ITESM CAMPUS PUEBLA

Manual de usuario

Sistema de consumo eléctrico

Diciembre 2021

Versión 1.0

Idea de implementación de la aplicación

¿Cuál es el objetivo de la aplicación?

El sistema tiene como objetivo tener el control de sus dispositivos electrónicos conectados, ya sea en su hogar, oficina u otro lugar que desee agregar. Cada dispositivo se almacena en su habitación correspondiente.
El sistema puede generar informes personalizados para conocer tu consumo eléctrico para el período que decidas, o incluso especificar un espacio o dispositivo.

¿Cómo funciona la aplicación?

El sistema creará automáticamente un escenario vacío para ti con una habitación vacía, pero puedes crear uno desde cero. Cada etapa tiene un número n de habitaciones y cada habitación tiene unos n dispositivos. Los dispositivos tienen una etiqueta para especificar si son focos, tv, etc., pero puede elegir "otros" si no encuentra su dispositivo.

Implementación del backend

API

Para realizar la API nosotros decidimos utilizar cosas que sean relativamente recientes, fáciles de usar, sencillas de modificar y escalar en el futuro.
Es por esto que para la API nos decidimos a usar GraphQL, el cual es un lenguaje de consulta para las APIs, que se ejecuta del lado del servidor por lo que permite a los usuarios solicitar sólo los datos que necesitan de la API.

GraphQL fue diseñado por Facebook para poder soportar aplicaciones muy robustas con gran flujo de datos, de igual forma es más flexible y permite la creación de APIs con Python, JavaScript, Java, Scala, etc.

En este caso nos apoyamos de Flask que está realizado en Python como framework para construir una aplicación web sencilla y que a futuro pueda escalarse.

Para poder usar GraphQL, debemos usar el lenguaje propio que es SDL o Schema Definition Language, el cual nos dice que debemos crear esquemas.

Dentro de estos esquemas, con las Queries el usuario puede solicitar toda la información que necesite, el formato es el siguiente:

```
type Query {
    todos: [Todo]!
}
```

Para poder modificar o añadir datos, se usan mutaciones, su escritura es similar a las queries.

```
type Mutation {
   createTodo(description: String!, dueDate: String!): Todo!
}
```

Siguiendo esta escritura podemos construir nuestros esquemas, a continuación se muestra el ejemplo de un esquema construido para la API, en este caso es para obtener y manejar los dispositivos del usuario.

```
extend type Query  {
    getDevicesFromUser: GetDevicesFromUserPayload!
}

extend type Mutation {
    upsertDevice(data: UpsertDeviceInput!): UpsertDevicePayload!
    deleteDevice(data: DeleteDeviceInput!): Boolean!
}

type Device {
    id: ID!
    name: String!
    user: User!
    creation_date: Datetime!
    min_c: Float!
    average_consumption: Float!
    max_c: Float!
    freq_time: Float!
    turn_off: Boolean!
    sort: String!
    metering: Int!
}

type UpsertDevicePayload {
    device: Device
    success: Boolean!
    errors: [String!]
}

type DevicePayload {
    device: Device
    success: Boolean!
    errors: [String!]
}

type GetDevicesFromUserPayload {
    devices: [Device!]
    success: Boolean!
    errors: [String!]
}
```

Base de datos

(Se puede conectar con Postgres, MySQL, etc)

Como queremos poder ver todos los elementos guardados en cualquier momento, guardamos los datos en una base de datos.

Nosotros elegimos PostgreSQL porque es un sistema de gestión de bases de datos robusto con muchas opciones y que a futuro permitirá almacenar una gran cantidad de datos.

De igual forma usamos Flask-SQLAlchemy que se integra a la perfección con Flask y también permite interactuar con las bases de datos usando Python.

Conexión

Para construir la API, usamos una librería de Python llamada Ariadne, la cual ayuda a construir servidores de GraphQL. Esta librería es del tipo de esquema primero, lo que significa que el esquema escrito en SDL es la fuente de los datos.

Para esto solo necesitamos haber definido el esquema del cual se obtienen los datos, realizar algunos resolvers que manejan los distintos campos en el esquema y realizar la conexión entre ambos.

```python
from ariadne import (    Import "ariadne" could not be resolved
    load_schema_from_path,
    make_executable_schema,
    snake_case_fallback_resolvers,
)
from api.resolvers import query, mutation, datetime_scalar

types = load_schema_from_path("./api/schema/types/")

schema = make_executable_schema(
    types, query, mutation, datetime_scalar, snake_case_fallback_resolvers
)
```

Manejo de errores

De igual forma pensamos que cualquier aplicación necesita tener un control sobre errores o excepciones, nosotros implementamos unas cuantas que son sencillas pero cumplen su función a la perfección.

