Universidad Tecnológica Nacional

Informática 2

SAPEI

Sistema Alternativo de Pago para el Estacionamiento Institucional

Cortesini Luciano - 402719
Gil Ignacio - 401891
Grasso Gaston - 401892
Noccetti Santino - 405947
Palombo Franco - 401910

19 / 11 / 2024

Motivación

El tránsito en los alrededores de la Universidad Tecnológica Nacional, Facultad Regional Córdoba, es algo que es imposible de resolver, en especial en horas pico, o de cambio de turno. Ya habiendo asistido durante dos años consecutivos a cursar la carrera de ingeniería electrónica, uno empieza a prestarle atención al detalle de los eventos del día a día mientras estamos presentes en la facultad.

Algo que como grupo hemos experimentado todos, es el tránsito que hay en las zonas circundantes a los ingresos para vehículos de la facultad. Es algo que se puede percibir siendo usuario del transporte público, peatón o conductor, ya que frenan todo. Este tránsito es perjudicial para todo aquel que quiera circular por cualquiera de las vías a las que se conecten los ingresos para vehículos.

Definición del problema

Como se mencionó previamente, el problema está en el tránsito generado por los vehículos. Este tránsito (en gran parte) es generado por la fila de autos que hay en el ingreso al estacionamiento de la facultad, lo que obstruye la vía pública, reduciendo el flujo de vehículos y produciendo embotellamientos. La razón por la cual se genera la fila de autos, es por la lentitud del sistema de ingreso al estacionamiento. Este sistema requiere de un operario que ingrese la patente del vehículo de manera manual, el cual procede a generar un ticket de ingreso, y automáticamente se le descuenta el monto correspondiente al usuario.

Solución propuesta

La alternativa que planteamos como solución, consta de un sistema Contactless, en el cual el usuario tiene en su posesión una tarjeta RFID, emitida por la entidad correspondiente, la cual tiene la identificación del mismo usuario. De este modo, no es necesario que el operario tenga que ingresar la patente del vehículo manualmente para poder generar el ticket, ya que el mismo no tiene razón de ser creado, debido a que para el ingreso o el egreso, se utiliza la tarjeta RFID. De tal forma, se puede mantener el conteo de vehículos en el estacionamiento, y además, se evita desperdiciar papel que se emite y se desecha en el mismo día.

Planificación

Para poder llevar a cabo esta idea, empezamos por la organización de la misma y las necesidades o posibles problemas del proyecto. Decidimos usar Github y su sistema de control de versiones creando así los repositorios necesarios para distribuir de forma efectiva las partes de trabajo y así el proyecto pueda avanzar en simultaneo, permitiendo que algunos integrantes se centren en avanzar en partes especificas del proyecto, sin interferir en los avances del resto de los integrantes. Ademas, esta separación, permitió que los problemas se mantengan aislados, por repositorio, evitando así la cruza de errores y fallas, y mejorar el mantenimiento del código.

Partes del Proyecto

La distribución de repositorios, y la distribución de trabajos, se realizo teniendo en cuenta las habilidades de cada uno de los integrantes. Las partes del proyecto y sus requisitos son los siguientes:

1. Hardware: Desarrollar el terminal SAPEI, montarlo en una protoboard y verificar la integridad y robustez del mismo.

2. Firmware: Desarrollar el firmware para el terminal SAPEI, administrar el GPIO por medio de registros, emplear interrupciones y crear un protocolo de comunicación e inicializacion del terminal SAPEI.

3. Core: Desarrollar herramientas de uso general que sirven para todas las partes del proyecto.

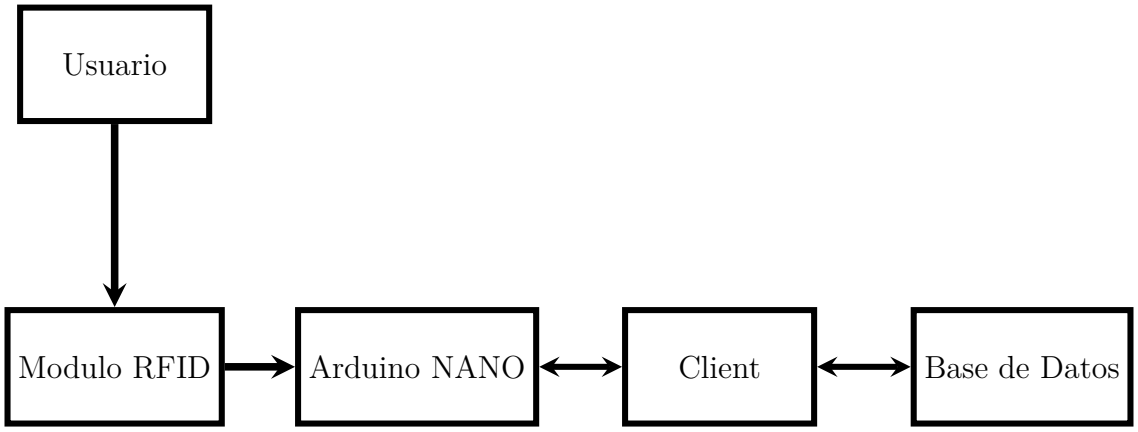
4. Client: Desarrollar una interfaz de usuario que interactúe de manera directa con el terminal y el Core, para administrar clientes, vehículos, carga de saldo y edición de propiedades.

Hardware

Los componentes usados para el hardware fueron:

1. Arduino Nano
2. RC522 (Modulo RFID)
3. transistor NPN BC548
4. 3 Leds
5. Buzzer Activo
6. Protoboard
7. Resistencias y cables

Diagrama Funcional



Programa

El programa tiene una interfaz de usuario simple pero potente. Este programa es el encargado de administrar todos los clientes, comunicarse con la terminal SAPEI para leer las tarjetas y enviar códigos de estado al cliente.

La interfaz esta compuesta por una sección de acciones, una ventana de estado, y una ventana de registro de operaciones.

Ademas de poder agregar clientes, el operario puede editarlos, agregarle o borrarle vehículos e incluso borrar clientes.

Firmware

1. SAPEIFirwmare.ino

```
#include <Arduino.h>
#include "src/rfid/MFRC522v2.h"
#include "src/rfid/MFRC522DriverSPI.h"
#include "src/rfid/MFRC522DriverPinSimple.h"
#include "src/rfid/MFRC522Debug.h"
#include "src/lib/gpio.h"
#include "src/SAPEICore/Error.h"

#define RED_LED PORTdBits.b5
#define YEL_LED PORTdBits.b6
#define GRN_LED PORTdBits.b7
#define BUZZER PORTbBits.b0

#define LEDS_PORT &PORTD
#define BUZZER_PORT &PORTB

#define RED 5
#define YEL 6
#define GRN 7
#define BZZ 0

#define SIMULTANEOUS_CONTROL_NUMBER 2

#define TIMER_INTERRUPT_DEBUG         0
#define _TIMERINTERRUPT_LOGLEVEL_     0
#define USE_TIMER_1 true

#include "TimerInterrupt.h"

MFRC522DriverPinSimple ss_1_pin(10);
MFRC522DriverSPI driver_1{ss_1_pin};
MFRC522 reader{driver_1};

void blink(uint8_t channel, uint8_t *port, uint8_t pin, uint16_t period, uint8_t repetitions = 0);
void flash(uint8_t channel, uint8_t *port, uint8_t pin, uint16_t period, uint16_t onTime, uint8_t
↪  repetitions = 0);
void pinsControlHandler();
void pinControlHandler(uint8_t channel);
void pinControl(uint8_t channel, uint8_t *port, uint8_t pin, uint8_t type, uint8_t repetitions,
↪  uint16_t period, uint16_t onTime = 0);

typedef struct{
  volatile bool alreadyControlling;
  volatile uint8_t *conPort;
  volatile uint8_t conPin;
  volatile uint8_t conType;
  volatile uint8_t conRepetitions;
  volatile uint16_t conPeriod;
  volatile uint16_t conOnTime;
  volatile uint16_t passes;
  volatile uint16_t repetitions;
} pinStatus_s;

pinStatus_s pins[SIMULTANEOUS_CONTROL_NUMBER];

void setup(){
  Serial.begin(115200);
  DDRd = 0xE0;
  DDRb |= _BV(PORTB0);

  Serial.print("INIT ");

  ITimer1.init();
  if (!ITimer1.attachInterruptInterval(10, pinsControlHandler)){
    Serial.println("FAIL");
    while(1);
  }

```

```
67    if(reader.PCD_Init() == 1){
68      Serial.println("OK");
69    } else {
70      Serial.println("FAIL");
71      while(1);
72    }
73
74    bool init = 0;
75    char rxbuff[50] = {0};
76    while (init == 0){
77      if (Serial.available()){
78        Serial.readBytesUntil('\n', rxbuff, 50);
79        if (strcmp(rxbuff, "SAPEI_INIT") == 0)
80          init = 1;
81        else
82          memset(rxbuff, 0, 50);
83      }
84      flash(0, LEDS_PORT, RED, 1000, 50);
85    }
86    Serial.println("CONNECTED");
87  }
88
89  void loop(){
90    static uint32_t readerTimestamp = millis();
91
92    if (reader.PICC_IsNewCardPresent() && reader.PICC_ReadCardSerial() && millis() >= readerTimestamp
      ↪  + 5000) {
93      MFRC522Debug::PrintUID(Serial, reader.uid);
94      Serial.println();
95      reader.PICC_HaltA();
96      reader.PCD_StopCrypto1();
97      readerTimestamp = millis();
98    }
99
100   if (Serial.available()){
101     char rxbuff[5] = {0};
102     Serial.readBytesUntil('\n', rxbuff, 5);
103     uint16_t code = atoi(rxbuff);
104     switch (code){
105         case OK:
106             blink(0, LEDS_PORT, GRN, 500, 10);
107             flash(1, BUZZER_PORT, BZZ, 100, 75, 1);
108             break;
109         case TR_NOT_ENOUGH_FUNDS:
110             blink(0, LEDS_PORT, RED, 500, 10);
111             flash(1, BUZZER_PORT, BZZ, 500, 300, 3);
112             break;
113         case TR_JUST_ENOUGH_FUNDS:
114             blink(0, LEDS_PORT, YEL, 500, 10);
115             flash(1, BUZZER_PORT, BZZ, 100, 75, 3);
116             break;
117         case TR_CLIENT_NOT_FOUND:
118             flash(0, LEDS_PORT, RED, 100, 50, 25);
119             flash(1, BUZZER_PORT, BZZ, 100, 50, 15);
120             break;
121         case TR_ERROR:
122             flash(0, LEDS_PORT, RED, 6000, 6000);
123             flash(1, BUZZER_PORT, BZZ, 2000, 1000, 3);
124         default:
125             break;
126     }
127   }
128 }
129
130 /**
131  * @brief Funcion tipo "wrapper" para parpadear una salida
132  */
133 void blink(uint8_t channel, uint8_t *port, uint8_t pin, uint16_t period, uint8_t repetitions){
134     pinControl(channel, port, pin, 1, repetitions, period);
135 }
136
137 /**
138  * @brief Funcion tipo "wrapper" para parpadear por un periodo determinado una salida
139  */
```

```
140 void flash(uint8_t channel, uint8_t *port, uint8_t pin, uint16_t period, uint16_t onTime, uint8_t
    ↪  repetitions){
141     pinControl(channel, port, pin, 2, repetitions, period, onTime);
142 }
143 
144 /**
145  * @brief Funcion que administra el estado de las salidas en base a su configuracion (blink o flash)
146  */
147 void pinControlHandler(uint8_t channel){ // ISR ONLY!!!
148   if(pins[channel].alreadyControlling == 1){
149     switch (pins[channel].conType){
150       case 1: // blink
151         if (pins[channel].passes < pins[channel].conPeriod/2)
152           *(pins[channel].conPort) |= _BV(pins[channel].conPin);
153         else
154           *(pins[channel].conPort) &= ~_BV(pins[channel].conPin);
155         break;
156       case 2: // flash
157         if (pins[channel].passes < pins[channel].conOnTime)
158           *(pins[channel].conPort) |= _BV(pins[channel].conPin);
159         else
160           *(pins[channel].conPort) &= ~_BV(pins[channel].conPin);
161         break;
162     }
163     if (pins[channel].passes >= pins[channel].conPeriod){
164       if (pins[channel].repetitions >= pins[channel].conRepetitions){
165         pins[channel].alreadyControlling = 0;
166         pins[channel].conPort == NULL;
167         pins[channel].conPin = 0;
168         pins[channel].conType = 0;
169         pins[channel].conPeriod = 0;
170         pins[channel].conOnTime = 0;
171         pins[channel].conRepetitions = 0;
172         pins[channel].passes = 0;
173         pins[channel].repetitions = 0;
174       } else {
175         pins[channel].repetitions++;
176         pins[channel].passes = 0;
177       }
178     } else {
179       pins[channel].passes++;
180     }
181   }
182 }
183 
184 /**
185  * @brief Funcion que define los parametros para el control de las salidas
186  */
187 void pinControl(uint8_t channel, uint8_t *port, uint8_t pin, uint8_t type, uint8_t repetitions,
    ↪  uint16_t period, uint16_t onTime){
188   if(pins[channel].alreadyControlling == 0){
189     pins[channel].conPort = port;
190     pins[channel].conPin = pin;
191     pins[channel].conType = type;
192     pins[channel].conPeriod = period/10;
193     pins[channel].conOnTime = onTime/10;
194     pins[channel].conType = type;
195     pins[channel].conRepetitions = repetitions;
196     pins[channel].repetitions = 0;
197     pins[channel].passes = 0;
198     pins[channel].alreadyControlling = 1;
199   }
200 }
201 
202 /**
203  * @brief Funcion para manipular todos los pines definidos al "mismo tiempo"
204  */
205 void pinsControlHandler(){
206   for (uint8_t i = 0; i < SIMULTANEOUS_CONTROL_NUMBER; i++)
207     pinControlHandler(i);
208 }
```

