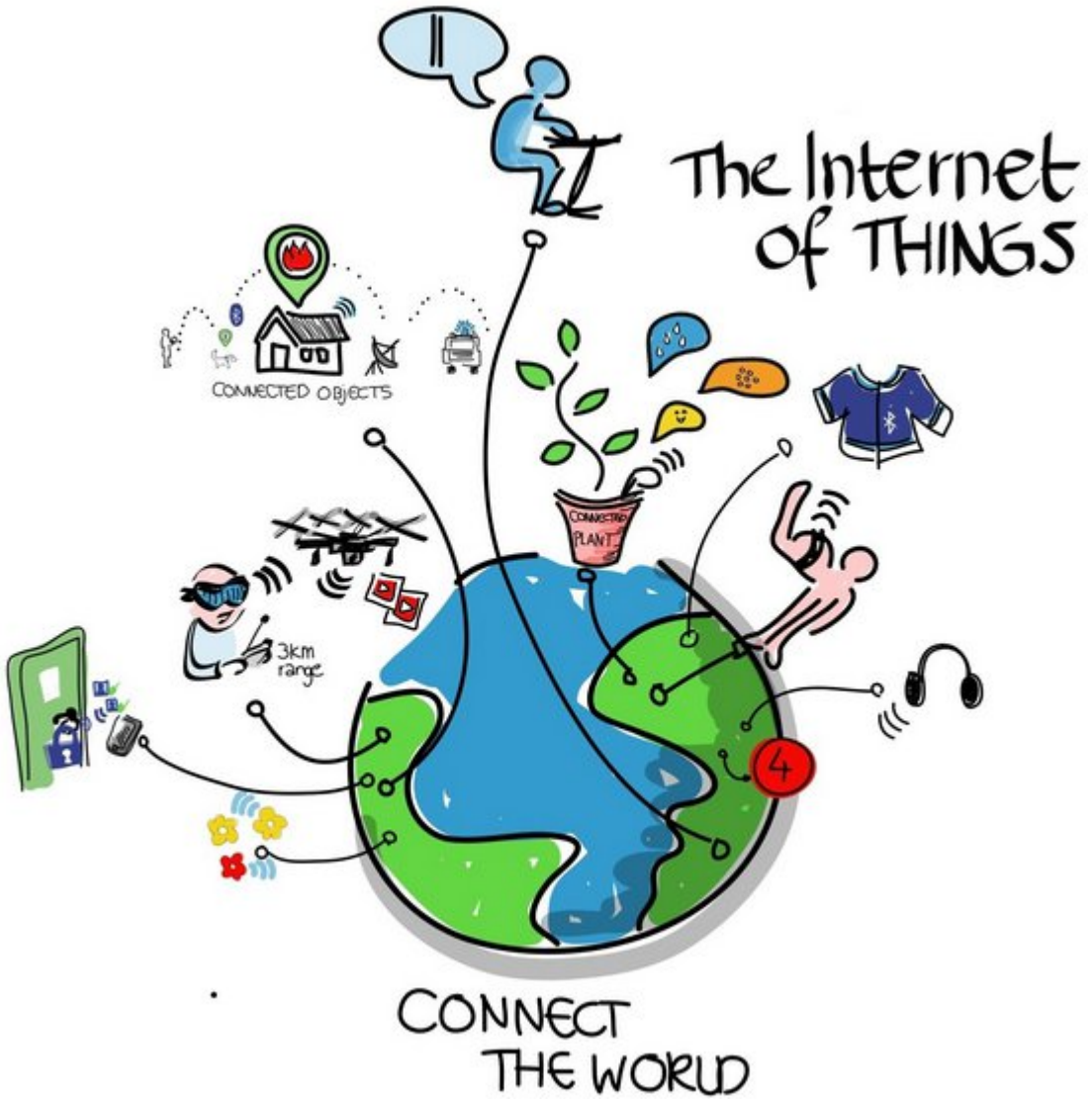
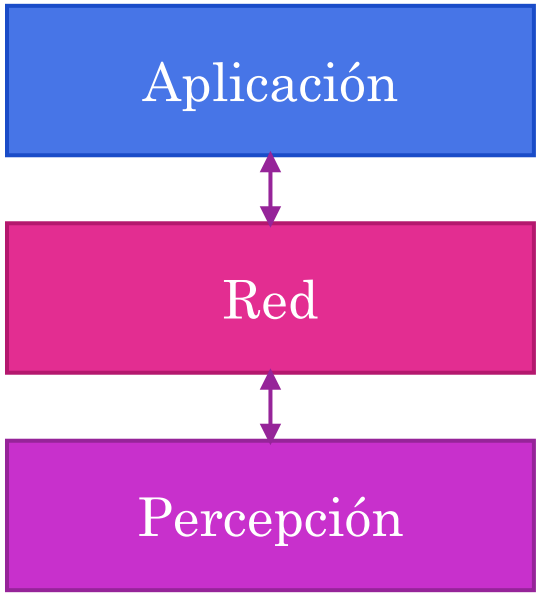
IoT

Una infraestructura de red global dinámica con capacidades de auto-configuración basada en protocolos de comunicación estándares e interoperables donde las cosas físicas y las virtuales tienen identidades, atributos físicos, personalidades virtuales y usan interfaces inteligentes, y están perfectamente integradas en la red de información.