Estas son para el manejo de datos, que se revise si los datos ingresados no son los correctos o se encuentran incompletos.

```python
class WrongData(Exception):
    def __init__(
        self,
        message="Data provided for request is incomplete or incorrect",
        details="No details provided",
    ):
        self.message = f"Error: {message} > Details: {details}"
        super().__init__(self.message)
```

Otro control es sobre el acceso, esto lo realizamos para el control de usuarios. Dentro de los planes originales se sugiere que cada usuario tenga sus propios dispositivos y solo ellos tengan acceso a ellos, a menos que se cambien los permisos.

```python
class UserNotAllowed(Exception):
    def __init__(
        self,
        message="User not allowed to access resource",
        details="No details provided",
    ):
        self.message = f"Error: {message} > Details: {details}"
        super().__init__(self.message)
```

El último control que se implementó es para manejar el inicio de sesión, comprobar si el usuario se encuentra registrado o no.

```python
class UserNotFound(Exception):
    def __init__(self, message="User not found", details="No details provided"):
        self.message = f"Error: {message} > Details: {details}"
        super().__init__(self.message)
```

Implementación del Frontend por parte de equipos anteriores

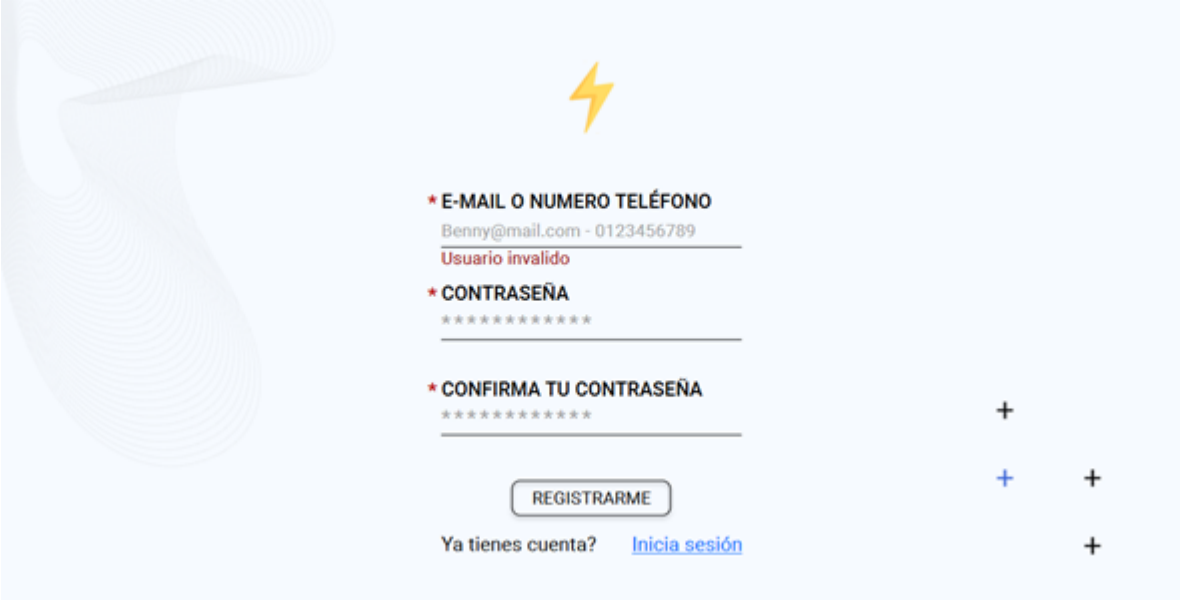
Iniciar sesión

Para iniciar sesión, debemos tener una cuenta ya registrada y debemos llenar los campos de e-mail y contraseña. Posteriormente, se debe de dar clic al botón “Entrar”.

