

**DESARROLLO DE APLICACIONES WEB AVANZADO**

**LABORATORIO N° 12**

**API con Express**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Alumno(s):*** |  | | | | | | ***Nota*** |  | |
| ***Grupo:*** |  | | ***Ciclo: III*** | | | | | | |
| ***Criterio de Evaluación*** | | ***Excelente (4pts)*** | | ***Bueno (3pts)*** | ***Requiere mejora (2pts)*** | ***No acept. (0pts)*** | | | ***Puntaje Logrado*** |
| Identifica los comandos básicos para manipular documentos con Mongoose | |  | |  |  |  | | |  |
| Gestiona la creación de tokens con JWT | |  | |  |  |  | | |  |
| Encripta contraseñas con el uso de bcrypt | |  | |  |  |  | | |  |
| Une proyectos React con Express | |  | |  |  |  | | |  |
| Es puntual y redacta el informe adecuadamente | |  | |  |  |  | | |  |

**Laboratorio 12: API con Express**

**Objetivos:**

Al finalizar el laboratorio el estudiante será capaz de:

* Identificar las principales características de una API con Express
* Instalar la base de datos MongoDB
* Identificar las librerías adicionales para verificación de usuarios
* Acceder a una API a través de Postman

**Seguridad:**

* Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
* No ingresar con líquidos, ni comida al aula de Laboratorio.
* Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

**Equipos y Materiales:**

* Una computadora con:
* Windows 7 o superior
* VMware Workstation 10+ o VMware Player 7+
* Conexión a la red del laboratorio
* Máquinas virtuales:
* Clon de Windows 7 Pro 64bits Español – Plantilla con el software MongoDB Instalado
* Lista de reproducción en Youtube
* Se adjunta una lista de videos guía para el siguiente laboratorio:
* <https://www.youtube.com/playlist?list=PLcU0WsRXAr7uq7wtEMaHlMSWmB7BmQwiW>

**Procedimiento:**

**Lab Setup**

1. **Inicio del equipo virtual**
   1. Encender el equipo
   2. Acceder empleando la cuenta de **usuario**: Tecsup, **contraseña**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   3. Iniciar el Software **VMWare**.
   4. Abrir la máquina virtual creada la sesión anterior: **C15-DAWA** almacenada en el folder: **D:\C15-DAWA**
   5. Iniciar el equipo virtual **C15-DAWA**
   6. Identifíquese con la cuenta de usuario: **Redes**. Contraseña: **RCDTecsup2**
   7. Inicie el **servidor** de la base de datos **MongoDB**
   8. Inicie el **Shell** de **MongoDB** y crear una base de datos de nombre: **Lab02**

**Adición de documentos**

1. **Configuración inicial del proyecto**
   1. Crearemos la carpeta lab12 y ejecutaremos el comando **npm init**



* 1. Esta acción, habrá creado un proyecto npm para poder gestionar. Ahora nuestros comandos de instalación de librerías variarán. Instale las siguientes librerías, con el argumento --save





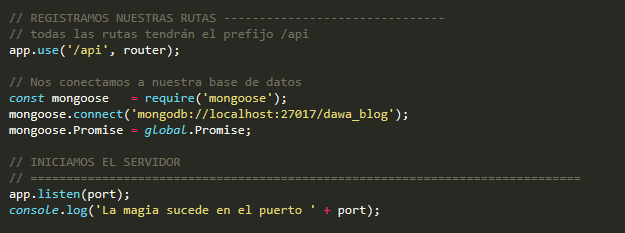






* 1. Procederemos a crear nuestro archivo principal. Cree el archivo server.js con el siguiente contenido.

