Sprint 3 – Ejercicios

Índice

Objetivo

El objetivo de este *sprint* es aprender sobre:

- Autenticación y Autorización.
- Passport.js.
- JWT.
- MongoDB.
- Deployment.

Comentarios generales

- Leer en detalle la pauta de cada ejercicio.
- Notar que algunos ejercicios requieren que sea haya dictado una clase previa (teórico) antes de poder resolverlos. Estos ejercicios estarán debidamente señalizados.
- En caso de dudas, pueden recurrir a sus compañeros, docentes (por Slack/Teams) y/o sitios en Internet (ej: Stack Overflow). Recuerden la importancia de apoyarse entre ustedes ya que una gran forma de aprender y reforzar conocimientos es explicarle a otro.
- También recomendamos tener una carpeta llamada `ha_bootcamp_sprint3` (o similar) para tener todos los ejercicios de este *sprint* juntos.

Ejercicio 1

Clase previa: “Autenticación”.

La idea de este ejercicio es seguir trabajando con el Blog con el que se había trabajado antes y crearle un sistema de autenticación.

Para eso seguirán trabajando con el mismo equipo con el cual hicieron la primer parte del Blog, pero con una gran diferencia:

- El equipo #1 trabajará con el código del equipo #2
- El equipo #2 trabajará con el código del equipo #3
- El equipo #3 trabajará con el código del equipo #4, y así sucesivamente
- El último equipo trabajará con el código del equipo #1

El objetivo de esta dinámica es experimentar el trabajo de desarrollar usando como base un código ajeno, algo muy común en el ambiente laboral.

Para eso deberán darse acceso a sus repositorios y deberán crear una *branch* llamada “v2” que contendrá todos los cambios respectivos a este ejercicio. Al finalizar el ejercicio se realizará un *Pull Request* y el equipo original (dueño del repositorio) deberá revisarlo y aceptarlo.

👉 Cuiden mucho la comunicación entre ustedes. Además de tratarse con respeto, mantengan informados a sus compañeros. Júntense físicamente y/o hagan videollamadas donde se cuenten sobre lo que están haciendo, en qué están trancados y qué planean hacer. La buena comunicación en el equipo es clave para el éxito del proyecto. Por favor, dediquen tiempo para hacer una buena planificación.

Pauta:

1. Modificar la tabla de autores para que se pueda guardar la contraseña de un autor.
2. Crear las siguientes rutas:
    a. `[GET]` http://localhost:3000/registro.
    b. `[POST]` http://localhost:3000/registro.
    c. `[GET]` http://localhost:3000/login.

   d. `[POST]` http://localhost:3000/login.
   e. `[GET]` http://localhost:3000/logout.
3. Para hacer login, un autor deberá ingresar su email y contraseña.
4. Una vez logueado, un autor podrá acceder a la sección de Admin del Blog. Dicha sección deberá permanecer “privada” para cualquier navegante que no esté logueado.
5. Si un usuario intenta ingresar a una página “privada” sin haber iniciado sesión, se lo debe redirigir a la página de *login*.
6. Un autor sólo podrá editar artículos de su autoría.
7. Será necesario estar logueado para realizar comentarios en un artículo.

Ejercicio 2

Clase previa: “Autenticación”.

En este ejercicio se seguirá trabajando con el ejercicio del Blog. Cada equipo deberá trabajar con su código original pero incluyendo la funcionalidad de autenticación realizada por el equipo “vecino”. Recordar que el ejercicio anterior debería haber finalizado con la aprobación del *Pull Request*.

⚠️ En caso de que el equipo esté escaso de tiempo, sugerimos saltear este ejercicio y pasar al siguiente.

Pauta: Agregar autenticación por Facebook y Google, usando Passport.js.

Ejercicio 3

Clase previa: “Autenticación”.

