PLATAFORMA DE GESTIÓN DE EVENTOS

Documentación de la Plataforma

Proyecto de Estancias 2

Se busca que cualquier persona pueda hacer uso de la documentación para el mantenimiento y mejora de la plataforma de gestión para eventos para la Universidad Politécnica de Quintana

Realizado por la carrera de Ingeniería en software
202200461 / 202200498 / 202200517

Contenido

Introducción

Este proyecto surge de la necesidad de optimizar la gestión de eventos en la universidad, integrando tecnología de hardware y software para el conteo automático de asistentes y el registro detallado de eventos. A partir de un prototipo inicial que utiliza una ESP32 y un sensor PIR para contar los asistentes, el enfoque se amplió hacia una plataforma web completa que facilita el control y organización de eventos, brindando herramientas de consulta y generación de reportes en tiempo

Resumen Ejecutivo

Este proyecto consiste en una plataforma web de gestión de eventos para la universidad, que permite organizar y controlar la asistencia a diversos eventos académicos y culturales. Integra un sistema de hardware con una ESP32 y un sensor PIR para el conteo automatizado de asistentes en los eventos, lo cual alimenta directamente la plataforma web, permitiendo una gestión integral de eventos y asistencia.

Antecedentes

La idea de esta plataforma surgió a partir de un prototipo en ESP32 con un sensor PIR que se utilizó para el conteo automático de asistentes en eventos universitarios. Con la finalidad de ofrecer una herramienta completa de gestión, el proyecto evolucionó hacia una solución web que permitiera no solo el conteo de asistentes, sino también el registro de eventos, la generación de reportes y la consulta de históricos.

Tecnologías principales

React: es una biblioteca de JavaScript de código abierto desarrollada por Facebook que permite construir interfaces de usuario interactivas y dinámicas.

Componentización: Permite dividir la aplicación en componentes reutilizables, lo que facilita su desarrollo y mantenimiento.

Virtual DOM: Mejora el rendimiento de la aplicación al minimizar la manipulación directa del DOM real.

Comunidad Activa: Una amplia base de desarrolladores y recursos facilita la solución de problemas y la implementación de nuevas características.

Ecosistema: Compatible con diversas bibliotecas y herramientas como Redux, Axios y Chart.js.

Aplicación en este proyecto: React se utilizó para construir la interfaz de usuario de la plataforma web. Esto incluye las vistas para gestionar eventos, visualizar estadísticas y generar reportes. Además, su capacidad para manejar estados y eventos permite actualizar la información en tiempo real, proporcionando una experiencia interactiva y fluida.

Firebase: Es una plataforma de desarrollo de aplicaciones móviles y web proporcionada por Google, que ofrece una variedad de servicios backend.

Servicios utilizados:

Realtime Database: Almacena y sincroniza datos entre los usuarios en tiempo real, lo que es ideal para gestionar el conteo de personas y los datos de los eventos.

Hosting: Proporciona un servicio de alojamiento seguro y rápido para la plataforma web.

ESP32: El ESP32 es un microcontrolador de bajo costo y alta versatilidad, diseñado para aplicaciones de IoT (Internet de las cosas).

Razones por las que se hizo uso de este:

Wi-Fi y Bluetooth integrados: Permite conectividad inalámbrica para enviar datos a la plataforma web.

Procesador Dual-Core: Ofrece alta capacidad de procesamiento para manejar múltiples tareas simultáneamente.

Compatibilidad con sensores y periféricos: Ideal para conectar dispositivos como el sensor PIR y la pantalla OLED.

Aplicación en este proyecto: El ESP32 se encargó de recopilar datos del sensor PIR, procesarlos y enviar los resultados a Firebase. Además, interactúa con la pantalla OLED para mostrar el conteo en tiempo real y con los botones para ofrecer funcionalidades manuales como el reinicio del contador.

Objetivo general

Desarrollar una plataforma web que permita a la universidad gestionar de manera eficiente sus eventos y asistentes, combinando software y hardware para así integra el control automatizado de las asistencias y proporcionando reportes detallados, lo que mejorará la organización y toma de decisiones.

Objetivos específicos

- Integrar el hardware ESP32 y el sensor PIR para automatizar el conteo de asistentes en eventos
- Implementar un sistema de registros para los eventos con detalles como nombre del evento, tipo, asistentes, representantes, entre otros.
- Desarrollar un formulario que permita registrar y gestionar la cantidad de asistentes por evento.
- Desarrollar una funcionalidad que permita visualizar el historial de eventos y generar reportes de asistencia detallados.
- Asegurar que la plataforma tenga una interfaz amigable y accesible, que permita una gestión rápida y eficiente de los eventos.
- Crear un módulo de edición de eventos para modificar datos ya registrados.
- Habilitar la generación de reportes mensuales y por historial de eventos en formato PDF y/o Excel.
- Visualizar el historial de eventos y generar reportes de asistencia detallados de manera mensual.

Metodología de Desarrollo: Scrum

Para el desarrollo del proyecto, se adoptó la metodología Scrum, que permitió un enfoque ágil y colaborativo. El trabajo se dividió en Sprint de dos semanas, donde se priorizaron tareas clave según las necesidades del sistema. Al final de cada sprint, se realizaron reuniones de retroalimentación y planificación para el siguiente ciclo.

Fases del Desarrollo del Proyecto

1. Planificación y Análisis de Requerimientos

Actividades realizadas:

Reuniones iniciales con los usuarios (administrativos y organizadores de eventos) para definir los requerimientos funcionales y no funcionales.

Identificación de necesidades clave, como registro de eventos, control de asistencias y generación de reportes en formatos PDF y Excel.

Priorización de funcionalidades usando un Product Backlog.

Entregables:

Lista detallada de historias de usuario con criterios de aceptación claros.

Diagrama de casos de uso para visualizar el alcance funcional del sistema.

2. Diseño de la Interfaz y Estructura del Sistema

Actividades realizadas:

Creación de prototipos de baja y alta fidelidad con herramientas como Figma.

Diseño de la experiencia de usuario (UX) para garantizar la usabilidad de la plataforma.

Estructura inicial de carpetas y componentes del sistema en React.

Entregables:

Prototipo funcional de la interfaz.

Esquema modular del sistema frontend (componentes reutilizables).

3. Diseño de la Base de Datos

Actividades realizadas:

Análisis de las relaciones entre datos clave, como eventos, asistentes y reportes.

Creación de diagramas entidad-relación (ERD) para Firebase o una base de datos alternativa.

Definición de reglas de seguridad en Firebase para proteger la integridad y confidencialidad de los datos.

Entregables:

Modelo de datos con colecciones/tables como events, attendees, reports.

4. Desarrollo del Frontend

Actividades realizadas:

Implementación de componentes reutilizables para el sistema, como formularios, tablas y vistas de reportes.

Integración de librerías como Chart.js para visualizaciones interactivas.

Diseño responsivo utilizando CSS y frameworks como Bootstrap o Tailwind CSS.

Entregables:

Vistas funcionales para la gestión de eventos y generación de reportes.

5. Desarrollo del Backend

Actividades realizadas:

Configuración de Firebase para autenticar usuarios y gestionar datos en tiempo real.

Implementación de funciones para operaciones CRUD (crear, leer, actualizar y eliminar eventos).

Configuración de APIs necesarias para la comunicación con dispositivos externos (ESP32).

Entregables:

Backend funcional con endpoints seguros.

6. Integración de Hardware (ESP32)

Actividades realizadas:

Programación del ESP32 para contar asistentes en tiempo real utilizando sensores PIR.

Configuración del ESP32 para enviar datos al backend mediante un protocolo como MQTT o HTTP.

Pruebas de sincronización entre el hardware y la plataforma web.

Entregables:

Comunicación fluida entre hardware y software.

Datos de asistencias registrados automáticamente en el sistema.

