Entrega y Corrección de la 2ª Iteración de la Práctica de Internet y Sistemas Distribuidos

Entrega

- Para realizar la entrega de la práctica **se creará el tag `it-2`** (segunda iteración) en el repositorio Git del proyecto.
    - La transparencia número 34 del tutorial de Git (disponible en moodle) ilustra una manera de crear un tag desde línea de comandos.
- **No se admiten modificaciones posteriores al día 18-12-2021.**
- Se recuerda que **los mensajes de los commits deberán seguir OBLIGATORIAMENTE el formato que se explica en el enunciado de la práctica.**
- **Es OBLIGATORIO tener subido al directorio raíz del repositorio Git un fichero de texto llamado "PRUEBAS.txt"** con los comandos Maven y la sentencia SQL necesarios para ejecutar las pruebas que se detallan en el apartado "Pruebas" de este documento (en ese apartado se proporcionan más detalles).

Corrección

La corrección de la práctica la realizarán los profesores de la asignatura a partir del **19 de Diciembre** (**las clases presenciales de prácticas finalizan el 16 de Diciembre**).

Para evaluar el software, el profesor lo ejecutará realizando los siguientes pasos:

- NOTA: para hacer posible que la práctica se pueda corregir en cualquier máquina (independientemente del sistema operativo instalado) de manera rápida:
    - Las BBDD deben llamarse **`ws`** y **`wstest`**, con usuario **`ws`** y contraseña **`ws`**.
    - MySQL se arrancará en el puerto por defecto.
- Utilizará su propio ordenador o portátil.
- Bajará la distribución fuente del repositorio Git. Ejemplo desde la línea de comandos:

```
git clone <<URL del repositorio git>> it-2
cd it-2
git checkout it-2
```

- Arrancará la BD (si no está ya arrancada).
- Inicializará la base de datos **`ws`**, y compilará el código de todos los subsistemas (ejecutará **`mvn sql:execute install`** desde el directorio raíz).
- Instalará la aplicación web en **Tomcat** (es decir, copiará el fichero WAR generado en **`ws-app-service/target/`** al directorio **`webapps`** de Tomcat) y lo arrancará.
- Configurará el cliente para utilizar la implementación REST de la capa Acceso a Servicios (o la implementación Thrift si se ha hecho el trabajo tutelado).
- Realizará ejecuciones de la **aplicación cliente** desde el directorio **`ws-app-client`** con los argumentos (valor del parámetro **`-Dexec.args`**) indicados en la siguiente sección de este documento:
    - `mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.XXXClient" -Dexec.args="..."`
- Utilizará la sentencia SQL indicada en el punto 2 de la siguiente sección de este documento para modificar la fecha de celebración de un evento y establecerla a una fecha pasada.
- Analizará el código y **los commits para comprobar que cada alumno ha contribuido significativamente a la implementación de todas las capas: capa Servicios, capa Acceso a Servicios y capa Interfaz de Usuario.**
    - **Si un alumno no tiene commits significativos en alguna de las capas enumeradas, no superará la práctica.**

    - **Si los mensajes de los commits de un alumno no siguen el formato explicado en el enunciado de la práctica, no superará la práctica.**
    - Aunque el trabajo se haya repartido entre los componentes del grupo, **la calidad del código de toda la práctica es responsabilidad de todos los componentes**.
- Entre otras cosas, se revisarán los siguientes aspectos:
    - Diseño de los DTOs de la capa Servicios.
        - Para cada DTO se revisará si tiene los atributos adecuados y su tipo.
    - Diseño e implementación de las operaciones REST.
        - Se revisarán los Servlet implementados para verificar que contengan las operaciones necesarias y ninguna más.
        - Se revisará como se ha modelado e implementado cada operación, tanto en lo referente a la petición (método HTTP, URL, datos recibidos como parámetros en la URL o en el cuerpo), como a la respuesta (código HTTP devuelto en caso de ejecución satisfactoria y en cada caso de error, envío de cabeceras HTTP, datos enviados en el cuerpo).
    - Diseño de los DTOs del Cliente.
        - Para cada DTO se revisará si tiene los atributos adecuados y su tipo.
    - Diseño de la capa Acceso a Servicios.
        - Se revisará que la interfaz declare las operaciones necesarias y ninguna más.
        - Para cada método se revisarán los parámetros recibidos, el tipo devuelto y las excepciones declaradas.
        - Uso de una factoría para instanciar la implementación a utilizar de la capa Acceso a Servicios
    - Implementación de la capa Acceso a Servicios.
        - Se revisará la lógica de cada operación, incluyendo, entre otras cosas, la creación de la excepción pertinente cuando el servicio REST devuelve un código de error.
    - Implementación de la capa Interfaz de Usuario (clase con método main).

