Alumna: Dafne Pineda

Matricula: T07077540

Maestro: Edgar Iván Patricio Aizpuru

Materia: Full Stack

Avance de proyecto 31/01/2026

Introducción

En la escuela de TecMilenio, las instalaciones del gimnasio son de uso común y general para los estudiantes y personal del campus. Sin embargo, recientemente se han tenido quejas de las personas que lo usan por la falta de avisos de cierre del gimnasio, que no haya instructores durante el tiempo que este abierto el campus, también, el personal se ha quejado de que los artículos prestados desaparecen o no son cuidados como debe ser y no se tiene una información constante de quienes entran para usar las instalaciones.

El objetivo de este proyecto es crear una página la cual permita a los usuarios estarse informados de los horarios de los instructores, los días de cierre por falta de instructor, rutinas recomendadas por el instructor, agregar la identificación de las personas que entran (Aunque la actual sea débil, existe), solicitudes de faltas por parte del instructor y para administradores la aceptación o rechazo de estas.

Desarrollo

Durante este avance se presentarán varias entidades:

- Usuarios que se registran
- Gimnasio y reglas de uso
- Instructores y sus horarios.
- Ingreso de usuarios al gimnasio.
- Rutinas del instructor
- Solicitudes de falta y aceptación por parte del instructor
- Artículos y gestión de préstamo

La entidad que se estará trabajando es la entidad del instructor el cual puede cambiar por semestre por lo que se filtra por activos, se necesita: Su nombre, correo, teléfono y su turno.

Iniciamos **npm** y lo configuramos para poder descargar **express**

```
Press ^C at any time to quit.
package name: (proyectofinal)
version: (1.0.0)
description: Proyecto del certificado FullStack
entry point: (instructores.js)
test command:
git repository:
keywords: node, express
author: Dafne Pineda
license: (ISC)
type: (commonjs)
```

Iniciamos y configuramos el servidor a partir de un puerto.

```
const express = require('express')
const app = express()
const PORT = 3000

let instructores = [
    //Nombre
    //(UK) correo
    //(UK) telefono
    //turno
    //activo
]

app.use(express.json())
```

Creamos el **require get** el cual su único objetivo es mostrar todos los instructores registrados.

```
app.get('/instructores',(req, res)=>{
    return res.json(instructores);
})
```

Para crear nuevos registros de instructores usamos el **require post**

```
app.post('/instructores',(req, res)=>{
    const nuevoInstructor = req.body
    let repetido = false

    if(!nuevoInstructor) return res.status(400).send('Body vacio')
    if(Array.isArray(nuevoInstructor)) return res.status(400).send("No arrays")
    if(!nuevoInstructor["nombre"]  || !nuevoInstructor['correo'] || !nuevoInstructor['telefono'] || !nuevoInstructor['turno']){
        return res.status(400).send("Bad request: nombre > 2, correo con @, telefono(10 digitos), turno(matutino/vespertino)")
    }
    if(nuevoInstructor['nombre'].length < 2 || nuevoInstructor['correo'].length < 3 || !nuevoInstructor['correo'].includes('@')
        || nuevoInstructor['telefono'].length !== 10 || (nuevoInstructor['turno'].toLowerCase() !== 'matutino'
        && nuevoInstructor['turno'].toLowerCase() !== 'vespertino')){
            return res.status(400).send('Bad request: nombre > 2, correo con @, telefono(10 digitos), turno(matutino/vespertino)')
    }
    instructores.forEach((instructor)=>{
        if(nuevoInstructor["correo"] === instructor["correo"] || nuevoInstructor["telefono"] === instructor["telefono"]){
            repetido = true
            return
        }
    })
    if(repetido) return res.status(409).send("correo o telefono ya existente")

    nuevoInstructor['activo'] = true
    instructores.push(nuevoInstructor)
    return res.status(201).json(instructores)
})
```

En la imagen se muestra que hay 5 filtros: Si el usuario no escribió nada, envió un arreglo, no ingreso alguno de los campos obligatorios, los campos no cumplen con los requisitos o los datos de teléfono y correo ya existen.

