



UNIVERSIDAD  
TECNOLÓGICA  
METROPOLITANA  
*del Estado de Chile*

# Informe Proyecto Rest

**Computación Paralela y Distribuida**

**Profesor:** Sebastián Salazar Molina

**Sección 411**

**Integrantes:**

- Roberto Castillo Riquelme
- Nahuel Espinoza Fuentes
- Cristian Montecinos Fuenzalida

**Fecha de entrega:** Lunes 12 de diciembre de 2022

## **CONTENIDO**

<b>Introducción</b>	<b>3</b>
<b>Marco Teórico</b>	<b>4</b>
Planteamiento del problema	4
Análisis del problema	6
Alcances	6
Limitaciones	6
Desarrollo del problema	6
Arquitectura	7
Infraestructura de software.	7
Desarrollo de aplicación.	8
Capa de aplicación.	8
Lenguaje de programación.	8
Framework de desarrollo.	8
Tablas.	8
Indicators.	8
<b>Funcionamiento de la aplicación</b>	<b>9</b>
<b>Protección del usuario</b>	<b>11</b>
<b>Conclusión</b>	<b>13</b>
<b>Bibliografía</b>	<b>14</b>
<b>Nodejs REST API con JWT y Roles (Autorización y Autenticación) &amp; MongoDB</b>	<b>14</b>

## **Introducción**

API (Application Programming Interface) es un conjunto de protocolos y definiciones que permiten la integración de software y la comunicación entre ellos. Digamos que es como un intermediario entre diferentes aplicaciones.

Lo más importante en el desarrollo de API es una base sólida y bien definida, porque al desarrollar una aplicación, no solo buscamos la funcionalidad, sino también que la aplicación pueda mejorarse o ampliarse fácilmente en el futuro, lo que reduce los costos con futuras extensiones, que los usuarios tengan una buena experiencia de uso, entre otras cosas.

Este trabajo consta de dos partes, la primera siendo la creación de una API Rest. El sistema debe contener los usuarios de la UTEM, registrar las consultas y debe poder registrar el voto. Los objetivos son:

- Comprender el funcionamiento del protocolo HTTP.
- Comprender el funcionamiento de aplicaciones stateless, mecanismos asíncronos y funcionamiento REST.
- Diseñar e implementar un proyecto de software con requerimientos específicos.

## **Marco Teórico**

### **Planteamiento del problema**

1. Usuarios: El sistema debe contener los usuarios de la UTEM, se necesita que el mecanismo de autenticación sea el provisto por Google (oauth 2.0), lo que garantiza la validez de la autenticación, es necesario además que exista un manejo de roles, para una personalización futura. Cada llamada a los servicios api, debe identificar al usuario y tener un access token, para validar la llamada contra los servicios de Google, esto se debe realizar usando una cabecera jwt.
2. Consultas: Se necesita registrar las consultas, los elementos mínimos necesarios son disponer de un token único que permita identificar esta consulta para la interacción de los sistemas y un nombre para identificar la consulta de forma humana y un flag que permita determinar si dicha consulta está activa o no. Por otro lado, cada opción asociada a la consulta debe tener un atributo numérico (que se usará como identificador de la opción) y un campo para el texto que describe la opción.  
En esta etapa se permite que la carga de las consultas y sus detalles, se realicen directamente a la capa de persistencia de datos (Por ejemplo, se permita la carga por un script sql).
3. Se debe poder registrar el voto, es necesario controlar que sólo se pueda realizar un voto por usuario y consulta, por otro lado el diseño debe garantizar el anonimato de la elección.

### **API**

El api a construir debe tener algunas consideraciones generales para su implementación.

- El mecanismo de transferencia debe ser Json.
- Cualquier error (controlado o no) debe producir salidas en formato Json.
- Las operaciones deben estar autenticadas.

## Autenticación

Se necesita una operación que permita la autenticación contra google y almacenar los datos necesarios para validar el token de acceso provisto por Google. Se recomienda que las operaciones de entrada, aquellas que interactúan con Google, están protegidas al menos por credencial/contraseña. La validación se realizará con un objeto jwt que se incluirá en una 1 cabecera “authorization”.

## Votación

Las operaciones mínimas necesarias, son:

### /voter/polls

Se requiere lo siguiente:

- La operación debe ser GET.
- Cabecera de autorización
  - Authorization: Bearer jwt.
    - Se debe validar en base a este token (provisto por Google), el usuario y si su token está activo o no.
- La salida debe indicar aquellas encuestas con al menos la siguiente información:
  - token. Identificador de encuesta.
  - nombre. El nombre de la encuesta.
  - activo. Flag que indica si la encuesta está vigente o no.
  - opciones. Listado de opciones asociadas a la encuesta.
    - número de la opción.
    - descripción de la opción.

