**CS544**

**Enterprise Architecture Final**

## True False Explain

T **F** Spring Enterprise Integration is essentially another name for Spring Remoting [JMS,AMQP,HTTP etc.]

**EXPLAIN:**

**False**: Spring features integration classes fo~~r~~ remoting support using various technologies. The remoting support eases the development of remote-enabled services, implemented by your usual (Spring) POJOs**.** Spring Enterprise Integration high-level abstraction over spring remoting.

T F Spring Integration is based on lightweight messaging which is based on JMS.

**EXPLAIN:**

**False** :Spring integration is yes, it is a lightweight messaging but is based on some declarative adapters like JMS, AMPQ and others not only JMS.

T F A Content Router basically determines routing based on IF-THEN-ELSE logic.

**EXPLAIN:**

**True**: What Content router does is that it determines if the message header and payload satisfies a condition and based on the satisfaction of the condition it will pass it to respective channel. The If Else condition might not be always one it will have continuity like IF-THEN-ELSE-IF-THEN-ELSE

T F - A Message Channel is a core component of Spring Integration

**EXPLAIN: False:** Spring integration has three main/core components Message, Message channel and Message Endpoint

T F A basic capability of ESB is transport conversion.

**EXPLAIN:**

True : ESB:- Enterprise service bus helps in converting message from one type to another while transporting it the other outlet or destination.

T **F** Spring security only supports the authentication model HTTP Basic defined by RFC 1945 which is the most popular authentication mechanism in the web.

**EXPLAIN**:

**False**: Not only HTTP Basic, it also supports HTTP Digest, HTTP X.509, JOSSO etc

**T** F Spring Security Groups can be used to implement RBAC within the Spring Framework.

**EXPLAIN**:

**True**: Spring security group allow authorization based on role level of the group.

T **F** Digest authentication uses Base64 encoding to transmit encrypted username/password

**EXPLAIN**:

**False**: IT is not Digest, it is Basic authentication that uses Base64 encoding to transmit username/password as plaint text

T **F** Authorization refers to validating unique identifying information about each system user.

**EXPLAIN:**

**False:** NO this is authentication. Authorization is the process of allowing a user to access a resource.

T **F** Permission Based Access allows for fine-grained access control.

**EXPLAIN:**

**False-** NO, custom based rule is the one that allow fine-grained access control.

T F Spring ACL is used to give permissions to access methods (i.e. specify who can execute which method)

EXPLAIN:

**T** F The ACLs or access control lists are specifically for assuring domain object security

EXPLAIN:

**True**: Access Control List (ACL) is a list of permissions attached to an object. An ACL specifies which identities are granted which operations on a given object.

Spring Security Access Control List is a Spring component which supports Domain Object Security. Simply put, Spring ACL helps in defining permissions for specific user/role on a single domain object – instead of across the board, at the typical per-operation level.

## 2. [10 points] Consider the following AOP Aspect:

**Create an Advice that executes before an** *annotated* **application joinpoint that has a pointcut that would identify service methods that have a OrderItem object as a parameter.**

**Note you need to create an additional pointcut AND the Advice.**

**The Advice method should calculate and print out the cost of the inventory using quantity and price fields from OrderItem.**

**Also show a complete implementation of an example join point.**

Two Pointcuts are provided for your use.

@Aspect

@Component

# AccountingAspect.java

@Pointcut("execution(\* edu.mum.service..\*(..))") **public** **void** accountingMethod() {}

@Pointcut("@annotation(edu.mum.validation.Accounting)")

**public** **void** accounting() {}

answer:

@Pointcut("args(orderItem)")

public void testOrderItem(OrderItem orderItem) {}

@Before("accountingMethod() && accounting() && testOrderItem(orderItem)")

public void adviceMethod(OrderItem orderItem) {

System.out.println();

System.out.println("Cost of inventory is: "+orderItem.getQuantity\*orderItem.getPrice);

System.out.println("Order Number: " + orderItem.getOrderNumber());

}

edu.mum.service.impl.OrderServiceImpl

@Accounting

public Order update(Order order) {

return orderDao.update(order);

}

1. [20 points] Spring has 3 SPEL

expressions that can be used to provide annotation-based access control on Service methods. We commonly used @PreAuthorize as the annotation.