Registrar nuevo usuario

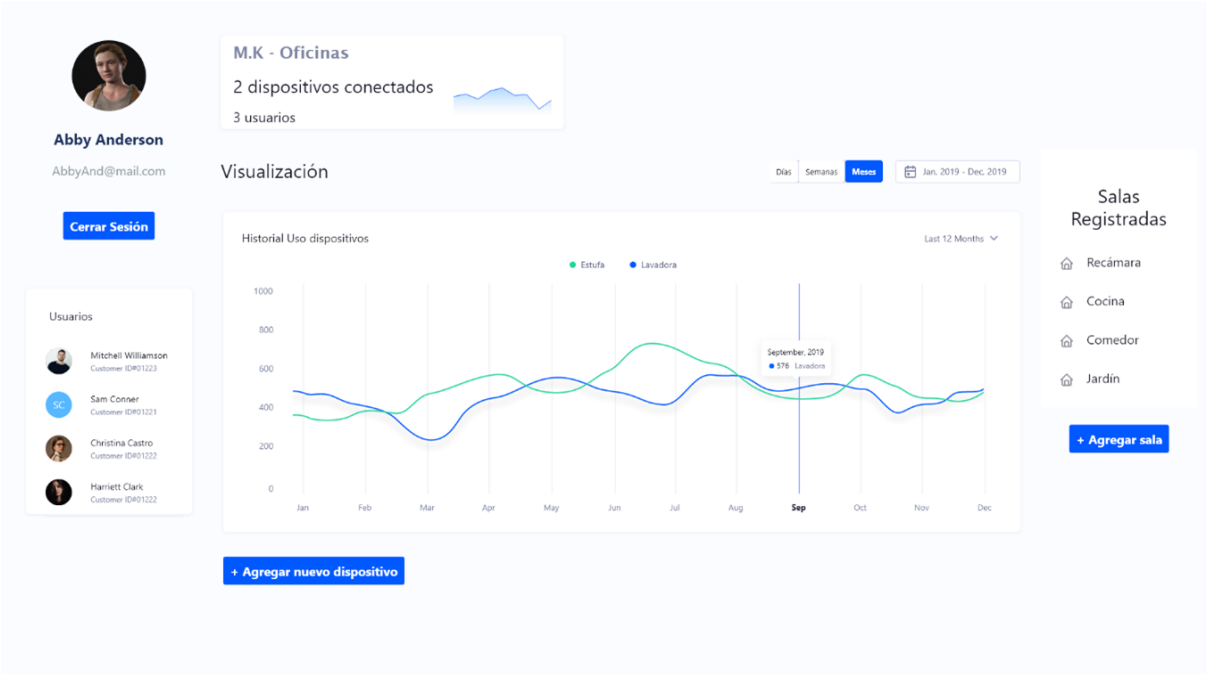
En caso de no tener registrada una cuenta, se puede crear una nueva cuenta al dar clic en la palabra regístrate en la pantalla de inicio de sesión.

Luego, debemos ingresar un correo electrónico y una contraseña.



