Rafael Lomeña Busta 42546730J

Iñigo Esteban Garcia 73056238G

Ingenieria Informática 2024/2025

Practica 3

Seguridad y API REST

Índice

[Introducción 3](#_Toc185364829)

[Componentes nuevos 3](#_Toc185364830)

[EC\_Registry.js 3](#_Toc185364831)

[EC\_CTC.js 7](#_Toc185364832)

[API\_CENTRAL 9](#_Toc185364833)

[Cambios EC\_Central.py y EC\_DE.py 11](#_Toc185364834)

# Introducción

Continuando con la práctica anterior implementaremos sobre los módulos de central y motor del taxi nuevas especificaciones como son la seguridad por la conexión de socket, tráfico de información cifrado central y taxi y, la central tendrá conectividad con dos API nuevas, la primera enviara las peticiones del tiempo actual en una ciudad y la segunda API se encargara de volcar todo la información de los taxis y los eventos que vayan ocurriendo en la ejecución en un front web sencillo.

# Componentes nuevos

## EC\_Registry.js

Importación de librerías necesarias: express para manejar la aplicación web, https y fs para configurar el servidor HTTPS con certificados SSL, mysql para conectar a la base de datos MySQL, uuid (paquete uuidv4) para generar identificadores únicos si fuese necesario y cors para permitir solicitudes entre diferentes dominios.

Texto

Descripción generada automáticamente

Puede recibir parámetros por consola(minimist) para conectar a una base de datos con esos atributos, seguidamente se crea la conexión a BD y después se prepara para recibir datos por el sistema middleware:

Texto

Descripción generada automáticamente

Tenemos los siguientes endPoints:

* Registrar un nuevo taxi (POST /taxis) como entrada: id\_taxi, nombre, ciudad en el cuerpo de la solicitud. Verifica que el id\_taxi sea enviado, comprueba si el id\_taxi ya existe en la base de datos, A continuación, agrega el taxi si el ID no existe en la BD y el token inicialmente se establece en null. Como respuesta obtendremos los códigos; 201 para un Taxi registrado correctamente, 400 cuando los datos son inválidos y 409 si el ID ya está en uso.

Texto

Descripción generada automáticamente

* Obtener un taxi específico (GET /taxis/:id\_taxi) como entrada; id\_taxi, busca el taxi en la base de datos. Y recibirá como valor 200 indicando que devuelve la información del taxi o 404 si el taxi no se ha encontrado.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

* Obtener todos los taxis (GET /taxis) recupera todos los registros en la tabla Taxis y devuelve una lista de taxis o un error en caso de problemas.

Texto

Descripción generada automáticamente

* Actualizar información de un taxi (PUT /taxis/:id\_taxi) recibe id\_taxi como parámetro y nombre, ciudad. Actualiza el registro y devuelve 200 si el taxi se ha actualizado correctamente o 404 si no se ha encontrado.

Pantalla de computadora con letras

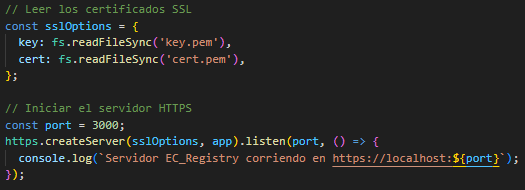
Descripción generada automáticamente con confianza media

* Eliminar un taxi (DELETE /taxis/:id\_taxi) como entrada recibe id\_taxi como parámetro. Elimina el registro asociado al id del taxi y devuelve 200 si el taxi se ha eliminado correctamente o 404 si el Taxi no se ha encontrado.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Por último se utilizan certificado y clave privada autogenerados para poder servir una petición https y se pone de escucha el puerto 3000



## EC\_CTC.js

Este nuevo módulo se encargará de pedir el tiempo en una ciudad en concreto a la api de openweather y que enviará el valor de la temperatura al modulo central indicando si se puede circular (>= 0ºC) o no (< 0ºC).

Inicio:

Necesitaremos las siguientes librerías: express para configurar un servidor HTTP, axios para realizar llamadas HTTP a la API de OpenWeather, readline para capturar entrada de usuario desde la consola y https para manejar conexiones HTTPS. Como configuración de API externa (OpenWeather) se utiliza una API Key y de endpoint de la API: <https://api.openweathermap.org/data/2.5/weather>. Con puerto del servidor: 4000. Además se usa certificado HTTPS.

Captura de pantalla de computadora

Descripción generada automáticamente

Petición al usuario:

Utiliza readline para pedir al usuario que introduzca un nombre de ciudad por consola. Y valida que la ciudad no puede estar vacía.