* + 1. Explain the use & give examples of the 3 expressions using the @PreAuthorize.
    2. Explain how Spring facilitates RBAC [include the use of 1 of the 3 expressions] ***by example***
    3. Explain how Spring facilitates ABAC [include the use of 1 of the 3 expressions] ***with example***
  1. @PreAuthorize(“hasRole(‘ROLE\_ADMIN’)”): grants authorization to the specified method based on the users authority level, checks if @PreAuthorize

(“hasAuthority(‘READ)”): grants authorization for the user based on the users permission level. The permissions can be assigned to a group or not. @PreAuthorize (“hasPermission(‘#comment, ‘update’)”): the authorization level is based on custom attributes that can be a constraint in the application.

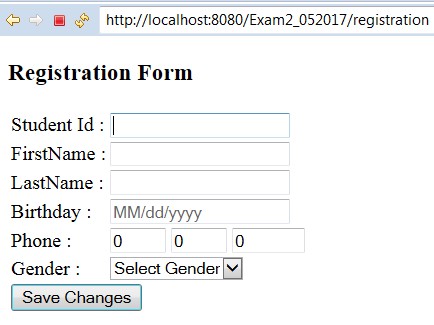
* 1. Role Based Access control: permissions are given to roles instead of users. Users have roles. It is better for a group of users. hasAuthority(‘READ)
  2. Attribute Based Access Control: Permissions are given based on user attributes, object attributes and environmental condition with requirements specified in access control policies.

1. [15 Points] This is a student Registration form.

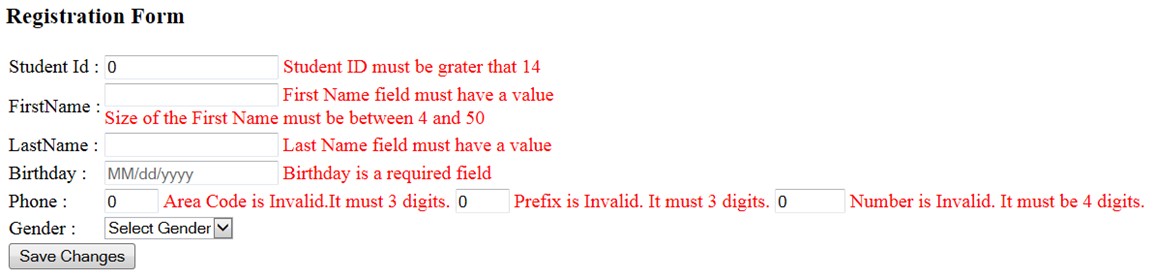
There is validation required before the student can be entered into the system successfully. If the student information is entered correctly then a JSP page success.jsp is displayed. Below you can see the error messages resulting from wrong input. Fill in ALL the content of the supplied resources. USE BEST PRACTICES…

**NOTE: In the interest of time, you only need to annotate ONE of the phone fields [all 3 look about the same]. For “3 digits”, test for values between 100 & 999.**

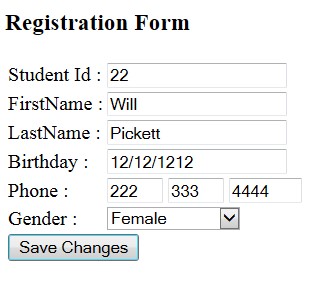
**INITIAL SCREEN:**



**NO INPUT & “SAVE CHANGES”**



**WITH VALID INPUT & “SAVE CHANGES”**



**SUCCESSFUL “SAVE CHANGES”**



# StudentController.java

@RequestMapping(value = “”, Method = RequestMethod.Get)

**public** String showForm(@ModelAttribute("newStudent") Student newStudent){

return “addStudent”;

}

@RequestMapping(value = "", method = RequestMethod.***POST***) **public** String processForm (@Valid @ModelAttribute("newStudent") Student studentToBeAdded, BindingResult result){

**if**(result.hasErrors)

**return** “addStudent”;

studentService.saveFull(studentToBeAdded);

**return** “redirect:/success/{studentToBeAdded.getId}”;

}

@RequestMapping(“/success/{id}”)

**public** String success(@PathVariable(“id”) Model model) { model.addAtribute(“newStudent”, studentService.findOne(id));

**return** “student”;

}

# Student.java

**public** **class** Student {

@Min(value = 14)

## private int StudentId;

@NotEmpty

@Size(min=4, max=50, message = “{Size.validation}” **private** String firstName = **null**;

@NotEmpty

**private** String lastName = **null**;

## private String gender = null;

@NotNull

**private** Date birthday;

@Valid

**private** Phone phone;

## Phone.java

### public class Phone {

@Range(100-999)

**private** Integer area;

**private** Integer prefix;

**private** Integer number;

# Errormessages.properties

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

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

Min= **{0}** must be greater than **{1}** Range = **{0}** is invalid. It must 3 digits.