Pauta:

1. Continuar con el ejercicio anterior (Blog), pero de forma personal, ya no en equipo. Para eso se recomienda crear una *branch* personal, así no trabajan todos sobre *master* (o *main*).
2. Agregar funcionalidad de “Roles”. En el blog deberán tener 4 tipos de roles, **aunque eventualmente podrán ser más roles en el futuro**. Un usuario sólo puede estar asignado a un rol. Notar que, a partir de ahora, tal vez ya no tiene sentido hablar de “autores” sino de “usuarios” (algunos de los cuales, crearán artículos). Analizar qué esfuerzo implica hacer este cambio y si vale la pena hacerlo.
3. Roles:
    a. Lector: es el rol por defecto que tiene un usuario al registrarse al blog. Puede hacer comentarios en cualquier artículo.
    b. Escritor: mismos permisos que el lector, pero además puede hacer CRUD (de sus propios artículos).
    c. Editor: mismos permisos que el escritor, y además editar artículos de cualquier escritor. No puede borrar artículos que no sean de su autoría. Además puede editar y/o borrar comentarios de cualquier artículo.
    d. Administrador: puede hacer CRUD de cualquier entidad; incluyendo, por ejemplo, eliminar usuarios.
4. Ahora será posible eliminar usuarios (ya sea porque un usuario se dio de baja a sí mismo o porque un administrador lo hizo). Al eliminar un usuario, se deberán eliminar todos los artículos pero sus comentarios deberán permanecer visibles.
5. Notar que para todos los puntos anteriores será necesario agregar nuevas vistas o hacer cambios a las vistas existentes.

Ejercicio 4

Clase previa: “Autenticación en APIs”.

Pauta:

1. Completar la API REST del Blog (algo ya tienen armado). Crear los *endpoints* necesarios para:
    a. Consultar listado de artículos.
        i. Filtrar artículos de determinado autor a través de su ID.
        ii. Filtrar artículos cuyo título tenga determinadas letras. Ej: obtener todos los artículos en cuyo título contenga la palabra “JavaScript”.
    b. Crear, editar, borrar un artículo.
    c. [Extra] Consultar listado de autores. Y crear, editar, borrar un autor.
    d. [Extra] Consultar listado de comentarios. Y crear, editar, borrar un comentario.
2. No olvidar hacer las validaciones necesarias a nivel de Back-End.
3. Agregarle autenticación a la API usando JWT. Solamente un usuario administrador podrá generar estos *tokens*, los cuales se podrán generar desde la GUI del Admin y/o desde un *endpoint* de la API (ej: `POST /tokens`), pasando email/password como credenciales.

Ejercicio 5

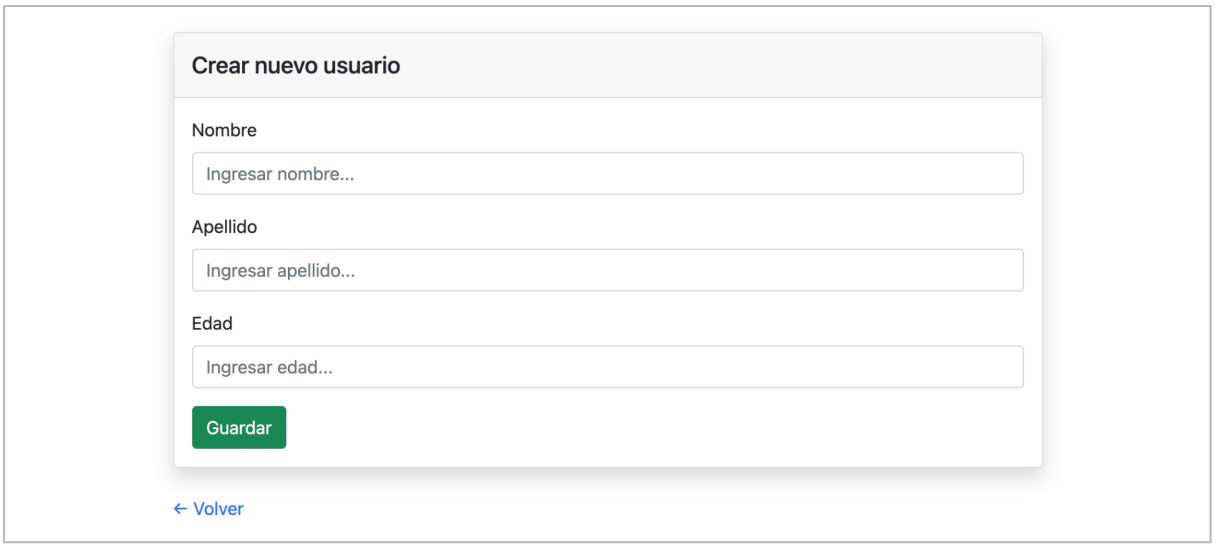
Clase previa: “MongoDB”.