2. gpio.h:

```
#ifndef CONFIG_H
#define CONFIG_H
#include <avr/io.h>

/**
 * @brief Estructura de bits de un byte.
 */
typedef struct {
  volatile unsigned b0 : 1;
  volatile unsigned b1 : 1;
  volatile unsigned b2 : 1;
  volatile unsigned b3 : 1;
  volatile unsigned b4 : 1;
  volatile unsigned b5 : 1;
  volatile unsigned b6 : 1;
  volatile unsigned b7 : 1;
} bitfield_s;

/**
 * @brief Registro de hardware personalizado.
 *
 * Union que permite acceder a los registros de hardware
 * de forma directa, como un registro de 8 bits, o de forma
 * parcial, modificando/leyendo un solo bit a la vez.
 */
typedef union {
  volatile uint8_t *REG;
  bitfield_s *bits;

} bitreg_u;

extern bitreg_u PORTb_bitreg; /// Registro personalizado para PORTB.
#define PORTb *(PORTb_bitreg.REG)
#define PORTbBits (*(PORTb_bitreg.bits))
extern bitreg_u PINb_bitreg; /// Registro personalizado para PINB.
#define PINb *(PINb_bitreg.REG)
#define PINbBits (*(PINb_bitreg.bits))
extern bitreg_u DDRb_bitreg; /// Registro personalizado para DDRB.
#define DDRb *(DDRb_bitreg.REG)
#define DDRbBits (*(DDRb_bitreg.bits))

extern bitreg_u PORTc_bitreg; /// Registro personalizado para PORTC.
#define PORTc *(PORTc_bitreg.REG)
#define PORTcBits (*(PORTc_bitreg.bits))
extern bitreg_u PINc_bitreg; /// Registro personalizado para PINC.
#define PINc *(PINc_bitreg.REG)
#define PINcBits (*(PINc_bitreg.bits))
extern bitreg_u DDRc_bitreg; /// Registro personalizado para DDRC.
#define DDRc *(DDRc_bitreg.REG)
#define DDRcBits (*(DDRc_bitreg.bits))

extern bitreg_u PORTd_bitreg; /// Registro personalizado para PORTD.
#define PORTd *(PORTd_bitreg.REG)
#define PORTdBits (*(PORTd_bitreg.bits))
extern bitreg_u PINd_bitreg; /// Registro personalizado para PINC.
#define PINd *(PINd_bitreg.REG)
#define PINdBits (*(PINd_bitreg.bits))
extern bitreg_u DDRd_bitreg; /// Registro personalizado para DDRD.
#define DDRd *(DDRd_bitreg.REG)
#define DDRdBits (*(DDRd_bitreg.bits))

#endif
```

3. gpio.cpp:

```
#include "gpio.h"

bitreg_u PORTb_bitreg = {&PORTB}; /// Registro personalizado para PORTB.
bitreg_u PINb_bitreg = {&PINB}; /// Registro personalizado para PINB.
```

```
5   bitreg_u DDRb_bitreg = {&DDRB}; /// Registro personalizado para DDRB.
6
7   bitreg_u PORTc_bitreg = {&PORTC}; /// Registro personalizado para PORTC.
8   bitreg_u PINc_bitreg = {&PINC}; /// Registro personalizado para PINC.
9   bitreg_u DDRc_bitreg = {&DDRC}; /// Registro personalizado para DDRC.
10
11  bitreg_u PORTd_bitreg = {&PORTD}; /// Registro personalizado para PORTD.
12  bitreg_u PINd_bitreg = {&PIND}; /// Registro personalizado para PINC.
13  bitreg_u DDRd_bitreg = {&DDRD}; /// Registro personalizado para DDRD.
```

Core

1. Client.h

```cpp
#ifndef CLIENT_H
#define CLIENT_H

#include "Person.h"
#include "Vehicle.h"
#include <vector>

#define NULLCLIENT Client()

/**
 * @brief Clase para administrar los datos de un cliente
 */
class Client : public Person {

friend std::ostream& operator<<(std::ostream& os, const Client& Client);

private:
    unsigned long long id; /**<ID relacionado a la tarjeta RFID unica de cada cliente*/
    std::vector<Vehicle> vehicleVector; /**<Vector que almacena objetos con los datos del vehiculo*/
    double balance; /**<Lleva el saldo del cliente, en pesos argentinos*/

public:
    Client( unsigned long long id = 0, const std::string& name ="", unsigned int age = 0, unsigned int dni = 0,
            const std::string& address ="", const std::string& email = "", const std::string& phone = "",
            const std::string& license = "", const std::string& type = "", const std::string& color = "",
            const std::string& brand = "", const std::string& model = "");

    unsigned long long getId() const;
    double getBalance() const;
    std::vector<Vehicle> getVehicles() const;

    void setId(unsigned long long id);
    void addVehicle(Vehicle vehicle);
    void removeVehicleByPos(int vehiclePos);
    void removeVehicleByLicense(std::string license);
    void setBalance(double balance);
    inline bool isNull() {return id ? false : true;}
};

#endif
```

2. Client.cpp

```cpp
#include "Client.h"
#include "Vehicle.h"

Client::Client(unsigned long long id, const std::string& name, unsigned int age,
               unsigned int dni,
               const std::string& address,
               const std::string& email, const std::string& phone,
               const std::string& license, const std::string& type,
               const std::string& color, const std::string& brand,
               const std::string& model)
    : Person(name, age, dni, address, email, phone), id(id){
    balance = 0;
    if(license != ""){
      Vehicle firstVehicle(license, type, color, brand, model, id);
      vehicleVector.push_back(firstVehicle);
    }
}

unsigned long long Client::getId() const {
    return id;
```

```cpp
}

double Client::getBalance() const {
    return balance;
}

std::vector<Vehicle> Client::getVehicles() const{
  return vehicleVector;
}

void Client::setId(unsigned long long id) {
    this->id= id;
    for(int i = 0; i <= vehicleVector.size(); i++)
      vehicleVector[i].updateClientId(id);
}

void Client::addVehicle(Vehicle vehicle) {
    vehicle.setClientId(this->id);
    vehicleVector.push_back(vehicle);
}

void Client::removeVehicleByPos(int vehiclePos){
  if(vehiclePos > 0 && vehiclePos <= vehicleVector.size())
    vehicleVector.erase(vehicleVector.begin() + vehiclePos-1);
}

void Client::removeVehicleByLicense(std::string license){
  for(int i = 0; i < vehicleVector.size(); i++)
    if(vehicleVector[i].getLicensePlate() == license)
      vehicleVector.erase(vehicleVector.begin() + i);
}

void Client::setBalance(double balance){
    this->balance = balance;
}

std::ostream& operator<<(std::ostream& os, const Client& client) {
    os << static_cast<const Person&>(client)
    << "ID: " << client.id << std::endl
    << "Saldo: " << client.balance << std::endl;
    for(int i = 0; i < client.vehicleVector.size(); i++){
      os << "Vehiculo N: " << i+1 << std::endl
      << client.vehicleVector.at(i) << std::endl;
    }

    return os;
}
```

3. Vehicle.h

```h
#ifndef VEHICLE_H
#define VEHICLE_H

#include <iostream>
#include <string>
#include "Error.h"

#define NULLVEHICLE Vehicle()

class Vehicle {

friend std::ostream& operator<<(std::ostream& os, const Vehicle& vehicle);
friend class Client;

private:
    std::string licensePlate;
    std::string type;
    std::string color;
    std::string brand;
    std::string model;
    unsigned long long clientId;
    void updateClientId(unsigned long long id);

public:
    Vehicle(const std::string& license = "", const std::string& type = "", const std::string& color = "", const std::string& brand = "",
            const std::string& model = "", unsigned long long clientId = 0);

    std::string getLicensePlate() const;
    std::string getType() const;
    std::string getColor() const;
    std::string getBrand() const;
    std::string getModel() const;
    unsigned long long getClientId() const;

    void setLicensePlate(const std::string& license);
    void setType(const std::string& type);
    void setColor(const std::string& color);
    void setBrand(const std::string& brand);
    void setModel(const std::string& model);
    void setClientId(unsigned long long id);
    inline bool isNull() const { return licensePlate.empty(); }

};

#endif
```

4. Vehicle.cpp

```cpp
#include "Vehicle.h"

using namespace std;

Vehicle::Vehicle(const std::string& license, const std::string& type, const std::string& color, const std::string& brand, const std::string& model, unsigned long long clientId)
    : licensePlate(license), type(type), color(color), brand(brand), model(model), clientId(clientId){
}

std::string Vehicle::getLicensePlate() const {
  return licensePlate;
}
std::string Vehicle::getType() const {
  return type;
}

std::string Vehicle::getColor() const {
  return color;
}

```

```cpp
std::string Vehicle::getBrand() const {
  return brand;
}

std::string Vehicle::getModel() const {
  return model;
}

unsigned long long Vehicle::getClientId() const {
  return clientId;
}

void Vehicle::setLicensePlate(const std::string& license) {
  this->licensePlate = license;
}

void Vehicle::setType(const std::string& type) {
  this->type = type;
}

void Vehicle::setColor(const std::string& color) {
  this->color = color;
}

void Vehicle::setBrand(const std::string& brand) {
  this->brand = brand;
}

void Vehicle::setModel(const std::string& model) {
  this->model = model;
}

void Vehicle::setClientId(unsigned long long clientId) {
  if (this->clientId == 0)
    this->clientId = clientId;
  else
    cerr << getErrMsg(VH_CLIENT_ASSOCIATE) << endl;
}

void Vehicle::updateClientId(unsigned long long clientId) {
    this->clientId = clientId;
}

std::ostream& operator<<(std::ostream& os, const Vehicle& vehicle) {
    os << "Patente: " << vehicle.licensePlate << std::endl
       << "Tipo: " << vehicle.type << std::endl
       << "Color: " << vehicle.color << std::endl
       << "Marca: " << vehicle.brand << std::endl
       << "Modelo: " << vehicle.model;
    return os;
}
```

5. Person.h

```h
#ifndef PERSON_H
#define PERSON_H

#include <iostream>
#include <string>

class Person {

friend std::ostream& operator<<(std::ostream& os, const Person& person);

private:
    std::string name;
    unsigned int age;
    unsigned int dni;
    std::string address;
    std::string email;
    std::string phone;

public:
    Person(const std::string& name = "", unsigned int age = 0, unsigned int dni = 0, const
    ↪  std::string& address = "",
            const std::string& email = "", const std::string& phone = "");

    std::string getName() const;
    unsigned int getAge() const;
    unsigned int getDni() const;
    std::string getAddress() const;
    std::string getEmail() const;
    std::string getPhone() const;

    void setName(const std::string&);
    void setAge(const unsigned int);
    void setDni(const unsigned int);
    void setAddress(const std::string&);
    void setEmail(const std::string&);
    void setPhone(const std::string&);
};

#endif // PERSON_H
```

6. Person.cpp

```cpp
#include "Person.h"


using namespace std;

Person::Person(const string& name, unsigned int age, unsigned int dni, const string& address, const
↪  string& email, const std::string& phone)
    : name(name), age(age), dni(dni), address(address), email(email), phone(phone){}

string Person::getName() const {
    return name;
}

unsigned int Person::getAge() const {
    return age;
}

unsigned int Person::getDni() const{
    return dni;
}

string Person::getAddress() const {
    return address;
}

string Person::getEmail() const {
    return email;
}
```

```cpp

string Person::getPhone() const {
    return phone;
}

void Person::setName(const std::string& name){
    this->name = name;
}

void Person::setAge(const unsigned int age){
    this->age = age;
}

void Person::setDni(const unsigned int dni){
    this->dni = dni;
}

void Person::setAddress(const std::string& address){
    this->address = address;
}

void Person::setEmail(const std::string& email){
    this->email = email;
}

void Person::setPhone(const std::string& phone){
    this->phone = phone;
}

std::ostream& operator<<(std::ostream& os, const Person& person) {
    os << "Nombre: " << person.name << std::endl;
    os << "Edad: " << person.age << std::endl;
    os << "DNI: " << person.dni << std::endl;
    os << "Dirección: " << person.address << std::endl;
    os << "email: " << person.email << std::endl;
    os << "telefono: " << person.phone << std::endl;
    return os;
}
```