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

StudentId = Student ID firstName = First Name lastName = Last Name birthday = Birthday

phone.area = Area Code

5. [20 points] The validation for the Student Registration in the previous problem [#4] deals with a Web UI scenario. Consider 2 other alternative scenarios for Student Registration: A REST service and a Batch service.

BATCH simply requires validating: lastName & phone

REST requires: ALL fields to be validated: The fields in problem 4 PLUS gender field. Assume the gender field requires a value.

Fields to be validated & when to validate them:

|  |  |  |  |
| --- | --- | --- | --- |
|  | BATCH | WEB UI | REST |
| studentId |  | **X** | **X** |
| firstName |  | **X** | **X** |
| lastName | X | **X** | **X** |
| gender |  |  | **X** |
| birthday |  | **X** | **X** |
| phone | X | **X** | **X** |

**Implement** Validation groups for the 3 options. Give examples of annotating the fields for each group. For example annotate firstName, lastName, gender.

WEB UI & REST can use the Spring MVC validation functionality for handling groups because they are HTTP based. On the other hand, BATCH group validation uses the Hibernate Validator. It is on a different layer of the N-tier architecture.

**Explain** a robust validation strategy related to an N-tier architecture. Describe the value of validation at different layers in the N-tier.

6. [15 points] Complete the following AMQP messaging application.

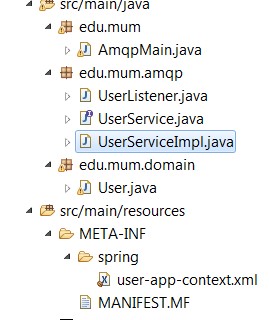
The project structure and UserService interface are provided.

To be completed:

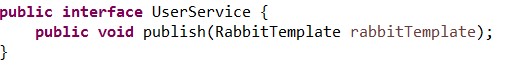
* user-app-context.xml
* UserServiceImpl.java
* UserListener.java

It is a direct Exchange. The exchange name is ***user***. There is a single queue named ***userQueue***. The binding between the exchange & queue is through ***user.key***.

**Project Structure:**



## UserService.java



## user-app-context.xml Fill in the parts indicated by underline \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<rabbit:connection-factory id=*"connectionFactory"* host=*"localhost"* username=*"joe"* password=*"joe"*/> <rabbit:admin connection-factory=*"connectionFactory"* />

**<!-- \*\*\*\*\*\*\*\*\*\*\* EXCHANGE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->**

<rabbit:queue name=*"\_\_\_\_\_\_\_\_userQueue\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"* durable=*"true"*/>

<rabbit:direct-exchange name=*"\_\_\_\_\_\_user\_\_\_\_\_\_\_\_\_\_\_\_\_\_"* durable=*"true"*>

<rabbit:bindings>

<rabbit:binding \_\_queue\_\_=“\_\_userQueue\_\_\_” key\_\_\_\_=“\_\_\_user.key\_\_\_\_” />

</rabbit:bindings>

</rabbit:direct-exchange>

**<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PRODUCER -->**  associated with it...] -->

<rabbit:template id=*"userTemplate"* connection-factory=*"connectionFactory"*

reply-timeout=*"2000"* routing-key = *"\_\_\_\_\_\_user.key\_\_\_\_\_\_\_\_\_\_\_"*

exchange=*"\_\_\_\_\_\_\_user\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"* />

**<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CONSUMER \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->**

<rabbit:listener-container connection-factory=*"connectionFactory"*>

<rabbit:listener ref=*"queueListener"* method=*"\_onMessage\_"* queue-names=*"\_\_userQueue\_\_\_\_\_"* </rabbit:listener-container>

**<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LISTENER -->**

<bean id=*"\_\_ userListener \_\_\_"* class=*"\_\_\_\_\_edu.mum.ampq.UserListener\_\_\_\_\_\_\_"* />

## UserServiceImpl.java

**public** **class** UserServiceImpl **implements** UserService { **public** **void** publish(RabbitTemplate rabbitTemplate) {

// Send 2 User messages

User user = **new** User("Bill", "Mee", "bMee@usa.com");

rabbitTemplate.convertAndSend(user);

user = **new** User("Paul", "Tree", "pTree@mav.com");

rabbitTemplate.convertAndSend(user);

