

Final exam

[10 minutes]

Explain clearly the difference between **conversation scope** and **flow scope** in **Spring WebFlow**

Data in conversation scope can also be accessed by the subflows. Data in flow scope cannot be accessed by subflows.

[10 minutes]

We learned that writing a web application with spring webflow is different than writing a web application with spring MVC. Explain clearly when you would choose for spring webflow compared to Spring MVC. In other words, for what kind of applications you would use Spring webflow?

Applications that have complex flow logic where what is the next page is decided by some conditional logic.

[35 minutes]

Suppose we have to develop a small SpringMVC REST application for a bank account application. A bank account consists of a accountNumber, accountHolderName and balance. Accounts should be stored in the database using SpringJPA.

Write the code for **all necessary classes** including the Spring annotations. Use the best practices we have learned in this course. Do NOT write getter and setter methods. The application is called through its REST interface. The rest interface should have the following methods:

```
void createAccount(int accountNumber, String accountHolderName)
Product getAccount(int accountNumber)
void deposit(int accountNumber, double amount)
void withdraw(int accountNumber, double amount)
void removeAccount(int accountNumber)
List<Product> findAccountsByAccountHolder(string accountHolderName)
```

```
@Entity
public class Account{

    @Id
    private int accountNumber;
    private String accountHolderName ;
    private double balance;

    public void deposit(double amount) {
        balance=balance+amount;
    }

    public void withdraw(double amount) {
        balance=balance-amount;
    }
    ...
}

public interface AccountRepository extends
    JpaRepository<Account, Integer>{

    List<Account> findByAccountHolderName(String name);
}
```

```

@Service
public class AccountService {

    public AccountService() {
    }

    @Autowired
    private AccountRepository accountRepository;

    public void createAccount(int accountNumber, String
accountHolderName) {
        accountRepository.save(new
Account(accountNumber, accountHolderName, 0.0));
    }

    public Account getAccount(int accountNumber) {
        Account a =
accountRepository.findById(accountNumber).get();
        System.out.println("account = "+a);
        return a;
    }

    public void deposit(int accountNumber, double amount){
        Account account = getAccount(accountNumber);
        account.deposit(amount);
        accountRepository.save(account);
    }

    public void withdraw(int accountNumber, double amount){
        Account account = getAccount(accountNumber);
        account.withdraw(amount);
        accountRepository.save(account);
    }

    public void removeAccount(int accountNumber){
        accountRepository.deleteById(accountNumber);
    }

    public List<Account> findAccountsByAccountHolder(String
accountHolderName){
        return
accountRepository.findByAccountHolderName(accountHolderName);
    }
}

```

```

@RestController
public class AccountController {
    @Autowired
    private AccountService accountService;

    @PostMapping("/create/{accountNumber}/{accountHolderName}")
    public void createAccount(@PathVariable int accountNumber,
    @PathVariable String accountHolderName) {
        accountService.createAccount(accountNumber,
accountHolderName);
    }

    @GetMapping("/{accountNumber}")
    public Account getAccount(@PathVariable int accountNumber) {
        return accountService.getAccount(accountNumber);
    }

    @PostMapping("/deposit/{accountNumber}/{amount}")
    public void deposit(@PathVariable int accountNumber,
    @PathVariable double amount){
        accountService.deposit(accountNumber, amount);
    }

    @PostMapping("/withdraw/{accountNumber}/{amount}")
    public void withdraw(@PathVariable int accountNumber,
    @PathVariable double amount){
        accountService.withdraw(accountNumber, amount);
    }

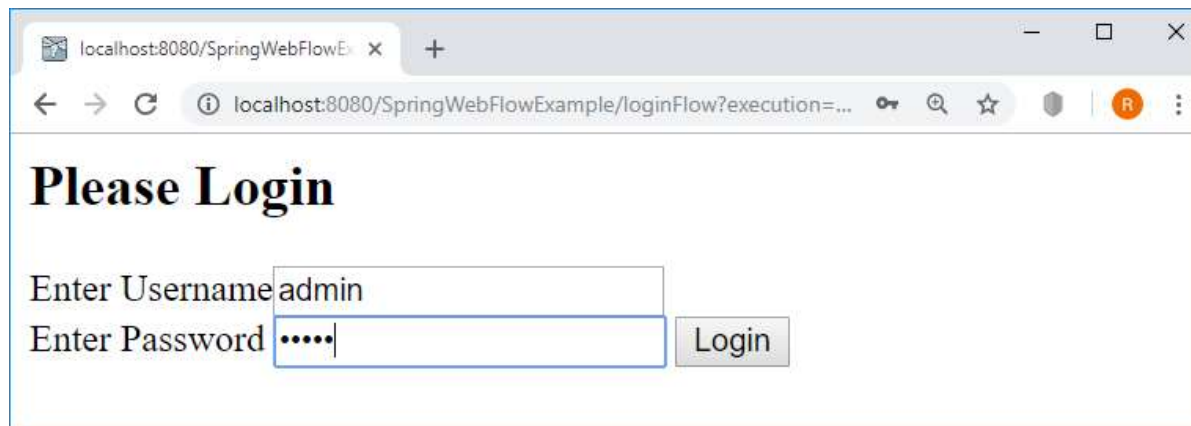
    @DeleteMapping("/{accountNumber}")
    public void removeAccount(@PathVariable int accountNumber){
        accountService.removeAccount(accountNumber);
    }

    @GetMapping("/getByHolder/{accountHolderName}")
    public List<Account>
findAccountsByAccountHolder(@PathVariable String
accountHolderName){
        return
accountService.findAccountsByAccountHolder(accountHolderName);
    }
}