Pruebas

- Es **OBLIGATORIO** tener subido al directorio raíz del repositorio Git un fichero de texto llamado "PRUEBAS.txt" con los comandos Maven y la sentencia SQL necesarios para ejecutar las pruebas que se enumeran a continuación, **de forma que el profesor pueda hacer copy&paste de esos comandos y sentencias en un terminal, sin necesidad de hacer ningún cambio, y se ejecute el comando maven o la sentencia SQL correspondiente.**
    - Los comandos Maven se ejecutarán desde el directorio **`ws-app-client`**, y para ejecutar la sentencia SQL se habrá conectado antes a la BD **`ws`** con el usuario **`ws`** (**`mysql -u ws ws --password=ws`**).
    - Nótese que como en un paso anterior se inicializó la base de datos **`ws`**, estará vacía en el momento de empezar a ejecutar las pruebas, y los identificadores empezarán a asignarse empezando por el 1.
    - Para cada comando Maven se muestran los valores de los **argumentos** que recibiría el comando (valor del parámetro **`-Dexec.args`**) con carácter ilustrativo, ya que el nombre de las opciones que acepta el cliente, el orden de los parámetros, el formato, etc., será diferente en la práctica de cada grupo.
    - La sentencia SQL también se muestra con carácter ilustrativo, ya que los nombres de las tablas y columnas serán diferentes en la práctica de cada grupo.

1. Añadir evento:

```
-addEvent <name> <description> <start_date> <end_date>
```

```
-addEvent 'Fiesta' 'Fiesta Verano' '2023-08-15T17:00' '2023-08-16T00:00' // eventId=1 creado
-addEvent 'Presentación' 'Presentación de producto' '2023-09-15T11:00' '2023-09-15T13:00'
                                                                        // eventId=2 creado
-addEvent 'Fiesta' 'Fiesta Otoño' '2023-10-15T17:00' '2023-10-16T00:00'  // eventId=3 creado

-addEvent ' ' 'Cena Otoño' '2023-10-01T21:00' '2023-10-02T00:00'    // Falla (nombre inválido)
-addEvent 'Cena' ' ' '2023-10-01T21:00' '2023-10-02T00:00'     // Falla (descripción inválida)
-addEvent 'Cena' 'Cena Otoño' '2022-08-01T21:00' '2022-08-02T00:00'  // Falla (fecha inválida)
-addEvent 'Cena' 'Cena Otoño' '2023-10-01T21:00' '2023-10-01T20:00'
                                                          // Falla (fecha fin <= fin inicio)
```

2. Responder a un evento:

```
-respond <userEmail> <eventId> <response>
```

```
-respond 'user1@udc.es' 1 true             // respId=1 creada
-respond 'user2@udc.es' 1 false            // respId=2 creada
-respond 'user1@udc.es' 3 false            // respId=3 creada
-respond 'user3@udc.es' 3 false            // respId=4 creada

-respond 'user1@udc.es' 3 true             // Falla (ya respondido)
-respond 'user1@udc.es' 9 true             // Falla (evento no exite)
```

[PASO NECESARIO PARA TENER UNA EVENTO PASADO QUE NO SE PUEDA RESERVAR NI CANCELAR] Incluir la sentencia SQL que permita establecer “2022-08-01 21:00” como la fecha y hora del evento con identificador 2 (los resultados de las siguientes pruebas asumen que se ha ejecutado este comando SQL contra la BD). Por ejemplo:

```
UPDATE Evento SET celebrationDate='2022-08-01 21:00' WHERE eventId=2;
```

Responder a un evento pasado:

```
-respond 'user4@udc.es' 2 true ' '  // Falla (fuera de plazo)
```

3. Cancelar un evento:

```
-cancel <eventId>
```

```
-cancel 3     // eventId=3 cancelado

-cancel 3     // Falla (evento cancelado)

-cancel 2     // Falla (fuera de plazo)

-cancel 9     // Falla (evento no existe)
```

4. Responder a un evento cancelado:

```
-respond <userEmail> <eventId> <response>
```

```
-respond 'user4@udc.es' 3 true    // Falla (evento cancelado)
```

5. Buscar eventos por fecha y descripción.

```
-findEvents <untilDate> [<keyword>]
```

```
-findEvents '2023-12-01'           // Devuelve eventos con id 1 y 3
-findEvents '2023-09-01'           // Devuelve evento con id 1
-findEvents '2023-12-01' 'Verano'  // Devuelve evento con id 1
-findEvents '2023-08-01'           // Devuelve lista vacía

-findEvents '2022-08-01'           // Falla (fecha pasada) o devuelve lista vacía
```

- Datos del evento con id 1:
  `Respuestas afirmativas: 1, Respuestas totales: 2, Cancelado: No, etc.`
- Datos del evento con id 3 (se canceló en el paso 3):
  `Respuestas afirmativas: 0, Respuestas totales: 2, Cancelado: Sí, etc.`

6. Buscar evento por identificador.

```
-findEvent <eventId>
```

```
-findEvent 2

-findEvent 9 // Falla (evento no existe)
```

- Datos del evento con id 2:
  `Fecha celebración: 2022-08-01 21:00, Respuestas afirmativas: 0, Respuestas totales: 0, Cancelado: No, etc.`

7. Buscar respuestas de un usuario.

```
- findResponses <userEmail> <onlyAffirmative>
```

```
-findResponses 'user1@udc.es' false // Devuelve respuestas con id 1 y 3
-findResponses 'user1@udc.es' true // Devuelve respuestas con id 1

-findResponses 'user6@udc.es' true // Devuelve lista vacía
```

- Datos de la respuesta con id 1:
  `respId: 1, eventId: 1, Asistencia: Sí, etc.`
- Datos de la respuesta con id 3:
  `respId: 3, eventId: 3, Asistencia: No, etc.`

Una vez creado el tag (y antes de que venza el plazo de entrega), **se recomienda reproducir los pasos que se seguirán para la corrección de la práctica**, de manera que se verifique que la entrega es correcta.