} }

## UserListener.java

**public** **class** UserListener {

**public** **void** onMessage(User user) {

System.***out***.println("AMQP User on DIRECT Queue: " + user.getFirstName);

}

1. [15 points] AOP is a Spring Core Technology**.**

**It is used in numerous places within the Spring Framework, itself. Explain the fundamentals of Spring’s AOP implementation; how it works, how it relates to AspectJ, with examples of its usage within Spring.**

**To help in your explanation of how it works consider the following use case:**

**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. Using AOP terminology, describe what would need to be implemented. Be specific with respect to Pointcut & Advice syntax.**

**For example:**

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);

}

}

**ANSWER:**

AOP is Aspect oriented programming that is defined as the implementation of cross cutting concerns.it means it defines a in one place, functionality that is need to be in a multiple places in the code through out the application.

Aop supports separation of concern that is it separate the business logic from the cross cutting concerns. It centralize and modularize codes that are intermixed and scattered through out the application so that you can easily maintain them and change in single place.

Spring Aop 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.

Like Spring AOP, AspectJ is implementation of cross cutting concern. But is is intended to be the complete AOP solution. Unlike spring AOP , Aspectj supports compile time waving and and field level advice. It is more powerful but complex than spring AOP. It works for any java code, not only in spring managed Bean as of Spring AOP.

@Aspect

@component

public class AspectLogging(){

@PointCut(“execution(\* edu.mum.service..save(object)”) public void loggingadvice(Object object){}

@Before(“loggingadvice(object)”)

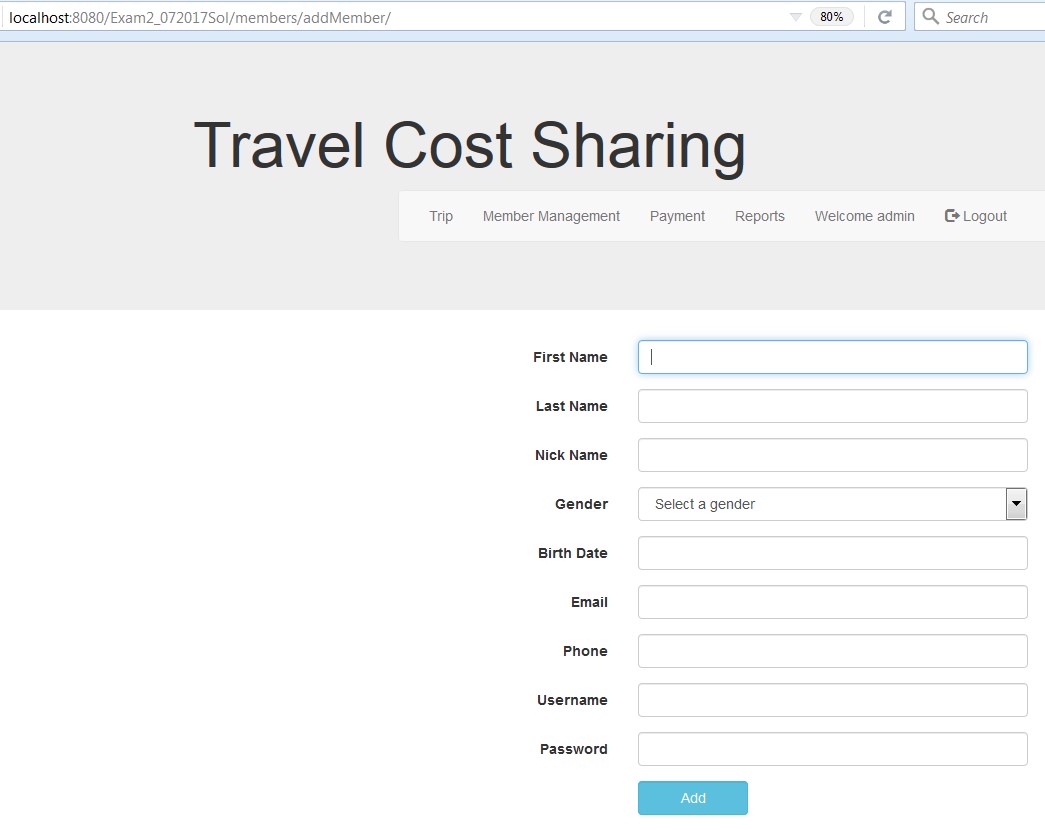
public void saveAdvice(JoinPoint joinpoint, Object object) throws Throwable{