### /voter/vote

Esta opción debe permitir registrar un voto, es importante que esto sea anónimo.

- La operación debe ser POST.
- Cabecera de autorización
  - Authorization: Bearer jwt.
    - Se debe validar en base a este token (provisto por Google), el usuario y si su token está activo o no.
- El cuerpo de la petición debe contener:
  - Token de la encuesta.
  - El número de la selección.
- La salida debe responder con código 200 ó 201 y un objeto que indica que la operación fue exitosa.

### /voter/{token}/results

Esta opción debe permitir desplegar los resultados del proceso.

- La operación debe ser GET.
- Cabecera de autorización
  - Authorization: Bearer jwt.
    - Se debe validar en base a este token (provisto por Google), el usuario y si su token está activo o no.
- Cómo parte de la url se debe indicar el uso del token de la encuesta.
- La salida debe responder con un código 200 e indicar:
  - El nombre de la encuesta.
  - Un listado con el nombre de la opción y la cantidad de votos acumulados.

## **Análisis del problema**

### Alcances

- La aplicación permite realizar un voto por alumno con correo utem
- La aplicación encripta la opción marcada por cada alumno

### Limitaciones

- La aplicación solo se ejecuta con una URL local (se puede usar de manera online igualmente)
- Solo posee una encuesta

## **Desarrollo del problema**

Instrucciones para uso online:

Ingresar al siguiente link para votar

<https://proyectoparalela-main-production.up.railway.app/>

Instrucciones para uso local:

Este proyecto fue desarrollado en Windows 10 64 Bits, Las instrucciones para su ejecución son las siguientes.

- Instalar visual studio code
- Instalar node.js
- Descargar el archivo del proyecto en GitHub
- Abrir el proyecto con visual studio code
- Abrir una terminal integrada de visual studio code
- Ejecutar el comando npm install
- Ejecutar el comando node app

```
run npm audit for details.  
PS C:\Users\nafee\Downloads\proyecto_paralela-main-main\proyecto_paralela-main-main> node app  
Servidor corriendo en puerto 8080  
Base de datos online
```

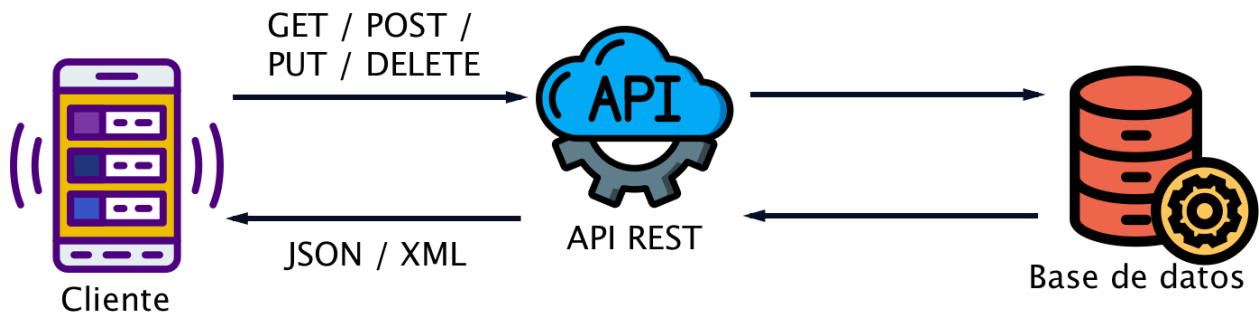
-Abrir la url en el navegador: <http://localhost:8080>

## Arquitectura

### Infraestructura de software.

La arquitectura de la solución fue planteada para la resolución del problema entregado en la asignatura de computación paralela y distribuida, para poder cumplir con todos los requerimientos exigidos. El stack tecnológico desarrollado es el siguiente:

- Windows 10 64 Bits.
- MongoDB Atlas Database<sup>1</sup>.
- Node.js (18.12.1 o superior)<sup>2</sup>. Es crítico para el sistema, que el contenedor sea levantado por un usuario sin permisos especiales.



El API REST es un backend capaz de contestar a las llamadas a las URLs en formato JSON y que también es capaz de recibir JSON para gestionar la información que le enviamos.

<sup>1</sup> <https://www.mongodb.com/>

<sup>2</sup> <https://nodejs.org/>

# Desarrollo de aplicación.

## Capa de aplicación.

### Lenguaje de programación.

Para el desarrollo de la aplicación, se escogió programar la solución de software en lenguaje JavaScript.