CERP-IoT. (2010). Vision and Challenges for Realising the Internet of Things. (H. G. Sundmaeker, Ed.) Brussels, Belgium: European Commission.


The Internet of THINGS
CONNECTED OBJECTS
3km range
CONNECT THE WORLD

Arquitectura



WiFi

- Wireless Access Point (access point o AP).
- Estación.
- Basic Service Set (BSS).
- Dirección MAC.
- Service Set Identifier (SSID).

ESP32 como estación

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "lwip/err.h"
#include "lwip/sys.h"

#define EXAMPLE_ESP_WIFI_SSID      "mi_ssid"
#define EXAMPLE_ESP_WIFI_PASS      "mi_contrasena"
#define EXAMPLE_ESP_MAXIMUM_RETRY  5

/* FreeRTOS event group to signal when we are connected*/
static EventGroupHandle_t s_wifi_event_group;

/* The event group allows multiple bits for each event, but we only care about two events:
 * - we are connected to the AP with an IP
 * - we failed to connect after the maximum amount of retries */
#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

static const char *TAG = "wifi station";
static int s_retry_num = 0;
```

```c
void app_main(void)
{
    //Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES ||
            ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    ESP_LOGI(TAG, "ESP_WIFI_MODE_STA");
    wifi_init_sta();
}
```

El propósito es guardar la configuración de WiFi (SSID, contraseña) y otros parámetros que facilitan la reconexión cuando el ESP32 se reinicie o apague.

```
void wifi_init_sta(void)
{
    s_wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_STA_GOT_IP,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_got_ip));

    wifi_config_t wifi_config = {
        .sta = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .password = EXAMPLE_ESP_WIFI_PASS,
            .threshold.authmode = WIFI_AUTH_WPA2_PSK,
        },
    };
```

```
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA) );
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
    ESP_ERROR_CHECK(esp_wifi_start() );

    ESP_LOGI(TAG, "wifi_init_sta finished.");

    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
            pdFALSE,
            pdFALSE,
            portMAX_DELAY);

    if (bits & WIFI_CONNECTED_BIT) {
        ESP_LOGI(TAG, "connected to ap SSID:%s password:%s",
                 EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS);
    } else if (bits & WIFI_FAIL_BIT) {
        ESP_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",
                 EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS);
    } else {
        ESP_LOGE(TAG, "UNEXPECTED EVENT");
    }
}
```

Revisar **wifi_event_t** en: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_wifi.html#_CPPv412wifi_event_t

```
static void event_handler(void* arg, esp_event_base_t event_base,
                          int32_t event_id, void* event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        if (s_retry_num < EXAMPLE_ESP_MAXIMUM_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG,"connect to the AP fail");
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}
```

esp_err_t **esp_event_handler_instance_register**(
esp_event_base_t **event_base**, /* La base de eventos que agrupa varios tipos de eventos relacionados. Por ejemplo WIFI_EVENT */
int32_t **event_id**, /* El ID específico del evento dentro de event_base que se quiere manejar */
esp_event_handler_t **event_handler**, /* Apuntador a la función que va a manejar el evento. Será invocada cuando ocurra el evento especificado */
void ***event_handler_arg**, /* Argumento opcional a pasar a la función manejadora de eventos. */
esp_event_handler_instance_t ***instance** /* Apuntador para almacenar la instancia del manejador de eventos. Puede ser NULL si no se quiere almacenar */
)

La firma del manejador de eventos es:

void (*esp_event_handler_t)(void* handler_arg, esp_event_base_t base, int32_t id, void* event_data)

esp_err_t **esp_wifi_set_mode**(
wifi_mode_t **mode** /* Modo de operación de WiFi */
)

enum `wifi_mode_t`

Values:

enumerator `WIFI_MODE_NULL`

null mode

enumerator `WIFI_MODE_STA`

WiFi station mode

enumerator `WIFI_MODE_AP`

WiFi soft-AP mode

enumerator `WIFI_MODE_APSTA`

WiFi station + soft-AP mode

enumerator `WIFI_MODE_NAN`

WiFi NAN mode

enumerator `WIFI_MODE_MAX`

esp_err_t **esp_wifi_set_config**(
wifi_interface_t **interface**, /* Interfaz WiFi: Usar WIFI_IF_STA para modo estación y WIFI_IF_AP para modo AP */
wifi_config_t ***conf** /* Parámetros de configuración de la interfaz */
)

esp_err_t **esp_wifi_start**(void) /* Inicia WiFi de acuerdo a la configuración actual */

ESP32 como servidor web