Texto

Descripción generada automáticamente

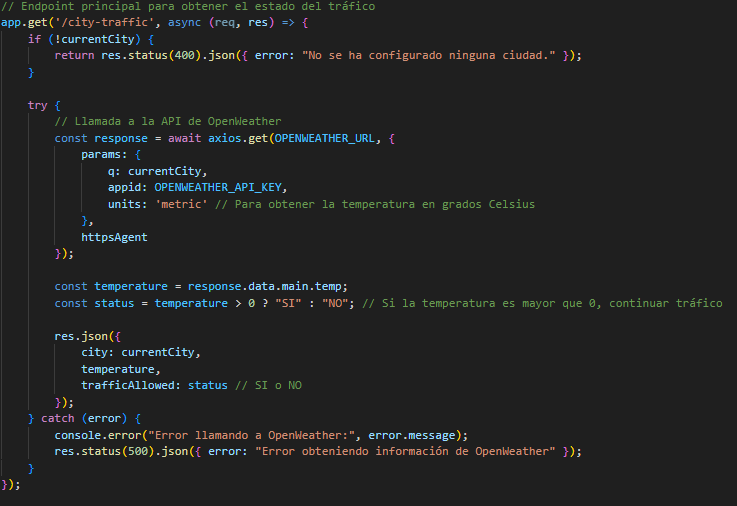
Omitimos la verificación de verificados por parte del cliente que se conecte a la API nuestra

Texto

Descripción generada automáticamente

Endpoint principal: GET /city-traffic

Con ruta /city-traffic, verifica si currentCity tiene valor, luego llama a la API de OpenWeather para obtener la temperatura de la ciudad pasado por consola para comprobar si la temperatura es mayor que 0 grados Celsius, el tráfico está permitido (SI) y en caso contrario, no está permitido (NO). Como parámetros de la llamada a OpenWeather tenemos “q” que es ciudad actual (currentCity), “appid” como la API key de OpenWeather y “units” (con valor 'metric' para devolver la temperatura en grados Celsius). Devuelve 200 si no hay fallos y en caso contrario se captura el error y se indica el error:



Por último, se lanza el servidor en el puerto 4000:

Texto

Descripción generada automáticamente con confianza media

## API\_CENTRAL

Este modulo se encargará de enviar toda la situación del sistema de gestión de taxis al front web, como son el mapa del comportamiento en tiempo real y logs de las situaciones en todo momento de la central, además de servir al front web conectándose a él.

Inicio:

Librerías requeridas express es framework para manejar HTTP, socket.io da soporte en tiempo real utilizando WebSockets, cors permite solicitudes desde diferentes orígenes, path que maneja rutas de archivos, multer gestiona la subida de archivos y fs manipula archivos del sistema.

Texto

Descripción generada automáticamente

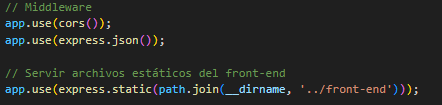
Creación de servidor HTTP y WebSockets con http.createServer(app) para crear el servidor HTTP y socketIo(server) que habilita WebSockets en el servidor.

Texto

Descripción generada automáticamente

Configuración de middleware:

Permite peticiones desde cualquier origen conparser permite recibir y procesar JSON en las peticiones HTTP y los archivos estáticos sirven recursos del directorio front-end.



Eventos de Socket.IO

Detecta cuando un cliente se conecta/desconecta y lo notifica en la consola.

Texto

Descripción generada automáticamente

Endpoints de la API:

Ruta para servir EC.html 🡪GET /EC.html que envía un archivo HTML al cliente (localizado en un directorio externo)

Texto

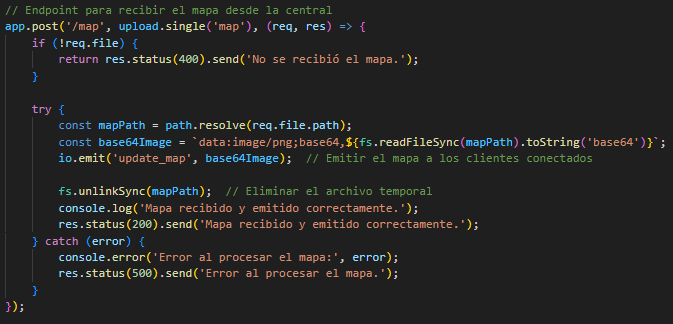
Descripción generada automáticamente

Recepción y difusión de logs (POST /logs) de parámetro un JSON con el campo message, devuelve un error 400 si no se proporciona el campo message, emite el log recibido a todos los clientes conectados mediante Socket.IO y responde con un estado 200 OK.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Recepción de mapas y transmisión (POST /map) con parámetro un archivo de imagen (mapa) mediante Multer, valida si se recibió un archivo, responde con 400 en caso contrario, convierte el archivo a base64 y lo emite a todos los clientes conectados con el evento update\_map, elimina el archivo temporal del servidor utilizando fs.unlinkSync() y devuelve 200 OK si el archivo se procesa con éxito o 500 en caso de error.