7. Error.h

```h
#ifndef ERRORS_H
#define ERRORS_H

#ifndef Arduino_h
#include <string>
#else
#include <Arduino.h>
#endif

enum errorCode{
    OK,
    //Database errors
    DB_LI = 1,
    DB_CLIENT_NOT_FOUND = DB_LI,
    DB_VEHICLE_NOT_FOUND,
    DB_DUPLICATE_CLIENT,
    DB_DUPLICATE_VEHICLE,
    DB_LS = DB_DUPLICATE_VEHICLE,
    //Client Errors
    CL_LI = 100,
    CL_SERVER_TIMEOUT = CL_LI,
    CL_LS = CL_SERVER_TIMEOUT,
    //Vehicle Errors
    VH_LI = 200,
    VH_CLIENT_ASSOCIATE = VH_LI,
    VH_LS = VH_CLIENT_ASSOCIATE,
    //Terminal errors (for)
    TR_NOT_ENOUGH_FUNDS = 300,
    TR_JUST_ENOUGH_FUNDS,
    TR_CLIENT_NOT_FOUND,
    TR_ERROR
};

// enumRange es el rango en donde hay enums definidos, donde se consideran
// pares de numeros para el rango, siendo el primero el limite inferior y
// el segundo el limite superior (NO EXCLUYENTES). "errLib" debe contener
// al menos una cadena vacia para asignar un espacio en el arreglo. En caso
// de no tener cadenas, el rango de aquellas debera ser excluido de enumRange
#ifndef Arduino_h
extern const int enumRange[];
extern const std::string errLib[][99];

bool checkEnumRange(int error);
std::string getErrMsg(int error);
#endif

#endif // ERRORS_H


```

8. Error.cpp

```cpp
#include "Error.h"

const std::string errLib[][99] = {
    {
        "OK",
        //DATABASE_ERROR [001 - 099]
        "Cliente no encontrado",
        "Vehiculo no encontrado",
        "Cliente existente",
        "Vehiculo existente",
    },
    {
        //CLIENT_ERROR  [100 - 199]
        "No se pudo conectar con el servidor"
    },
    {
        //VEHICLE_ERROR  [200 - 299]
        "Este vehiculo ya tiene un cliente asociado"
```

```
    },
    {
        //TERMINAL_ERROR [300 - 399]
        // No tiene sentido que haya cadenas. La terminal no tiene pantalla.
        // Debe existir la cadena "" para que se ocupe el espacio y se pueda
        // indexar el arreglo de strings.
        ""
    }
};

const int enumRange[] = {DB_LI, DB_LS, CL_LI, CL_LS, VH_LI, VH_LS};


bool checkEnumRange(int error){
    for(int i = 0; i<(sizeof(enumRange)/sizeof(*enumRange)); i += 2) {
        if(enumRange[i] <= error && enumRange[i+1] >= error)
            return true;
    }
    return false;
}

std::string getErrMsg(int error){
    if (checkEnumRange(error) == 0)
        return "NO_ERR_MSG";

    int errType = error / 100;
    int errId = error % 100;
    return errLib[errType][errId];
}
```

9. Database.h

```
#ifndef DATABASE_H
#define DATABASE_H

#include <iostream>
#include <vector>
#include <sqlite3.h>
#include <string>
#include <fstream>
#include "Client.h"
#include "Vehicle.h"
#include "Error.h"

/**
 * @brief Clase para administrar la base de datos
 *
 * Esta clase utiliza la SQLite3 para almacenar los datos de los clientes
 * Permite facilmente interactuar mediante la clase Client
 *
 */
class DataBase {
private:
    sqlite3* db; /**<Puntero para interactuar con la base de datos*/
    char* errMsg = 0; /**<Cadena para guardar los mensajes de error*/
    void createClientTable();
    void addMultipleVehicles(const unsigned long long client_id, const std::vector<Vehicle>&);
    void createVehicleTable();
    void exportTableToCSV(const std::string& tableName, const std::string& outputFile);
    static int callback(void* data, int argc, char** argv, char** azColName);

    template <typename T>
    int checkExistence(const std::string& table, const std::string& by, T value);

    template <typename T>
    Client getClient(const std::string&, T);
public:
    DataBase(const std::string& dbName = "SAPEI.db");
    ~DataBase();

    void addClient(const Client&);
    int updateClient(const Client&);
    void rmClient(const unsigned long long id);

    void showClientById(const unsigned long long);
    void showClientByName(const std::string&);
    void showClients();

    void updateBalance(const unsigned long long id, double balance);
    double getBalance(const unsigned long long id);

    Client getClientById(const unsigned long long id);
    Client getClientByName(std::string name);
    Client getClientByDni(unsigned int dni);

    Vehicle getVehicleByPlate(const std::string& plate);
    std::vector<Vehicle> getVehiclesByClientId(unsigned long long vehicleId);
    std::vector<Vehicle> getVehiclesByClientName(const std::string& vehicleName);

    std::vector<Client> getAllClients();
    std::vector<Vehicle> getAllVehicles();

    void addVehicle(const unsigned long long client_id, const Vehicle&);
    void rmVehicle(const std::string& license);
    void showVehicles();
    void showVehicleByLicense(const std::string& license);
    void showVehiclesByClientId(const unsigned long long client_id);

    void exportClientsToCSV(const std::string& = "clientes.csv");
    void exportVehiclesToCSV(const std::string& = "vehiculos.csv");
};

template <typename T>
int DataBase::checkExistence(const std::string& table, const std::string& by, T value) {
```

```
73      sqlite3_stmt* stmt;
74      int exists = 0;
75      std::string sqlSmtp = "SELECT 1 FROM " + table + " WHERE " + by + " = ";
76
77      if constexpr (std::is_same<T, std::string>::value){
78          sqlSmtp = sqlSmtp +  "'" + value + "'";
79      } else if constexpr (std::is_same<T, const char*>::value || std::is_same<T, char*>::value){
80          sqlSmtp = sqlSmtp +  "'" + std::string(value) + "'";
81      } else if constexpr (std::is_floating_point<T>::value || std::is_integral<T>::value){
82          sqlSmtp = sqlSmtp +  std::to_string(value);
83      } else {
84          std::cerr << "Tipo de dato no soportado." << std::endl;
85          return -1;
86      }
87
88      if (sqlite3_prepare_v2(db, sqlSmtp.c_str(), -1, &stmt, nullptr) != SQLITE_OK) {
89          std::cerr << sqlite3_errmsg(db) << std::endl;
90          return -1;
91      }
92
93      if (sqlite3_step(stmt) == SQLITE_ROW) exists = 1;
94
95      sqlite3_finalize(stmt);
96      return exists;
97  }
98
99  /**
100  * @brief Busca a travez de calquier atributo un cliente en la tabla.
101  *
102  * Devuelve un NULLCLIENT si el cliente no es encontrado.
103  *
104  * @param atribute Atributo para la busqueda (dni, id, etc).
105  * @param value Valor del atributo seleccionado para la busqueda.
106  */
107 template <typename T>
108 Client DataBase::getClient(const std::string& atribute, T value) {
109     if(!checkExistence("client", atribute, value)){
110         std::cerr << getErrMsg(DB_CLIENT_NOT_FOUND) << std::endl;
111         return NULLCLIENT;
112     }
113
114     sqlite3_stmt* stmt;
115     std::string sqlQuery =
116         "SELECT client.id, client.name, client.age, client.dni, client.address, client.email,
        ↪  client.phone, "
117         "vehicle.license, vehicle.type, vehicle.color, vehicle.brand, vehicle.model, "
118         "client.balance "
119         "FROM client "
120         "LEFT JOIN vehicle ON client.id = vehicle.client_id WHERE client." + atribute + " = ";
121
122     if constexpr (std::is_same<T, std::string>::value){
123         sqlQuery = sqlQuery +  "'" + value + "'";
124     } else if constexpr (std::is_same<T, const char*>::value || std::is_same<T, char*>::value){
125         sqlQuery = sqlQuery +  "'" + std::string(value) + "'";
126     } else if constexpr (std::is_floating_point<T>::value || std::is_integral<T>::value){
127         sqlQuery = sqlQuery +  std::to_string(value);
128     } else {
129         std::cerr << "Tipo de dato no soportado." << std::endl;
130         return 0;
131     }
132
133     if (sqlite3_prepare_v2(db, sqlQuery.c_str(), -1, &stmt, nullptr) != SQLITE_OK) {
134         std::cerr << sqlite3_errmsg(db);
135     }
136
137     Client client;
138     bool clientInitialized = false;
139
140     while (sqlite3_step(stmt) == SQLITE_ROW) {
141         if (!clientInitialized) {
142             unsigned long long clientId = sqlite3_column_int64(stmt, 0);
143             std::string clientName(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1)));
144             unsigned int age = sqlite3_column_int(stmt, 2);
145             unsigned int dni = sqlite3_column_int(stmt, 3);
```

```
146             std::string address(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 4)));
147             std::string email(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 5)));
148             std::string phone(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 6)));
149             double balance = sqlite3_column_double(stmt, 12);
150 
151             client = Client(clientId, clientName, age, dni, address, email, phone);
152             client.setBalance(balance);
153             clientInitialized = true;
154         }
155 
156         const unsigned char* licenseText = sqlite3_column_text(stmt, 7);
157         if (licenseText != nullptr) {
158             std::string license(reinterpret_cast<const char*>(licenseText));
159             std::string type(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 8)));
160             std::string color(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 9)));
161             std::string brand(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 10)));
162             std::string model(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 11)));
163 
164             Vehicle vehicle(license, type, color, brand, model);
165             client.addVehicle(vehicle);
166         }
167     }
168 
169     sqlite3_finalize(stmt);
170 
171     return client;
172 }
173 
174 #endif // DATABASE_H
```

