Definición Entrega Julio
Interfaces de Usuario
Curso 2024-2025

Tipología

Entrega individual.

La entrega tiene una dedicación de 70 horas.

Definición

Se solicita, siguiendo los principios de construcción de interfaces de usuario explicados en teoría de la asignatura junto con los principios de codificación seguidos en la práctica de la asignatura, hacer una arquitectura de clases con gestión multidioma (español e inglés) para la generación automática de la interfaz de las funcionalidades de muestra de los datos y de ADD, EDIT, DELETE, SEARCH y SHOWCURRENT para cualquier entidad definida en un fichero nombreentidad_estructura.js con el formato que se acompaña a esta definición de entrega. Además debe existir la funcionalidad de ejecución de tests definidos en las estructuras nombreentidad_def_tests, nombreentidad_tests y nombreentidad_tests_files con el formato que se acompaña a la definición de esta entrega.

Se solicita también general para todas la entidades:

Utilizar una clase Entidad_Abstracta en un fichero Entidad_Abstract_class.js con los métodos comunes a todas las entidades y de la cual heredan todas las clases de entidad.

Utilizar una clase Validaciones_Atomicas en un fichero Validaciones_class.js con los metodos de validaciones para las validaciones atómicas definidas en el fichero de estructura.

Utilizar una clase test_IU en un fichero Test_class.js para que verifique el formato de test y pruebas y ejecute todas las pruebas de test de las definiciones de test definidas para cada atributo (incluidas las de fichero) y saque su resultado en un componente modal con scroll para ser visualizado por el usuario. La acción de test para cada entidad debe estar disponible en la parte superior del index cuando se entra en la gestión de una entidad.

Implementar un método createForm() en la clase Dom en un fichero Dom_class.js (y métodos accesorios si es necesario) que, a partir de los datos de la estructura de datos de la entidad, genere los formularios de ADD, SEARCH, EDIT, DELETE y SHOWCURRENT. Este método debe permitir que si existe un método cargar_formulario_html() en la clase de la entidad se ejecute y cargue el contenido html del formulario y si no existe se cree dinámicamente el formulario a partir de la estructura de datos en un método cargar_formulario_dinamico() de la clase Dom.

Implementar un método load_data() en la clase Dom en un fichero Dom_class.js que rellene los valores necesarios en los formularios.

Implementar un método load_validations() en la clase Dom_validations en el fichero Dom_validations_class.js que compruebe la información introducida en los campos del formulario y un método submit_test() en la misma clase que si todos los datos son correctos permita realizar la llamada a BACK.

La llamada a BACK se realizará mediante el método back_request() en una clase ExternalAccess en un fichero ExternalAccess_class.js,

Las columnas a mostrar de la tabla de datos deben poder seleccionarse de un select con el nombre de los atributos que se muestra en la interfaz cuando abrimos una entidad para su gestión. Debe existir una indicación de qué columnas a mostrar por defecto cuando se abre una entidad por primera vez.

Se solicita también si una entidad particular lo requiere:

Implementar un método change_value_IU() en la clase Entidad_Abstracta sin codificar pero que se invoque siempre y por lo tanto que permita que se refactorice dicho método en una clase de entidad particular para modificar un valor de presentación en interfaz de un atributo de la entidad.

Implementar un método check_special_tests() en la clase Dom_validations que se invoque siempre y que verifique la existencia de cualquier método en la clase de entidad particular con un nombre check_special_NOMBREATRIBUTO() y lo ejecute para validar dicho atributo.

Debe existir un fichero Julio_Datos_*NombreAlumno*.js con una variable de tipo array con el nombre info_entrega con la siguiente información:

Entrega,
DNI alumno,
nombre alumno,
horas dedicadas por el alumno,

En la página index.html debe existir un icono que lleve a una página API.html en donde se describen las clases desarrolladas y los métodos desarrollados indicando y describiendo para cada una de ellas los parámetros de entrada que utilizan y la información que devuelven a nivel de nombre, tipo de dato y descripción. Las clases deben estar separadas en función de si se utilizan para los test, para modificar el html o para realizar validaciones. Este fichero debe estar a nivel de index.html.

Debe existir un fichero IU.css en el directorio /css que contenga todas las reglas de estilo aplicables en el interfaz. La interfaz de las páginas web debe poder visualizarse desde 1920x1080 píxeles hasta 960×540 píxeles de forma dinámica. A partir de 960×540 píxeles debe cambiar al formato móvil de una columna con la resolución 640×480 píxeles como tamaño mínimo. El formato de index.html debe tener un encabezado, un pie de página, un icono de menú debajo de la cabecera (para mostrar las entidades que se pueden seleccionar) y una zona de trabajo. Se mostrará en un modal con scroll vertical y

horizontal los resultados de test y se mostrarán la tabla y los formularios (modales) y los mensajes de acción (modales).