## Cambios EC\_Central.py y EC\_DE.py

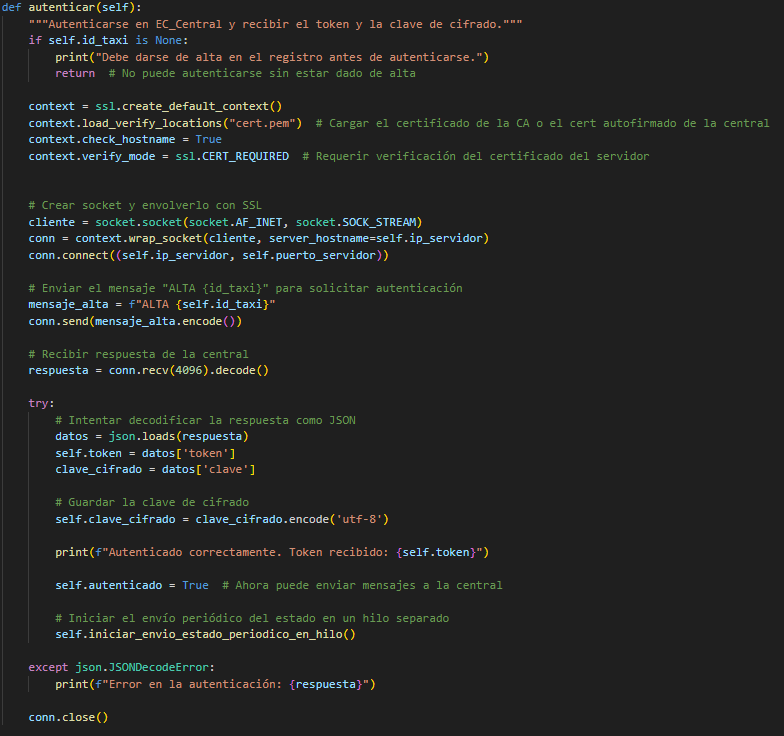
En este modulo hemos cambiado la implementación del registro de taxis ya que no proveerá de id al taxi sino de un token cuando se comuniquen por primera vez y se cifrarán las comunicaciones entre la central y los taxis.

Implementación de socket seguros:

En EC\_Central (método iniciar) usamos la configuración SSL para crear un contexto SSL usando ssl.create\_default\_context(), carga los certificados de servidor (cert.pem y key.pem), para la configuración del socket se crea un socket TCP/IP con socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) se vincula a una IP y puerto específicos con bind() e inicia la escucha de conexiones con listen(5) (cola de 5 conexiones). Entramos a un bucle de conexiones usa accept() para esperar conexiones entrantes y se envuelve cada conexión con SSL usando context.wrap\_socket(). Creamos un hilo para cada conexión usando threading.Thread(), cada hilo ejecuta manejar\_solicitud() para procesar la conexión que anteriormente ya se implemento en la anterior práctica

Texto

Descripción generada automáticamente

Para el taxi (método autenticar) también usamos la configuración SSL creando un contexto SSL con ssl.create\_default\_context(), carga y verifica el certificado del servidor (load\_verify\_locations("cert.pem")), se configura la verificación de hostname y modo de certificado, se crea la conexión con socket; Llamamos a un socket TCP/IP que esta envuelto por SSL usando context.wrap\_socket() y se conecta al servidor central con conn.connect(). Para el envío de datos para la autenticación se envía mensaje "ALTA {id\_taxi}" al servidor y recibe una respuesta de autenticación con token y clave de cifrado.

Para el taxi se de alta ahora, es decir, que se le asigne una id y que conste en la central, se utiliza la API EC\_Registry explicada anteriormente, se hace una petición por https al modulo cargando la ip y el puerto, se pide al usuario que id desea tener y se pregunta al método get de taxis del registry junto con la ciudad del taxi y el nombre, recibirá el resultado explicado en registry indicando si se ha dado de alta correctamente o no.

Texto

Descripción generada automáticamente

Para adecuar que los cambios que se hagan en los taxis se hagan por medio del resgitry se han cambiado los métodos actualizartaxi y eliminar para que hagan llamadas a la API de registry de la misma forma que el anterior método:

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Para el envío de mensajes de la central por Kafka, tenemos el método enviar\_estado\_kafka adaptado para que se cifren los mensajes mediante la librería fernet que previamente se habrá ayudado de la clave de cifrado que se le ha pasado en la primera comunicación con la central por socket. Esto se aplicara tanto en los metodos escuchar\_asignacion\_cliente como en escuchar\_incidencias.

Texto

Descripción generada automáticamente

En la central se le enviará la clave de cifrado al taxi

Texto

Descripción generada automáticamente

Se le consultara a la base de datos mediante el uso de un cursor haciendo una simple consulta

Captura de pantalla de computadora

Descripción generada automáticamente

# Guia de despliegue

Siguiendo la guía de la practica anterior mediante scripts ejecutamos Kafka y resolvemos la incompatibilidad de Python con Kafka-python.

Para la creación de la API iniciamos:

npm init

npm install express resquest body-parser mysql socket.io multer https minimist

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout certServ.pem -out certServ.pem

node EC\_Registry.js y node server.js

Además instalamos los paquetes:

pip install kafka-python==2.0.0 request mysql-connector-python cryptography Fernet