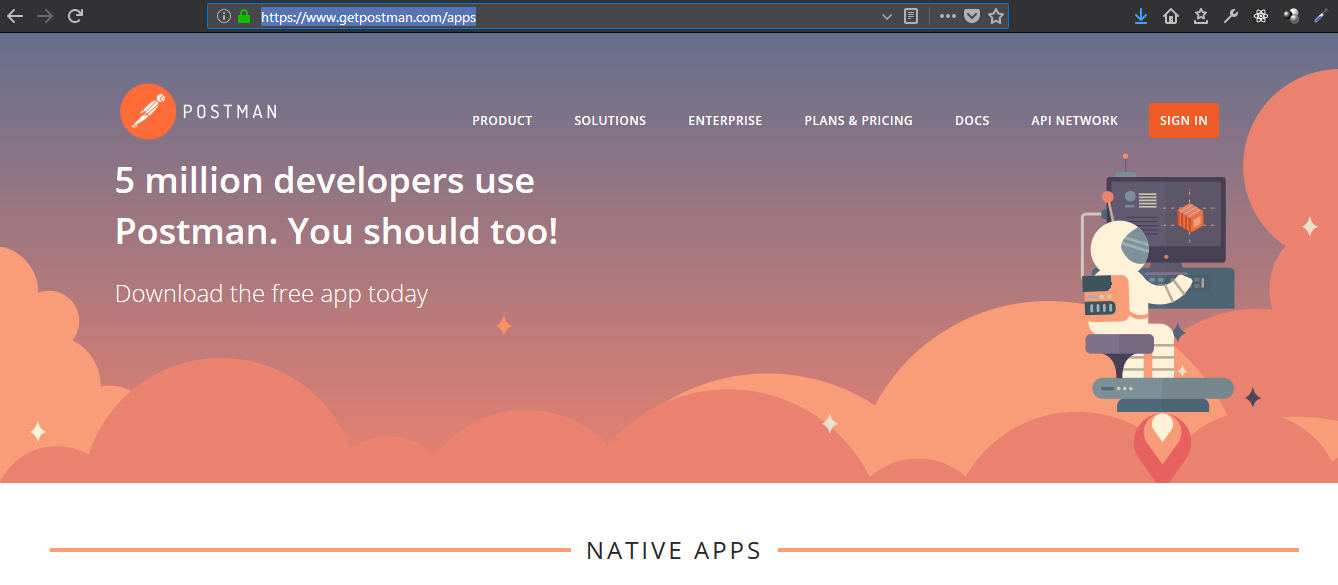




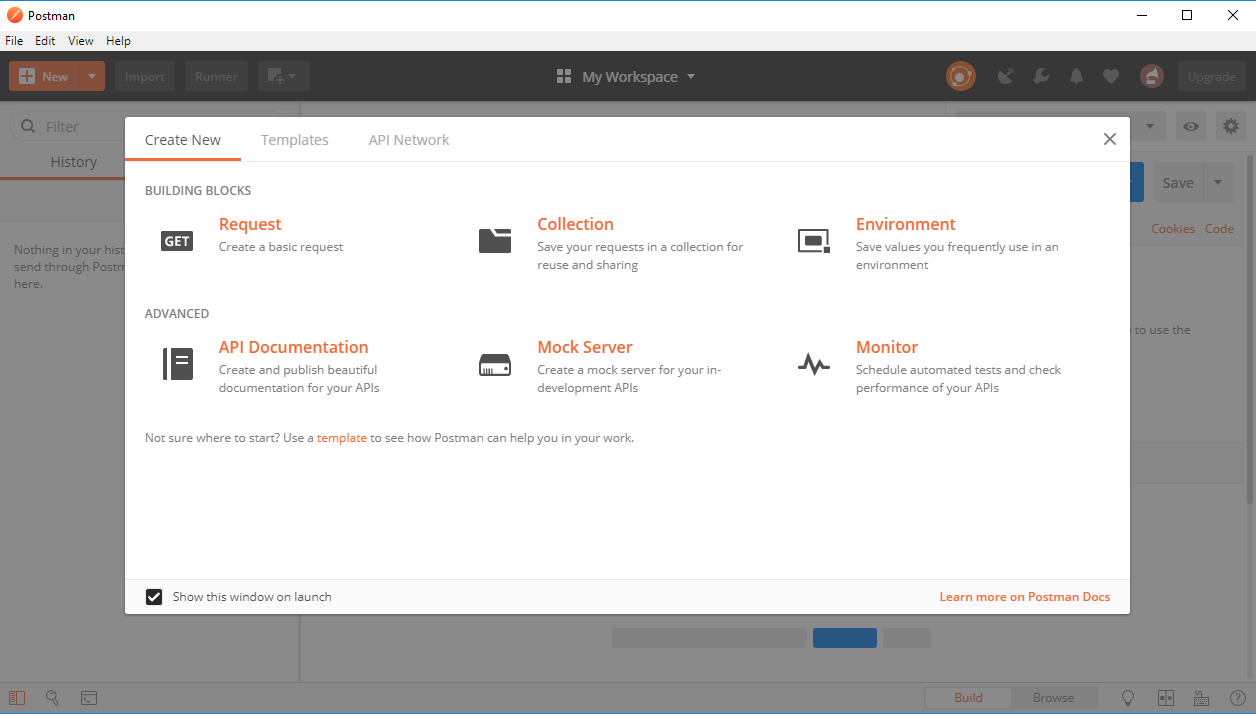
* 1. Al ejecutar **npm start** deberemos ver nuestro mensaje en la consola.