10. Database.cpp

```
1   #include "DataBase.h"
2   
3   using namespace std;
4   
5   /**
6    * @brief Abre la base de datos
7    *
8    * Abre la base de datos con el nombre proporcionado y crea las tablas de cliente y vehículo.
9    *
10   * @param dbName Nombre del archivo de la base de datos.
11   * @throws runtime_error Si hay un error al abrir la base de datos.
12   */
13  DataBase::DataBase(const string& dbName) {
14      if (sqlite3_open(dbName.c_str(), &db)) {
15          string errorMsg = "Error opening database: ";
16          errorMsg += sqlite3_errmsg(db);
17          throw runtime_error(errorMsg);
18      } else {
19          cout << "Database opened successfully." << endl;
20      }
21      createClientTable();
22      createVehicleTable();
23  }
24  
25  /**
26   * @brief Cierra la conexión a la base de datos.
27   */
28  DataBase::~DataBase() {
29      sqlite3_close(db);
30  }
31  
32  /**
33   * @brief Crea la tabla de clientes en la base de datos.
34   *
35   * La tabla solo se crea si no existe.
36   *
37   * @throws runtime_error Si hay un error al crear la tabla.
38   */
39  void DataBase::createClientTable(){
40      const char* sqlCreateClientTable =
```

```
41          "CREATE TABLE IF NOT EXISTS client ("
42          "id INTEGER PRIMARY KEY,"
43          "balance REAL NOT NULL,"
44          "name TEXT NOT NULL,"
45          "age INTEGER NOT NULL,"
46          "dni INTEGER NOT NULL,"
47          "address TEXT,"
48          "email TEXT,"
49          "phone TEXT"
50          "); ";
51
52      if (sqlite3_exec(db, sqlCreateClientTable, nullptr, nullptr, &errMsg) != SQLITE_OK) {
53          throw runtime_error(errMsg);
54          sqlite3_free(errMsg);
55      }
56  }
57
58  /**
59   * @brief Crea la tabla de vehículos en la base de datos.
60   *
61   * La tabla solo se crea si no existe.
62   *
63   * @throws runtime_error Si hay un error al crear la tabla.
64   */
65  void DataBase::createVehicleTable(){
66      const char* sqlCreateVehicleTable =
67          "CREATE TABLE IF NOT EXISTS vehicle ("
68          "license TEXT PRIMARY KEY,"
69          "client_id INTEGER NOT NULL,"
70          "type TEXT,"
71          "color TEXT,"
72          "brand TEXT,"
73          "model TEXT,"
74          "FOREIGN KEY(client_id) REFERENCES client(id)"
75          "); ";
76
77      if (sqlite3_exec(db, sqlCreateVehicleTable, nullptr, nullptr, &errMsg) != SQLITE_OK) {
78          throw runtime_error(errMsg);
79          sqlite3_free(errMsg);
80      }
81  }
82
83  /**
84   * @brief Agrega un vehículo a la base de datos.
85   *
86   * Si el vehículo ya existe, no se insertará.
87   *
88   * @param client_id ID del cliente asociado al vehículo.
89   * @param vh Objeto de tipo Vehicle que contiene la información del vehículo a agregar.
90   * @throws runtime_error Si hay un error al insertar el vehículo.
91   */
92  void DataBase::addVehicle(const unsigned long long client_id, const Vehicle& vh) {
93      string sqlInsert = "INSERT OR IGNORE INTO vehicle (license, client_id, type, color, brand,
      ↪  model) VALUES ('"
94        + vh.getLicensePlate() + "', "
95        + to_string(client_id) + ", '"
96        + vh.getType() + "', '"
97        + vh.getColor() + "', '"
98        + vh.getBrand() + "', '"
99        + vh.getModel() + "');";
100
101     if (sqlite3_exec(db, sqlInsert.c_str(), nullptr, nullptr, &errMsg) != SQLITE_OK) {
102         throw runtime_error(errMsg);
103         sqlite3_free(errMsg);
104     }
105 }
106
107 void DataBase::addMultipleVehicles(const unsigned long long client_id, const vector<Vehicle>& vh){
108     for(int i = 0; i < vh.size(); i++){
109         addVehicle(client_id, vh.at(i));
110     }
111 }
112
113 /**
```

```
114     * @brief Elimina un vehículo de la base de datos por su licencia.
115     *
116     * @param license Licencia del vehículo a eliminar.
117     * @throws runtime_error Si hay un error al eliminar el vehículo.
118     */
119    void DataBase::rmVehicle(const string& license) {
120        string sqlRmVehicle = "DELETE FROM vehicle WHERE license = '" + license + "';";
121        if (sqlite3_exec(db, sqlRmVehicle.c_str(), nullptr, nullptr, &errMsg) != SQLITE_OK) {
122            throw runtime_error(errMsg);
123            sqlite3_free(errMsg);
124        }
125    }
126
127    /**
128     * @brief Muestra todos los vehículos de la base de datos.
129     *
130     * @throws runtime_error Si hay un error al ejecutar la consulta.
131     */
132    void DataBase::showVehicles() {
133        if (sqlite3_exec(db, "SELECT * FROM vehicle;", callback, nullptr, &errMsg) != SQLITE_OK) {
134            throw runtime_error(errMsg);
135            sqlite3_free(errMsg);
136        }
137    }
138
139    /**
140     * @brief Muestra un vehículo específico por su licencia.
141     *
142     * @param license Licencia del vehículo a buscar.
143     * @throws runtime_error Si hay un error al ejecutar la consulta.
144     */
145    void DataBase::showVehicleByLicense(const string& license) {
146        string sql = "SELECT * FROM vehicle WHERE license = '" + license + "';";
147
148        if (sqlite3_exec(db, sql.c_str(), callback, nullptr, &errMsg) != SQLITE_OK) {
149            throw runtime_error(errMsg);
150            sqlite3_free(errMsg);
151        }
152    }
153
154    /**
155     * @brief Muestra todos los vehículos de un cliente específico.
156     *
157     * @param client_id ID del cliente cuyos vehículos se desean mostrar.
158     * @throws runtime_error Si hay un error al ejecutar la consulta.
159     */
160    void DataBase::showVehiclesByClientId(const unsigned long long client_id) {
161        string sql = "SELECT * FROM vehicle WHERE client_id = " + to_string(client_id) + ";";
162
163        if (sqlite3_exec(db, sql.c_str(), callback, nullptr, &errMsg) != SQLITE_OK) {
164            throw runtime_error(errMsg);
165            sqlite3_free(errMsg);
166        }
167    }
168
169    /**
170     * @brief Agrega un cliente a la base de datos.
171     *
172     * También agrega el vehículo asociado al cliente.
173     *
174     * @param cl Objeto de tipo Client que contiene la información del cliente a agregar.
175     * @throws runtime_error Si hay un error al insertar el cliente.
176     */
177    void DataBase::addClient(const Client& cl) {
178        string sqlInsert = "INSERT OR IGNORE INTO client (id, balance, name, age, dni, address, email,
        ↪  phone) VALUES ("
179            + to_string(cl.getId()) + ", '"
180            + to_string(cl.getBalance()) + "', '"
181            + cl.getName() + "', "
182            + to_string(cl.getAge()) + ", "
183            + to_string(cl.getDni()) + ", '"
184            + cl.getAddress() + "', '"
185            + cl.getEmail() + "', '"
186            + cl.getPhone() + "');";
```

```
187
188     if (sqlite3_exec(db, sqlInsert.c_str(), nullptr, nullptr, &errMsg) != SQLITE_OK) {
189         throw runtime_error(errMsg);
190         sqlite3_free(errMsg);
191     }
192
193     addMultipleVehicles(cl.getId(), cl.getVehicles());
194 }
195
196 int DataBase::updateClient(const Client& cl) {
197     if(!checkExistence("client","id",cl.getId())) return DB_CLIENT_NOT_FOUND;
198
199     rmClient(cl.getId());
200     addClient(cl);
201     return 0;
202 }
203
204 /**
205  * @brief Elimina un cliente de la base de datos por su ID.
206  *
207  * @param id ID del cliente a eliminar.
208  * @throws runtime_error Si hay un error al eliminar el cliente.
209  */
210 void DataBase::rmClient(const unsigned long long id) {
211     string sqlRmClient = "DELETE FROM client WHERE id = " + to_string(id) + ";";
212     if (sqlite3_exec(db, sqlRmClient.c_str(), 0, 0, &errMsg) != SQLITE_OK) {
213         throw runtime_error(errMsg);
214         sqlite3_free(errMsg);
215     }
216     string sqlRmVehicles = "DELETE FROM vehicle WHERE client_id = " + to_string(id) + ";";
217     if (sqlite3_exec(db, sqlRmVehicles.c_str(), 0, 0, &errMsg) != SQLITE_OK) {
218         throw runtime_error(errMsg);
219         sqlite3_free(errMsg);
220     }
221 }
222
223 /**
224  * @brief Muestra por consola todos los clientes de la base de datos.
225  *
226  * @throws runtime_error Si hay un error al ejecutar la consulta.
227  */
228 void DataBase::showClients() {
229     if (sqlite3_exec(db, "SELECT * FROM client;", callback, nullptr, &errMsg) != SQLITE_OK){
230         throw runtime_error(errMsg);
231         sqlite3_free(errMsg);
232     }
233 }
234
235 /**
236  * @brief Muestra por consola un cliente específico por su ID.
237  *
238  * Llama a un callback para procesar cada fila de resultados.
239  *
240  * @param id ID del cliente a buscar.
241  * @throws runtime_error Si hay un error al ejecutar la consulta.
242  */
243 void DataBase::showClientById(const unsigned long long id) {
244     string sql = "SELECT * FROM client WHERE id = " + to_string(id) + ";";
245
246     if (sqlite3_exec(db, sql.c_str(), callback, nullptr, &errMsg) != SQLITE_OK) {
247         throw runtime_error(errMsg);
248         sqlite3_free(errMsg);
249     }
250 }
251
252 /**
253  * @brief Muestra por consola un cliente específico por su nombre.
254  *
255  * Llama a un callback para procesar cada fila de resultados.
256  *
257  * @param name Nombre del cliente a buscar.
258  * @throws runtime_error Si hay un error al ejecutar la consulta.
259  */
260 void DataBase::showClientByName(const string& name) {
```

```
261     string sql = "SELECT * FROM client WHERE name = '" + name + "';";
262
263     if (sqlite3_exec(db, sql.c_str(), callback, nullptr, &errMsg) != SQLITE_OK) {
264         throw runtime_error(errMsg);
265         sqlite3_free(errMsg);
266     }
267 }
268
269
270 /**
271  * @brief Callback que se llama para cada fila de resultados de una consulta.
272  *
273  * @param data Puntero a datos adicionales (no utilizado aquí).
274  * @param argc Número de columnas en la fila.
275  * @param argv Array de punteros a los valores de las columnas.
276  * @param azColName Array de punteros a los nombres de las columnas.
277  * @return 0 para continuar.
278  */
279 int DataBase::callback(void* data, int argc, char** argv, char** azColName) {
280     for (int i = 0; i < argc; i++) {
281         cout << azColName[i] << ": " << (argv[i] ? argv[i] : "NULL") << " | ";
282     }
283     cout << endl;
284     return 0;
285 }
286
287 /**
288  * @brief Obtiene un cliente de la base de datos por su ID.
289  *
290  * @param id ID del cliente a buscar.
291  * @return Objeto Client con la información del cliente.
292  * @throws runtime_error Si hay un error al ejecutar la consulta.
293  */
294
295 void DataBase::updateBalance(const unsigned long long id, double balance){
296     sqlite3_stmt* stmt;
297     const char* sqlQuery = "UPDATE client SET balance = ? WHERE id = ?";
298
299     if (sqlite3_prepare_v2(db, sqlQuery, -1, &stmt, nullptr) != SQLITE_OK) {
300         throw runtime_error(sqlite3_errmsg(db));
301     }
302
303     sqlite3_bind_double(stmt, 1, balance);
304     sqlite3_bind_int64(stmt, 2, id);
305
306     sqlite3_step(stmt);
307     sqlite3_finalize(stmt);
308 }
309
310 double DataBase::getBalance(const unsigned long long id){
311     sqlite3_stmt* stmt;
312     double balance;
313     const char* sqlQuery = "SELECT client.balance FROM client WHERE id = ?";
314
315     if (sqlite3_prepare_v2(db, sqlQuery, -1, &stmt, nullptr) != SQLITE_OK) {
316         throw runtime_error(sqlite3_errmsg(db));
317     }
318
319     sqlite3_bind_int64(stmt, 1, id);
320
321     if (sqlite3_step(stmt) == SQLITE_ROW)
322         balance = sqlite3_column_double(stmt, 0);
323
324     sqlite3_finalize(stmt);
325
326     return balance;
327 }
328
329 /**
330  * @brief Busca un cliente por su id.
331  */
332 Client DataBase::getClientById(const unsigned long long id){
333     return getClient("id",id);
334 }
```

```cpp

/**
 * @brief Busca un cliente por su nombre.
 */
Client DataBase::getClientByName(std::string name){
    return getClient("name",name);
}

/**
 * @brief Busca un cliente por su dni.
 */

Client DataBase::getClientByDni(unsigned int dni){
    return getClient("dni",dni);
}

/**
 * @brief Exporta una tabla a un archivo CSV.
 *
 * @param tableName El nombre de la tabla a exportar (ej: "client", "vehicle").
 * @param outputFile El nombre del archivo CSV al que se exportarán los datos.
 * @throws runtime_error Si ocurre un error en la consulta o al escribir el archivo.
 */
void DataBase::exportTableToCSV(const string& tableName, const string& outputFile) {
    ofstream file(outputFile);
    if (!file.is_open()) {
        throw runtime_error("Error al abrir el archivo para escribir.");
    }

    string sqlQuery = "SELECT * FROM " + tableName + ";";
    sqlite3_stmt* stmt;

    if (sqlite3_prepare_v2(db, sqlQuery.c_str(), -1, &stmt, nullptr) != SQLITE_OK) {
        throw runtime_error(sqlite3_errmsg(db));
    }

    // Escribir los nombres de las columnas en la primera línea del CSV
    int numCols = sqlite3_column_count(stmt);
    for (int i = 0; i < numCols; i++) {
        file << sqlite3_column_name(stmt, i);
        if (i < numCols - 1) file << ",";
    }
    file << "\n";  // Nueva línea después de las cabeceras

    // Escribir los datos de las filas
    while (sqlite3_step(stmt) == SQLITE_ROW) {
        for (int i = 0; i < numCols; i++) {
            const char* colValue = reinterpret_cast<const char*>(sqlite3_column_text(stmt, i));
            file << (colValue ? colValue : "NULL");  // Manejar valores NULL
            if (i < numCols - 1) file << ",";
        }
        file << "\n";  // Nueva línea después de cada fila
    }

    sqlite3_finalize(stmt);
    file.close();
}

/**
 * @brief Exporta la tabla de clientes a un archivo CSV.
 */
void DataBase::exportClientsToCSV(const string&){
    exportTableToCSV("client", "clients.csv");
}

/**
 * @brief Exporta la tabla de vehículos a un archivo CSV.
 */
void DataBase::exportVehiclesToCSV(const string&){
    exportTableToCSV("vehicle", "vehicles.csv");
}

/**
 * @brief Obtiene los datos de todos los clientes.
```

```
409  *
410  * Obtiene solo los datos del cliente, mas no sus vehiculos, sirve para listarlos.
411  */
412 vector<Client> DataBase::getAllClients() {
413     std::vector<Client> clients;
414     sqlite3_stmt* stmt;
415     const char* sqlQuery = "SELECT id, name, age, dni, address, email, phone, balance FROM client;";
416 
417     if (sqlite3_prepare_v2(db, sqlQuery, -1, &stmt, nullptr) != SQLITE_OK) {
418         throw std::runtime_error(sqlite3_errmsg(db));
419     }
420 
421     while (sqlite3_step(stmt) == SQLITE_ROW) {
422         unsigned long long id = sqlite3_column_int64(stmt, 0);
423         std::string name(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1)));
424         int age = sqlite3_column_int(stmt, 2);
425         unsigned int dni = sqlite3_column_int(stmt, 3);
426         std::string address(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 4)));
427         std::string email(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 5)));
428         std::string phone(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 6)));
429         double balance = sqlite3_column_double(stmt, 7);
430 
431         Client client(id, name, age, dni, address, email, phone);
432         client.setBalance(balance);
433         clients.push_back(client);
434     }
435 
436     sqlite3_finalize(stmt);
437     return clients;
438 }
439 
440 /**
441  * @brief obtiene un vehiculo por su patente
442  *
443  * Devuelve NULLVEHICLE si no existe un vehiculo con esa patente
444  *
445  */
446 Vehicle DataBase::getVehicleByPlate(const std::string& plate){
447     if(!checkExistence("vehicle","license",plate)){
448         cerr << getErrMsg(DB_VEHICLE_NOT_FOUND) << endl;
449         return NULLVEHICLE;
450     }
451 
452     sqlite3_stmt* stmt;
453     string sqlQuery = "SELECT client_id, license, type, color, brand, model "
454         "FROM vehicle WHERE license = '"+ plate +"' ;";
455 
456     Vehicle vehicle;
457 
458     if (sqlite3_prepare_v2(db, sqlQuery.c_str(), -1, &stmt, nullptr) != SQLITE_OK) {
459         throw std::runtime_error(sqlite3_errmsg(db));
460     }
461 
462     while (sqlite3_step(stmt) == SQLITE_ROW) {
463         unsigned long long id = sqlite3_column_int64(stmt, 0);
464         std::string license(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1)));
465         std::string type(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 2)));
466         std::string color(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 3)));
467         std::string brand(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 4)));
468         std::string model(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 5)));
469 
470         vehicle = Vehicle(license, type, color, brand, model, id);
471     }
472 
473     sqlite3_finalize(stmt);
474 
475     return vehicle;
476 }
477 
478 /**
479  * @brief Obtiene todos los vehiculos de un cliente por su id
480  */
481 vector<Vehicle> DataBase::getVehiclesByClientId(unsigned long long clientId) {
482     Client client = getClientById(clientId);
```

```
483     return client.getVehicles();
484 }
485 
486 /**
487  * @brief Obtiene todos los vehiculos de un cliente por su nombre
488  */
489 vector<Vehicle> DataBase::getVehiclesByClientName(const std::string& clientName) {
490     Client client = getClientByName(clientName);
491     return client.getVehicles();
492 }
493 
494 vector<Vehicle> DataBase::getAllVehicles() {
495     sqlite3_stmt* stmt;
496     const char* sqlQuery = "SELECT client_id, license, type, color, brand, model FROM vehicle;";
497 
498     std::vector<Vehicle> vehicles;
499 
500     if (sqlite3_prepare_v2(db, sqlQuery, -1, &stmt, nullptr) != SQLITE_OK) {
501         throw std::runtime_error(sqlite3_errmsg(db));
502     }
503 
504     while (sqlite3_step(stmt) == SQLITE_ROW) {
505         unsigned long long id = sqlite3_column_int64(stmt, 0);
506         std::string license(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1)));
507         std::string type(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 2)));
508         std::string color(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 3)));
509         std::string brand(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 4)));
510         std::string model(reinterpret_cast<const char*>(sqlite3_column_text(stmt, 5)));
511 
512         Vehicle vehicle(license, type, color, brand, model);
513         vehicles.push_back(vehicle);
514     }
515 
516     sqlite3_finalize(stmt);
517 
518     return vehicles;
519 }
```