```
#include <esp_wifi.h>
#include <esp_event.h>
#include <esp_log.h>
#include <esp_spiffs.h>
#include <nvs_flash.h>
#include <esp_netif.h>
#include <esp_http_server.h>

static const char *TAG = "web_server";
```

```
void app_main(void) {
    // Inicializar NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES
        || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ESP_ERROR_CHECK(nvs_flash_init());
    }

    // Inicializar SPIFFS
    init_spiffs();

    // Inicializar WiFi en modo Access Point
    wifi_init_softap();

    // Iniciar el servidor web
    start_webserver();
}
```

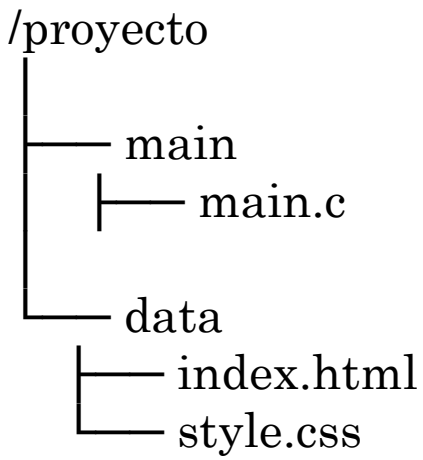
```c
void init_spiffs(void) {
    esp_vfs_spiffs_conf_t conf = {
        .base_path = "/spiffs",
        .partition_label = NULL,
        .max_files = 5,
        .format_if_mount_failed = true
    };

    esp_err_t ret = esp_vfs_spiffs_register(&conf);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Error al inicializar SPIFFS (%s)", esp_err_to_name(ret));
        return;
    }

    size_t total = 0, used = 0;
    ret = esp_spiffs_info(NULL, &total, &used);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "No se pudo obtener la información de la partición SPIFFS (%s)", esp_err_to_name(ret));
    } else {
        ESP_LOGI(TAG, "SPIFFS: Tamano total: %d, Usado: %d", total, used);
    }
}
```

```c
void wifi_init_softap(void) {
    // Inicializar WiFi
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());

    esp_netif_create_default_wifi_ap();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    const char mi_ssid[] = "mi_esp_ap";

    wifi_config_t wifi_config = {
        .ap = {
            .ssid_len = strlen(mi_ssid),
            .channel = 1,                       // Canal WiFi
            .password = "123456789",            // Contraseña del AP
            .max_connection = 4,                // Número máximo de conexiones permitidas
            .authmode = WIFI_AUTH_WPA_WPA2_PSK  // Modo de autenticación
        },
    };

    memcpy(wifi_config.ap.ssid, mi_ssid,  strlen(mi_ssid)); // Nombre de la red WiFi

    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));  // Configurar como AP
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());

    ESP_LOGI(TAG, "ESP32 AP iniciado. SSID:%s password:%s channel:%d",
             wifi_config.ap.ssid, wifi_config.ap.password, wifi_config.ap.channel);
}
```

```c
/* Configurar y registrar los handlers */
void start_webserver(void) {
    httpd_handle_t server = NULL;
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    if (httpd_start(&server, &config) == ESP_OK) {
        httpd_uri_t html_uri = {
            .uri      = "/",
            .method   = HTTP_GET,
            .handler  = html_get_handler,
            .user_ctx = NULL
        };
        httpd_register_uri_handler(server, &html_uri);

        httpd_uri_t css_uri = {
            .uri      = "/style.css",
            .method   = HTTP_GET,
            .handler  = css_get_handler,
            .user_ctx = NULL
        };
        httpd_register_uri_handler(server, &css_uri);