1. Instalación de Postman
   1. Entre a la página web <https://www.getpostman.com/apps> y descargue la versión adecuada para su sistema operativo.



* 1. Instale como cualquier programa común con las opciones por defecto.
  2. Después de iniciado el programa, veremos una pantalla parecida a la siguiente, la cual saltaremos para iniciar nuestra prueba.



* 1. En la parte del protocolo, lo dejaremos por defecto con GET y pondremos la siguiente URL para testear.

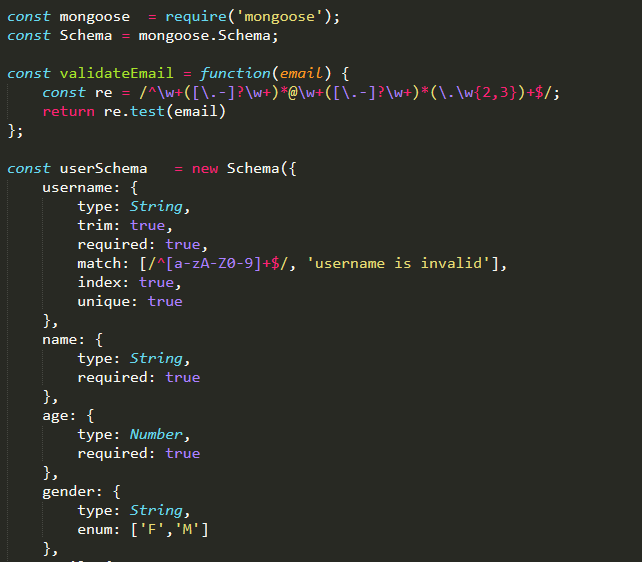


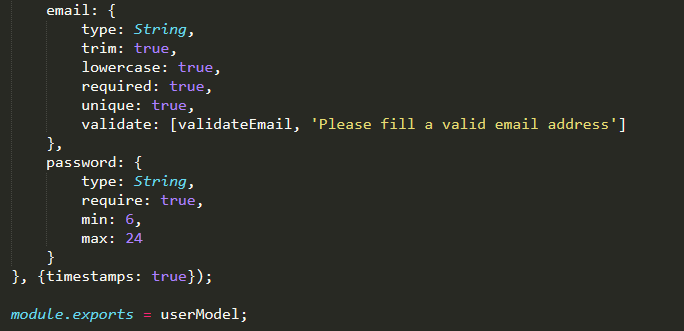
* 1. Haga click en Send y adjunte los resultados

1. Funciones de nuestra API
   1. La forma de nuestra API será la siguiente:

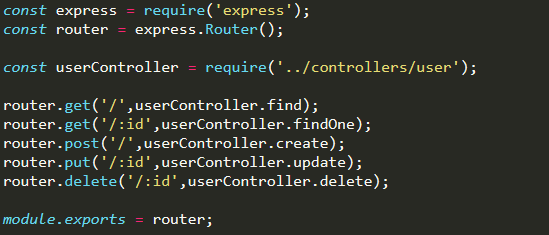
|  |  |  |
| --- | --- | --- |
| Ruta | Acción HTTP | Descripción |
| /api/user | GET | Obtiene todos los usuarios |
| /api/user/:id | GET | Obtiene la data de un usuario |
| /api/user | POST | Crea un nuevo usuario |
| /api/user/:id | PUT | Actualiza un usuario |
| /api/user/:id | DELETE | Elimina un usuario |

* 1. Cree la carpeta models y dentro de ella al archivo user.js con el siguiente contenido





* 1. Cree la carpeta routes con el archivo user.js con el siguiente contenido



* 1. Cree la carpeta controllers con el archivo user.js con el siguiente contenido

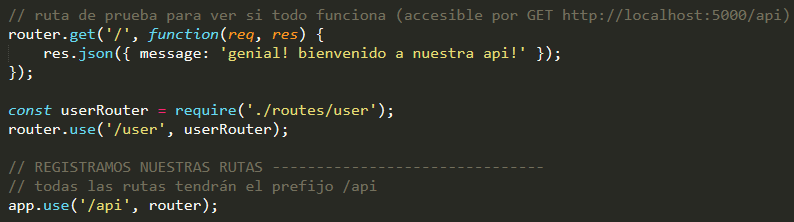




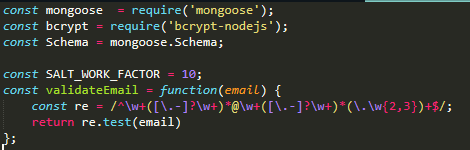




* 1. Modifique nuestro archivo server.js para que luzca de la siguiente manera



1. Encriptación de contraseña
   1. Modificaremos el inicio de nuestro modelo user.js



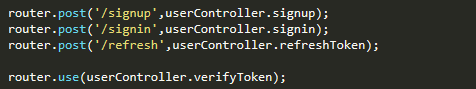
* 1. Así mismo, modificaremos el final del mismo archivo para que luzca de la siguiente manera