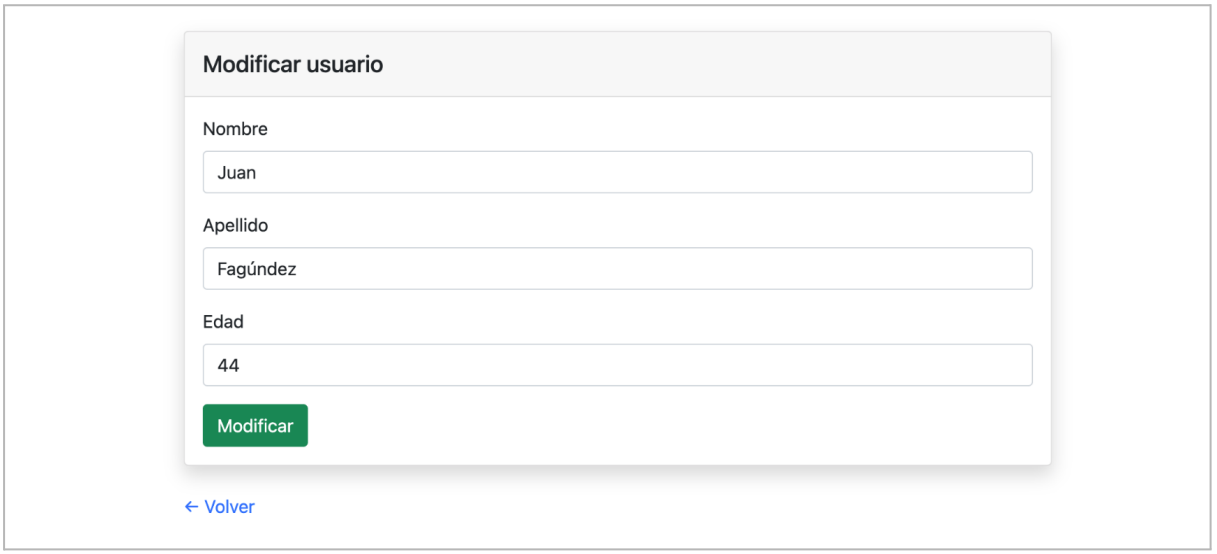
Este ejercicio es muy parecido a un ejercicio del sprint anterior. La diferencia es que en lugar de utilizar MySQL se utilizará MongoDB.

Pauta:

1. Crear una carpeta llamada `ejercicio_mongodb`.
2. Inicializar un proyecto con el comando `npm init` y crear archivo `index.js`.
3. Instalar los módulos `express`, `mongoose` y `ejs`.
4. Replicar una página como la que se ve a continuación.

Importante: Los `id` no tienen por qué ser números naturales. Pueden dejar los `id` que genera MongoDB.


Crear nuevo usuario
Nombre
Ingresar nombre...
Apellido
Ingresar apellido...
Edad
Ingresar edad...
Guardar
← Volver


Modificar usuario
Nombre
Juan
Apellido
Fagúndez
Edad
44
Modificar
← Volver

Ejercicio 6

Clase previa: “MongoDB”.

Pauta:

Investigar sobre SQL y NoSQL.

- ¿Qué es cada uno?
- ¿Qué ventajas / desventajas tiene cada uno?
- ¿Cuándo usar uno y cuándo usar el otro? ¿Se puede prescindir de alguno?
- ¿Se puede trabajar con ambos en una misma aplicación?