        ESP_LOGI(TAG, "Web server iniciado");
    }
}
```

```c
/* Handler para servir archivos HTML */
esp_err_t html_get_handler(httpd_req_t *req) {
    FILE* file = fopen("/spiffs/index.html", "r");
    if (file == NULL) {
        ESP_LOGE(TAG, "Error al abrir el archivo para lectura");
        httpd_resp_send_404(req);
        return ESP_FAIL;
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        httpd_resp_sendstr_chunk(req, line);
    }
    httpd_resp_sendstr_chunk(req, NULL); // Finalizar la respuesta
    fclose(file);

    return ESP_OK;
}
```

```c
/* Handler para servir archivos CSS */
esp_err_t css_get_handler(httpd_req_t *req) {
    FILE* file = fopen("/spiffs/style.css", "r");
    if (file == NULL) {
        ESP_LOGE(TAG, "Error al abrir el archivo para lectura");
        httpd_resp_send_404(req);
        return ESP_FAIL;
    }

    httpd_resp_set_type(req, "text/css");
    char line[256];
    while (fgets(line, sizeof(line), file)) {
        httpd_resp_sendstr_chunk(req, line);
    }
    httpd_resp_sendstr_chunk(req, NULL);
    fclose(file);

    return ESP_OK;
}
```

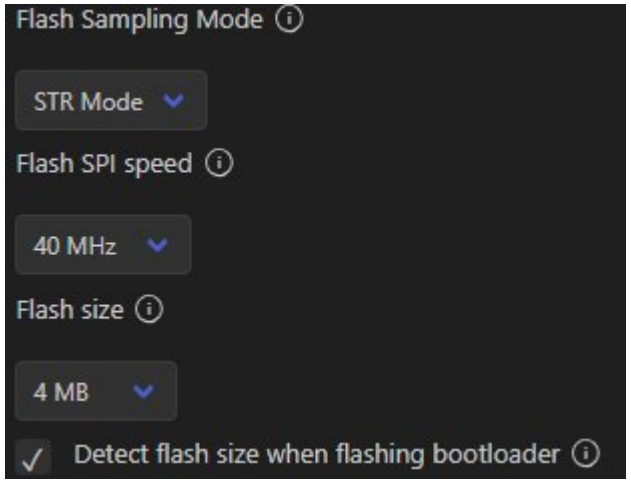

/proyecto
main
main.c
data
index.html
style.css

Archivo **partitions.csv**

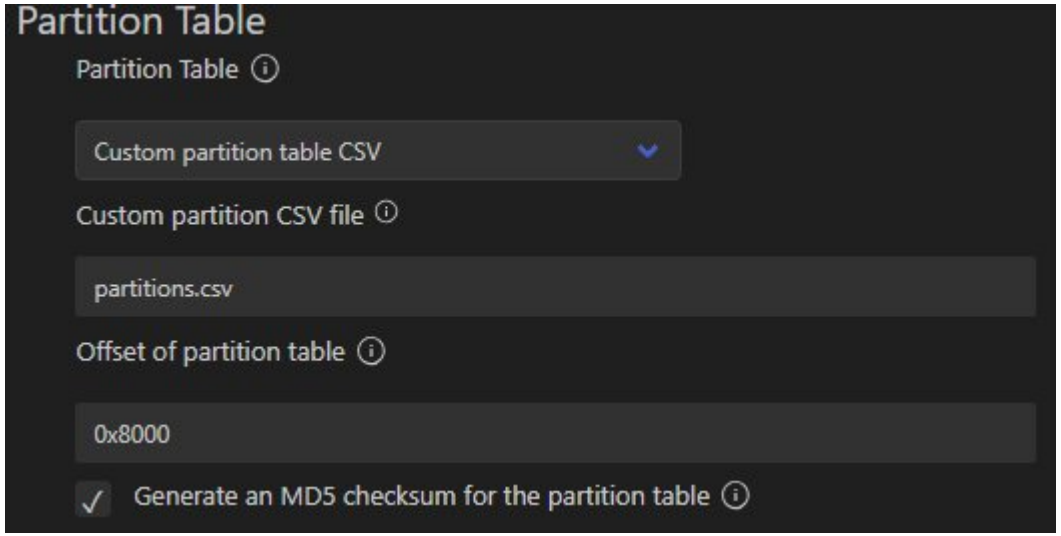
```
# Name,   Type, SubType, Offset,  Size, Flags
# Note: if you change the phy_init or app partition offset, make sure to change the offset in Kconfig.projbuild
nvs,      data, nvs,     ,        0x6000,
phy_init, data, phy,     ,        0x1000,
factory,  app,  factory, ,        1M,
storage,  data, spiffs,  ,        1M
```

1. Abre el archivo CMakeLists.txt que está en el folder main y agrega está línea al final:
spiffs_create_partition_image(storage ../data FLASH_IN_PROJECT)

2. Click en menuconfig: 
3. Selecciona Serial Flasher config.
4. Configura los siguientes parámetros:



5. Selecciona Partition Table.
6. Indica los siguientes parámetros para usar las particiones de partitions.csv



Tarea:

- Revisar la documentación del ESP-IDF sobre WiFi, HTTP y SPIFFS.
  - https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/api-reference/network/esp_wifi.html
  - https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/api-reference/protocols/esp_http_server.html
  - https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/api-reference/storage/spiffs.html

- Ejecutar los ejemplos vistos en clase.