Diagramas de flujos

Diagrama de flujo para la creación de un evento

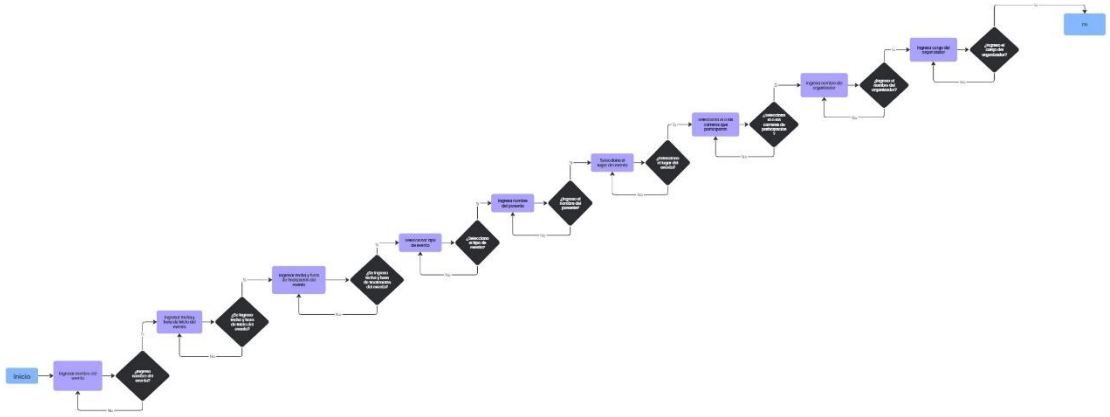


Diagrama de flujo para editar la información de un evento

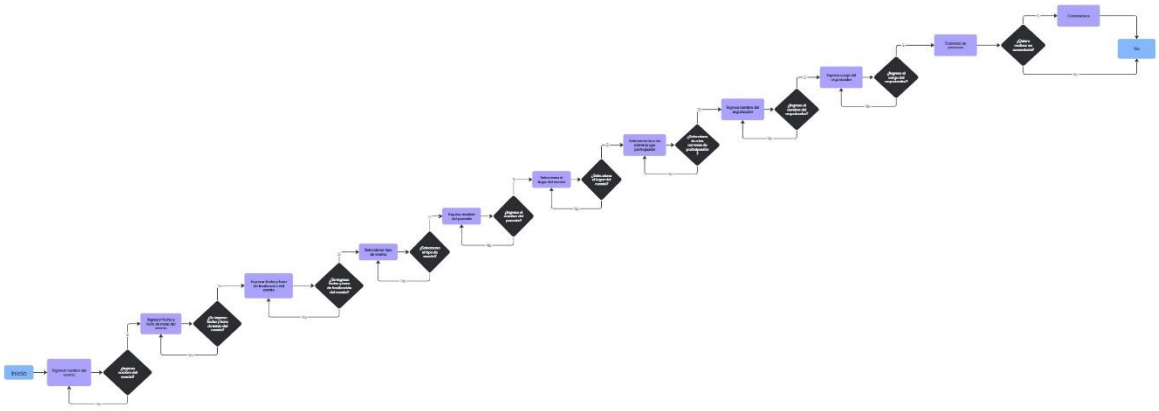
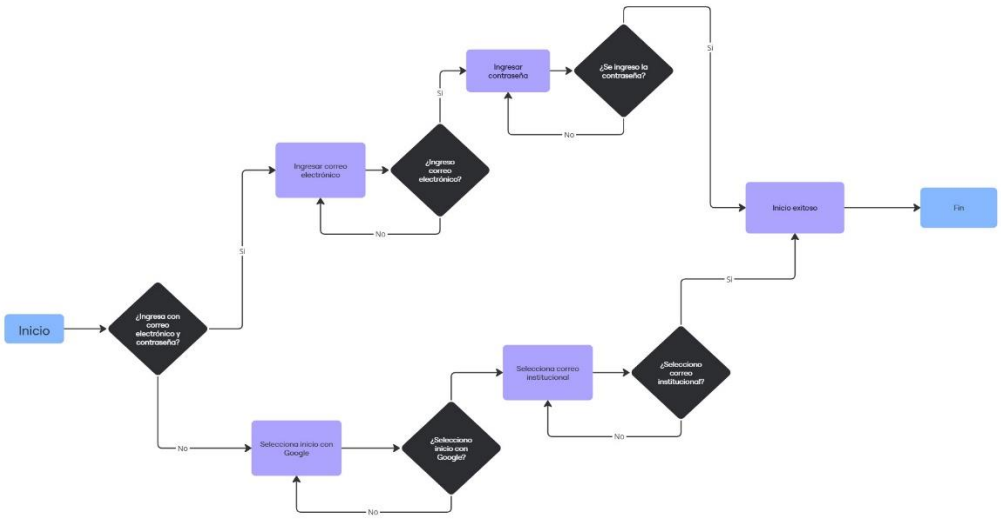


Diagrama de flujo de inicio de sesión



Arquitectura del sistema

Estructura del Frontend:

Rutas principales (App.js):

Autenticación con rutas protegidas:

ProtectedRoute: Valida si el usuario está autenticado antes de permitir acceso a ciertas rutas. Utiliza el contexto AuthContext para manejar el estado de autenticación.

Layout con navegación lateral:

Layout: Componente principal que integra el SideNavBar (barra lateral) y renderiza las rutas hijas mediante <Outlet />.

Rutas y módulos:

/login: Pantalla de login.

/: Página principal protegida (EventosPrincipal).

/mod3: Componente de formulario desplegable (MyDropdown).

/mod5: Tabla de reportes de eventos (EventReportTable).

/mod6: Historial de eventos (Historial).

/mod7: Vista previa (PrevView).

/mod4/eventos/:id_evento: Edición de eventos (Myformularioedit).

Librerías utilizadas y su propósito:

Autenticación y Rutas:

1. `react-router-dom:`
- Maneja rutas dinámicas en la aplicación.
- Proporciona componentes como Router, Routes, Route, y Navigate.
2. `jwt-decode:`
- Permite decodificar tokens JWT para verificar la autenticación y extraer información del usuario.
- Estilo y Componentes UI
3. `@fortawesome/fontawesome-free:`
- Incluye iconos para enriquecer la interfaz visual.
4. `sweetalert2:`
- Librería para mostrar alertas modales personalizadas, útil para confirmar o informar al usuario.
5. `react-icons:`
- Proporciona un conjunto amplio de iconos listos para usar en React.
- Tablas y Gráficos
6. `react-data-table-component:`
- Herramienta para crear tablas interactivas y personalizables.
7. `chart.js y react-chartjs-2:`
- chart.js: Librería para crear gráficos (barras, líneas, etc.).

- react-chartjs-2: Adaptador que permite usar chart.js directamente en React.
- Exportación de Documentos

8. `jspdf y jspdf-autotable:`

- jspdf: Permite generar archivos PDF.
- jspdf-autotable: Extensión para crear tablas dentro de documentos PDF.

9. `xlsx:`

- Permite generar y leer archivos Excel (formato .xlsx).
- Interacciones del Usuario

10. `react-toastify:`

- Muestra notificaciones o alertas no intrusivas para informar cambios o errores.

11. `react-to-print:`

- Habilita la funcionalidad de imprimir elementos en la página.
- Conexión con APIs

12. `axios:`

- Cliente HTTP para realizar solicitudes a APIs REST.

Estructura del Backend:

Este proyecto es un servidor backend desarrollado en Node.js utilizando Express.js como framework principal. El servidor proporciona funcionalidades esenciales para manejar roles, usuarios, eventos, tipos de eventos, y conexión con Firebase. Se configura con CORS para permitir solicitudes de orígenes específicos y soporta intercambio de datos en formato JSON.

Estos son los archivos principales que tenemos en el servidor Web:

- index.js: Es el archivo principal que inicializa el servidor y configura las rutas para manejar distintas funcionalidades del sistema.

Funcionalidades principales:

- Configuración de Express: Maneja datos en formato JSON y formularios codificados en URL.
- Middleware CORS: Permite solicitudes de un origen específico (http://localhost:3000) o de cualquier origen si se utiliza sin opciones específicas.

Carga de rutas:

Importa y utiliza rutas definidas para las siguientes funcionalidades:

Roles (routes/roles)

Firebase (routes/firebase)

Usuarios (routes/usuarios)

Eventos (routes/eventos)

Tipos de eventos (routes/tiposeventos)

DataFire (routes/datafire)

Servidor HTTP:

Inicia el servidor en el puerto definido en las variables de entorno (process.env.port) y escucha las solicitudes entrantes.

Código en Index.js:

```
const express = require("express");
const cors = require("cors");
const app = express();

// Configuración de middlewares
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cors());
```

```
// Configuración de rutas
app.use(require('./routes/roles'));
app.use(require('./routes/firebase'));
app.use(require('./routes/usuarios'));
app.use(require('./routes/eventos'));
app.use(require('./routes/tiposeventos'));
app.use(require('./routes/datafire'));

// Configuración del servidor
const port = process.env.port || 3000; // Puerto por defecto
app.listen(port, () => {
    console.log(`El servidor escucha en el puerto ${port}`);
});

module.exports = app;
```

Dependencias principales:

1. **express (^4.21.1):**
- Framework web rápido y flexible para Node.js.
- Uso: Para manejar rutas, middlewares, y solicitudes HTTP.
2. **cors (^2.8.5):**
- Middleware para habilitar CORS (Cross-Origin Resource Sharing).
- Uso: Permitir solicitudes desde http://localhost:3000 o cualquier origen.
3. **firebase (^11.0.1):**
- SDK oficial de Firebase para Node.js.
- Uso: Conexión y manipulación de datos almacenados en Firebase.
4. **google-auth-library (^9.15.0):**
- Biblioteca para manejar autenticación con Google.
- Uso: Probablemente para validar solicitudes a Firebase o Google APIs.
5. **mysql (^2.18.1):**
- Cliente MySQL para Node.js.
- Uso: Manejo de consultas y conexión con bases de datos MySQL.