}

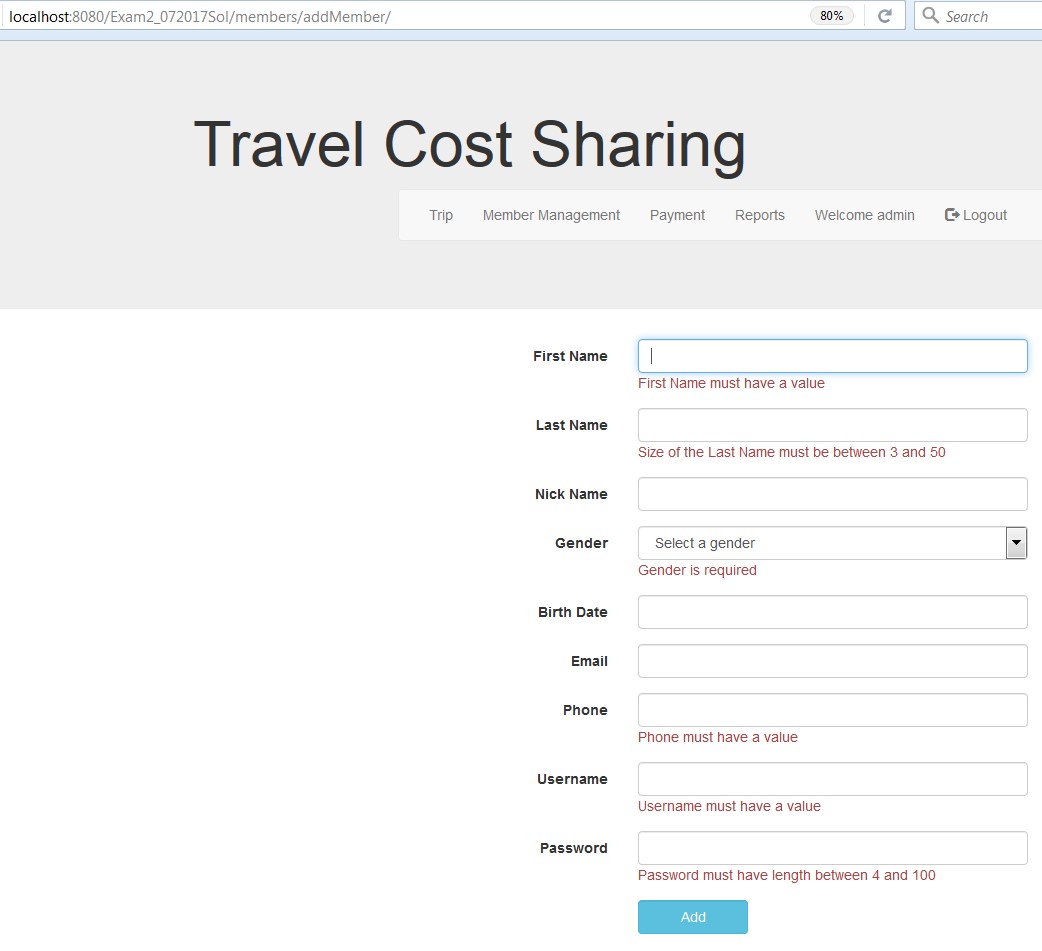
## [20 Points] This is a Member Registration form.

## There is validation required before the member can be entered into the system successfully. If the member information is entered correctly then a JSP page members.jsp is displayed. Below you can see the error messages resulting from wrong input. Fill in ALL the content of the supplied resources. USE BEST PRACTICES…