Client

1. main.cpp

```cpp
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    MainWindow window;
    window.resize(400, 300);
    window.show();

    return app.exec();
}

```

2. addcarddialog.h

```cpp
/**
 * @file addcarddialog.h
 * @brief Declaración de la clase AddCardDialog.
 *
 * Este archivo contiene la declaración de la clase AddCardDialog,
 * que proporciona una interfaz de diálogo para agregar información de una tarjeta y sus detalles asociados.
 */

#ifndef ADDCARDIALOG_H
#define ADDCARDIALOG_H

#include <QDialog>

namespace Ui {
class AddCardDialog;
}

/**
 * @class AddCardDialog
 * @brief Clase que representa un cuadro de diálogo para agregar información de una tarjeta.
 *
 * La clase AddCardDialog hereda de QDialog y proporciona un diálogo en la interfaz gráfica
 * para ingresar detalles como nombre, apellido, DNI, email, teléfono, dirección, edad,
 * información de licencia de vehículo y datos del vehículo (marca, modelo, color y tipo).
 */
class AddCardDialog : public QDialog {
    Q_OBJECT

public:
    /**
     * @brief Constructor de la clase AddCardDialog.
     * @param parent Puntero al widget padre. Si no se pasa, el valor predeterminado es nullptr.
     */
    explicit AddCardDialog(QWidget *parent = nullptr);

    /**
     * @brief Destructor de la clase AddCardDialog.
     */
    ~AddCardDialog();

    /**
     * @brief Obtiene el nombre ingresado en el diálogo.
     * @return Cadena de texto con el nombre.
     */
    QString getName() const;

    /**
     * @brief Obtiene el apellido ingresado en el diálogo.
     * @return Cadena de texto con el apellido.
     */
    QString getLastName() const;

    /**
     * @brief Obtiene el DNI ingresado en el diálogo.
     * @return Cadena de texto con el DNI.
     */
    QString getDNI() const;

    /**
     * @brief Obtiene el email ingresado en el diálogo.
     * @return Cadena de texto con el email.
     */
    QString getEmail() const;

    /**
     * @brief Obtiene el teléfono ingresado en el diálogo.
     * @return Cadena de texto con el número de teléfono.
     */
    QString getPhone() const;

```

```
71      /**
72       * @brief Obtiene la dirección ingresada en el diálogo.
73       * @return Cadena de texto con la dirección.
74       */
75      QString getAddress() const;
76
77      /**
78       * @brief Obtiene la edad ingresada en el diálogo.
79       * @return Número entero con la edad.
80       */
81      int getAge() const;
82
83      /**
84       * @brief Obtiene la licencia ingresada en el diálogo.
85       * @return Cadena de texto con el número de licencia.
86       */
87      QString getLicense() const;
88
89      /**
90       * @brief Obtiene el tipo de vehículo ingresado en el diálogo.
91       * @return Cadena de texto con el tipo de vehículo.
92       */
93      QString getType() const;
94
95      /**
96       * @brief Obtiene el color del vehículo ingresado en el diálogo.
97       * @return Cadena de texto con el color del vehículo.
98       */
99      QString getColor() const;
100
101     /**
102      * @brief Obtiene la marca del vehículo ingresada en el diálogo.
103      * @return Cadena de texto con la marca del vehículo.
104      */
105     QString getBrand() const;
106
107     /**
108      * @brief Obtiene el modelo del vehículo ingresado en el diálogo.
109      * @return Cadena de texto con el modelo del vehículo.
110      */
111     QString getModel() const;
112
113 private:
114     Ui::AddCardDialog *ui;
115 };
116
117 #endif // ADDCARDIALOG_H
```

3. addcarddialog.cpp

```
1   #include "addcarddialog.h"
2   #include "ui_addcarddialog.h"
3
4   AddCardDialog::AddCardDialog(QWidget *parent) :
5       QDialog(parent),
6       ui(new Ui::AddCardDialog) {
7       ui->setupUi(this);
8   }
9
10  AddCardDialog::~AddCardDialog() {
11      delete ui;
12  }
13
14  QString AddCardDialog::getName() const {
15      return ui->nameLineEdit->text();
16  }
17
18  QString AddCardDialog::getLastName() const {
19      return ui->lastNameLineEdit->text();
20  }
21
22  QString AddCardDialog::getDNI() const {
```

```
23      return ui->dniLineEdit->text();
24  }
25
26  QString AddCardDialog::getEmail() const {
27      return ui->emailLineEdit->text();
28  }
29
30  QString AddCardDialog::getPhone() const {
31      return ui->phoneLineEdit->text();
32  }
33
34  QString AddCardDialog::getAddress() const {
35      return ui->addressLineEdit->text();
36  }
37
38  int AddCardDialog::getAge() const {
39      return ui->ageSpinBox->value();}
40
41  QString AddCardDialog::getLicense() const {
42      return ui->licenseLineEdit->text();
43  }
44
45  QString AddCardDialog::getType() const {
46      return ui->typeLineEdit->text();
47  }
48
49  QString AddCardDialog::getColor() const {
50      return ui->colorLineEdit->text();
51  }
52
53  QString AddCardDialog::getBrand() const {
54      return ui->brandLineEdit->text();
55  }
56
57  QString AddCardDialog::getModel() const {
58      return ui->modelLineEdit->text();
59  }
```

4. addvehicledialog.h

```
#ifndef ADDVEHICLEDIALOG_H
#define ADDVEHICLEDIALOG_H

#include <QDialog>

namespace Ui {
class AddVehicleDialog;
}

class AddVehicleDialog : public QDialog {
    Q_OBJECT

public:
    explicit AddVehicleDialog(QWidget *parent = nullptr);
    ~AddVehicleDialog();

    QString getLicense() const;
    QString getType() const;
    QString getColor() const;
    QString getBrand() const;
    QString getModel() const;

private:
    Ui::AddVehicleDialog *ui;
};

#endif // ADDVEHICLEDIALOG_H
```

5. addvehicledialog.cpp

```
#include "addvehicledialog.h"
#include "ui_addvehicledialog.h"

AddVehicleDialog::AddVehicleDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::AddVehicleDialog) {
    ui->setupUi(this);
}

AddVehicleDialog::~AddVehicleDialog() {
    delete ui;
}

QString AddVehicleDialog::getLicense() const {
    return ui->licenseLineEdit->text();
}

QString AddVehicleDialog::getType() const {
    return ui->typeLineEdit->text();
}

QString AddVehicleDialog::getColor() const {
    return ui->colorLineEdit->text();
}

QString AddVehicleDialog::getBrand() const {
    return ui->brandLineEdit->text();
}

QString AddVehicleDialog::getModel() const {
    return ui->modelLineEdit->text();
}

```

6. balancehandler.h

```
#ifndef BALANCEHANDLER_H
#define BALANCEHANDLER_H

#include <QObject>
#include <QDebug>
#include <QFile>
#include <QTextStream>
#include <QFileInfo>
#include <QDialog>
#include <QCloseEvent>
#include "QString"
#include "lib/SAPEICore/DataBase.h"
#include "lib/SAPEICore/Error.h"
#include "ui_balancehandlerdialog.h"

class BalanceHandler : public QObject {
    Q_OBJECT

signals:
    void balanceUpdated(const QString &message);
    void balanceUpdateFailed(const QString &message);
    void windowClosed();
    void clientNotFound();


public:
    explicit BalanceHandler(DataBase *db, QObject *parent = nullptr);

    int credit(unsigned long long clientId, double amount);
    int credit(const QString &name, double amount);
    int debit(unsigned long long clientId, double amount);
    int debit(const QString &name, double amount);
    double loadPrice();
    void savePrice(double price);
    void setPrice(double newPrice);
    void completeName(const QString &name, Ui::BalanceHandlerDialog &ui);
    int openDialog(int isCharging = 0, const QString &name = "");

    Ui::BalanceHandlerDialog ui;

private:
    QDialog dialog;
    int updateBalance(Client &client, double amount);
    double price;
    DataBase *db;

protected:
    bool eventFilter(QObject *watched, QEvent *event) override;
};

#endif // BALANCEHANDLER_H
```