Dependencias de desarrollo:

1. **dotenv (^16.4.1):**
- Biblioteca para cargar variables de entorno desde un archivo .env.
- Uso: Facilitar la configuración del puerto, credenciales, y claves de API.
2. **nodemon (^3.1.7):**
- Herramienta para desarrollo que reinicia automáticamente el servidor al detectar cambios en los archivos.
- Uso: Mejora la productividad durante el desarrollo.

Ruta de Eventos del Servidor:

La ruta eventos implementa la gestión de eventos mediante métodos HTTP para operaciones CRUD (crear, leer, actualizar y eliminar) en la tabla datosevento de la base de datos. Esta ruta conecta con las tablas relacionadas tiposeventos y datos para obtener información adicional asociada a cada evento.

Métodos implementados

1. GET - Obtener todos los eventos

Endpoint: /eventos

Descripción: Recupera todos los eventos de la base de datos, incluyendo detalles de tipo de evento y datos relacionados.

Consulta SQL

```
SELECT
    ev.id_evento,
    ev.nombre_evento,
    ev.fechahora_inicio,
    ev.tipo_evento,
    te.tipo_evento AS nombre_tipo_evento,
    ev.nombre_ponente,
    ev.lugar,
    ev.tipo_carrera,
    ev.nombre_organizador,
    ev.cargo_organizador,
    ev.comentarios,
    ev.fechahora_fin,
    ev.tipo_dato,
    d.count AS datos_count,
    d.end_date AS datos_end_date
FROM datosevento ev
INNER JOIN tiposeventos te ON ev.tipo_evento = te.id
INNER JOIN datos d ON ev.tipo_dato = d.id;
```

Respuesta exitosa (HTTP 200): Lista de eventos en formato JSON.

Tabla de Eventos: GetEventos

http://localhost:8888/eventos


```
[
    {
        "id_evento": 1,
        "nombre_evento": "Evento Prueba",
        "fechahora_inicio": "2024-11-14T07:04:00.000Z",
        "tipo_evento": 1,
        "nombre_tipo_evento": "Conferencia",
        "nombre_ponente": "Prueba",
        "lugar": "Biblioteca",
```

```
        "tipo_carrera": "Ingeniería en Biotecnología",
        "nombre_organizador": "PruebaOrganizador",
        "cargo_organizador": "Licenciado",
        "comentarios": "comentario",
        "fechahora_fin": "2024-11-15T07:04:00.000Z",
        "tipo_dato": "EVE0002",
        "datos_count": 35,
        "datos_end_date": "2024-11-14T05:00:00.000Z"
    },
    {
        "id_evento": 34,
        "nombre_evento": "PruebaIntegracion",
        "fechahora_inicio": "2024-12-08T06:12:00.000Z",
        "tipo_evento": 1,
        "nombre_tipo_evento": "Conferencia",
        "nombre_ponente": "Prueba",
        "lugar": "Auditorio",
        "tipo_carrera": "Público en General",
        "nombre_organizador": "Moises Zetina",
        "cargo_organizador": "Maestro",
        "comentarios": "N/A",
        "fechahora_fin": "2024-12-09T06:12:00.000Z",
        "tipo_dato": "EVE0020",
        "datos_count": 1,
        "datos_end_date": "2024-12-03T05:00:00.000Z"
    }
]
```


2. GET - Obtener un evento por ID

Endpoint: /eventos/:id

Descripción: Recupera un evento específico utilizando su ID.

Consulta SQL:


```
[
    {
        "id_evento": 1,
        "nombre_evento": "Evento Prueba",
        "fechahora_inicio": "2024-11-14T07:04:39.000Z",
        "tipo_evento": 1,
        "nombre_tipo_evento": "Conferencia",
        "nombre_ponente": "Prueba",
        "lugar": "Biblioteca",
        "tipo_carrera": "Ingeniería en Software",
        "nombre_organizador": "Prueba",
        "cargo_organizador": "Licenciado",
        "comentarios": null,
        "fechahora_fin": "2024-11-15T07:04:39.000Z",
        "tipo_dato": "EVE0002",
        "datos_count": 35,
        "datos_end_date": "2024-11-14T05:00:00.000Z"
    }
]
```

Parámetro URL: id - Identificador único del evento.
Respuesta exitosa (HTTP 200): Detalles del evento solicitado.

3. POST - Crear o actualizar un evento

Endpoint: /eventos

Descripción:

Si el cuerpo de la solicitud incluye la acción insert, se crea un nuevo evento.

De lo contrario, se actualizan los datos de un evento existente.

Parámetros del cuerpo (JSON):
http://localhost:8888/eventos


```
{
       "action”: "insert",
       "nombre_evento": "Evento Prueba2",
        "fechahora_inicio": "2024-11-14T07:04:39.000Z",
        "tipo_evento": 3,
        "nombre_ponente": "Prueba",
        "lugar": "Biblioteca",
        "tipo_carrera": "Ingeniería en Software",
        "nombre_organizador": "Prueba",
        "cargo_organizador": "Licenciado",
        "comentarios": null,
        "fechahora_fin": "2024-11-15T07:04:39.000Z",
        "tipo_dato": "EVE0004",
        "datos_end_date": "2024-11-14T05:00:00.000Z"
}
```


PostEventos con el action “UPDATE”
http://localhost:8888/eventos


```
{
       "action”: "update",
       "nombre_evento": "Evento Prueba2.2",
        "fechahora_inicio": "2024-11-14T07:04:39.000Z",
        "tipo_evento": 3,
        "nombre_ponente": "Prueba2",
        "lugar": "Biblioteca",
        "tipo_carrera": "Ingeniería en Software",
        "nombre_organizador": "Prueba2",
        "cargo_organizador": "Licenciado2",
        "comentarios": null,
        "fechahora_fin": "2024-11-15T07:04:39.000Z",
        "tipo_dato": "EVE0004",
        "datos_end_date": "2024-11-14T05:00:00.000Z",
        "id_evento": "2"
}
```


- Consultas SQL:

- INSERT INTO datosevento
  (nombre_evento, fechahora_inicio, tipo_evento, nombre_ponente, lugar, tipo_carrera, nombre_organizador, cargo_organizador, comentarios, fechahora_fin, tipo_dato)
  VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
  _____DESPUÉS EL UPDATE_____
- UPDATE datosevento
  SET nombre_evento = ?, fechahora_inicio = ?, tipo_evento = ?, nombre_ponente = ?, lugar = ?, tipo_carrera = ?, nombre_organizador = ?, cargo_organizador = ?, comentarios = ?, fechahora_fin = ?, tipo_dato = ?
  WHERE id_evento = ?;

Estas son las Respuesta exitosa (HTTP 201):
- Para inserciones: { "Item añadido correctamente": 1 }
- Para actualizaciones: { "Item editado correctamente": 1 }

4. DELETE - Eliminar un evento
Endpoint: /eventos/:id
Descripción: Elimina un evento específico utilizando su ID.
Consulta SQL:
- DELETE FROM datosevento WHERE id_evento = ?;

Mensaje:
```
{
    "Item eliminado": 1
}
```

Básicamente así funciona con las demás rutas, aquí te dejo las consultas de Endpoint de las demás rutas:

Get Usuarios haciendo una llave foránea con la tabla roles, peticiones
http://localhost:8888/usuarios
```
[
    {
        "id_usuario": 1,
        "nombre_usuario": "Nombre",
        "correo": "correo@gmail.com",
        "contraseña": 12345678,
        "cargo_usua": 2,
        "tipo_rol": "Usuario"
    },
    {
        "id_usuario": 2,
        "nombre_usuario": "Nombre2",
        "correo": "nombre2@gmail.com",
        "contraseña": 654321,
        "cargo_usua": 2,
        "tipo_rol": "Usuario"
    }
```

]