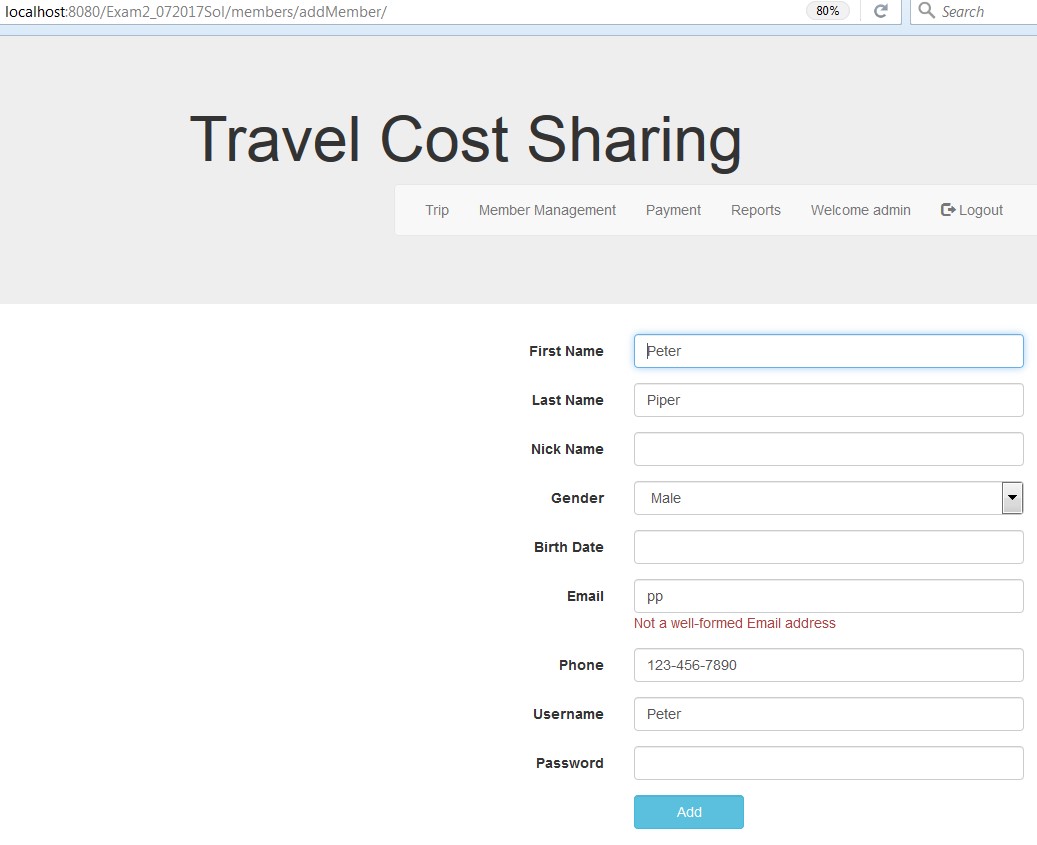
**INITIAL SCREEN:**



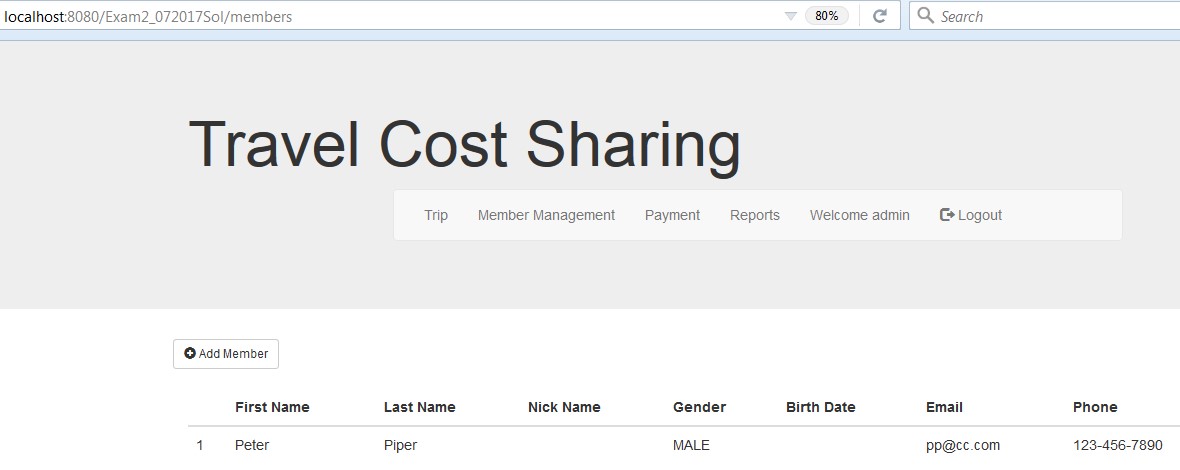
**NO INPUT & “ADD CHANGES”**



**Invalid Email input**



**Successful Member Addition:**



**MemberController.java**

@Controller

@RequestMapping(“/members”)

**public** **class** MemberController {

@Autowired

private MemberService memberService;

@RequestMapping(value=””, method=RequestMethod.GET) **public** String getMembers( Model model ) {

model.addAttribute(“listMember”, memberService.findAll());

return = “members”;

}

@RequestMapping(value=”/add”, method=RequestMethod.GET)

**public** String addMember( @ModelAttribute(“newMember”) Member member) {

return “addMember”;