7. balancehandler.cpp

```
#include "balancehandler.h"

BalanceHandler::BalanceHandler(DataBase *db, QObject *parent)
    : QObject(parent), price(600), db(db) {
      this->ui.setupUi(&dialog);
      dialog.installEventFilter(this);
    }

int BalanceHandler::credit(unsigned long long clientId, double amount) {
    Client client = db->getClientById(clientId);
    if (client.isNull()) {
        qDebug() << "Cliente no encontrado con ID:" << clientId;
        return TR_CLIENT_NOT_FOUND;
    }
    return updateBalance(client, amount);
}
```

```
17
18
19
20  int BalanceHandler::credit(const QString &name, double amount) {
21      Client client = db->getClientByName(name.toStdString());
22      if (client.isNull()) {
23          qDebug() << "Cliente no encontrado con nombre:" << name;
24          return TR_CLIENT_NOT_FOUND;
25      }
26      return updateBalance(client, amount);
27  }
28
29  int BalanceHandler::debit(unsigned long long clientId, double amount) {
30      Client client = db->getClientById(clientId);
31      if (client.isNull()) {
32          qDebug() << "Cliente no encontrado con ID:" << clientId;
33          emit clientNotFound();
34          return TR_CLIENT_NOT_FOUND;
35      }
36      return updateBalance(client, -amount);
37  }
38
39  int BalanceHandler::debit(const QString &name, double amount) {
40      Client client = db->getClientByName(name.toStdString());
41      if (client.isNull()) {
42          qDebug() << "Cliente no encontrado con nombre:" << name;
43          return TR_CLIENT_NOT_FOUND;
44      }
45      return updateBalance(client, -amount);
46  }
47
48  int BalanceHandler::updateBalance(Client &client, double amount) {
49      double newBalance = client.getBalance() + amount;
50
51      if (newBalance < 0) {
52          emit balanceUpdateFailed("Fondos insuficientes para el cliente: " +
            ↪  QString::fromStdString(client.getName()));
53          return TR_NOT_ENOUGH_FUNDS;
54      }
55
56      client.setBalance(newBalance);
57      try {
58          db->updateClient(client);
59          emit balanceUpdated("Balance actualizado para el cliente: " +
            ↪  QString::fromStdString(client.getName()) + " Nuevo balance: " +
            ↪  QString::number(newBalance));
60          if(newBalance < price)
61            return TR_JUST_ENOUGH_FUNDS;
62          return OK;
63      } catch (const std::runtime_error &e) {
64          emit balanceUpdateFailed("Error al actualizar el balance: " +
            ↪  QString::fromStdString(e.what()));
65          return TR_ERROR;
66      }
67  }
68
69
70  double BalanceHandler::loadPrice() {
71      QFile file("culocacapis.txt");
72      if (file.exists()) {
73          if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
74              QTextStream in(&file);
75              in >> price;
76              file.close();
77              qDebug() << "Precio cargado desde archivo:" << price;
78              return price;
79          } else {
80              qDebug() << "No se pudo abrir el archivo para lectura.";
81          }
82      } else {
83          savePrice(price);
84          qDebug() << "Archivo no encontrado. Precio por defecto establecido a:" << price;
85          return price;
86      }
```

```
87      return 0;
88  }
89
90  void BalanceHandler::savePrice(double price) {
91      QFile file("culocacapis.txt");
92      if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
93          QTextStream out(&file);
94          out << price;
95          file.close();
96          qDebug() << "Precio guardado en archivo:" << price;
97      } else {
98          qDebug() << "No se pudo abrir el archivo para escritura.";
99      }
100 }
101
102 void BalanceHandler::setPrice(double newPrice) {
103     savePrice(newPrice);
104 }
105
106
107
108
109 int BalanceHandler::openDialog(int isCharging,const QString &name) {
110   if(!isCharging){
111     this->ui.clientNameLineEdit->setText("");
112     this->ui.amountSpinBox->setValue(0.00);
113
114     // Configurar conexiones, por ejemplo, conectar el botón de aceptar o rechazar
115     if (dialog.exec() == QDialog::Accepted) {
116       QString clientName = ui.clientNameLineEdit->text();
117       double amount = QLocale::system().toDouble(ui.amountSpinBox->text());
118
119     return credit(clientName, amount);
120     }
121   }
122   else{
123     completeName(name, this->ui);
124   }
125 }
126
127 void BalanceHandler::completeName(const QString &name, Ui::BalanceHandlerDialog &ui){
128   ui.clientNameLineEdit->setText(name);
129 }
130
131 bool BalanceHandler::eventFilter(QObject *watched, QEvent *event) {
132     if (watched == &dialog && event->type() == QEvent::Close) {
133         emit windowClosed();
134         return false;
135     }
136     return QObject::eventFilter(watched, event);
137 }
```

8. clientlistdialog.h

```h
#ifndef CLIENTLISTDIALOG_H
#define CLIENTLISTDIALOG_H

#include <QDialog>
#include <QMessageBox>
#include <QString>
#include <QListWidgetItem>
#include "lib/SAPEICore/Client.h"
#include "lib/SAPEICore/DataBase.h"
#include "editclientdialog.h"


namespace Ui {
class ClientListDialog;}

class ClientListDialog : public QDialog
{
    Q_OBJECT

public:
    explicit ClientListDialog(DataBase* db, QWidget* parent = nullptr);
    ~ClientListDialog();

private slots:
    void updateClientList();
    void onSearchTextChanged(const QString &text);
    void onClientDoubleClicked(QListWidgetItem* item);
    void onEditClientButtonClicked();
    void onDeleteClientButtonClicked();
private:
    Ui::ClientListDialog *ui;
    DataBase *database;
    void updateClient();
    void deleteClient();
};
#endif // CLIENTLISTDIALOG_H
```

9. clientlistdialog.cpp

```cpp
#include "clientlistdialog.h"
#include "ui_clientlistdialog.h"

ClientListDialog::ClientListDialog(DataBase* db, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::ClientListDialog),
    database(db)
{
    ui->setupUi(this);

    updateClientList();

    connect(ui->searchLineEdit, &QLineEdit::textChanged, this,
    ↪  &ClientListDialog::onSearchTextChanged);
    connect(ui->clientListWidget, &QListWidget::itemDoubleClicked, this,
    ↪  &ClientListDialog::onClientDoubleClicked);

    connect(ui->editClientButton, &QPushButton::clicked, this,
    ↪  &ClientListDialog::onEditClientButtonClicked);
    connect(ui->deleteClientButton, &QPushButton::clicked, this,
    ↪  &ClientListDialog::onDeleteClientButtonClicked);
}

ClientListDialog::~ClientListDialog()
{
    delete ui;
}

void ClientListDialog::updateClientList()
{
```

```
27      ui->clientListWidget->clear();
28
29      std::vector<Client> clients = database->getAllClients();
30      for (const Client &client : clients) {
31      QListWidgetItem *item = new QListWidgetItem(QString::fromStdString(client.getName()) + " - DNI:
        ↪  " + QString::number(client.getDni()));
32          ui->clientListWidget->addItem(item);
33      }
34  }
35
36  void ClientListDialog::onSearchTextChanged(const QString &text)
37  {
38      ui->clientListWidget->clear();
39
40      std::vector<Client> clients = database->getAllClients();
41      for (const Client &client : clients) {
42          QString name = QString::fromStdString(client.getName());
43          QString dni = QString::number(client.getDni());
44
45          if (name.contains(text, Qt::CaseInsensitive) || dni.contains(text, Qt::CaseInsensitive)) {
46              QString displayText = name + " - DNI: " + dni;
47              QListWidgetItem *item = new QListWidgetItem(displayText);
48              ui->clientListWidget->addItem(item);
49          }
50      }
51  }
52
53  void ClientListDialog::onClientDoubleClicked(QListWidgetItem *item) {
54      QString clientText = item->text();
55      QStringList parts = clientText.split(" - DNI: ");
56
57      std::string clientName = parts[0].toStdString();
58
59      Client client = database->getClientByName(clientName);
60
61      ui->nameLabel->setText("Nombre: " + QString::fromStdString(client.getName()));
62      ui->dniLabel->setText("DNI: " + QString::number(client.getDni()));
63      ui->emailLabel->setText("Email: " + QString::fromStdString(client.getEmail()));
64      ui->phoneLabel->setText("Teléfono: " + QString::fromStdString(client.getPhone()));
65      ui->addressLabel->setText("Dirección: " + QString::fromStdString(client.getAddress()));
66      ui->ageLabel->setText("Edad: " + QString::number(client.getAge()));
67      ui->idLabel->setText("ID: " + QString::number(client.getId()));
68      ui->hexIdLabel->setText("ID Hexadecimal: " + QString::number(client.getId(), 16).toUpper());
69      ui->balanceLabel->setText("Saldo: " + QString::number(client.getBalance()));
70
71      std::vector<Vehicle> vehicles = database->getVehiclesByClientId(client.getId());
72
73   ui->vehicleTableWidget->setRowCount(0);
74      int rowCount = static_cast<int>(vehicles.size());
75      ui->vehicleTableWidget->setRowCount(rowCount);
76      for (size_t row = 0; row < vehicles.size(); ++row) {
77          const Vehicle &vehicle = vehicles[row];
78
79   ui->vehicleTableWidget->setItem(static_cast<int>(row), 0, new
     ↪  QTableWidgetItem(QString::fromStdString(vehicle.getLicensePlate())));
80          ui->vehicleTableWidget->setItem(static_cast<int>(row), 1, new
            ↪  QTableWidgetItem(QString::fromStdString(vehicle.getBrand())));
81          ui->vehicleTableWidget->setItem(static_cast<int>(row), 2, new
            ↪  QTableWidgetItem(QString::fromStdString(vehicle.getModel())));
82          ui->vehicleTableWidget->setItem(static_cast<int>(row), 3, new
            ↪  QTableWidgetItem(QString::fromStdString(vehicle.getColor())));
83          ui->vehicleTableWidget->setItem(static_cast<int>(row), 4, new
            ↪  QTableWidgetItem(QString::fromStdString(vehicle.getType())));
84      }
85  }
86
87  void ClientListDialog::onEditClientButtonClicked(){
88
89    updateClient();
90
91  }
92
93
94  void ClientListDialog::onDeleteClientButtonClicked(){
```

```
95      deleteClient();
96  }
97
98  void ClientListDialog::deleteClient(){
99      QListWidgetItem *selectedItem = ui->clientListWidget->currentItem();
100     if (!selectedItem) {
101         QMessageBox::warning(this, "Advertencia", "Por favor, selecciona un cliente para
            ↪  eliminar.");
102         return;
103     }
104
105     QString clientText = selectedItem->text();
106     QStringList parts = clientText.split(" - DNI: ");
107     std::string clientName = parts[0].toStdString();
108     Client client = database->getClientByName(clientName);
109     unsigned long long clientId = client.getId();
110
111     QString clientNameQString = QString::fromStdString(clientName);
112
113     QMessageBox::StandardButton reply;
114     reply = QMessageBox::question(this, "Confirmar eliminación",
115                                   QString("¿Estás seguro de que deseas eliminar al cliente
                                      ↪  '%1'?").arg(clientNameQString),
116                                   QMessageBox::Yes | QMessageBox::No);
117     if (reply == QMessageBox::Yes) {
118         database->rmClient(clientId);
119         updateClientList();
120     }
121 }
122
123 void ClientListDialog::updateClient(){
124     QListWidgetItem *selectedItem = ui->clientListWidget->currentItem();
125     if (!selectedItem) {
126         QMessageBox::warning(this, "Advertencia", "Por favor, selecciona un cliente para editar.");
127         return;
128     }
129
130     QString clientText = selectedItem->text();
131     QStringList parts = clientText.split(" - DNI: ");
132     std::string clientName = parts[0].toStdString();
133     Client client = database->getClientByName(clientName);
134
135     EditClientDialog editDialog(this);
136     editDialog.setName(QString::fromStdString(client.getName()));
137     editDialog.setDNI(QString::number(client.getDni()));
138     editDialog.setEmail(QString::fromStdString(client.getEmail()));
139     editDialog.setPhone(QString::fromStdString(client.getPhone()));
140     editDialog.setAddress(QString::fromStdString(client.getAddress()));
141     editDialog.setAge(client.getAge());
142
143     if (editDialog.exec() == QDialog::Accepted) {
144
145             client.setName(editDialog.getName().toStdString());
146             client.setAge(editDialog.getAge());
147             client.setDni(editDialog.getDNI().toUInt());
148             client.setAddress(editDialog.getAddress().toStdString());
149             client.setEmail(editDialog.getEmail().toStdString());
150             client.setPhone(editDialog.getPhone().toStdString());
151
152         database->updateClient(client);
153         updateClientList();
154     }
155 }
```

10. editclientdialog.h

```h
#ifndef EDITCLIENTDIALOG_H
#define EDITCLIENTDIALOG_H

#include <QDialog>

namespace Ui {
class EditClientDialog;
}

class EditClientDialog : public QDialog
{
    Q_OBJECT

public:
    explicit EditClientDialog(QWidget *parent = nullptr);
    ~EditClientDialog();

    QString getName() const;
    QString getDNI() const;
    QString getEmail() const;
    QString getPhone() const;
    QString getAddress() const;
    int getAge() const;

    void setName(const QString &name);
    void setDNI(const QString &dni);
    void setEmail(const QString &email);
    void setPhone(const QString &phone);
    void setAddress(const QString &address);
    void setAge(int age);

private:
    Ui::EditClientDialog *ui;
};

#endif // EDITCLIENTDIALOG_H
```

11. editclientdialog.cpp

```cpp
#include "editclientdialog.h"
#include "ui_editclientdialog.h"

EditClientDialog::EditClientDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::EditClientDialog)
{
    ui->setupUi(this);

    connect(ui->saveButton, &QPushButton::clicked, this, &QDialog::accept);
    connect(ui->cancelButton, &QPushButton::clicked, this, &QDialog::reject);
}

EditClientDialog::~EditClientDialog()
{
    delete ui;
}

QString EditClientDialog::getName() const {
    return ui->nameLineEdit->text();
}

QString EditClientDialog::getDNI() const {
    return ui->dniLineEdit->text();
}

QString EditClientDialog::getEmail() const {
    return ui->emailLineEdit->text();
}

```

```
31  QString EditClientDialog::getPhone() const {
32      return ui->phoneLineEdit->text();
33  }
34
35  QString EditClientDialog::getAddress() const {
36      return ui->addressLineEdit->text();
37  }
38
39  int EditClientDialog::getAge() const {
40      return ui->ageSpinBox->value();
41  }
42
43  void EditClientDialog::setName(const QString &name) {
44      ui->nameLineEdit->setText(name);
45  }
46
47  void EditClientDialog::setDNI(const QString &dni) {
48      ui->dniLineEdit->setText(dni);
49  }
50
51  void EditClientDialog::setEmail(const QString &email) {
52      ui->emailLineEdit->setText(email);
53  }
54
55  void EditClientDialog::setPhone(const QString &phone) {
56      ui->phoneLineEdit->setText(phone);
57  }
58
59  void EditClientDialog::setAddress(const QString &address) {
60      ui->addressLineEdit->setText(address);
61  }
62
63  void EditClientDialog::setAge(int age) {
64      ui->ageSpinBox->setValue(age);
65  }
```

