# Documentación del Proyecto: Sistema de Gestión de Parqueadero

Miguel Caiza
Universidad de las Fuerzas Armadas - ESPE
Email: macaiza16@espe.edu.ec

***Abstract*—Este documento describe el diseño, implementación y funcionalidades de un sistema de gestión de parqueadero desarrollado en C++. El sistema incluye registro de vehículos, historial de eventos, visualización en consola en formato circular, creación de respaldos y restauración de datos. Se detalla el diseño de cada clase y las interacciones entre los componentes del sistema.**

## I. Introducción

El sistema de gestión de parqueadero es una solución orientada a objetos diseñada para facilitar la administración de vehículos en un entorno controlado. Este documento detalla el diseño modular, las principales funcionalidades y los aspectos técnicos del proyecto.

## II. Estructura General del Proyecto

El proyecto se compone de los siguientes módulos principales:

- **Clase `ParkingManager`:** Controla la lógica principal del sistema.
- **Clase `Vehicle`:** Representa los datos de un vehículo registrado.
- **Clase `HistoryManager`:** Gestiona el historial de eventos.
- **Clase `Cypher`:** Cifrado con modelo de César.
- **Clase `TokenManager`:** Tokens para facilitar procesos de recolección de datos.
- **Clase `InputValidator`:** Valida las entradas del usuario.
- **Archivos de datos:** Almacenan la información persistente.

El flujo principal del programa incluye el registro de vehículos, consulta de información, generación de respaldos y recuperación de datos.

## III. Clases del Proyecto

### A. Clase `Vehicle`

La clase `Vehicle` almacena la información de un vehículo, incluyendo la placa, el propietario y el tipo de vehículo.

Listing 1. Declaración de `Vehicle`

```
class Vehicle {
private:
    std::string plate;
    std::string ownerName;
    std::string ownerId;
    std::string ownerPhone;
    std::string vehicleType;

public:
    Vehicle(const std::string& plate, const std::
        string& ownerName,
            const std::string& ownerId, const std::
                string& ownerPhone,
            const std::string& vehicleType);

    std::string getPlate() const;
    std::string getOwnerName() const;
    std::string serializeBasic() const;
};
```

### B. Clase `HistoryManager`

Gestiona el historial de eventos, incluyendo entradas y salidas de vehículos.

Listing 2. Declaración de `HistoryManager`

```
class HistoryManager {
private:
    std::string historyFile;

public:
    explicit HistoryManager(const std::string&
        historyFile);
    void logEvent(const std::string& plate, const
        std::string& eventType);
    std::vector<ParkingEvent> getAllEvents() const;
};
```

### C. Clase `InputValidator`

Valida las entradas del usuario, como placas, nombres y fechas.

Listing 3. Declaración de `InputValidator`

```
class InputValidator {
public:
    static std::string getValidatedPlate();
    static bool isValidPlate(const std::string&
        plate);
    static std::string getValidatedText();
};
```

## IV. Implementación Técnica

### A. Validación de Entradas

La función `getValidatedPlate()` valida que una placa solo contenga letras y números. También permite borrar caracteres y verifica que el campo no esté vacío.

Listing 4. Fragmento de `getValidatedPlate`

```
std::string InputValidator::getValidatedPlate() {
    std::string input;
    char ch;

    while (true) {
        input.clear();
        while ((ch = _getch()) != '\r') {
            if (std::isalnum(ch)) {
                ch = std::toupper(ch);
                input += ch;
                std::cout << ch;
            } else if (ch == '\b' && !input.empty())
                 {
                input.pop_back();
                std::cout << "\b \b";
            }
        }

        if (!input.empty()) return input;
        std::cout << "\nEl campo no puede estar
            vac o.\n";
    }
}
```

### B. *Respaldo y Restauración de Datos*

El sistema incluye la generación de respaldos automáticos con marca de tiempo. Esto permite recuperar datos en caso de fallos.

Listing 5. Fragmento de `createBackup`

```
void ParkingManager::createBackup() {
    auto now = std::chrono::system_clock::now();
    std::ostringstream timestamp;
    timestamp << std::put_time(std::localtime(&
        now_time), "%Y-%m-%d_%H-%M-%S");

    std::string backupDir = dataDir + "backup/";
    std::filesystem::create_directories(backupDir);

    std::string vehiclesBackupFile = backupDir + "
        vehicles_backup_" + timestamp.str() + ".txt"
        ;
    std::filesystem::copy_file(vehiclesFile,
        vehiclesBackupFile);
}
```

## V. Resultados

El sistema fue probado bajo diferentes escenarios:

- Registro y búsqueda de vehículos.
- Generación y restauración de respaldos.
- Visualización del parqueadero en formato circular con diferentes configuraciones.

## VI. Conclusión

El sistema de gestión de parqueadero proporciona una solución robusta y flexible para manejar entradas, salidas y datos de vehículos. Su diseño modular permite extender fácilmente las funcionalidades.