}

@RequestMapping(value=”/add”, method=RequestMethod.POST)

**public** String addMember(@ModelAttribute(“newMember”) @ValidMember member, BindingResutl result) {

if(result.hasErrors()){

return “addMember”;

}

memberService.save(member);

return “redirect:/members”;

}

**Here is the relevant part of the Member Domain Class:**

@Entity **public** **class** Member {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***) **private** Long id;

@NotEmpty

@Column(length = 20)

**private** String firstName;

// @Size(min =3, max=50, message =”{lastName.Size}”)

@EmptyOrSize(min = 3, max = 50, message = "{size.name.validation}") **private** String lastName;

**private** String nickName;

@NotNull

**private** Gender gender;

@Email

**private** String email;

@NotEmpty

**private** String phone;

@Temporal(TemporalType.***DATE***) @DateTimeFormat(pattern = "yyyy-MM-dd") **private** Date birthDate;

@valid

@OneToOne(cascade = CascadeType.***ALL***)

@JoinColumn(name = "credential\_id")

Credential credential;

**Here is the Credentials Class:**

@Entity

**public** **class** Credential {

@NotEmpty

@Id

**private** String username;

@Size(min-4, max=100, message =”{password.Size}”)

**private** String password;

**private** Boolean enabled = Boolean.***TRUE***;

@OneToOne(mappedBy = "credential", cascade = CascadeType.***PERSIST***) **private** Member member;

@valid

@OneToMany(fetch = FetchType.***EAGER***, cascade = CascadeType.***ALL***, orphanRemoval = **true**)

@JoinColumn(name = "username")

**private** List<Authority> authorities = **new** ArrayList<Authority>();

@Transient

**private** List<String> authorityList = **new** ArrayList<>();

**ErrorMessage.properties**

NotEmpty = {0} field must have a value NotNull = {0} is required

size.name.validation = Size of the {0} must be between {2} and {1} Email = {0} must have a valid syntax

password.Size = {0} must have a length between {2} and {1}

firstName = First Name lastName = Last Name nickname = Nick Name birthdate = Birth Date gender = Gender email = Email phone = Phone

1. [15 points] Messaging is basic to scalable enterprise architectures**.**

**We covered two messaging technologies, JMS & AMQP. Explain the fundamentals of messaging.**

**Be sure to cover: the types of messaging, the messaging architecture, and the differences between the two, JMS & AMQP and how they are implemented.**

**Be specific. Give examples. Diagrams are good; be sure to explain them.**

**Answer:**

Messaging is a loosely coupled asynchronous and reliable communication between applications.

The main type of messaging are

1. point to point
2. publish –subscribe

**Point to point**

Message is sent from one application to another application via a queue. Here there is only one consumer but we can have multiple producers.

**Publish – subscribe**

Message (publish) is sent from producer (publisher) to multiple consumers (subscribers) via topic. Here there is only one producer.

The publisher is not required to know information about the subscribers. It is easily deployable, dynamic and flexible.

**The main difference between JMS and AMQP**

* 1. JMS has queue and topic but AMQP has only queue and exchange.
  2. Unlike JMS, in AMQP message is not published directly to the queue, it is first publishes to the exchange.
  3. JMS is an API and AMQP is a protocol. Therefore, JMS client can use AMQP as a protocol.
  4. Unlike AMQP, JMS is not required to know how the message is formed or created.
  5. AMQP supports multiple languages and it is a cross-platform messaging standard.
  6. AMQP has a high quality messaging system and more secured than JMS.

1. [15 points] Enterprise Integration Patterns [EIP**]**

**are a fundamental definition of how to do integration in a company of any significant size. Spring Integration implements those patterns.**

**Explain the fundamental aspects of Spring Integration.**

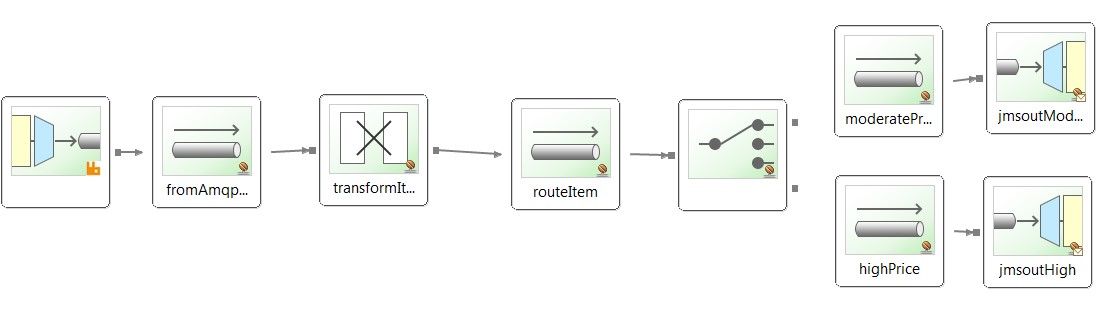
**Why is it necessary & valuable?**

**Describe the 3 main components.**

**Drawing on the demo from class [Routing an order through the “enterprise”], give details on some of the EIP components.**