12. editvehicledialog.h

```h
#ifndef EDITVEHICLEDIALOG_H
#define EDITVEHICLEDIALOG_H

#include <QDialog>

namespace Ui {
class EditVehicleDialog;
}

class EditVehicleDialog : public QDialog
{
    Q_OBJECT

public:
    explicit EditVehicleDialog(QWidget *parent = nullptr);
    ~EditVehicleDialog();

    void setLicensePlate(const QString &licensePlate);
    void setType(const QString &type);
    void setColor(const QString &color);
    void setBrand(const QString &brand);
    void setModel(const QString &model);

    QString getLicensePlate() const;
    QString getType() const;
    QString getColor() const;
    QString getBrand() const;
    QString getModel() const;

private:
    Ui::EditVehicleDialog *ui;
};

#endif // EDITVEHICLEDIALOG_H
```

13. editvehicledialog.cpp

```cpp
#include "editvehicledialog.h"
#include "ui_editvehicledialog.h"

EditVehicleDialog::EditVehicleDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::EditVehicleDialog)
{
    ui->setupUi(this);

    connect(ui->buttonBox, &QDialogButtonBox::accepted, this, &QDialog::accept);
    connect(ui->buttonBox, &QDialogButtonBox::rejected, this, &QDialog::reject);
}

EditVehicleDialog::~EditVehicleDialog()
{
    delete ui;
}

void EditVehicleDialog::setLicensePlate(const QString &licensePlate) {
    ui->licensePlateLineEdit->setText(licensePlate);
}

void EditVehicleDialog::setType(const QString &type) {
    ui->typeLineEdit->setText(type);
}

void EditVehicleDialog::setColor(const QString &color) {
    ui->colorLineEdit->setText(color);
}

void EditVehicleDialog::setBrand(const QString &brand) {
    ui->brandLineEdit->setText(brand);
```

```
33  }
34
35  void EditVehicleDialog::setModel(const QString &model) {
36      ui->modelLineEdit->setText(model);
37  }
38
39  QString EditVehicleDialog::getLicensePlate() const {
40      return ui->licensePlateLineEdit->text();
41  }
42
43  QString EditVehicleDialog::getType() const {
44      return ui->typeLineEdit->text();
45  }
46
47  QString EditVehicleDialog::getColor() const {
48      return ui->colorLineEdit->text();
49  }
50
51  QString EditVehicleDialog::getBrand() const {
52      return ui->brandLineEdit->text();
53  }
54
55  QString EditVehicleDialog::getModel() const {
56      return ui->modelLineEdit->text();
57  }
```

14. mainwindow.h

```cpp
/**
 * @file mainwindow.h
 * @brief Declaración de la clase MainWindow.
 *
 * Este archivo contiene la declaración de la clase MainWindow, que es la ventana principal de la aplicación.
 * Maneja la interacción con la interfaz gráfica de usuario, la comunicación serial, y la gestión de la base de datos.
 */

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "serialhandler.h"
#include "clientlistdialog.h"
#include "vehiclelistdialog.h"
#include "addcarddialog.h"
#include "addvehicledialog.h"
#include "balancehandler.h"
#include "lib/SAPEICore/DataBase.h"
#include "lib/SAPEICore/Client.h"
#include <QMainWindow>
#include <QListWidget>
#include <QListWidgetItem>
#include <QMessageBox>
#include <QDebug>
#include <QInputDialog>
#include <QLineEdit>
#include <cstdint>
#include <QMovie>
#include <QTimer>
#include <QGraphicsTextItem>
#include <QFile>
#include <QTextStream>
#include <QFileInfo>
#include <QLabel>
#include <QGraphicsView>
#include <QVBoxLayout>
#include <QResizeEvent>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

/**
 * @class MainWindow
 * @brief Clase principal que maneja la interfaz gráfica y la lógica de la aplicación.
 *
 * La clase MainWindow es responsable de interactuar con el usuario, manejar la recepción de IDs a través del puerto serie,
 * y gestionar la base de datos de tarjetas. Además, permite agregar nuevas tarjetas y seleccionar el puerto serie.
 */
class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    /**
     * @brief Constructor de la clase MainWindow.
     * @param parent Puntero al widget padre. Si no se pasa, el valor predeterminado es nullptr.
     */
    MainWindow(QWidget *parent = nullptr);

    /**
     * @brief Destructor de la clase MainWindow.
     */
    ~MainWindow();

    /**
     * @brief Valida el ID recibido para determinar si es correcto.
     * @param id Cadena de texto con el ID a validar.
     * @return true si el ID es válido, false en caso contrario.
```

```
69          */
70         bool validateId(const QString &id);
71
72         /**
73          * @brief Agrega una tarjeta a la base de datos utilizando el ID proporcionado.
74          * @param id Cadena de texto con el ID de la tarjeta.
75          */
76         bool addCard(const QString &id);
77
78
79         void addVehicleToClient();
80         void changePrice();
81         void updatePriceDisplay();
82         void openBalanceDialog();
83
84     private slots:
85         /**
86          * @brief Slot que maneja el evento cuando se presiona el botón para agregar una tarjeta.
87          */
88         void on_addCardButton_clicked();
89
90
91         void onAddVehicleButtonClicked();
92         /**
93          * @brief Slot que maneja el evento cuando se recibe un ID desde el puerto serie.
94          * @param id Cadena de texto con el ID recibido.
95          */
96         void onIdReceived(const QString &id);
97
98         /**
99          * @brief Slot que maneja la selección del puerto serie.
100         */
101        void onSelectSerialPortClicked();
102        void onClientListButtonClicked();
103        void onVehicleListButtonClicked();
104        void updateConnectionStatus();
105        void onConfirmPriceChangeClicked();
106        void onBalanceUpdated(const QString &message);
107        void onBalanceUpdateFailed(const QString &message);
108        void updateAddCardState();
109        void onClientNotFound();
110    private:
111        void onClosedChargeWindow();
112        SerialHandler *serialHandler; ///< Manejador de comunicación serial.
113        DataBase *db; ///< Puntero a la base de datos utilizada para almacenar los clientes y tarjetas.
114        BalanceHandler *balanceHandler;  // BalanceHandler como atributo
115        QString currentId; ///< Almacena temporalmente el ID de la tarjeta actual.
116        bool isAddingCardMode; ///< Indica si la aplicación está en modo de agregar tarjeta.
117        bool isChargingMode; ///< Indica si la aplicación está en modo de carga de saldo.
118        Ui::MainWindow *ui; ///< Puntero a la interfaz gráfica de la ventana principal.
119        void setupFloatingGif();
120    protected:
121        void resizeEvent(QResizeEvent *event) override;
122
123    };
124
125    #endif // MAINWINDOW_H
```