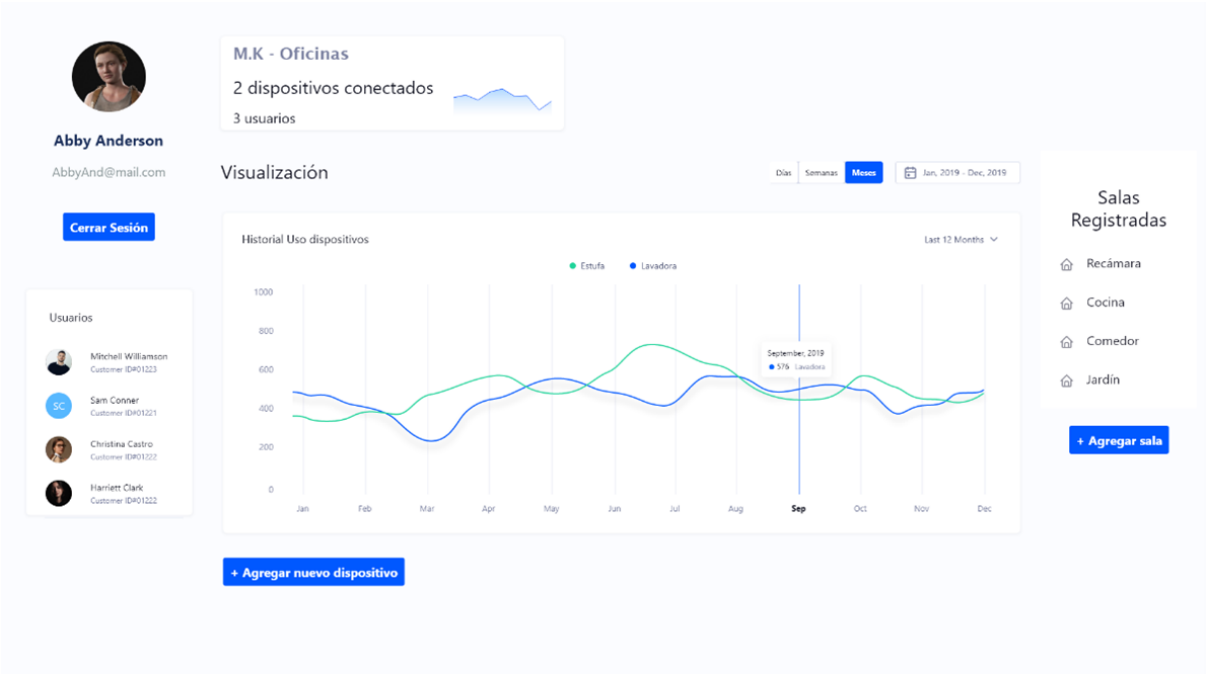
Página principal

Después de haber iniciado sesión, podemos observar la pantalla principal de la aplicación, que nos muestra datos diferentes, como lo son los datos de la cuenta, dispositivos que tenemos registrados y del lado derecho se tiene una barra con los espacios que registramos, junto con sus dispositivos.



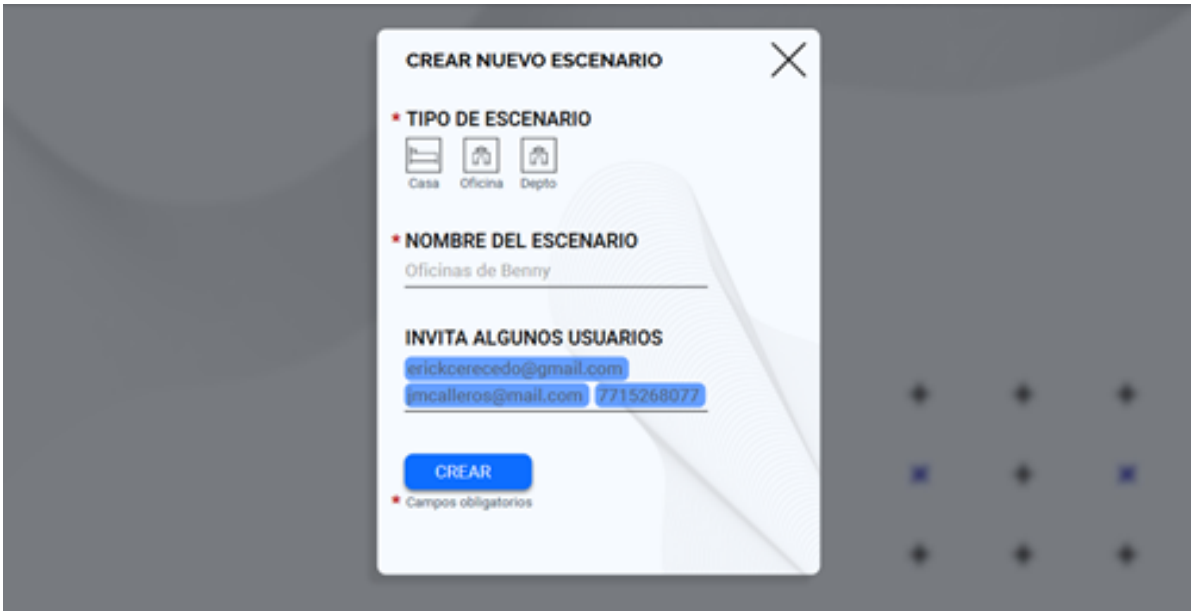
Registrar un nuevo escenario

Para poder registrar un nuevo escenario desde la página principal, debemos dirigirnos al lado derecho de la pantalla y le damos clic al botón de color azul que dice agregar sala.



Después de dar clic, nos va a aparecer una nueva ventana que nos va a pedir el nombre que le queremos colocar a este nuevo escenario, se recomienda que el nombre sea significativo de un espacio físico como puede ser una sala, una habitación, cocina, etc.

Luego de haberle puesto nombre a nuestro escenario, se le da clic al botón de crear.



Una vez creado el escenario, la aplicación nos va a mandar a la pantalla de inicio y del lado derecho ahora podemos ver el nuevo escenario ya agregado.