=\"hidden\" name=\"fullname\" value=\"" + name + "\"/>");

out.println("<input type=\"hidden\" name=\"country\" value=\"" + country + "\"/>");

out.println("<input type=\"submit\" value=\"Submit to Stage 2\"></form>");

out.close();

}

}

(7 Marks)

**package** com.ee417;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

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

**import** javax.servlet.http.Cookie;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

@WebServlet("/Stage2Servlet")

**public** **class** Stage2Servlet **extends** HttpServlet {

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

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

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

out.println("<h2>Values from Session</h2>");

out.println("<br/>Name: " + session.getAttribute("name"));

out.println("<br/>Country: " + session.getAttribute("country"));

out.println("<h2>Values from Hidden Form</h2>");

out.println("<br/>Name: " + request.getParameter("fullname"));

out.println("<br/>Country: " + request.getParameter("country"));

out.println("<h2>Values from Cookies</h2>");

// First we grab all our cookies

Cookie[] cookies = request.getCookies();

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

**for** (**int** i=0; i< cookies.length; i++) {

String name = cookies[i].getName();

String value = cookies[i].getValue();

out.println("<br/>" + name + " : " + value);

}

}

out.close();

}

}

(6 Marks)

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

Discussion to cover the following:

Hidden Form Fields: Limited to a series of forms, complex where a lot of the data is involved. Strings only.

URL Rewriting: Limited to a series of dynamic documents (breaks for static documents), cumbersome to ensure all links are rewritten. Strings only.

Persistent Cookies: principal limitation is where individuals have these cookies turned off in browser privacy settings. Strings only.

Session API: few limitations. Sessions are typically lost on browser crash/shutdown.

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

### Web application frameworks provide “plumbing” facilities for building web applications. Certain tasks, such as validation, connection pooling, security, filtering, logging can be cumbersome to build manually every time a web application is built. Frameworks provide such facilities in a simple and effective manner, saving the developer time, which is the largest cost for a software development company. One example of this is the Grails Framework.

Client-side libraries, such as JavaScript libraries provide ready-to-use components for building the front end of web applications. For example, jQuery can be used to simplify a number of client-side tasks around referencing the DOM, manipulating pages, animation etc. Other libraries like jQuery-UI allow the rapid development of widgets which improve the look and feel of our web applications.

## [End of Solutions of Question 2]

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

### Q 3(a) [8 Marks]

### <html>

### <head>

### <title>Sample Layout</title>

### <style>

### div.box1-outer { background-color:#888888; float:left;width:40%;padding:45px;margin:20px; border:2px solid black;}

### div.box1-inner1 { background-color:#cccccc; float:left; width:25%;padding-top:15px; border:3px solid green; text-align:left; }

### div.box1-inner2 { background-color:#cccccc; float:right;width:25%;padding-top:15px; border:3px solid green; text-align:right;}

### div.box1-inner3 { background-color:#cccccc; float:left; width:100%; margin-top:50px; border:4px solid green; text-align:center;}

### </style>

### </head>

### <body>

### <div class="box1-outer">

### <div class="box1-inner1">Inner1</div>

### <div class="box1-inner2">Inner2</div>

### <div class="box1-inner3">Inner3</div>

### </div>

### </body>

### </html>

### Q 3(b) [9 Marks]

<html>

<head>

<title>Sample Layout</title>

<style>

div.box1-outer { background-color:#888888; float:left;width:40%;padding:45px;margin:20px; border:2px solid black;}

div.box1-inner1 { background-color:#cccccc; float:left; width:25%;padding-top:15px; border:3px solid green; text-align:left; }

div.box1-inner2 { background-color:#cccccc; float:right;width:25%;padding-top:15px; border:3px solid green; text-align:right;}

div.box1-inner3 { background-color:#cccccc; float:left; width:100%; margin-top:50px; border:4px solid green; text-align:center;}

</style>

<script>

function handleClick(which) {

for (var i=1;i<4;i++) {

if (document.getElementById("span" + i).innerHTML == "Clicked!")

document.getElementById("span" + i).innerHTML = "Old Click!";

}

document.getElementById("span" + which).innerHTML = "Clicked!";

}

</script>

</head>

<body>

<div class="box1-outer">

<div class="box1-inner1" onclick="handleClick('1')">

<span id="span1">Inner1</span>

</div>

<div class="box1-inner2" onclick="handleClick('2')">

<span id="span2">Inner2</span>

</div>

<div class="box1-inner3" onclick="handleClick('3')">

<span id="span3">Inner3</span>

</div>

</div>

</body>

</html>

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

The WSDL (Web Services Description Language) describes the following:

 The WSDL is a document written in XML, describing the following:

* **What a service does -**the methods that the service provides
* **How a service is accessed -**details of the data formats and protocols necessary to access a service operations
* **Where a service is located -**details of the protocol specific network address, such as a URL

### Q 3(d) [2 Marks]

HTTP Version 1.1 introduced the concept of persistent connections. Previously each resource (e.g. javascript files, css files, images etc.) were downloaded over a separately opened HTTP connection. With HTTP1.1 the connection is persistent, allowing a number of resources to be sent over the single connection. This connection typically times out after a couple of seconds, which provides enough time for an entire web page to be transferred.

### Q 3(e) [2 Marks]

HTTP status codes in a server response is a 3-digit integer which describes the success or otherwise of the request. For example, requesting a URL using the GET method will typically return the content of that URL together with a 200 (OK) response. However, if the URL did not exist, it would return a 404 Page Not Found status code.

