Green-E-Mind. Especificación de Requerimientos de Sistema Back-end.

Elaborado para Carlos Arturo Dounce Pérez Tagle

Proyectos de Ingeniería Mecatrónica

Diego Oseguera

Abril 26, 2021

Introducción

Este documento presenta el objetivo de la integración del back-end, las entradas y salidas respecto al resto del sistema Green-e-mind, las principales funcionalidades que ofrece, las reglas de negocio que debe cumplir y el detalle de su implementación.

Este documento está dirigido a gestores de producto, desarrolladores y personal técnico involucrado en la implementación del sistema Green-e-mind.

Objetivo

Desarrollar el software necesario para integrar la conexión entre el dispositivo Green-e-mind y la interfaz de usuario correspondiente (aplicación web). Almacenar, depurar, manejar y securizar la información relevante para cumplir con las reglas de negocio.

Alcance

Desplegar una API-REST capaz de interactuar con el resto del sistema y mediante un desarrollo CRUD e interactuar con las siguientes entidades: Usuarios, Sensores y Productos. Facilitar la conexión entre el dispositivo Green-e-mind (mediante el protocolo MQTT) y la interfaz de usuario correspondiente (HTTP, WebSockets). Depurar, almacenar y modificar la información necesaria según las reglas de negocio.

Contexto

Hoy en día las aplicaciones más exitosas cuentan con una API-REST escalable e independiente a la interfaz de usuario, que permite mantener una gestión exclusiva de dicho desarrollo. Además de que permite integrar esta API-REST con diferentes interfaces de usuario (de escritorio, móviles, etc.) concentrando todo la regla de negocio en una sola implementación.

Las bases de datos no relacionales están teniendo un gran desarrollo en los últimos tiempos por la diversificación y escalabilidad que representan.

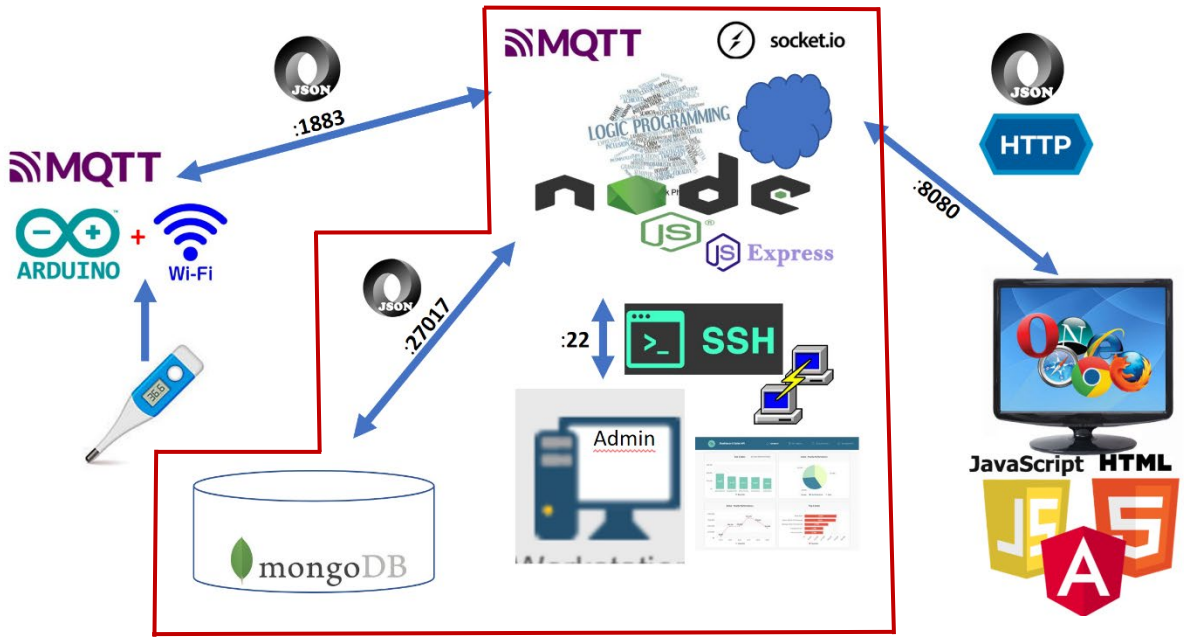
Recientes protocolos de comunicación nos permitirán mejorar la experiencia de usuario, haciendo uso de tecnologías de IOT como MQTT. MQTT (Message Queue Telemetry Transport) es un protocolo de transporte de mensajes Cliente/Servidor basado en publicaciones y subscripciones a los denominados “tópicos”. Cada vez que un mensaje es publicado será recibido por el resto de dispositivos adheridos a un tópico del protocolo. El protocolo MQTT funciona sobre TCP/IP o sobre otros protocolos de red con soporte bi-direccional y sin pérdidas de datos.