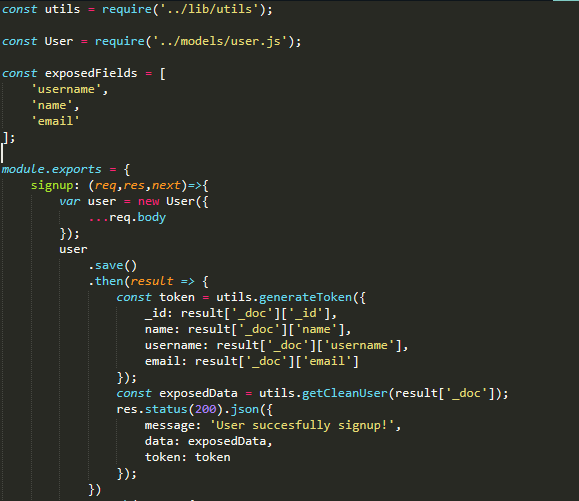
1. Creación de JWT
   1. Crearemos la carpeta lib con el archivo utils.js



* 1. Agregaremos las siguientes rutas dentro de nuestro controlador. Es importante que estas sean colocadas antes de las rutas anteriormente declaradas (sobre todo para que funcione el middleware de verificación de token “verifyToken”)

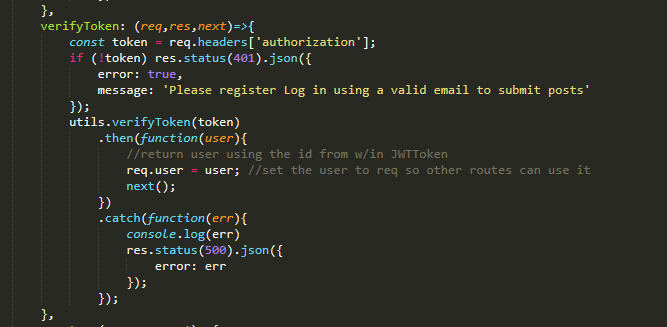


* 1. Así mismo, modificaremos nuestro archivo user.js dentro de controllers para que ya contenga las funciones de las rutas que acabamos de agregar.

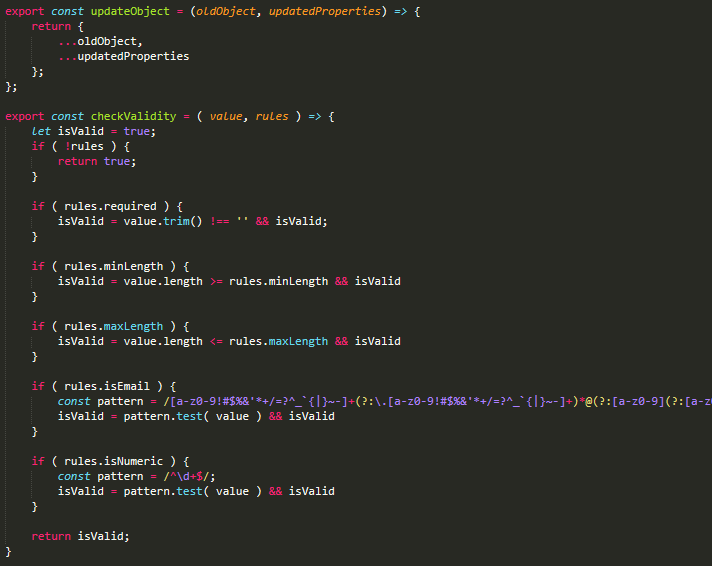








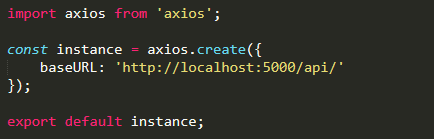
1. Modificación de React
   1. Copiaremos el contenido de lo desarrollado en el laboratorio 11 para crear una carpeta llamada **lab11\_v2**
   2. Empezaremos por crear una carpeta **shared** dentro de la carpeta src con un archivo llamado **utility.js**. Este archivo nos servirá para contener funciones de ayuda (helpers) a través de todo nuestro proyecto React. El contenido será el siguiente