GetUsuarios Id, se busca el usuario por medio de su clave única
http://localhost:8888/usuarios/1


```
[
    {
        "id_usuario": 1,
        "nombre_usuario": "Nombre",
        "correo": "correo@gmail.com",
        "contraseña": 12345678,
        "cargo_usua": 2,
        "tipo_rol": "Usuario"
    }
]
```


postUsuarios, se insertan los datos en la tabla de usuarios con el action "INSERT"
http://localhost:8888/usuarios


```
{
"action": "insert",
"nombre_usuario": "Nombre3",
"correo":"Nombre3@gmial.com",
"contraseña": "87654321",
"cargo_usua": "2"
}
```


PostUsuarios, se insertan saldo en la tabla de usuarios con el action "UPDATE"

http://localhost:8888/usuarios


```
{
"action": "update",
"nombre_usuario": "Nombre3",
"correo":"Nombre3@gmial.com",
"contraseña": "87654321",
"cargo_usua": "2",
"id_usuario": "4"
}
```


DelUsuario, se eliminan los datos de la tabla de usuarios
http://localhost:8888/usuarios/4

Mensaje:


```
{
    "Item eliminado": 1
}
```

Código en Arduino

Este programa cuenta eventos detectados mediante un sensor PIR o incrementados manualmente con botones. Utiliza un módulo OLED para mostrar el conteo y se conecta a Firebase para registrar los eventos en tiempo real. Además, gestiona el fin del día almacenando los datos recopilados y manteniendo un contador de eventos.

1. Arduino IDE

- Instalación de las siguientes bibliotecas:
  - ✓ Adafruit_GFX
  - ✓ Adafruit_SSD1306
  - ✓ FirebaseESP32
  - ✓ Configuración de la placa ESP32 en el gestor de tarjetas de Arduino IDE.

2. Conexión Wi-Fi

- Red Wi-Fi disponible con las credenciales especificadas en el código:
  - ✓ SSID: nombredelared
  - ✓ Contraseña: contraseñ@delared

3. Firebase

- Configuración de una base de datos en tiempo real. Obtener:
  - ✓ URL del host: https://arquitectura-8d40d-default-rtdb.firebaseio.com
  - ✓ Token de autenticación: YB2udBASW9hVPIOodZZPfFjeoRm3W56jL9YOEJFn.

Configuraciones Importantes

1. Pantalla OLED:

- Dimensiones: 128x64 píxeles.
- Dirección I2C: 0x3C.
- Se inicializa con la biblioteca Adafruit_SSD1306.

2. Sensor PIR:

- Conectado al pin digital 18.
- Detecta movimiento y activa incrementos automáticos del contador.

3. Botones:

- Botón Incrementar: conectado al pin 22.
- Botón Reiniciar: conectado al pin 21.
- Funcionan con resistencias pull-up internas.

4. Firebase:

- Al iniciar, recupera el último valor del contador de eventos desde Firebase para mantener la continuidad de los datos.
- Actualiza los datos en tiempo real cada vez que hay un incremento (manual o por sensor) o un reinicio.

Código en Arduino

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <WiFi.h>
#include "FirebaseESP32.h"
#include <time.h>
//ESTE EL ES EL BUENO
// CREDENCIALES WIFI
#define WIFI_SSID "rectoria"
#define WIFI_PASSWORD "R3ct0r1@"

#define FIREBASE_HOST "https://arquitectura-8d40d-default-
rtdb.firebaseio.com"
#define FIREBASE_AUTH "YB2udBASW9hVPIOodZZPfFjeoRm3W56jL9YOEJFn"

FirebaseData firebaseData;
FirebaseConfig config;
FirebaseAuth auth;

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

const int pirPin = 18;
const int buttonIncrementPin = 22;
const int buttonResetPin = 21;
//se cambio los botones
int pirState = LOW;
int count = 0;
int eventCounter = 1;
bool manualMode = false;
bool lastButtonIncrementState = HIGH;
bool lastButtonResetState = HIGH;

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = -18000;
const int daylightOffset_sec = 0;

void setup() {
  Serial.begin(115200);
  Wire.begin(19, 23);

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
```

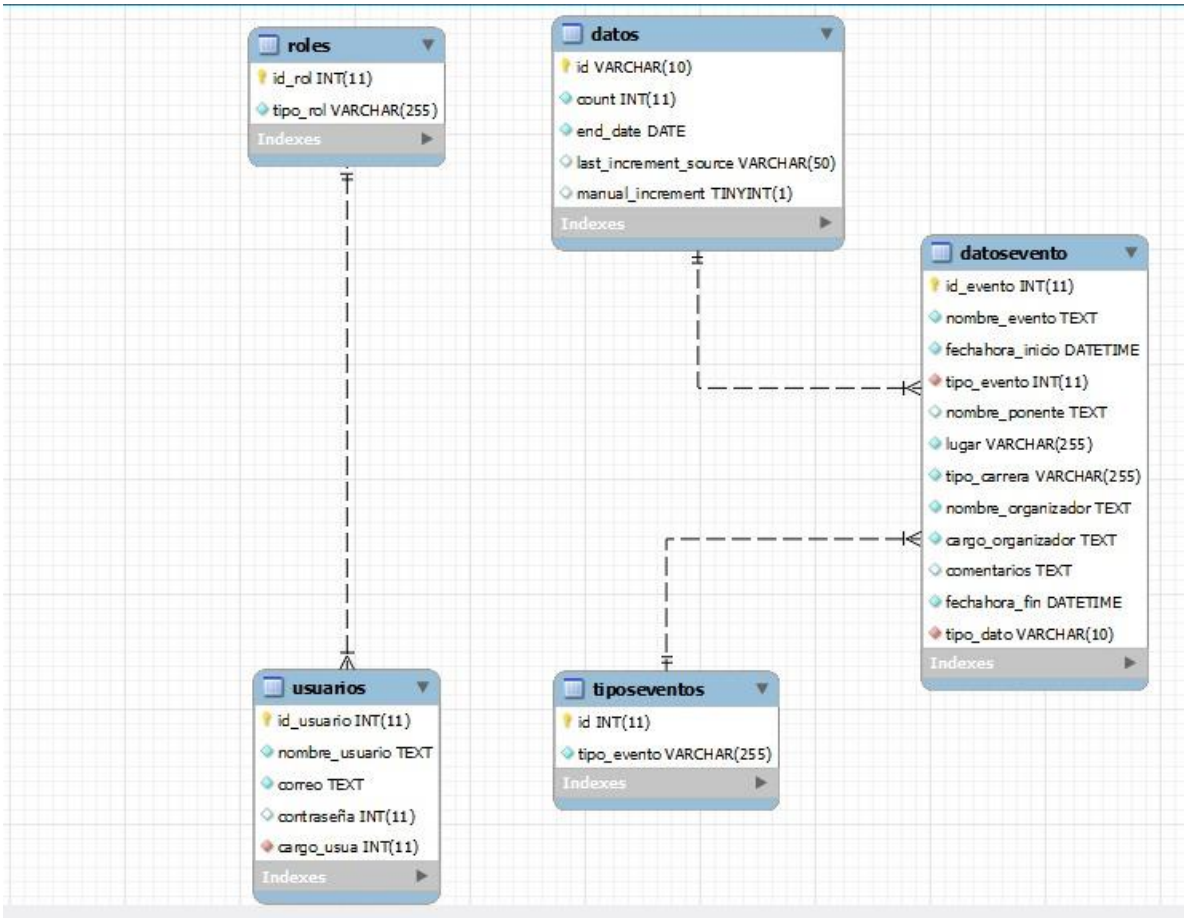
```
    for (;;);
  }

  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Contador:");
  display.setCursor(0, 30);
  display.print(count);
  display.display();

  pinMode(pirPin, INPUT);
  pinMode(buttonIncrementPin, INPUT_PULLUP);
  pinMode(buttonResetPin, INPUT_PULLUP);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Conectando al Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.println("Conectado al Wi-Fi");

  config.host = FIREBASE_HOST;
  config.signer.tokens.legacy_token = FIREBASE_AUTH;
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);

  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

  // Recupera el último eventCounter desde Firebase
  if (Firebase.ready() && Firebase.getInt(firebaseData,
"/lastEventCounter")) {
    eventCounter = firebaseData.intData() + 1;  // Usa el siguiente valor
disponible
    Serial.println("Último contador de eventos cargado: " +
String(eventCounter));
  } else {
    Serial.println("No se pudo obtener el contador de eventos, iniciando
en 1.");
    eventCounter = 1;
  }
}
```