Si pasa por todos los filtros, se agrega el campo de “Activo” agrega al arreglo.

Los instructores pueden cambiar de datos, entonces se usa un **require put**.

```
app.put('/instructores/:id', (req, res) => {
  const id = req.params.id
  const modified = req.body
  const instructor = instructores[id]
  let repetido = false

  if (!modified) {
    return res.status(400).send('Body vacio')
  }
  if (!instructor || instructor.activo === false) {
    return res.status(404).send('Not found')
  }
    instructores.forEach((instru)=>{
        if(modified["correo"] === instru["correo"] || modified["telefono"] === instru["telefono"]){
            repetido = true
            return
        }
    })
    if(repetido) return res.status(409).send("correo o telefono ya existente")
  let wasModified = false
  if (modified.nombre && modified.nombre.length > 2) {
    instructor.nombre = modified.nombre
    wasModified = true
  }
  if (modified.correo && modified.correo.length >= 3 && modified.correo.includes('@') ) {
    instructor.correo = modified.correo
    wasModified = true
  }
  if (modified.telefono && modified.telefono.length === 10) {
    instructor.telefono = modified.telefono
    wasModified = true
  }
  if ( modified.turno && (modified.turno.toLowerCase() === 'matutino' || modified.turno.toLowerCase() === 'vespertino')) {
    instructor.turno = modified.turno.toLowerCase()
    wasModified = true
  }
  if (!wasModified) {
    return res.status(400).send('Bad request: datos inválidos o sin cambios')
  }
  return res.status(200).json(instructor)
})
```

Los filtros que contiene la edición, son: El id existe, enviaron algún dato que permita la modificación, el instructor esta activo, no existe otro numero o correo al que se quiere actualizar.

Si pasa todos los filtros, debe de verificarse que sí se envió ese campo a modificar y cumple con los requisitos, si no los cumple, se avisa que el cambio no se hizo.

Para desactivar a un instructor se llama al **require delete**.

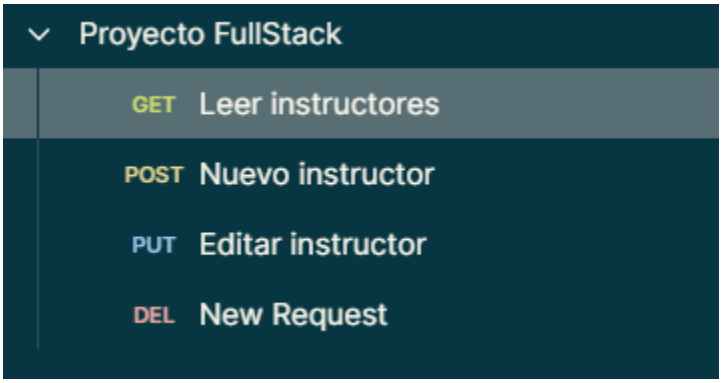
```
app.delete('/instructores/:id',(req, res)=>{
    const id = req.params.id
    if(!instructores[id]) return res.status(404).send('Not found')
    if(instructores[id]['activo']===false) return res.status(404).send('Not found')
    instructores[id]['activo'] = false
    return res.status(200).send('Desactivado')
})
```

Se identifica que sí exista el id y que no este desactivado previamente, en caso de pasar los filtros, se actualiza el estado de “Activo”.

Listen para escuchar cuando se accede al portal.

```
app.listen(PORT,()=>{
    console.log(`http://localhost:${PORT}`)
})
```

Para poder interactuar con el backend usaremos **PostMan** con sus respectivos request



Enlace público de la documentación de Postman:
https://www.postman.com/dafpineda-1123676/workspace/apispublicas/collection/51906938-d29d333a-ecbd-4711-a7c9-5343e42e0f1b?action=share&creator=51906938

Conclusión

El backend es bastante interesante, más en el ámbito de seguir usando el lenguaje de programación de java con la diferencia de apoyarse con librerías. No me interesa en ningún aspecto el entender backend sin usar librerías. Me gusto que nos recomendaran la plataforma de PostMan para poder interactuar con el backend, pues lo hace sencillo y vuelve la forma más cómoda y flexible de las pruebas.