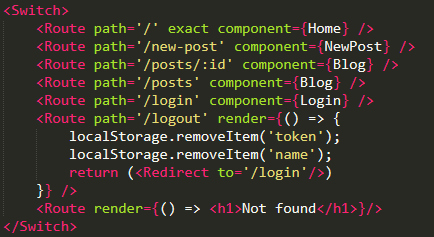
* 1. Debido a las limitaciones de este formato, les agrego el contenido de la línea de validación del email (la que aparece incompleta en la imagen)

const pattern = /[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?/;

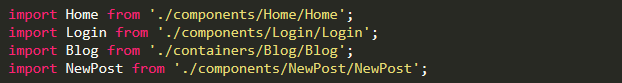
* 1. Crearemos así mismo una instancia axios de conexión con nuestro servidor express. Creemos el archivo **axios-dawa.js** con el siguiente contenido.



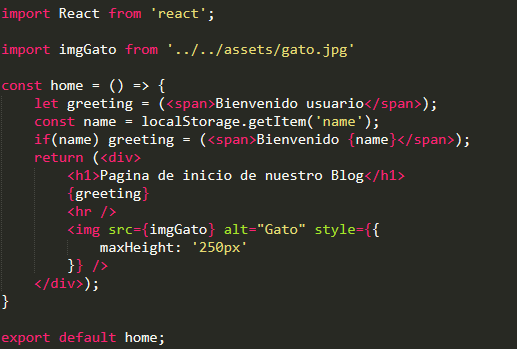
* 1. Modificaremos el componente Switch dentro del archivo App.js



* 1. En el mismo archivo, modificaremos los componentes importados, agregando nuestro nuevo componente **Login**



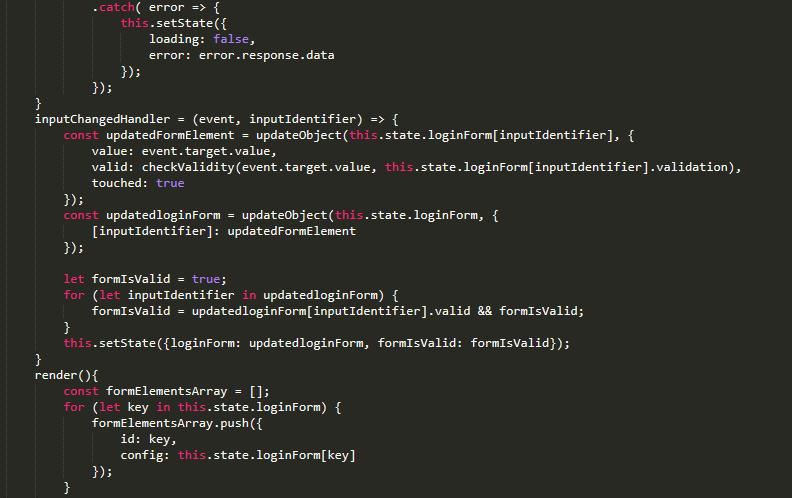
* 1. Antes de proceder con la creación de nuestro componente Login, haremos una modificación en nuestro componente Home para que nos salude en caso estemos autenticados.



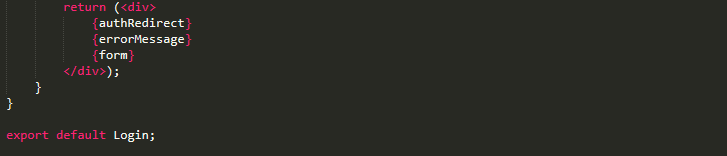
* 1. Procedemos a crear el archivo **Login.js** dentro de src/components/Login con el siguiente contenido.











* 1. Este extenso componente tiene toda la funcionalidad necesaria para la validación y al mismo tiempo la creación de un formulario con campos que pueden ser validados por separado. Esta estrategia suele ser beneficiosa ya que escribir los formularios directamente en cada componente pueden llevar a errores, pero si delegamos la tarea de validación a un componente en partículas cuyo única función es validar, estaremos más seguros.
  2. Crearemos dicho componente en **src/components/UI/Input** con el nombre de **Input.css**



* 1. Crearemos dicho componente en **src/components/UI/Input** con el nombre de **Input.js**



* 1. Verifique que el funcionamiento de ambos proyectos coincide con los videos proporcionados.

**Conclusiones:**

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.

|  |
| --- |
|  |