**Be specific. Give examples. Diagrams are good but be sure to explain them.**

**Here is a diagram that you should use to describe [some], components and an ESB type flow:**



**Answer:**

Spring integration is a concise and clear approach of Enterprise Application Integration. It is a lightweight messaging system that allow integration capability on our spring application. As a messaging strategy, it allows a transfer of information quickly and with a high level of decupling between different components in our application.

Provide a simple model for implementing complex enterprise integration solutions

Facilitate asynchronous, message-driven behavior within a Spring-based application.

Promote intuitive, incremental adoption for existing Spring users.

Main components are:

Message: contains information that is transferred between different part of an application or sent to an external system. It consists of Header and payload. Header is a wrapper of map that contains meta-information. Payload is a pure java class that contains information.

Message Channel: it is a pipe that contains two end-points and through which message travel.

Message Endpoint: it is an abstraction layer between application code and messaging framework. It is message consumers and message producers.

Components:

Channel adapter: Connects the application to an external system (unidirectional).

Gateway: Connects the application to an external system (bidirectional).

Service Activator: Can invoke an operation on a service object.

Transformer: Converts the content of a message.

Filter: Determines if a message can continue its way to the output channel.

Router: Decides to which channel the message will be sent.

Splitter: Splits the message in several parts.

Aggregator: Combines several messages into a single one.

1. [15 points] The Spring framework is the “example” architecture that we used in this course.

It emphasizes good design, best practices and use of design patterns.

Explain the value of the framework. Things to consider:

N-Tier; Separation of Concerns; Different types of N-tier; Distributed capabilities;

The characteristics & value of a framework.

Be specific. Give examples. Diagrams are good but be sure to explain them.

**Answer:**

Spring Framework Is a lightweight architecture with there tire architecture mainly called presentation tier, business tier and persistent tier.

Spring framework make enterprise java application development as easy as possible by following good programming practice such as

1. POSO based programming

2. Separation of concern

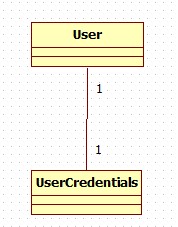
3. Flexibility.

The advantages of spring framework are:

1. Spring make application development simple
2. DI give flexibility in bean wiring
3. Separates business logic with cross cutting concern
4. Adhere DRY principle
5. Increase reliability and reduce programming time.
6. Helps enforce best practice and rules
7. Standard based
8. Design pattern based.

12. [15 Points] Annotate for validation both the User and UserCredentials from the Auction System. The action that triggers validation is an invocation of - userService.save(user);

It is NOT necessary to invoke validation, just annotate the domain Models AND externalize the messages in errorMessages.properties [space left at end of question.]



Here are the generated error messages when validation fails:

First Name field must have a value

Size of the Last Name must be between 5 and 9

Email must have valid syntax

Ranking must be between 4 and 6

Password must have at least 6 characters

Size of the Login User Name must be between 6 and 16

**Here is the relevant part of the User Domain Class:**

@Entity

@Table(name = "USERS")

**public** **class** User **implements** Serializable {

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

@Column(name = "USER\_ID") **private** Long id = **null**;

@Version

**private** **int** version = 0;

@Column(name = "FIRSTNAME", nullable = **false**) **private** String firstName;

@Column(name = "LASTNAME", nullable = **false**) **private** String lastName;

@Column(name = "EMAIL", nullable = **false**) **private** String email;

@Column(name = "RANK", nullable = **false**) **private** **int** ranking = 0;

@Column(name = "IS\_ADMIN", nullable = **false**) **private** **boolean** admin = **false**;

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

@JoinColumn(name="userId")

**private** UserCredentials userCredentials;

**Here is the UserCredentials:**

@Entity(name = "Authentication") **public** **class** UserCredentials {

@Id

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

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

String password;

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

String verifyPassword;

Boolean enabled;

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

**ErrorMessage.properties**