```

[25 minutes]

In the lab about webflow we implemented the following application:



localhost:8080/SpringWebFlowE x +

localhost:8080/SpringWebFlowExample/loginFlow?execution=...

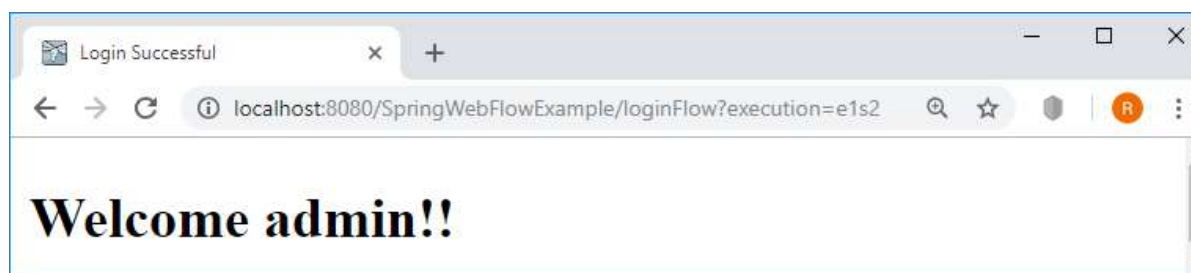
Please Login

Enter Username admin

Enter Password

Login

When you login with the correct username and password, you see the following welcome page:



When you login with the wrong username and password, you see the following error page:



The application uses the following Login domain class:

```
package edu.mum.domain;

public class Login implements Serializable{
    private String userName;
    private String password;

    //getter and setter methods are not shown
```

```
}
```

Given are the following web pages:

login.jsp

```
<html>
<body>
    <h2>Please Login</h2>
    <form method="post" action="{flowExecutionUrl}">
        <input type="hidden" name="_eventId" value="performLogin">
        <input type="hidden" name="_flowExecutionKey" value="{flowExecutionKey}" />

        Enter Username<input type="text" name="userName" maxlength="40"><br>
        Enter Password <input type="password" name="password" maxlength="40">
        <input type="submit" value="Login" />
    </form>
</body>
</html>
```

failure.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Invalid Credentials</title>
</head>
<body>
<h2>Invalid username or password. Please try again! </h2>
</body>
</html>
```

success.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Successful</title>
</head>
<body>
<h2>Welcome ${login.userName}!!</h2>
</body>
</html>
```

Also the following Spring service is given:

loginServiceImpl.java

```
@Service
public class LoginServiceImpl implements LoginService {
    public String verify(Login login){
        String userName = login.getUserName();
        String password = login.getPassword();

        if(userName.equals("admin") && password.equals("admin")){
            return "true";
        }
        else{
            return "false";
        }
    }
}
```