```
void updateFirebase(int count, const char* source) {
  if (Firebase.ready()) {
    char paddedCounter[5];
    snprintf(paddedCounter, sizeof(paddedCounter), "%04d", eventCounter);
    String eventPath = "/EVE" + String(paddedCounter) + "/";
    Firebase.setInt(firebaseData, eventPath + "count", count);
    Firebase.setBool(firebaseData, eventPath + "manualIncrement", (source
== "manual"));
    Firebase.setString(firebaseData, eventPath + "lastIncrementSource",
source);
  }
}

void saveEndOfDay() {
  char dateString[20];
  struct tm timeInfo;
  if (getLocalTime(&timeInfo)) {
    strftime(dateString, sizeof(dateString), "%Y-%m-%d", &timeInfo);
  }

  char paddedCounter[5];
  snprintf(paddedCounter, sizeof(paddedCounter), "%04d", eventCounter);
  String eventPath = "/EVE" + String(paddedCounter) + "/";
  Firebase.setString(firebaseData, eventPath + "endDate", dateString);
  Serial.println("Fecha de fin del evento guardada: " +
String(dateString));

  // Guarda el nuevo eventCounter en Firebase para el siguiente evento
  Firebase.setInt(firebaseData, "/lastEventCounter", eventCounter);
  eventCounter++;
}


void loop() {
  int val = digitalRead(pirPin);
  bool buttonIncrementState = digitalRead(buttonIncrementPin);
  bool buttonResetState = digitalRead(buttonResetPin);

  if (lastButtonIncrementState == HIGH && buttonIncrementState == LOW) {
    count++;
    display.clearDisplay();
    display.setCursor(0, 0);
    display.print("Contador:");
    display.setCursor(0, 30);
    display.print(count);
    display.display();
```

```
    Serial.println("Contador incrementado manualmente. Total: " +
String(count));
    updateFirebase(count, "manual");
    delay(100);
  }

  if (lastButtonResetState == HIGH && buttonResetState == LOW) {
    saveEndOfDay();
    count = 0;
    display.clearDisplay();
    display.setCursor(0, 0);
    display.print("Contador:");
    display.setCursor(0, 30);
    display.print(count);
    display.display();
    Serial.println("Contador reiniciado. Nuevo evento: " +
String(eventCounter));
    updateFirebase(count, "manual");
    delay(100);
  }

  lastButtonIncrementState = buttonIncrementState;
  lastButtonResetState = buttonResetState;

  if (val == HIGH) {
    if (pirState == LOW) {
      if (!manualMode) {
        count++;
        display.clearDisplay();
        display.setCursor(0, 0);
        display.print("Contador:");
        display.setCursor(0, 30);
        display.print(count);
        display.display();
        Serial.println("Movimiento detectado! Total: " + String(count));
        updateFirebase(count, "sensor");
        delay(500);
      }
    }
    pirState = HIGH;
  } else {
    if (pirState == HIGH) {
      Serial.println("Sin movimiento");
    }
    pirState = LOW;
  }
```