Objetivos

1) Debe funcionar para cualquier tabla de una base de datos. Toda la información necesaria debe estar en las estructuras de datos definidas.

2) Deben poder definirse definiciones de test y pruebas de test para todos los campos de la tabla incluyendo los campos de tipo file.

3) La realización de los test debe realizarse en index.html

4) Debe utilizar los códigos definidos en textos_ES y textos_EN para la información de interfaz.

5) Las acciones deben estar representadas por iconos.

7) Debe utilizarse todo lo desarrollado para implementar los test y la gestión y acceso a BACK de las tablas analysis_preparation, project (ambas ya utilizadas en entregas previas y disponibles en moovi) y characteristic (definida al final del documento), los cuales deben ser accesibles desde index.html.

Historias de usuario a cumplir

Particulares de entrega (Obligatorio. Si se incumple alguno de estos criterios la nota será 0)
1. Los ficheros tienen el nombre, formato y tipo indicado en la entrega
2. El directorio a entregar existe y tiene el nombre indicado en la entrega
3. Se han indicado el número de horas utilizadas en la realización de la entrega por cada alumno.

Por cada error en los ficheros, clases, métodos o variables solicitadas : 0,1

Por cada error en lo solicitado en la información de API.html mostrado desde el index.html : 0,1

Por cada error en los test : 0,1

Por cada error en los formularios (a nivel de datos o de validaciones) : 0,1 [un error en submit se asumirá como un error en cada uno de los campos del formulario]

Por cada error en la interacción en interfaz : 0,1

Por cada error en la apariencia de la interfaz : 0,1

Forma de entrega

1) Colocar todo el código en un directorio con nombre CODIGO

2) Crear un directorio con el nombre Julio_*NombreAlumno* y poner en su interior el directorio CODIGO

3) Comprimir el directorio Julio_*NombreAlumno* en formato .rar con el nombre Julio_*NombreAlumno.*rar

4) Subir el fichero Julio_*NombreGrupo*.rar al ejercicio Julio en Moovi.

Modo de corrección

Se descomprime el fichero .rar proporcionado.
Se coloca en un directorio de la máquina virtual. NO necesariamente en el directorio raíz.
Se verifica la información proporcionada a nivel de entrega, estructuras de datos, test y api.
Se verifica el funcionamiento para las entidades propuestas.
Se verifica el funcionamiento para una tabla nueva usando la estructura de datos de la entrega.
Se verifica la presentación de la interfaz conforme a las indicaciones de corrección proporcionadas.

(SE ENTREGA ANTES DEL VIERNES DÍA 20 DE JUNIO A LAS 23:59 HORAS)

```
CREATE TABLE `characteristic` (
  `id_characteristic` int(11) NOT NULL AUTOINCREMENT,
  `name_characteristic` varchar(100) NOT NULL,
  `description_characteristic` varchar(5000) NOT NULL,
  `data_type_characteristic` enum('number','text','set') NOT NULL,
  `category_characteristic` enum('soil_site','soil_chem','soil_bio') NOT NULL,
  `bibref_characteristic` varchar(200) NOT NULL,
  `file_characteristic` varchar(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```

id_characteristic	dígitos min 1 max 11
name_characteristic	alfabéticos y espacios sin acentos ni ñ, min 8 max 100
description_characteristic	alfabéticos y espacios sin acentos ni ñ, min 80 max 5000
data_type_characteristic	
category_characteristic	
bibref_characteristic	alfabéticos con acentos, ñ, espacios y signos de puntuación, min 16 max 200
file_characteristic	alfabéticos con punto sin acentos ni ñ ni espacios min 7 max 100. Solo los mime type pdf, doc o docx y tamaño de fichero menor de 200000 bytes.

Las definiciones para las pruebas estarán en un fichero “test_definitions.js”.

Las definiciones de tests se crearán, para los campos del formulario, mediante un array de nombre ‘nombreentidad_def_tests’ que contenga:

la entidad,
el campo,
el número de definición de test (secuencial desde 1 hasta el final)
la descripción del test
la acción a realizar
el resultado esperado para este test (boolean/string)
el mensaje de respuesta asociado al resultado.

Las pruebas se crearán, para los campos del formulario que no sean input tipo file, mediante un array de nombre ‘nombreentidad_tests’ que contenga:

la entidad,
el campo,
el número de definición de test,
el número de prueba (secuencial desde 1 hasta el final)
la acción a realizar
el valor a probar
los valores extra si son necesarios
el código asociado de error/valor true de éxito

Las pruebas para tipo file se crearán mediante un array de nombre ‘nombrentidad_tests_files’ que contenga:

la entidad,
el campo,
el número de definición de test,
el número de prueba (secuencial desde 1 hasta el final)
la acción a realizar
el parámetro a probar (max_size_file, type_file, format_name_file)
el valor de parámetro a probar
el codigo asociado de error/valor true de exito