Tarea final del curso

Github

Las aplicaciones a desarrollar deben ser subidas a Github. No deben ser subidas en un único commit cuando la aplicación esté finalizada, se debe realizar un commit cada vez que se finaliza una funcionalidad.

En la entrega se debe incluir el enlace al GIT del alumno, si los proyectos son privados añadir al usuario **cursoprogramacionuo** **https://github.com/cursoprogramacionuo/**

Introducción

La tarea final del curso consiste en desarrollar una aplicación web completa que incluirá:

- Diseño de una base de datos
- Aplicación backend node.js y express
- Aplicación frontend en React

La temática de la aplicación será la misma que se presentó en las tareas del **tema 4**. Se trata de crear una aplicación que permita a los usuarios crear "listas de regalos" para que sus amigos puedan ver que quieren que les regalen.

La aplicación va a permitir:

- El **registro** e **identificación** de usuarios.
- Desde una sección de la web el usuario podrá introducir información de los **regalos que quiere**, podrá incluir múltiples regalos, es decir tendrá una lista de regalos deseados.
    - Cada regalo registrado debe poder ser **modificado** y **eliminado**.
- Desde otra sección el usuario podrá introducir el **correo electrónico de sus amigos**, esta sección mostrará una lista con los correos electrónicos de todos sus amigos.
    - En esta lista se deben poder **eliminar** los correos electrónicos.
- **Ver los regalos publicados por un amigo**, habrá una sección en la web en la que se podrá introducir el correo electrónico de un usuario en un input, si ese usuario tiene agregado nuestro email a su lista de amigos podremos ver todos los regalos que ha creado.
- **Seleccionar un regalo para regalarlo**, desde una lista de regalos de un amigo podremos pulsar el botón **regalar**, esta acción registrará que nosotros vamos a realizar ese regalo.

Detalles sobre el backend

Se debe desarrollar una aplicación Node.js con express que cuente con los requisitos funcionales expresados a continuación. Además de la funcionalidad se deben:

- Utilizar una correcta estructura de división de la aplicación en **routers.**

- Utilizar de la forma más apropiada los verbos de http **GET, POST, PUT y DELETE**
- **Validar los datos** de entrada de los endpoints y generar los mensajes de error apropiados si no se reciben los datos o si estos son incorrectos
- En caso de que se produzcan **errores relativos a la lógica de negocio** de la aplicación se debe generar un error, por ejemplo, al intentar registrar el email de un usuario que ya está registrado.
- Utilizar un **middleware** para proteger las URLS que requieran validación previa del usuario
- Basarse en los **tokens JWT** para la identificación de los usuarios

Los servicios a implementar serán los siguientes:

Gestión básica de usuarios

1. **POST /users/** Crear usuarios (email, nombre y password), no pueden existir 2 usuarios con el mismo email en la aplicación. Los usuarios deben almacenarse en la base de datos. Se establecerá un criterio de seguridad por el que el password debe tener al menos 5 caracteres.

2. **POST /users/login/** Hacer login y generar una apiKey (JWT). Si la identificación es correcta se debe almacenar la apiKey en la lista de claves activas. Como mínimo dentro de la apiKey se deberá almacenar la id y el email del usuario.

3. **POST /users/disconnect/**, Cerrar sesión, se debe eliminar la apiKey de la lista de claves activas.

Gestión básica de regalos por parte de su dueño

4. **POST /presents/** Crear regalo, un usuario identificado en el sistema (apiKey) podrá crear un regalo (id AUTOINCREMENT, nombre, descripción, URL y precio). En la base de datos se va a almacenar toda la información del regalo, y también el email del usuario que lo creo (que se obtendrá el email directamente de la apiKey) .

   Además, en la base de datos incluiremos un campo textual llamado, en el futuro sé incluirá aquí el email de la persona que quiere hacerlos este regalo, este campo va a estar originalmente vacío cuando se crea un nuevo regalo.