```
}
```

Modelado de la base de datos



Hardware:

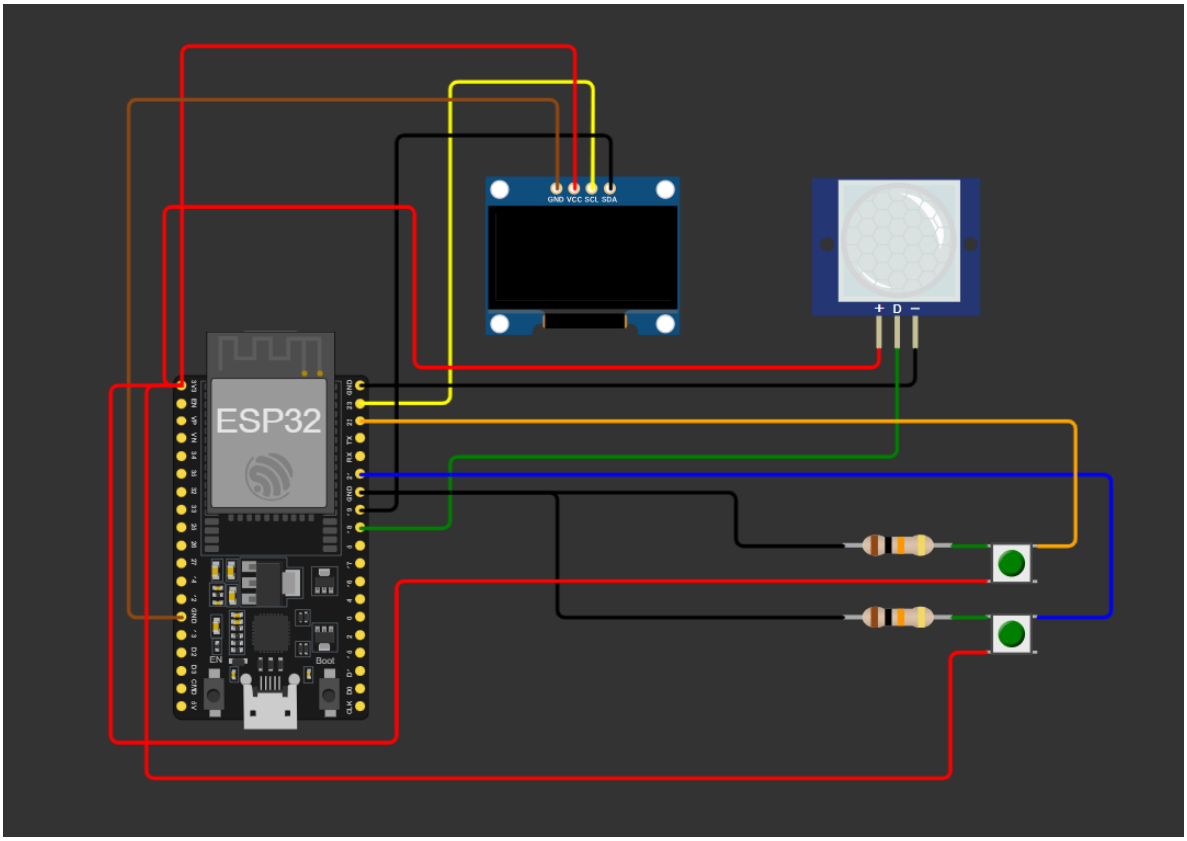
- Materiales:
  Cable de transferencia de datos
- ESP32
- Pantalla oled 0.96 pulgadas
- Cables
- Sensor PIR Hc-sr501
- Botones pulsadores
- Protoboard
- Resistencias 10K

El listado anterior es sobre los materiales con los cuales se realizó el circuito, sin embargo, entre otros materiales para su aseguramiento en calidad se usaron los siguientes

  - Cautín
  - Pasta para soldar
  - Estaño
  - Cinta aislante
  - Cinta doble cara
  - Cable para pelar El diseño del sistema se basa en la integración de varios componentes electrónicos conectados a un protoboard y controlados por un ESP32. La pantalla OLED mostrará el número de personas detectadas y el ganador al azar. Los botones permitirán reiniciar el contador y realizar un conteo manual en caso de fallos en el sensor PIR.
  - **Diagrama de Conexión**

- Pantalla OLED:
  - VCC -> 3.3V del ESP32
  - GND -> GND del ESP32
  - SDA -> GPIO 21 del ESP32
  - SCL -> GPIO 22 del ESP32
- Botones:
  - Un botón conectado a un pin digital (ej. GPIO 14) y GND con una resistencia de pull-up.
  - Otro botón conectado a otro pin digital (ej. GPIO 27) y GND con una resistencia de pull-up.
- Sensor de movimiento PIR:
  - VCC -> 3V del ESP32
  - OUT -> D18 del ESP32
  - GND -> GND del ESP32

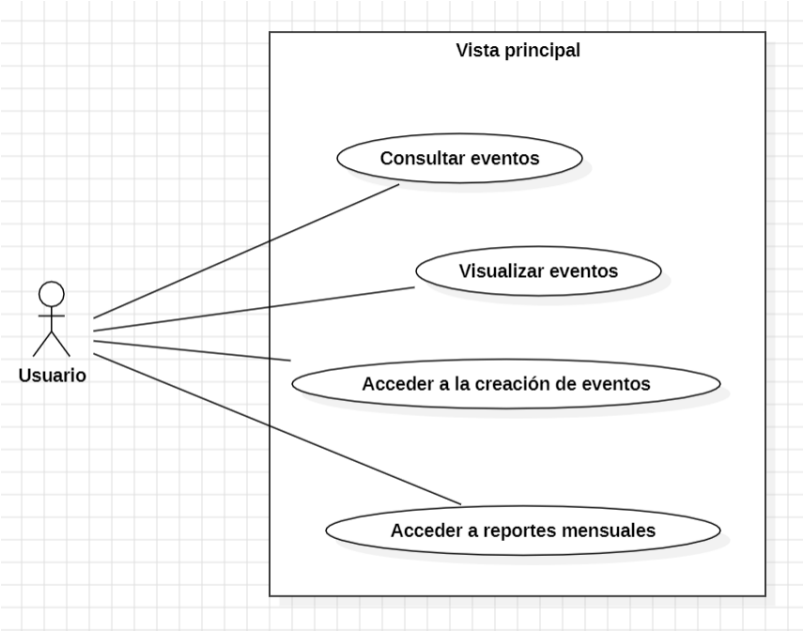
Diagrama del Hardware



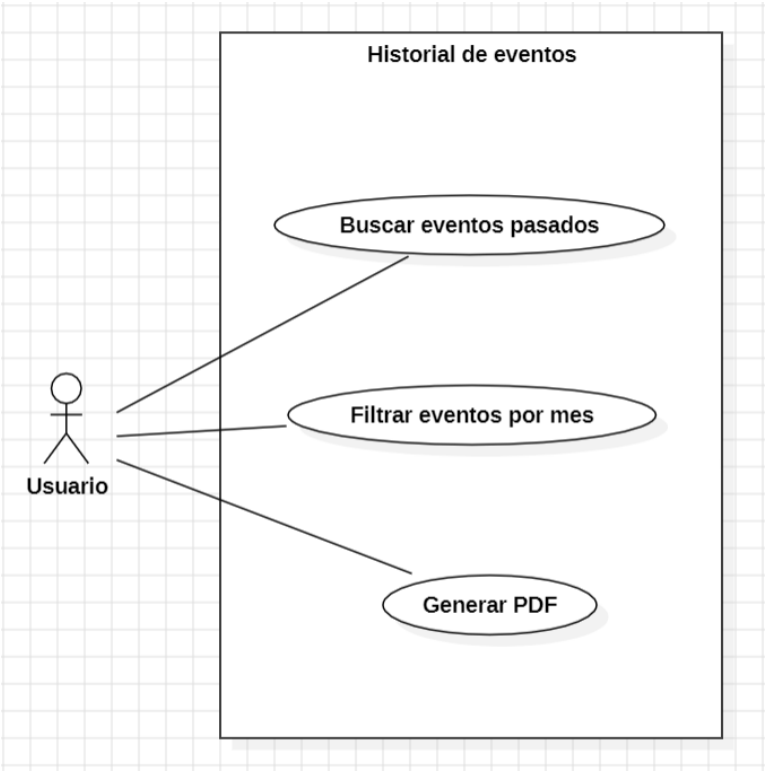
Si necesita más información ver el anexo [2]

Casos de uso

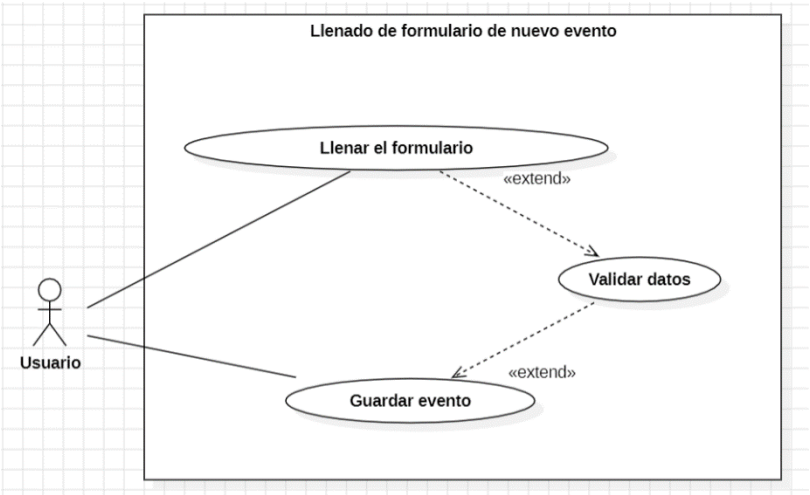
Vista Principal:



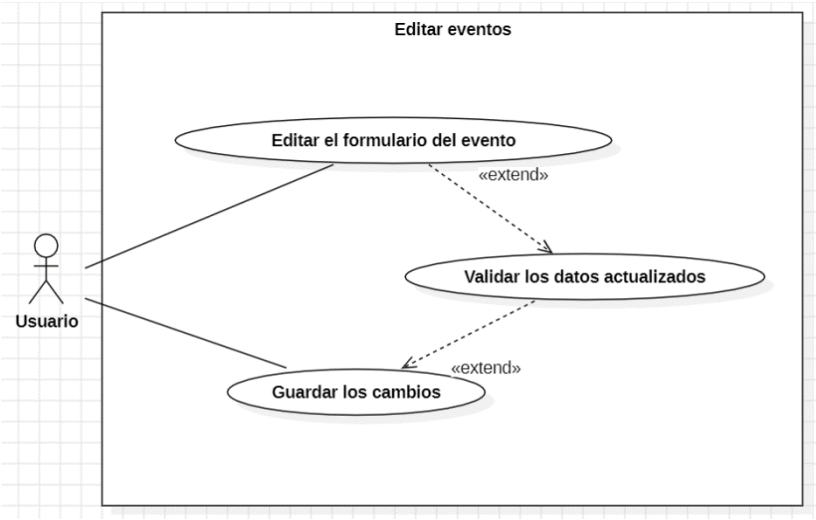
Historial de eventos:



Registro de un nuevo evento:



Edición de un evento:

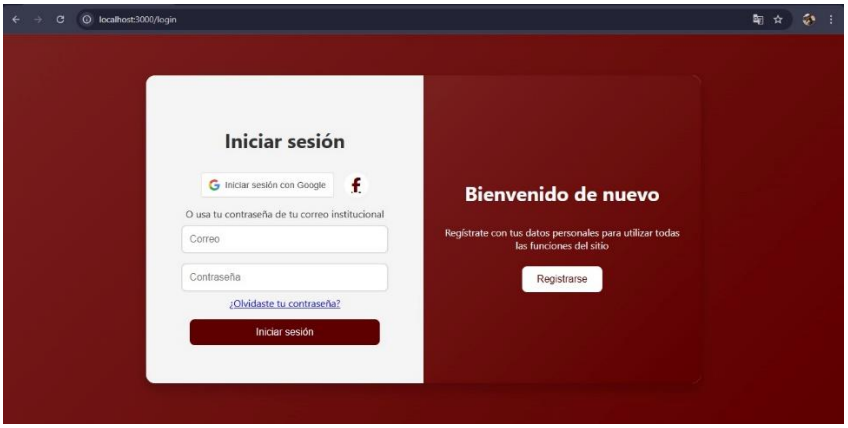


Pruebas y Validación (UNITARIAS Y DE INTEGRACIÓN)

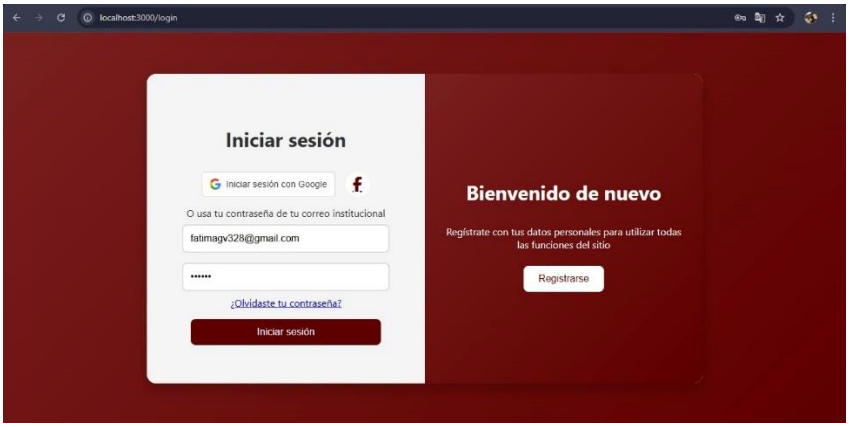
Actividades realizadas:

Pruebas unitarias: Validación de componentes frontend y funciones backend.

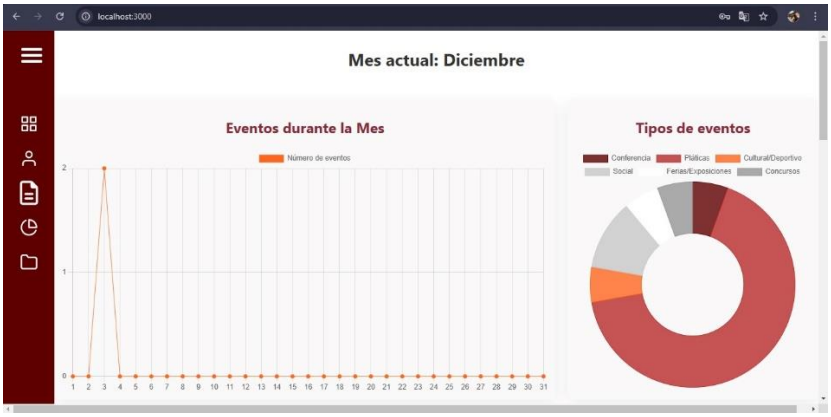
UNITARIA	
Proceso	Inicio de sesión
Descripción	Verificar que el sistema permita iniciar sesión con datos válidos
Entrada	Correo: fatimagv328@gmail.com
Resultado esperado	El sistema permite el acceso de manera correcta y nos dirige al inicio de la página
Resultado obtenido	El sistema nos permitió acceder de manera exitosa.
Código a validar	



La siguiente ventana será donde encontraremos dos formas de inicio de sesión. En esta prueba se llevó a cabo los datos que nos piden los campos.

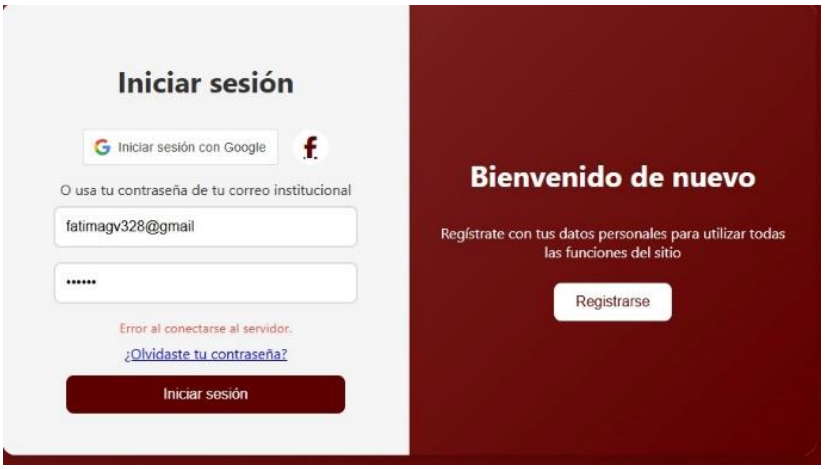


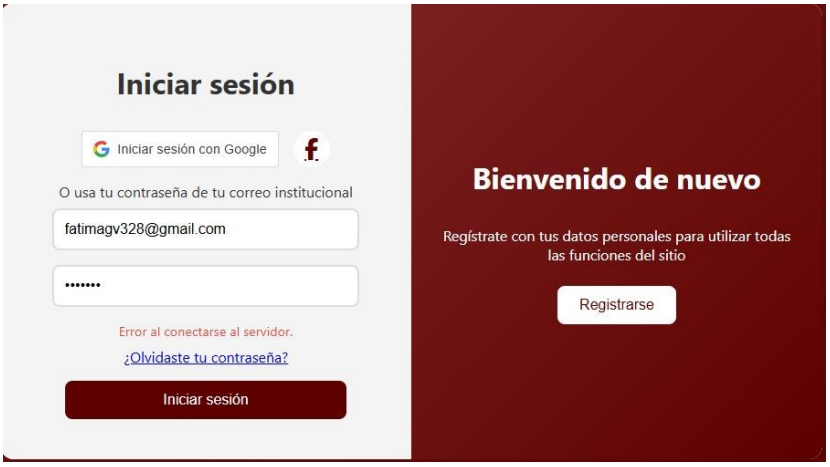
Ingresamos los datos que nos piden los campos, en este caso el correo electrónico y la contraseña.



Cuando dimos clic en iniciar sesión los datos en los campos fueron validados con la BD. Dando así un inicio de sesión exitoso.

Posibles advertencias en el inicio de sesión:

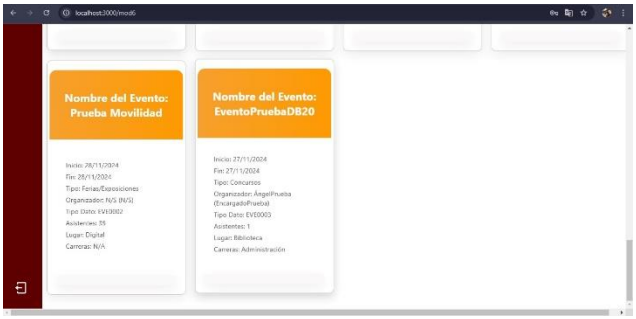




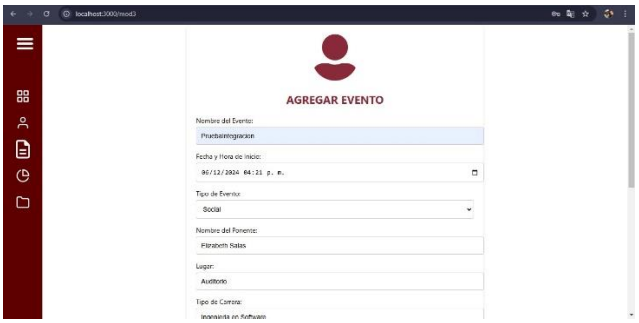
Si al dar clic en iniciar sesión nos marca el mensaje de “Error al conectarse al servidor”, quiere decir que ya sea el correo electrónico o la contraseña no coindicen con la información que está registrada en la BD.

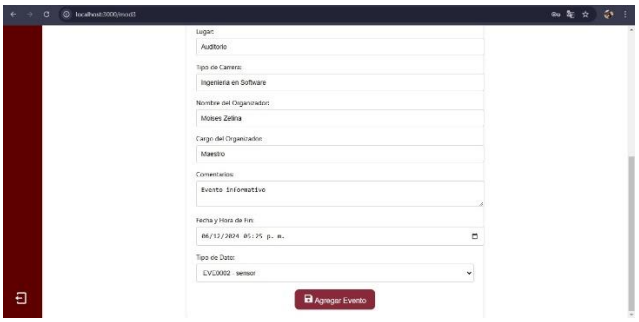
Pruebas de integración: Validación de flujos completos, como registro de eventos y generación de reportes.

INTEGRACIÓN	
Proceso	Comunicación entre dos módulos
Descripción	Verificar la comunicación entre el módulo Agregar evento y Visualizar los eventos
Entrada	Información de prueba para el registro de un evento
Resultado esperado	Al agregar un evento este se muestre dónde están todos los demás eventos que ya fueron registrados.
Resultado obtenido	Se agrego y visualizo con éxito el evento
Código a validar	



En esta ventana podemos ver los eventos que ya existen en el sistema.





Realizamos el registro del evento de prueba, ingresamos todos los datos que nos pide el formulario y le damos clic en “Agregar Evento”.

El evento que se agregó aparece en el módulo de visualización. En el módulo de visualización podrá mostrarse toda la información con la que se registró, pero además también con el número de asistentes.

Pruebas de usuario: Feedback del personal administrativo para mejorar la experiencia de uso.

Entregables:

Informe de pruebas con errores identificados y corregidos.

Validación final con usuarios clave.

Requisitos:

Requisitos funciones

- RF01 – Registro de Eventos: El sistema permitirá registrar eventos con detalles como nombre, tipo de evento, ponentes, fecha, lugar, duración, y número de asistentes esperados.
- RF02 – Registro de Asistencias: El sistema permitirá registrar la asistencia de las personas a los eventos mediante el conteo automático usando un sensor PIR.
- RF03 – Edición de Eventos: Los administradores podrán editar los eventos ya creados, modificando la fecha, hora, lugar, tipo de evento y número de asistentes.