### Framework de desarrollo.

El framework de desarrollo es Express JS, Express es un marco de aplicación web de Node.js mínimo y flexible que proporciona un conjunto sólido de funciones para aplicaciones web, con el se tienen métodos de utilidad HTTP y middleware a para su utilización, y esto nos sirve para crear una API.

```
var express = require('express')  
var app = express()
```

## Tablas.

### Indicators.

Almacena la información de un usuario .

Columna.	Tipo.	Definición.
_id	texto	es el identificador de un usuario
estado	booleano	muestra si un estudiante ya voto
google	booleano	es para saber si se registró con google
voto	texto	la respuesta del usuario
correo	texto	correo del usuario
img	texto	imagen del correo del usuario




__v	entero	entero generado por la base de datos
-----	--------	--------------------------------------

## Funcionamiento de la aplicación

Luego de ingresar a la URL nos mostrará la siguiente página

# Google Signin

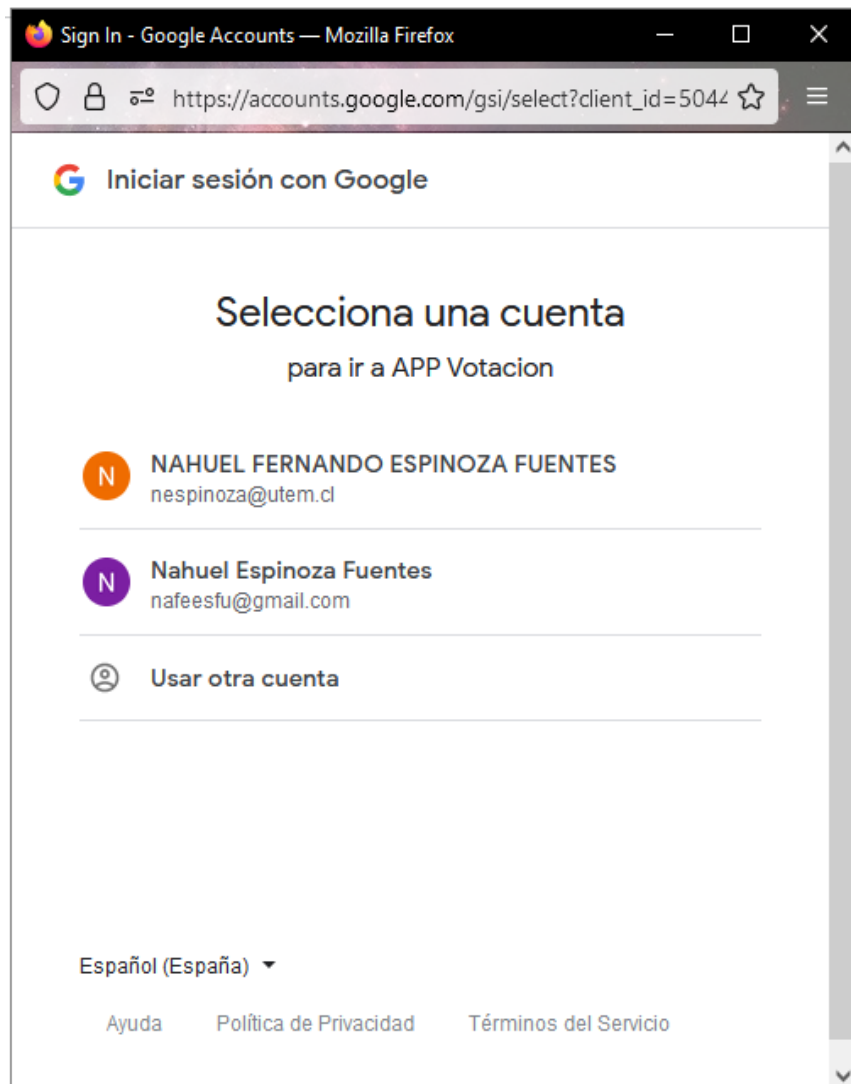
---

 Iniciar sesión con Google

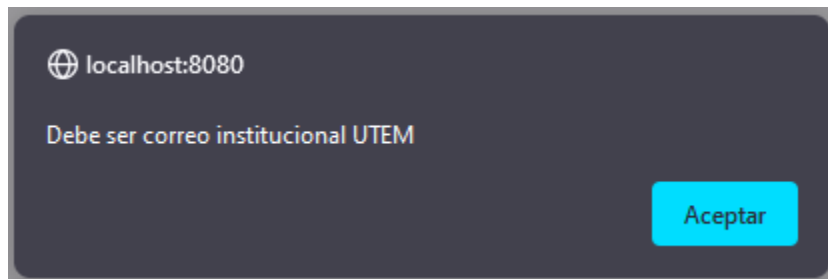
---

Sign Out

Al hacer clic en Iniciar sesión con Google se nos desplegará la siguiente ventana