   La tabla de la base de datos debería tener un aspecto similar al siguiente:

   presents

id	userId	name	description	url	price	chosenBy
1	1	car	Car des	http	122	NULL

5. **GET /presents/**, Listar mis regalos, este servicio deberá retornar todos los regalos creados por el usuario que ejecuta el servicio, la identificación del usuario se realizará por medio de su apiKey, la cual contiene el email del usuario en cuestión.

6. **GET /presents/:id**, retorna un único regalo, el que tiene la id indicada, es importante verificar que ese regalo pertenece al que ejecuta el servicio, la identificación del usuario se realizará por medio de su apiKey, la cual contiene la id del usuario en cuestión.

7. **DELETE /presents/:id** eliminar regalo, un usuario identificado en el sistema (apiKey) podrá eliminar un regalo que haya sido creado por el mismo. El regalo se identificará por su ID.

   Es muy importante comprobar que el usuario (obtener el usuario utilizando la apiKey) es el creador del regalo que se quiere eliminar.

8. **PUT/presents/:id** modificar regalo, un usuario identificado en el sistema (apiKey) podrá modificar un regalo que haya sido creado por el mismo. El regalo se identificará por su ID, y la petición contendrá en el body los parámetros a modificar (nombre, descripción, URL y precio).

   Es muy importante comprobar que el usuario (obtener el usuario utilizando la apiKey) es el creador del regalo que se quiere modificar.

Gestión básica de amigos

9. **POST /friends/** Agregar amigo, cada usuario tendrá una lista de amigos, amigos serán las personas que podrán acceder a la lista de regalos del usuario. Agregar un amigo consistirá únicamente en añadir su dirección de correo electrónico en una tabla, no requiere de confirmación por parte del amigo ni nada similar. Un usuario podrá agregar amigos utilizando en endpoint

   Este endpoint recibirá a través del body el email del amigo.

   El backend guardará las relaciones de amistad en una nueva **tabla (en la base de datos)** con 2 campos. El email del usuario que crea los regalos, email del amigo. A continuación, se muestra un ejemplo de la tabla con posibles valores:

   friends

emailMainUser	emailFriend
me@me.com	email1@email1.com
me@me.com	email2@email2.com
me@me.com	email3@email3.com
me@me.com	email4@email4.com

   Esta tabla no expresa relaciones con doble sentido, en el ejemplo mostrado anteriormente me@me.com quiere que (email1@email1.com, email2@email2.com, etc.) puedan ver sus regalos, pero no contrarío. Sí se quisiera incluir también esa relación deberían incluirse más datos en la tabla, por ejemplo:

   friends

emailMainUser	emailFriend
me@me.com	email1@email1.com
email1@email1.com	me@me.com

10. **GET /friends/** Ver lista de amigos, el usuario identificado por medio de la apiKey podrá obtener su lista de amigos.

11. **DELETE /friends/:email** Eliminar el amigo enviado como parámetro (:email) del usuario que esta identificado en sesión (por medio de su apiKey)

Selección de regalos por parte de un amigo

12. **GET /presents?userEmail=<email>** Listar los regalos de un, este servicio deberá retornar todos los regalos creados por el usuario recibido como parámetro. Se utilizará el email del usuario como parámetro. La llamada a este servicio con este parámetro se utiliza para ver regalos creados por otros usuarios.

    Esta ampliación del servicio GET /presents puede necesitar ejecutar más de una consulta
    a. ¿El usuario recibido como parámetro **userEmail**, me tiene en su lista de amigos?, habrá que consultar la tabla Friends, el email del usuario que realiza la petición se obtiene a través de la apiKey
    b. Obtener los regalos del usuario recibido como parámetro. Hay que tener en cuenta que la tabla presents identifica al dueño de los regalos por la **userId** NO por el email. Una solución puede ser buscar la id del usuario con ese email, esto podría hacerse en dos pasos, 2 consultas o por medio de una subconsulta.