Los WebSockets es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.

HTTP, de sus siglas en inglés: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. Así, una página web completa resulta de la unión de distintos sub-documentos recibidos, como, por ejemplo: un documento que especifique el estilo de maquetación de la página web (CSS), el texto, las imágenes, vídeos, scripts, etcétera.

Propuesta de solución

Se crea una API-REST en desarrollada en NodeJS auxiliada de frameworks como Express y Socket.io. Se implementa una base de datos no relacional en MongoDB para coleccionar la información de los usuarios, sensores y productos. Se integró lo enmarcado en un cuadro rojo del siguiente diagrama:



Principales características

Express permite desde NodeJS instanciar un servidor y su configuración básica con apenas unas cuantas líneas de código. Además es uno de los frameworks más utilizados para este fin. Socket.io nos permitió crear una conexión bidereccional entre el cliente y el servidor en tiempo real, sin necesidad de lanzar peticiones cada determinado tiempo. Se decidió utilizar una API-REST para poder utilizar este servicio fácilmente desde cualquier interfaz de usuario deseada, ya sea de escritorio o móvil.

La base de datos relacional, como se mencionó con ateriodidad, permite la fácil escalabilidad de los datos a través del tiempo.

Usuarios

Se contemplan 2 roles de usuarios: cliente y administrador. A diferencia del usuario cliente, el administrador tiene privilegios más extendidos dentro de la implementación. Cualquier usuario requiere de autenticación para poder interactuar con el servicio.

Entorno de operación

Se instanció una máquina virtual en la nube gestionada por Microsoft Azure donde se aloja tanto el entorno del front-end como el del back-end. Se contrató el dominio greenemind.com para acceder directamente a la plataforma desplegada. Mediante un servidor Web Nginx es posible dar soporte a la interfaz web de nuestra aplicación.

Restricciones y Reglas de Negocio

Autenticación. Es necesario contar con la autenticación de usuarios en nuestra plataforma, para así conocer a qué usuario pertenece cada dispositivo. Además de llevar una relación de quién está conectado en determinado momento.

Autorización. Es necesario contar con la autorización de roles de usuario para cada usuario. Así solo usuarios administradores podrán mantener ciertos privilegios sobre el resto de los usuarios.

Dependencias

1. Máquina Virtual y servicios Azure.
2. Mantenimiento a librerías como:
    a. Mongoose
    b. Express
    c. Helmet
    d. Socket.io
    e. MQTT
3. Mantenimiento a framework
    a. Angular

Especificaciones de Hardware

Gracias al contrato instanciado con Microsoft Azure, del lado del backend no contamos con software específico por parte del equipo de Green-e-mind.

Especificaciones de Software

1. NodeJS v.10.19.0
2. ExpressJS v.4.17.1
3. Socket.io v.3.1.2
4. Cors v.2.8.5
5. JSONWebToken v.8.5.1
6. Mongoose v.5.12.3
7. Morgan v.1.10.0
8. MQTT v.4.2.6
9. Socketio-jwt v.4.6.2
10. MongoDB v.4.4.5
11. Mosquitto v.1.6.9

Modelo de Datos

Usuarios

Nombre	String
Apellidos	String
Fecha de nacimiento	Date
Correo electrónico	String
Apodo	String
Contraseña	String
Fecha de registro	Date
Status	Char
Socket_ID	ObjectID