***[End of Solutions of Question 3]***

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

### Q 4(a) [19 Marks]

### The particular coding example does not matter here – the important element is to write any skeleton Java class which provides an empty (or at least minimal, non-functional) method. A test method should then be written and the test-runner used to demonstrate that the test fails. Following failure, the minimal amount of code should be written in the Java class to ensure that the test functions correctly. This iteration should be repeated twice as requested in the question.

To provide an example:

**package** com.ee417;

**public** **class** Client{

**private** String firstname;

**private** String surname;

**public** Client(String firstname, String surname) {

**this**.firstname = firstname;

**this**.surname = surname;

}

**public** String getFirstname() { **return** firstname; }

**public** **void** setFirstname(String firstname) { **this**.firstname = firstname; }

**public** String getSurname() { **return** surname; }

**public** **void** setSurname(String surname) { **this**.surname = surname; }

**public** String getInitials() { **return** **null**; }// empty for now

**public** **void** setInitials(String initials) {} // empty for now

}

Now we will write a test:

**import org.junit.Test;**

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

**public class ClientTest {**

**@Test**

**public void testInitialsStandard() {**

**System.out.println("ClientTest: Inside testIntialsStandard()");**

Client client = new Client("David", "Molloy");

    assertEquals("DM",client.getInitials());

**}**

**}**

This test will fail initially. This can be demonstrated using the TestRunner class.

Now we implement the code to fix up this example:

public class Client{

     private String firstname;

     private String surname;

     private String initials;

     public Client(String firstname, String surname) {

         this.firstname = firstname;

         this.surname = surname;

     this.initials = firstname.substring(0,1) + surname.substring(0,1);

     }

      public String getFirstname() {   return firstname;   }

      public void setFirstname(String firstname) {   this.firstname =

firstname;     }

      public String getSurname() {   return surname;     }

      public void setSurname(String surname) {    this.surname =

surname;     }

   public String getInitials() { return initials; }

public void setInitials(String initials)

{    this.initials=initials;}

}

Now let’s add in a second test – to test the scenario where a null value is passed in for firstname or surname:

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class ClientTest {

  @Test

  public void testInitialsStandard() {

System.out.println("ClientTest: Inside testIntialsStandard()");

  Client client = new Client("David", "Molloy");

    assertEquals("DM",client.getInitials());

  }

  @Test

  public void testGetInitialsWhereNull() {

    System.out.println("ClientTest: Inside testGetInitialsWhereNull()");

    Client client = new Client(null, null);

    assertEquals("",client.getInitials());

    client= new Client("David", null);

    assertEquals("D",client.getInitials());

    client= new Client(null, "Molloy");

    assertEquals("M",client.getInitials());

 }

}

This test is run and should fail!

So now we write our code to satisfy this test:

public class Client{

     private String firstname;

     private String surname;

     private String initials;

     public Client(String firstname, String surname) {

         this.firstname = firstname;

         this.surname = surname;

         initials = "";

     if (firstname!=null) initials = initials + firstname.substring(0,1);

         if (surname!=null) initials = initials + surname.substring(0,1);

     }

     public String getFirstname() {   return firstname;   }

     public void setFirstname(String firstname) {   this.firstname = firstname;     }

     public String getSurname() {   return surname;     }

     public void setSurname(String surname) {      this.surname = surname;     }

  public String getInitials() { return initials; }

  public void setInitials(String initials) {  this.initials=initials;}

}

We run our tests and all pass. This completes our second iteration.

### Q 4(b) [6 Marks]

The PAD architecture refers to the logical separation of Presentation, Application and Data within a client/server information system. The Presentation refers to the user interface, application refers to the business logic at the center of the system and the data refers to the data store (e.g. database) for the system.

This logical separation is better achieved in three-tier systems than in two tier systems as shown in the diagram below. By separating into three tiers it means our presentation logic can be handled on the client, business logic on an application server and database logic on a database server. This gives a clear separation of responsibilities and improves later modification or replacement of tiers. E.g. by separating the presentation logic from the business logic we can easily create new UIs for future devices.

# http://sites.google.com/site/webappdevdcu/_/rsrc/1299580712551/course-content/section-1-introduction-to-web-application-development/2tier_vs_3tier.gif

## [End of Solutions of Question 4]

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

### Q 5(a) [11 Marks]

Discussion of this network in terms of:

* Application server is in the public network, open to potential attack from outside, visible IP address
* Database server could be nailed down further to only allow port 1521 requests and only from the ApplicationServerSecurityGroup.
* Positive that we have at least put the database server into a private subnet, which means it can’t be accessed from outside

To consider it to be production level:

* We should look at moving the application server to a private subnet
* Create a single load balancer instead as our only public interface, hiding our server IP and connectivity
* Need to use a Bastion connection approach to make connections with this server
* Need to use NAT Gateway approach in order to allow the application server to perform updates

### Q 5(b) [8 Marks]

**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) [6 Marks]

### Modify the HibernateUtil to add the main method to do the following:

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

System.***err***.println("Recreating database!");

HibernateUtil.*recreateDatabase*();

Session session = *beginTransaction*();

Account account1=**new** Account(**true**,"Current","Euro",(**float**)123.45,**new** Date());

session.save(account1);

Account account2=**new** Account(**true**,"Savings","Euro",(**float**)43.05,**new** Date());

session.save(account2);

Account account3=**new** Account(**false**,"Current","Sterling",(**float**)0,**new** Date());

session.save(account3);

*commitTransaction*();

System.***err***.println("Completed!");

}

## [End of Solutions of Question 5]

## [END OF SOLUTIONS]