In sakai, write the login-flow.xml file so that the application works correctly using spring webflow.

```
<var name="login" class="edu.mum.domain.Login" />

<view-state id="displayLoginView" view="jsp/Login.jsp"
model="login">
    <transition on="performLogin" to="performLoginAction" />
</view-state>

<action-state id="performLoginAction">
    <evaluate expression="loginServiceImpl.verify(login)" />

    <transition on="true" to="displaySuccess" />
    <transition on="false" to="displayError" />

</action-state>

<view-state id="displaySuccess" view="jsp/success.jsp"
model="login"/>

<view-state id="displayError" view="jsp/failure.jsp" />
</flow>
```

[10 minutes]

Select all statements that are correct.

- ☐ A. Authorization means we need to check if the user is who he/she says he is.
- ☐ B. Spring security is based on servlet filters
- ☐ C.

To add security to a REST interface we can use form based authentication

- ☐ D. Angular version 2 or higher also supports Dependency Injection
- ☐ E. In JPA a one-to-many relationship can be mapped in the database either by a join column or a join table.
- ☐ F. In JPA a many-to-many relationship can be mapped in the database either by a join column or a join table.
- ☐ G. The Post-Redirect-Get (PRG) pattern should always be applied to a REST interface to avoid posting the same request twice
- ☐ H. Ajax calls are full duplex
- ☐ I. Reactive forms in Angular support 2 way binding
- ☐ J. Template-driven forms in Angular support 2 way binding

[5 minutes]

Suppose we have the following JPA entity class:

```
@Entity  
public class Order{  
    @Id  
    private int orderNr;  
  
    @OneToMany()  
    @JoinColumn(name="orderId")  
    private List orderlines ;  
    ...  
}
```

Select all statements that are correct

☐ A.

When we load an Order object from the database with JPA, then the corresponding orderlines are also loaded from the database

☐ B. When we load an Order object from the database with JPA, then the corresponding orderlines are not loaded from the database

☐ C. When we save an Order object to the database with JPA, then the corresponding orderlines are also saved to the database

☐ D. When we save an Order object to the database with JPA, then the corresponding orderlines are not saved to the database

☐ E. When we update an Order object in the database with JPA, then the corresponding orderlines are also updated in the database

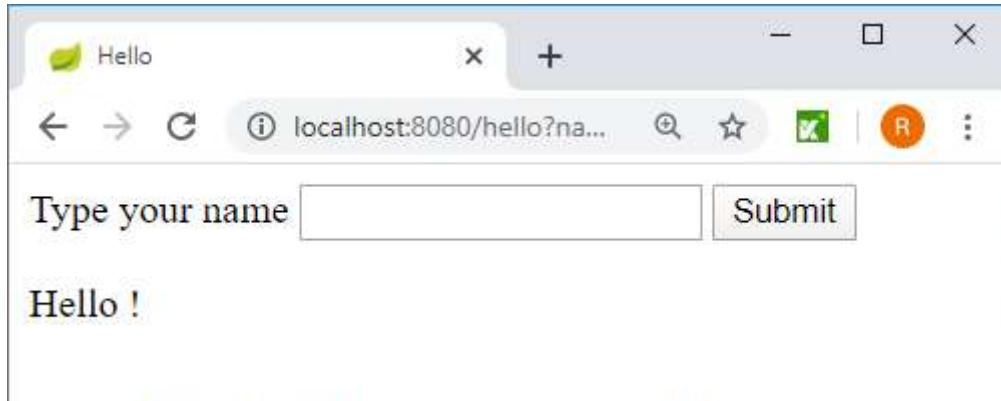
☐ F. When we update an Order object in the database with JPA, then the corresponding orderlines are not updated in the database

☐ G. When we delete an Order object from the database with JPA, then the corresponding orderlines are also deleted from the database

☐ H. When we delete an Order object from the database with JPA, then the corresponding orderlines are not deleted from the database