Sensores

Product_ID	*ObjectID*
Nombre	*String*
Lecturas	*SubDocument*

Productos

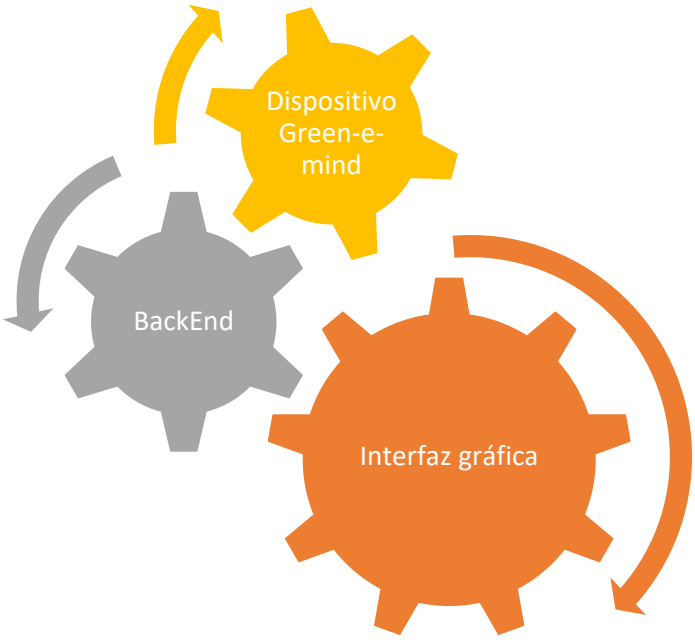
Apodo	*String*
Status	*Char*
Fecha de registro	*Date*
User_ID	*ObjectID*

Lecturas

Fecha y Hora	*Date*
Valor	*Decimal128*

Requerimientos Funcionales

El desarrollo permite crear, leer, actualizar y eliminar cualquier tipo de registro que pertenezca a las entidades descritas en la página anterior. Permite la conexión y almacenamiento de los datos que fluyen desde el dispositivo hacia la interfaz gráfica. Soporta la conexión (una vez el usuario ha sido autenticado) para mantener el envío de notificaciones del dispositivo hacia la interfaz. Autentica y autoriza a los usuarios según sus credenciales.



Criterios de Validación

Casos de Prueba

#	Prueba	Resultado Esperado	Resultado	Notas

1	Recepción de mensajes desde el dispositivo	Los mensajes se reciben en el backend con el siguiente formato:
		{
		“productID”: “XxXxX”,
		“temp”: “18”,
		“hum”: “46”
		}
			Los mensajes se reciben en el backend con el siguiente formato:
			{
			“productID”: “XxXxX”,
			“temp”: “18”,
			“hum”: “46”
			}	

2	Almacenamiento de mensajes	Los mensajes se almacenan en las colecciones y documentos correctos.	Los mensajes se almacenan en las colecciones y documentos correctos.	
3	Autenticación de usuarios	Se genera un JSONWebToken para identificar la sesión de cada usuario.	Se genera un JSONWebToken para identificar la sesión de cada usuario.	
4	Integración WebSockets	Se instancia un WebSocket del lado del cliente para probar la comunicación bidireccional con el servidor.	Se instancia un WebSocket del lado del cliente para probar la comunicación bidireccional con el servidor.	
5	Actualización del despliegue de mensajes en Chat	Se define la semántica para los mensajes enviados al cliente a partir de los datos recopilados por los sensores.	Se define la semántica para los mensajes enviados al cliente a partir de los datos recopilados por los sensores.	

Apéndices

Fallas Conocidas

- La correcta funcionalidad de ciertos protocolos de comunicación están sujetas a la correcta conectividad del cliente/dispositivo con internet.
- La autenticación de usuarios está sujeta al correcto ingreso de credenciales a través de la plataforma.
- La disponibilidad de los servicios en línea está sujeta al estado de funcionamiento de los servidores de Microsoft Azure.

Referencias

- "MQTT - the Standard for IoT Messaging." *Mqtt.org*, 2020, mqtt.org/. Accessed 10 May 2021.
- Colaboradores de los proyectos Wikimedia. "Web Socket." *Wikipedia.org*, Wikimedia Foundation, Inc., 6 Dec. 2010, es.wikipedia.org/wiki/WebSocket. Accessed 10 May 2021.
- "TST – MQTT." *Tst-Sistemas.es*, 2021, www.tst-sistemas.es/mqtt/#:~:text=MQTT%20(Message%20Queue%20Telemetry%20Transport,a%20los%20denominados%20%E2%80%9Ct%C3%B3picos%E2%80%9D.&text=Uso%20de%20mensajes%20%E2%80%9Cbroadcast%E2%80%9D%20para,con%20independencia%20de%20la%20aplicaci%C3%B3n.. Accessed 10 May 2021.
- "Generalidades Del Protocolo HTTP - HTTP | MDN." *Mozilla.org*, 9 May 2021, developer.mozilla.org/es/docs/Web/HTTP/Overview. Accessed 10 May 2021.

Control de Versiones

Versión	Descripción	Responsable	Fecha
0.1.0	Template	Edgar Luque	26/04/2021
0.1.1	Draft	Diego Oseguera	26/04/2021
0.1.2	Entregable	Edgar Luque y Diego Oseguera	10/05/2021