👉 Un alumno (elegido al azar) deberá contarle a toda la clase el resultado de su investigación. Si lo desean, pueden preparar una pequeña presentación con diapositivas (ej: Google Slides).

Ejercicio 7

Clase previa: “MongoDB”.

En este ejercicio deberá construir un clon de Twitter (simplificado).

Es un ejercicio largo y deberán trabajar en equipo (armados previamente por el docente y comunicados por Slack/Teams). Deberán dividirse las tareas tratando de que cada integrante realice aproximadamente la misma cantidad de trabajo e intentando que las mismas sean variadas (es decir, no vale que uno haga todo el Front-End y otro haga todo el Back-End). Tal vez les puede ser útil apoyarse en una herramienta como Trello o usar la herramienta de gestión de proyectos integrada en GitHub.

👉 Cuiden mucho la comunicación entre ustedes. Además de tratarse con respeto, mantengan informados a sus compañeros. Júntense físicamente y/o hagan videollamadas donde se cuenten sobre lo que están haciendo, en qué están trancados y qué planean hacer. La buena comunicación en el equipo es clave para el éxito del proyecto. Por favor, dediquen tiempo para hacer una buena planificación.

Deberán tener un repositorio privado en GitHub donde compartirán su código.

Hay partes de la pauta que no están especificadas al 100%, por lo que el equipo deberá tomar ciertos supuestos y/o hacerle preguntas al(los) docente(s), que oficiará(n) de Product Owner en el proyecto.

El equipo también deberá ser responsable de definir cómo organizar su código, elegir nombres para las URLs, elegir nombres para tablas/columnas de la base de datos, etc. Justifiquen bien todas las decisiones que tomen.

A la hora de crear vistas, se sugiere dividir las mismas en *partials*. Por ejemplo, se podría crear un `tweet.ejs`, que es algo que se repite varias veces y aparece en más de una página. Además de evitar código repetido, esto es muy útil para dividirse mejor las tareas de maquetación.

Pauta:

1. Crear una carpeta llamada `ejercicio_twitter`.
2. Realizar *setup* inicial e instalar dependencias necesarias.
3. Deberán utilizar una base de datos MongoDB.
4. Deberán existir páginas de login y registro de usuarios:
    a. `[GET]` http://localhost:3000/login.
    b. `[GET]` http://localhost:3000/registro.
5. El login se deberá hacer con username o email, y contraseña.
6. De cada tweet se deberá conocer:
    a. Texto (hasta 140 caracteres).
    b. Autor (usuario que lo creó).
    c. Fecha de creación.
    d. Likes.
7. De cada usuario se deberá conocer:
    a. Nombre.
    b. Apellido.
    c. Username.
    d. Email.
    e. Descripción (*Bio*).
    f. Foto de perfil.
    g. Lista de tweets.
    h. Lista de usuarios “seguidos” (*following*).
    i. Lista de usuarios “seguidores” (*followers*).
8. Cada usuario deberá tener una página de perfil en la URL:
    `[GET]` http://localhost:3000/username.
    En esta página se verán los datos del usuario y sus últimos 20 tweets, ordenados por fecha de creación, de forma descendente.
9. Un usuario no podrá editar sus tweets, pero sí podrá borrarlos.
10. La Home deberá mostrar:
    a. En caso de usuarios no logueados: una página de bienvenida, con un botón de Login y un botón de Registro.
    b. En caso de usuarios logueados: los últimos 20 tweets de sus usuarios “seguidos” (*following*). Opcionalmente, pueden crear algún mecanismo para mostrar tweets adicionales. Ejemplo: un paginador o scroll infinito.

Simplificaciones. No será necesario implementar las siguientes funcionalidades:

- Mencionar, es decir “arrobar”, a otros usuarios. Tampoco se podrán responder *tweets*.
- Hacer *Retweets*.
- Agregar *hashtags* en los *tweets*.

- Agregar fotos y/o videos en los *tweets*.
- Búsqueda de *tweets* y/o usuarios.
- Sugerencias de usuarios a seguir.
- *Trending Topics*.
- Mensajes directos entre usuarios.
- Notificaciones.
- Etc.