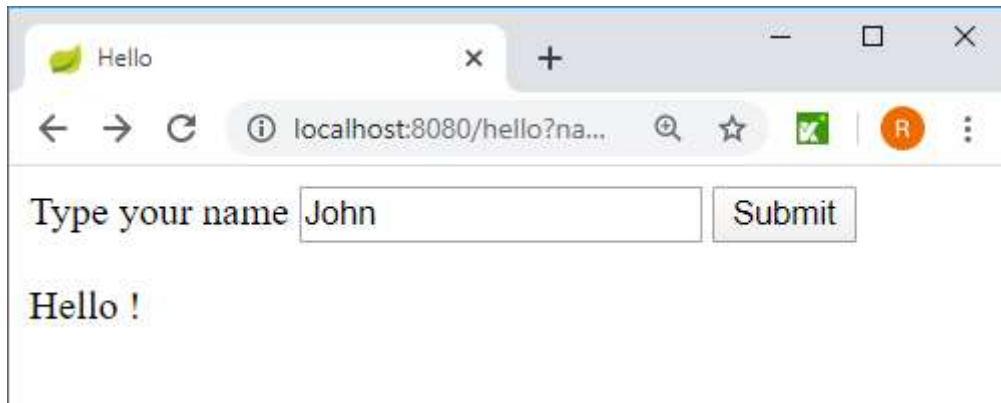
[15 minutes]

In the lab we wrote the following web application with Spring Boot and Thymeleaf:



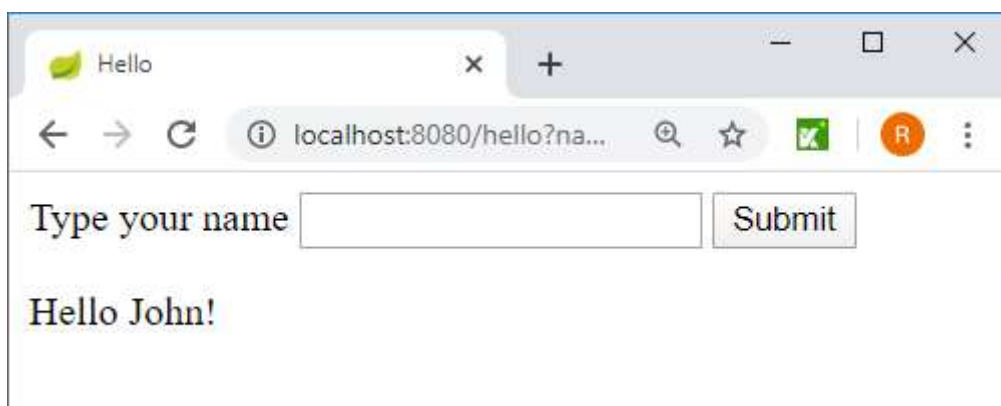
A screenshot of a web browser window. The title bar shows a green icon and the word "Hello". The address bar shows "localhost:8080/hello?na...". The page content includes a form with the label "Type your name" followed by an empty text input field and a "Submit" button. Below the form, the text "Hello !" is displayed.

If you do not type your name, the page shows the text “Hello !”



A screenshot of a web browser window. The title bar shows a green icon and the word "Hello". The address bar shows "localhost:8080/hello?na...". The page content includes a form with the label "Type your name" followed by a text input field containing the word "John" and a "Submit" button. Below the form, the text "Hello !" is displayed.

If you do type your name, the page shows “Hello <name> !”



A screenshot of a web browser window. The title bar shows a green icon and the word "Hello". The address bar shows "localhost:8080/hello?na...". The page content includes a form with the label "Type your name" followed by an empty text input field and a "Submit" button. Below the form, the text "Hello John!" is displayed.

Given is the hello.html webpage:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Hello</title>
</head>
<body>
    <form action="/hello">
        Type your name <input name="name" />
        <input type="submit" name="Say Hello" />
    </form>
    <p th:text="'Hello ' + ${name}+'!'" />
</body>
</html>
```

Write **ALL** necessary Java code so that this application works correctly according to the screenshots given above. For this exercise **you need to use Spring Boot**.

```
@Controller
public class HelloController {
    @RequestMapping("/hello")
    public String hello(Model model, @RequestParam(value =
"name", required = false) String name) {
        if (name == null) name="";
        model.addAttribute("name", name);
        return "hello";
    }
}

@SpringBootApplication
public class SpringBootProjectApplication {

    public static void main(String[] args) {

        SpringApplication.run(SpringBootProjectApplication.class,
args);
    }

}
```