Aquí podemos seleccionar el correo a utilizar para la encuesta, si seleccionamos un correo que no sea @utem, la página nos devolverá un mensaje alertando que el correo debe ser institucional



Si seleccionamos un correo @utem, nos mostrará las opciones de votación

Después de votar la página nos devolverá un mensaje donde se informa que el voto se realizó correctamente



Una vez realizado el voto se nos mostraran los resultados en ese momento de las votaciones

## Resultados votación

Votos Si: 1

Votos No: 0

Votos Nulo: 0

Si tratamos de ingresar con un correo que ya realizó su voto, el sistema nos informará de esto y no permitirá otro voto



Si el usuario ingresa con su correo pero no realiza ningún voto, podrá volver a ingresar para realizar su voto

## Protección del usuario

Para proteger los datos del usuario, una vez se realizó el voto, en la base de datos de MongoDB, aparece en la parte de usuarios la información del votante

```
_id: ObjectId('63969e6581de6151eda970b1')
estado: false
google: true
voto: "$2a$10$gTmIt3A.IdEUkyG0h7.RFeWoLBue4OoUqdQ4T07z..jtB0tcMKFMG"
nombre: "NAHUEL FERNANDO ESPINOZA FUENTES"
correo: "nespinoza@utem.cl"
img: "https://lh3.googleusercontent.com/a/AEdFTp5FIzAQjwyfNJA9CZwwcoI6h4aeDM..."
__v: 0
```

En esta imagen se pueden los datos correspondiente al correo [nespinoza@utem.cl](mailto:nespinoza@utem.cl), como se observa en la imagen, tenemos el identificador, el estado del correo en donde si aparece “true”, es porque el correo no registra un voto, y un “false”, si este ya fue realizado, en la parte de voto, vemos que la opción de este está encriptada lo que impide ver que opción marco, y en la parte de nombre, vemos el nombre completo asociado al correo institucional

Si el usuario ingresó al sistema de voto, pero no marcó ninguna preferencia, su estado será “true”, lo que significa que utilizó un correo válido, pero no marcó ninguna opción, lo que significa que aún puede votar.

```
_id: ObjectId('6396a0c381de6151eda970b2')
estado: true
google: true
voto: "$2a$10$MlMzXuAvrf0ZT6UnUIEBf0oUGQ5X0RfPYgZJx/p5FnnSGsR52P5g0"
nombre: "NAHUEL FERNANDO ESPINOZA FUENTES"
correo: "nespinoza@utem.cl"
img: "https://lh3.googleusercontent.com/a/AEdFTp5FIzAQjwyfNJA9CZwwcoI6h4aeDM..."
__v: 0
```

## **Conclusión**

El desarrollo de este proyecto fue llevado a cabo con la intención de crear una aplicación capaz de llevar a cabo una votación para los estudiantes de la utem, se hizo esta aplicación con la intención de que los estos estudiantes puedan hacer un voto para ver si estos están de acuerdo con algún paro o no. Para ello estos ingresan con su cuenta de utem, y se manda su respuesta a una api que se conecta con la base de datos y envía una respuesta los estudiantes no pueden votar dos veces ya que al momento de votar se cambia el estado de su cuenta y no lo pueden volver hacer, además su votación es secreta ya que esta es encriptada y en la base de datos no queda registro de cuál fue su votación. Durante el desarrollo de este proyecto, enfrentamos las diferentes dificultades que implican la encriptación de una operación con el fin de proteger los datos de los votantes, así mismo, gran parte del desarrollo consistió en crear un servidor y una api, y en menor medida la creación del login, aunque en un todo, implica una gran dificultad al no estar familiarizados con el desarrollo de este tipo de aplicaciones, lo que significó que ciertas funciones no fueran incluidas con tal de conservar el resto de funcionalidades y entregar un proyecto funcional.

## Bibliografía

*Aplicación Node.js*

<https://nodejs.org/en/>

*Aplicación MongoDB*

<https://www.mongodb.com/es>

*Nodejs REST API con JWT y Roles (Autorización y Autenticación) & MongoDB*

[https://www.youtube.com/watch?v=IV7mxivGX\\_I&list=PLo5IAe9kQrwpjb0R6CwUCFjWpZ7sl7Dyo&index=26](https://www.youtube.com/watch?v=IV7mxivGX_I&list=PLo5IAe9kQrwpjb0R6CwUCFjWpZ7sl7Dyo&index=26)