**CS544**

**Enterprise Architecture**

## Exam 2 April 2020

Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student ID \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**NOTE: This material is private and confidential. It is the property of MUM and is not to be disseminated.**

1. 1. Identify which of the following are TRUE/FALSE.EXPLAIN.  
     
   1. RESTful web services are stateless.

True: RESTful Web Service should not keep a client state on the server. That is why it is statelessness.  
  
2. The REST operations, GET and a DELETE are both are idempotent. Neither one of them  
changes the state of the server.

True: The GET, PUT and DELETE methods are idempotent, meaning that applying them multiple times to a resource result in the same state change of the resource as applying them once, though the response might differ.

AOP is a Spring Core Technology. It is used in numerous places with the Spring Framework, itself. For example, a client application needs to access a server application over the network. For monitoring purposes, it is necessary to log calls to save [save(Object object) ] methods at the service tier.  
Consider the following example of ***one***of the join points:  
  
Class FooServiceImpl {  
  
***public void save (Foo foo) {  
      fooDao.save (foo);  
}***  
Public List<Foo> findAll() {  
return fooDao.findAll();  
}  
  
Public Foo findOne(Long id) {  
return fooDao.findAll(id);  
}  
}  
  
Given this example of one of the join points, Implement the necessary pointcut[s], advice and advice method signature to complete the Aspect.

***Explain your solution using the appropriate AOP terminology.***

ANSWER GOES HERE

AOP is Aspect oriented programming that is defined as the implementation of cross cutting concerns. In OOP we faced problem with scattering and tangling type codes. Using AOP solve the problem. It centralize and modularize codes that tangled and scattered so you can easily maintain them and change in single place.

AOP is mostly used to provide declarative enterprise services such as declarative transaction

management and it allows users to implement custom aspects.

AOP implemented static and dynamic ways. Static is hard code and it is become part of your code so it is give better performance. (compile time weaving) Dynamic AOP is Spring AOP that is a proxy-based approach.

That is it wraps the orginal object by a proxy object. At the time of DAO injection under service object spring find out that there is some aspect is configured for DAO, so it injects the proxy object instead of the actual object. Now when the actual call is made to any method inside DAO, proxy applies the aspect and then call the actual target object. This is called runtime waving.

@Aspect

@component

public class AspectLogging(){

@PointCut(“execution(\* edu.mum.service..save(object)”)

public void loggingAdvice(Object object){}

@Before(“loggingadvice(object)”)

public void saveAdvice(JoinPoint joinpoint, Object object) throws Throwable{

***AOP terminology***

* **Join Point** is any point in your program such as method execution, exception handling, field access etc. Spring supports only method execution join point. Our case it is edu.mum.service inside save method with object parameter.
* **Advice** represents an action taken by an aspect at a particular join point. There are different types of advices: It is called also additional behavior. Our case loggingAdvice(Object object) it is give logging behavior.
* **Pointcut** is an expression language of AOP that matches join points. / @PointCut(“execution(\* edu.mum.service..save(object)”) /

1. RBAC [Role Based Access Control] is widely used in Corporate Enterprises.  
   It has its limitation.  
     
   Explain the alternative Access Control Methodology, ABAC based on RBAC.  
   Be specific. Give an example [use case/scenario].  
   Diagrams are excellent but be sure to explain them.  
   What are the Authorization features of Spring that accommodate an ABAC implementation?

ANSWER GOES HERE

Role based access control (RBAC) is helps reduce the complexity of security administration and supports the review of permissions assigned to users.

RBAC Model:Users are assigned to Roles. Permissions are given to a Roles. User has one or more Roles. RBAC is essentially a subset of ABAC with one attribute [Role].

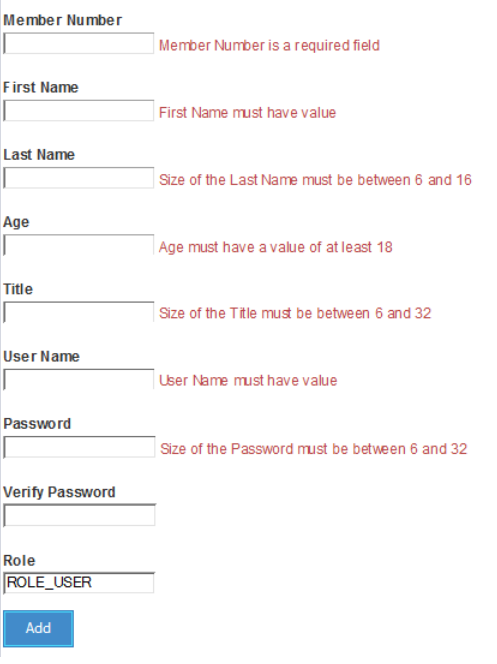
It has difficulty of setting up an initial role structure and inflexibility in changing domains. Attribute-based access control (ABAC) make RBAC more simple and flexible. ABAC is a rule-based approach to access control that can be easy to set up but complex to manage.

Attribute Based Access Control: Permissions are given based on user attributes, object attributes and environmental condition with requirements specified in access control policies.

1. The following screen displays the member entry screen. Annotate The Member & User Credentials domain

objects to reflect the Validation requirements.

***NOTE: ANSWER goes in ATTACHMENT***



@Entity

**public** **class** Member {

@Id

@GeneratedValue(strategy=GenerationType.***AUTO***)

**private** **long** id;

@Column(length = 16)

@NotEmpty(message= "{NotEmpty}")

**private** String firstName;

@Column(length = 16)

@Size(min=6, max = 16, message= "{Size.name.validation}")

**private** String lastName;

@Min(value = 18, message = "{MinNumber}")

**private** Integer age;

@Column(length = 32)

@Size(min=6, max = 32, message= "{Size.name.validation}")

**private** String title;

@NotEmpty(message= "{NotEmpty}")

**private** Integer memberNumber;

@OneToOne(fetch=FetchType.***EAGER***, cascade = CascadeType.***ALL***)

@JoinColumn(name="member\_id")

@Valid

UserCredentials userCredentials;

**Here is the UserCredentials:**

@Entity(name = "Authentication")

**public** **class** UserCredentials {

@Id

@Column(name = "USER", nullable = **false**, unique = **true**, length = 127)

@NotEmpty(message= "{NotEmpty}")

String userName;

@Column(name = "PASSWORD", nullable = **false**, length = 32)

@Size(min=6, max = 32, message= "{Size.name.validation}")

String password;

@Column( nullable = **false**, length = 32)

String verifyPassword;

Boolean enabled;

@OneToOne(mappedBy="userCredentials", cascade = {CascadeType.***PERSIST***, CascadeType.***MERGE***})

**private** User user;

**ErrorMessage.properties**

NotNull= **{0}** is a required field

NotEmpty= **{0}** field must have a value

Size.name.validation = Size of the **{0}** must be between **{2}** and **{1}**

MinNumber = **{0}** must have a valueof at least **{1}**