- RF04 – Eliminación de Eventos: El sistema permitirá eliminar eventos registrados.
- RF05 – Generación de Reportes: El sistema generará reportes detallados en formato PDF y Excel con la información de los eventos, incluyendo asistentes y ponentes.
- RF06 – Historial de Eventos: El sistema permitirá visualizar eventos pasados en función de la fecha y otros criterios.
- RF07 – Visualización de Estadísticas: El sistema mostrará estadísticas de eventos y asistentes para facilitar la toma de decisiones.
- RF08 – Integración con hardware.: Configuración y conexión del ESP32 y el sensor PIR para el conteo de asistentes.

Requisitos no funcionales

- RNF01 – Rendimiento: La plataforma debe poder manejar la gestión de hasta 100 eventos mensuales con un máximo de 500 asistentes por evento, sin comprometer el tiempo de respuesta del sistema, que debe ser inferior a 3 segundos para las consultas.
- RNF02 – Escalabilidad: El sistema debe ser escalable, permitiendo añadir más eventos y asistentes sin afectar su rendimiento.
- RNF03 – Seguridad: El sistema debe proteger los datos personales de los asistentes y administradores mediante encriptación de las contraseñas y acceso restringido con autenticación.
- RNF04 – Compatibilidad: El sistema debe ser compatible con los navegadores más utilizados (Chrome, Firefox, Edge, Safari) y debe adaptarse a diferentes dispositivos (responsive design).
- RNF05 – Usabilidad: La interfaz debe ser amigable y fácil de usar, con una curva de aprendizaje mínima para los administradores que gestionarán los eventos.
- RNF06 – Mantenibilidad: El código debe estar bien documentado para facilitar su mantenimiento y posibles ampliaciones futuras.
- RNF07 – Disponibilidad: La plataforma debe estar disponible en cualquier momento siempre que se cuente con una buena conexión a Internet, garantizando que los usuarios puedan acceder a ella en cualquier momento.

Mejoras a futuro y próximos usos

El ESP32 requiere una conexión estable al Internet, por lo que causa problemas si no se tiene una buena conexión, así que en recomendación del equipo desarrollador para algún próximo desarrollador o desarrolladores es hacer un cambio de tarjeta para que el funcionamiento sea sin Internet durante el conteo de los asistentes, así mismo poder registrar entrada y salida como opción extra dado que actualmente es únicamente de entrada y no cuenta con una estructura comercial de diseño.

Como siguiente mejora se sugiere poder implementar un registro más exacto sobre los estudiantes con el uso de credenciales o matrículas (esto mismo podría aplicarse a una solución de seguridad dentro del plantel universitario).

Anexos

Anexo 1. Funcionamiento del hardware

El sensor PIR (Passive Infrared) es un dispositivo que detecta la radiación infrarroja emitida por los objetos en movimiento. En el caso de nuestro sistema de conteo de personas, el sensor PIR se utiliza para detectar la presencia de personas en un área determinada.

Cuando una persona entra en el rango de detección del sensor PIR, su cuerpo emite radiación infrarroja que es detectada por el sensor. Esto genera una señal eléctrica que es enviada al ESP32, que procesa la información y envía un pulso al contador.

El contador es un circuito que incrementa un valor cada vez que recibe un pulso del ESP32. En este caso, el contador incrementa el número de personas detectadas cada vez que el sensor PIR detecta movimiento. La pantalla OLED muestra el número total de personas detectadas en tiempo real, actualizando la información cada vez que se produce un nuevo movimiento.

La pantalla OLED es una pantalla de visualización que muestra el número de personas detectadas y otros mensajes relevantes. Está conectada al ESP32 y se actualiza en tiempo real según la información recibida.

El sistema también cuenta con dos botones: el botón de conteo manual y el botón de reinicio. El botón de conteo manual permite al usuario incrementar el contador manualmente, en caso de que el sensor PIR no detecte movimiento. El botón de reinicio permite restablecer el contador y la pantalla OLED a su estado inicial.

El ESP32 es el cerebro del sistema, que coordina todas las funciones y procesa la información recibida del sensor PIR, los botones y la pantalla OLED. También se encarga de enviar los datos del contador a una página web asociada, donde se pueden visualizar gráficos y estadísticas detalladas

Anexo 2. Glosario

ESP32: Un ESP32 es un microcontrolador de bajo costo y bajo consumo de energía que incorpora tecnología Bluetooth y Wi-Fi.

Sensor PIR: Dispositivo utilizado para detectar movimiento mediante la radiación infrarroja emitida por los cuerpos.

OLED: Pantalla de diodos orgánicos que emite luz y se utiliza para mostrar información visual.