13. **PUT /presents/:id** vamos a ampliar este endpoint para darle más funcionalidad, actualmente permite modificar (name,description,url,Price) si el endpoint es ejecutado por parte del dueño del regalo.

    Si este endpoint lo ejecuta otro usuario que no es el dueño, no va a enviar nada en el body. La única modificación que va a realizar la ejecución de este servicio por parte de un amigo es incluir su email en la columna **chosenBy** del regalo en la base de datos, hasta ahora nunca habíamos utilizado esa columna. Recuerda que el email del amigo se obtiene a través de la apiKey nunca se envía como parámetro.

    Seguramente la modificación de este servicio va a tener que realizar varias consultas en la base de datos, tales como:
    - ¿Cuál es el email del dueño de ese regalo?, dado que tenemos solo la id del regalo, seguramente sea necesario realizar un JOIN entre la tabla presents y users
    - ¿El dueño del regalo es amigo del usuario que ejecuta la petición?

    Esta modificación solo debe poder realizarse sí:
    - Existe la amistad entre los usuarios

- El regalo no tiene ya un valor en la columna **chosenBy**, si ya tuviese un valor quiere decir que otro usuario ya va a hacer ese regalo.
- El usuario que ha creado el regalo no es el mismo que va a regalarlo (**chosenBy)**

React

La aplicación React se debe conectar al backend y facilitar al usuario las siguientes funcionalidades:

1. Crear un nuevo usuario
2. Identificarnos
3. Cerrar una sesión de un usuario identificado
4. Crear un regalo
5. Ver todos los regalos que hemos creado
6. Eliminar los regalos que hemos creado
7. Modificar los regalos que hemos creado
8. Agregar un correo electrónico a nuestra lista de amigos
9. Ver todos los correos electrónicos que tenemos en nuestra lista de amigos
10. Eliminar un correo electrónico de nuestra lista de amigos
11. Buscar los regalos creado por otro usuario, introduciendo su email en un buscador (esto funciona si el usuario nos ha agregado a su lista de amigos)
12. Seleccionar los regalos de nuestro amigo para indicar que vamos a regalárselo

Se debe incluir:

- Validación de los campos de entrada en el cliente (aplicación React)
- Mostrar los errores enviados desde el backend, cuando se producen errores o validaciones incorrectas en la parte del backend
- Un menú que muestre las opciones más adecuadas dependiendo de si el usuario está identificado o no
- Notificaciones cuando el usuario realice acciones de forma correcta
- Realizar las redirecciones automáticas necesarias para facilitar la experiencia de usuario.

Sobre la interfaz de usuario y parte gráfica de la aplicación

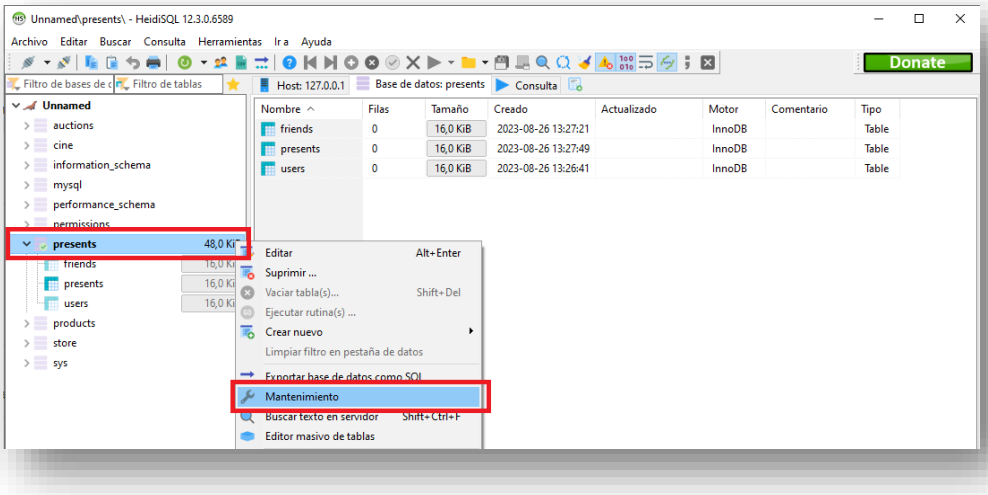
Utilizaremos la librería de componentes gráficos **Ant Design**, si lo deseamos es posible añadir nuestros propios estilos para modificar alguno de los elementos de Ant Design. Debe incluir como mínimo los siguientes componentes:

Layout, Menu, Card, Input, Button, Table, List, Description y Notification.
Al menos 1 componente de cada tipo, aunque de algunos podría haber más, también puedes considerar utilizar otros componentes no vistos en los ejemplos y que resulten de utilidad
https://ant.design/components/overview/

Exporta la base de datos

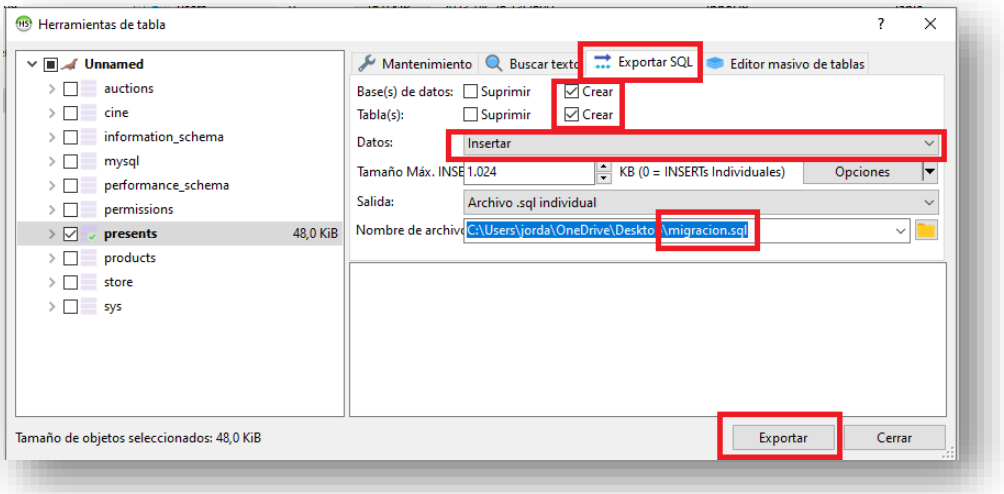
Durante el desarrollo de la aplicación backend se creará una base de datos, esa base de datos debe ser **exportada** a un fichero .sql , el fichero resultante debe ser **adjuntado a la tarea**.

Abrir el asistente de Heidi, pulsar click derecho sobre la base de datos a exportar y luego seleccionar **Mantenimiento**.



Seleccionar la opción **Exportar SQL**, indicaremos que queremos Crear tanto la **base de datos** como las **tablas**, también que queremos los Datos, seleccionando **Insertar**.

Finalmente elegiremos donde se guardará el archivo .sql que contendrá los datos de nuestra base de datos y pulsaremos en **Exportar**.



Entrega

Debe incluir lo siguiente:

- Código de todas las aplicaciones (comprimido y sin la carpeta **node_modules**)
    - Backend

    - React Ant Design
- Enlace al github donde se pueda ver el repositorio de cada una de las aplicaciones
- Fichero .sql con la exportación de la base de datos
- *Documento explicativo, si se desea incluir alguna aclaración o instrucción

Evaluación

Criterios mínimos para superar el curso

- Cumplir con **todos los requisitos explicados** en los puntos previos de este documento.
- **Correcto diseño de la base de datos**, tablas, columnas, tipos, claves primerias y externas.
- Cumplir **con los criterios de implementación** de código adecuado explicados durante el curso, nombrado de variables, formateado de código.