NPM: Gestor de paquetes utilizado para administrar dependencias en proyectos de JavaScript.

Plataforma web: Interfaz que permite visualizar las estadísticas y datos del sistema en tiempo real

Casos de uso: Un caso de uso es una lista de tareas que los actores pueden realizar para lograr un objetivo con un sistema. Son la base para el desarrollo de un producto y se utilizan para crear la estructura y el flujo de trabajo.

Diagrama de procesos: Un diagrama de procesos, también conocido como diagrama de flujo, es una representación visual de los pasos que se siguen para completar una tarea o lograr un resultado. Es una herramienta que ayuda a identificar ineficiencias, redundancias y etapas críticas en un proceso.

APIs: Una API, o interfaz de programación de aplicaciones, es un conjunto de reglas y protocolos que permiten a las aplicaciones informáticas comunicarse entre sí e intercambiar datos. Es decir, son un software intermediario que facilita la integración de funcionalidades en estructuras ya existentes. En ocasiones los desarrolladores pueden crear su propia API para que otras aplicaciones se comuniquen con sus aplicaciones.

Servicios de APIs: Un servicio en una API es una funcionalidad o conjunto de funcionalidades específicas que una API expone para que otras aplicaciones o sistemas puedan interactuar con ellas. Las API permiten la comunicación entre diferentes softwares mediante peticiones y respuestas.

MANUAL DE USO

PLATAFORMA DE GESTIÓN DE EVENTOS

Documentación de la Plataforma de Gestión de Eventos

Configuración de la Página React:

1. Asegurarse de tener Node.js y npm instalados.
2. En la raíz del proyecto React, instala las dependencias: `npm install react-router-dom`
3. Crea los archivos necesarios:
   - Componentes:
     - ✓ SideNavBar: Barra lateral con enlaces a las diferentes rutas.
     - ✓ Contextos:
     - ✓ AuthContext: Maneja la autenticación de usuarios.
   - Páginas:
     - ✓ EventosPrincipal: Página principal con datos del sistema.
     - ✓ MyDropdown: Formulario interactivo.
     - ✓ EventReportTable: Reportes de eventos.
     - ✓ Historial: Historial de datos.
     - ✓ PrevView: Vista previa de configuraciones.
     - ✓ Login: Pantalla de inicio de sesión.
   - Estilos:
     - ✓ Crea un archivo de estilos CSS para personalizar la interfaz.
4. Configura las rutas en el archivo App.js como se muestra en el código proporcionado. Asegúrate de:
   - Proteger las rutas principales con ProtectedRoute.
   - Usar Layout para incluir la barra lateral y renderizar el contenido dinámicamente.

Ejecución

1. Inicia el servidor de React:
   npm start
2. Accede a http://localhost:3000 en tu navegador.
3. Ingresa tus credenciales de usuario para acceder al sistema.

   Navega por las diferentes rutas:

   - / para la página principal.
   - /mod2 para el formulario.
   - /mod3 para el formulario.
   - /mod4 para el formulario.
   - /mod5 para los reportes.
   - /mod6 para el historial.
   - /mod7 para el formulario.

Sincronización entre Arduino y React

- El ESP32 envía los datos a Firebase en tiempo real.
- La página React consulta los datos desde un servidor Web que consume los datos que existen en la Firebase y los muestra en las diferentes secciones.

Instrucciones de uso del Servidor:

1. Configuración inicial:

- Clona el repositorio y navega al directorio del proyecto: https://github.com/EliSalas1/plataformaeventos.git
- Instala las dependencias ejecutando: `npm install`

2. Configurar variables de entorno:

- Crea un archivo .env en la raíz del proyecto.
- Configura el puerto y otras claves necesarias: `port=3000`

3. Iniciar el servidor:

En desarrollo:

- bash
- Copiar código: `npm run start`

Con nodemon:

- bash
- Copiar código: `npx nodemon index.js`

Pasos para la Ejecución en el código en Arduino:

1. Preparación del Hardware:
- Conectar el ESP32 con los periféricos según los pines definidos.
- Asegurarse de que la pantalla OLED y el sensor PIR estén funcionales.
- Conectar los botones de manera estable.
2. Configuración del Software:
- Subir el código al ESP32 utilizando Arduino IDE.
- Asegurarse de que las credenciales de Wi-Fi sean correctas.
- Configurar Firebase con el host y el token proporcionados.
3. Verificación:
- Al iniciar, el ESP32 intentará conectarse a la red Wi-Fi. Se mostrará un mensaje en el monitor serie indicando si la conexión fue exitosa.
- La pantalla OLED mostrará el contador inicial en 0.
- Incrementar manualmente o con el sensor PIR actualizará el contador en pantalla y en Firebase.

Advertencias

- Conexión Wi-Fi: El programa no funcionará correctamente si no puede conectarse a la red Wi-Fi.
- Firebase: Asegúrate de que la base de datos esté configurada y los permisos sean correctos para escritura.