15. mainwindow.cpp

```
1   #include "mainwindow.h"
2   #include "ui_mainwindow.h"
3
4   MainWindow::MainWindow(QWidget *parent)
5       : QMainWindow(parent),
6         serialHandler(new SerialHandler(this)),
7         db(new DataBase("database.db")),
8         balanceHandler(new BalanceHandler(db, this)),
9         isAddingCardMode(false),
10        isChargingMode(false),
11        ui(new Ui::MainWindow) {
12        ui->setupUi(this);
```

```
13
14      setupFloatingGif();
15
16      updateAddCardState();
17
18      try {
19          ui->textBrowser->append("Base de datos inicializada correctamente.");
20      } catch (const std::runtime_error& e) {
21          ui->textBrowser->append("Error al inicializar la base de datos: " +
            ↪  QString::fromStdString(e.what()));
22      }
23
24      connect(ui->selectSerialPortButton, &QPushButton::clicked, this,
        ↪  &MainWindow::onSelectSerialPortClicked);
25      connect(serialHandler, &SerialHandler::idReceived, this, &MainWindow::onIdReceived);
26      connect(ui->addVehicleButton, &QPushButton::clicked, this,
        ↪  &MainWindow::onAddVehicleButtonClicked);
27      connect(ui->clientListButton, &QPushButton::clicked, this,
        ↪  &MainWindow::onClientListButtonClicked);
28      connect(ui->vehicleListButton, &QPushButton::clicked, this,
        ↪  &MainWindow::onVehicleListButtonClicked);
29      connect(ui->confirmPriceButton, &QPushButton::clicked, this,
        ↪  &MainWindow::onConfirmPriceChangeClicked);
30      connect(ui->chargeBalanceButton, &QPushButton::clicked, this, &MainWindow::openBalanceDialog);
31      connect(balanceHandler, &BalanceHandler::balanceUpdated, this, &MainWindow::onBalanceUpdated);
32      connect(balanceHandler, &BalanceHandler::balanceUpdateFailed, this,
        ↪  &MainWindow::onBalanceUpdateFailed);
33      connect(balanceHandler, &BalanceHandler::windowClosed, this, &MainWindow::onClosedChargeWindow);
34      connect(balanceHandler, &BalanceHandler::clientNotFound, this, &MainWindow::onClientNotFound);
35      connect(balanceHandler->ui.acceptButton, &QPushButton::clicked, this,
        ↪  &MainWindow::onClosedChargeWindow);
36      connect(balanceHandler->ui.cancelButton, &QPushButton::clicked, this,
        ↪  &MainWindow::onClosedChargeWindow);
37
38
39      QTimer *connectionStatusTimer = new QTimer(this);
40      connect(connectionStatusTimer, &QTimer::timeout, this, &MainWindow::updateConnectionStatus);
41      connectionStatusTimer->start(1000);
42      QString configFileName = "configfile.conf";
43      QFile configFile(configFileName);
44
45      if (!QFileInfo::exists(configFileName)) {
46          if (configFile.open(QIODevice::WriteOnly | QIODevice::Text)) {
47              QTextStream out(&configFile);
48              out << 600;
49              configFile.close();
50          }
51      }
52
53      if (configFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
54          QTextStream in(&configFile);
55          double fileprice;
56          in >> fileprice;
57          configFile.close();
58
59          updatePriceDisplay();
60      }
61
62  }
63
64  MainWindow::~MainWindow() {
65      delete ui;
66  }
67
68  void MainWindow::onSelectSerialPortClicked() {
69      serialHandler->selectSerialPort();
70  }
71
72  void MainWindow::resizeEvent(QResizeEvent *event) {
73      QMainWindow::resizeEvent(event);
74
75      QGraphicsView *connectionStatusView = findChild<QGraphicsView *>("connectionStatusView");
76      if (!connectionStatusView) return;
77
```

```
78      QLabel *floatingGif = connectionStatusView->findChild<QLabel *>("floatingGif");
79      if (!floatingGif) return;
80
81      int gifWidth = 100;
82      int gifHeight = 100;
83      floatingGif->setGeometry(connectionStatusView->width() - gifWidth,
84                               0,
85                               gifWidth,
86                               gifHeight);
87  }
88
89  void MainWindow::setupFloatingGif() {
90      QGraphicsView *connectionStatusView = findChild<QGraphicsView *>("connectionStatusView");
91
92      QLabel *floatingGif = new QLabel(connectionStatusView);
93      floatingGif->setAttribute(Qt::WA_TranslucentBackground); // Fondo transparente
94      floatingGif->setAttribute(Qt::WA_NoSystemBackground);
95      floatingGif->setObjectName("floatingGif");
96
97      QMovie *gifMovie = new QMovie(":/utnlogo.gif");
98
99      int gifWidth = 100;
100     int gifHeight = 100;
101     gifMovie->setScaledSize(QSize(gifWidth, gifHeight));
102
103     floatingGif->setMovie(gifMovie);
104     gifMovie->start();
105
106     floatingGif->setGeometry(connectionStatusView->width() - gifWidth,
107                              0,
108                              gifWidth,
109                              gifHeight);
110
111     floatingGif->raise();
112 }
113
114 void MainWindow::updateConnectionStatus() {
115     QGraphicsScene* scene = new QGraphicsScene(this);
116     QGraphicsTextItem* textItem = new QGraphicsTextItem;
117
118     if (serialHandler->isConnected()) {
119         textItem->setPlainText("Conectado");
120         textItem->setDefaultTextColor(Qt::green);
121     } else {
122         textItem->setPlainText("Desconectado");
123         textItem->setDefaultTextColor(Qt::red);
124     }
125
126     QFont font = textItem->font();
127     font.setPointSize(24);
128     font.setBold(true);
129     textItem->setFont(font);
130
131     scene->addItem(textItem);
132     ui->connectionStatusView->setScene(scene);
133     updateAddCardState();
134 }
135
136 void MainWindow::openBalanceDialog(){
137   isChargingMode = true;
138   QString msg = QString::number(balanceHandler->openDialog());
139   serialHandler->sendToArduino(msg);
140 }
141
142 void MainWindow::updatePriceDisplay()
143 {
144     double price = 0;
145     price = balanceHandler->loadPrice();
146     ui->priceDisplay->display(price);
147 }
148
149 void MainWindow::on_addCardButton_clicked() {
150     isAddingCardMode = true;
151     ui->textBrowser->append("Modo de agregar tarjeta activado. Escanee una tarjeta...");
```

```
152
153     if (serialHandler->isConnected()) {
154         qDebug() << "Puerto serie conectado correctamente";
155     } else {
156         qDebug() << "No se pudo conectar al puerto serie";
157     }
158 }
159
160 void MainWindow::onAddVehicleButtonClicked()
161 {
162     addVehicleToClient();
163 }
164
165 void MainWindow::onClientListButtonClicked()
166 {
167     ClientListDialog dialog(db, this);
168     dialog.exec();
169 }
170
171 void MainWindow::onVehicleListButtonClicked()
172 {
173     VehicleListDialog dialog(db, this);
174     dialog.exec();
175 }
176
177 void MainWindow::onConfirmPriceChangeClicked() {
178   changePrice();
179   updatePriceDisplay();
180 }
181
182 void MainWindow::onBalanceUpdated(const QString &message) {
183     ui->textBrowser->append("Éxito: " + message);
184 }
185
186 void MainWindow::onBalanceUpdateFailed(const QString &message) {
187     ui->textBrowser->append("Error: " + message);
188 }
189
190 void MainWindow::onClientNotFound(){
191     ui->textBrowser->append("Cliente no registrado");
192 }
193
194 void MainWindow::changePrice(){
195
196     bool ok;
197     double newPrice = ui->priceInput->text().toDouble(&ok);
198     if (newPrice >= 0) {
199         ui->textBrowser->append("Precio de estacionamiento actualizado a: $" +
    ↪  QString::number(newPrice, 'f', 2));
200         balanceHandler->savePrice(newPrice);
201     } else {
202         QMessageBox::warning(this, "Error de entrada", "El precio debe ser un valor positivo.");
203     }
204 }
205
206 bool MainWindow::validateId(const QString &id) {
207     if (id.length() != 8) {
208         qDebug() << "ID no válido";
209         return false;
210     }
211     return true;
212 }
213
214 void MainWindow::onIdReceived(const QString &id) {
215     qDebug() << "ID recibido";
216     currentId = id;
217     bool ok;
218
219     if (validateId(currentId)) {
220         unsigned long long idInt = id.toULongLong(&ok, 16);
221         qDebug() << "ID recibido: " << currentId;
222         Client client = db->getClientById(idInt);
223         double currentBalance = db->getBalance(idInt);
224         double chargeAmount = balanceHandler->loadPrice();
```

```
225
226         if(!(isAddingCardMode || isChargingMode)){
227           QString msg = QString::number(balanceHandler->debit(idInt, chargeAmount));
228           serialHandler->sendToArduino(msg);
229         }
230         if(isChargingMode){
231           QString nombre = QString::fromStdString(client.getName());
232           balanceHandler->openDialog(isChargingMode, nombre);
233         }
234         else if(isAddingCardMode){
235           if(!client.isNull()){
236             ui->textBrowser->append("ID ya existe en la base de datos. No se añade la tarjeta. El
                ↪  cliente es " + QString::fromStdString(client.getName()));
237           }
238           else if(!addCard(currentId)){
239             ui->textBrowser->append("No se pudo agregar la tarjeta.");
240           }
241           else {
242           ui->textBrowser->append("Tarjeta añadida con éxito.");
243           }
244           isAddingCardMode = false;
245         }
246       }
247       else {
248        ui->textBrowser->append("ID no válido: " + id);
249       }
250   }
251
252   bool MainWindow::addCard(const QString &id){
253     AddCardDialog dialog(this);
254     if (dialog.exec() == QDialog::Accepted) {
255       QString name = dialog.getName();
256       QString lastName = dialog.getLastName();
257       QString dniString = dialog.getDNI();
258       QString email = dialog.getEmail();
259       QString phone = dialog.getPhone();
260       QString address = dialog.getAddress();
261       int age = dialog.getAge();
262       QString license = dialog.getLicense();
263       QString type = dialog.getType();
264       QString color = dialog.getColor();
265       QString brand = dialog.getBrand();
266       QString model = dialog.getModel();
267
268       bool ok;
269       unsigned long long idInt = id.toULongLong(&ok, 16);
270       if (!ok) {
271           qDebug() << "Error al convertir el ID de hexadecimal a entero";
272       }
273
274       unsigned int dni = dniString.toUInt(&ok);
275       if (!ok) {
276           qDebug() << "Error al convertir el DNI a entero";
277       }
278
279       QString fullName = name + " " + lastName;
280
281       Client newClient(idInt, fullName.toStdString(), age, dni, address.toStdString(),
282                        email.toStdString(), phone.toStdString(),
283                        license.toStdString(), type.toStdString(),
284                        color.toStdString(), brand.toStdString(),
285                        model.toStdString());
286
287       newClient.setBalance(0);
288
289       try {
290           db->addClient(newClient);
291           ui->textBrowser->append("Cliente agregado: " + name + " " + lastName + " con ID: " + id);
292           return 1;
293       } catch (const std::runtime_error& e) {
294           ui->textBrowser->append("Error al agregar el cliente: " + QString::fromStdString(e.what()));
295           return 0;
296       }
297     }
```

```
298 }
299
300 void MainWindow::addVehicleToClient() {
301     bool ok;
302     QString inputName = QInputDialog::getText(this, "Agregar Vehículo",
303                                               "Ingrese el nombre del cliente:",
304                                               QLineEdit::Normal, "", &ok);
305
306     if (!ok) {
307         ui->textBrowser->append("Operación de agregar vehículo cancelada.");
308         return;
309     }
310
311     Client client;
312
313     if ((client = db->getClientByName(inputName.toStdString())).isNull()) {
314         QMessageBox::warning(this, "Cliente no encontrado",
315                              "El cliente con el nombre ingresado no existe en la base de datos.");
316         return;
317     } else {
318         std::cout << client << std::endl;
319     }
320
321     unsigned long long clientId = client.getId();
322
323     AddVehicleDialog vehicleDialog(this);
324     if (vehicleDialog.exec() == QDialog::Accepted) {
325         QString license = vehicleDialog.getLicense();
326         QString type = vehicleDialog.getType();
327         QString color = vehicleDialog.getColor();
328         QString brand = vehicleDialog.getBrand();
329         QString model = vehicleDialog.getModel();
330
331         Vehicle newVehicle(license.toStdString(), type.toStdString(),
332                            color.toStdString(), brand.toStdString(),
333                            model.toStdString());
334
335         try {
336             db->addVehicle(clientId, newVehicle);
337             ui->textBrowser->append("Vehículo agregado al cliente " + inputName);
338         } catch (const std::runtime_error& e) {
339             ui->textBrowser->append("Error al agregar el vehículo: " +
                ↪  QString::fromStdString(e.what()));
340         }
341     } else {
342         ui->textBrowser->append("Operación de agregar vehículo cancelada.");
343     }
344 }
345
346 void MainWindow::onClosedChargeWindow(){
347   isChargingMode = false;
348 }
349
350 void MainWindow::updateAddCardState(){
351   if(serialHandler->isConnected()){
352     ui->addCardButton->setEnabled(true);
353     ui->addCardButton->setStyleSheet(
354     "QPushButton {"
355         "background-color: #2d2d2d;"
356         "color: #ffffff;"
357         "border: 1px solid #3c3c3c;"
358         "padding: 5px;"
359         "border-radius: 4px;"
360     "}"
361     "QPushButton:hover {"
362         "background-color: #3c3c3c;"
363     "}"
364     );
365   }
366   else{
367     ui->addCardButton->setEnabled(false);
368     ui->addCardButton->setStyleSheet("background-color: #1c1c1c; color: #eeeeee;");
369   }
```

```
370 }
```

16. serialhandler.h

```cpp
/**
 * @file serialhandler.h
 * @brief Declaración de la clase SerialHandler.
 *
 * Este archivo contiene la declaración de la clase SerialHandler, responsable de manejar la comunicación con el puerto serie,
 * enviar y recibir datos, y procesar los mensajes recibidos desde un dispositivo conectado por serial.
 */

#ifndef SERIALHANDLER_H
#define SERIALHANDLER_H

#include <QSerialPort>
#include <QObject>

/**
 * @class SerialHandler
 * @brief Clase que maneja la comunicación serie con un dispositivo (por ejemplo, Arduino).
 *
 * La clase SerialHandler permite seleccionar un puerto serie, conectarse a él, enviar mensajes,
 * y recibir datos desde el dispositivo conectado. También proporciona una señal que emite cuando se recibe un ID.
 */
class SerialHandler : public QObject {
    Q_OBJECT

public:
    /**
     * @brief Constructor de la clase SerialHandler.
     * @param parent Puntero al objeto padre. Si no se pasa, el valor predeterminado es nullptr.
     */
    SerialHandler(QObject *parent = nullptr);

    /**
     * @brief Selecciona el puerto serie para establecer la conexión.
     *
     * Este método abre un diálogo para que el usuario seleccione el puerto serie al que desea conectarse.
     */
    void selectSerialPort();

    /**
     * @brief Conecta al puerto serie especificado.
     * @param portName Nombre del puerto serie al que se conectará.
     */
    int connectSerialPort(const QString &portName);

    /**
     * @brief Envía un mensaje al dispositivo conectado por el puerto serie.
     * @param message Cadena de texto que contiene el mensaje a enviar.
     */
    void sendToArduino(const QString &message);

    /**
     * @brief Verifica si hay una conexión activa con el puerto serie.
     * @return true si está conectado, false en caso contrario.
     */
    bool isConnected() const;

signals:
    /**
     * @brief Señal que se emite cuando se recibe un ID desde el dispositivo serie.
     * @param id Cadena de texto que contiene el ID recibido.
     */
    void idReceived(const QString &id);

private slots:
    /**
     * @brief Slot privado que maneja la lectura de los datos entrantes desde el puerto serie.
     *
```

```
68          * Este método es llamado automáticamente cuando hay datos disponibles para leer en el puerto
            ↪ serie.
69          */
70          void handleReadyRead();
71
72      private:
73          QSerialPort *serial;
74      };
75
76      #endif // SERIALHANDLER_H
```

17. serialhandler.cpp

```
1       #include "serialhandler.h"
2       #include <QDebug>
3       #include <QDialog>
4       #include <QVBoxLayout>
5       #include <QComboBox>
6       #include <QDialogButtonBox>
7       #include <QSerialPortInfo>
8       #include <QFile>
9
10      SerialHandler::SerialHandler(QObject *parent)
11          : QObject(parent), serial(new QSerialPort(this)) {
12      }
13
14      void SerialHandler::selectSerialPort() {
15          QDialog dialog;
16          dialog.setWindowTitle("Seleccionar Puerto Serie");
17
18          // Aplicar tema oscuro
19          dialog.setStyleSheet(R"(
20              QDialog {
21                  background-color: #2e2e2e;
22                  color: #ffffff;
23              }
24              QLabel, QComboBox, QPushButton {
25                  color: #ffffff;
26              }
27              QComboBox {
28                  background-color: #3c3c3c;
29                  border: 1px solid #5e5e5e;
30              }
31              QComboBox QAbstractItemView {
32                  background-color: #3c3c3c;
33                  selection-background-color: #5e5e5e;
34                  color: #ffffff;
35              }
36              QDialogButtonBox QPushButton {
37                  background-color: #5e5e5e;
38                  border: 1px solid #5e5e5e;
39                  color: #ffffff;
40              }
41              QDialogButtonBox QPushButton:hover {
42                  background-color: #757575;
43              }
44          )");
45
46          QVBoxLayout layout(&dialog);
47
48          // ComboBox para listar los puertos serie disponibles
49          QComboBox comboBox;
50
51          const auto ports = QSerialPortInfo::availablePorts();
52          for (const QSerialPortInfo &port : ports) {
53              comboBox.addItem(port.portName());
54          }
55
56          // Agregar puertos PTY manualmente
57          for (int i = 0; i < 256; ++i) {
58              QString ptName = QString("/dev/pts/%1").arg(i);
59              if (QFile::exists(ptName)) {
```

```
60              comboBox.addItem(ptName);
61          }
62      }
63
64      layout.addWidget(&comboBox);
65
66      QDialogButtonBox buttonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel);
67      connect(&buttonBox, &QDialogButtonBox::accepted, &dialog, &QDialog::accept);
68      connect(&buttonBox, &QDialogButtonBox::rejected, &dialog, &QDialog::reject);
69      layout.addWidget(&buttonBox);
70
71      if (dialog.exec() == QDialog::Accepted) {
72          QString selectedPortName = comboBox.currentText();
73          if (!selectedPortName.isEmpty()) {
74              if(connectSerialPort(selectedPortName))
75                qDebug() << "Error para conectarse a: " << selectedPortName;
76              else
77                qDebug() << "Conectado a: " << selectedPortName;
78          } else {
79              qDebug() << "No se seleccionó ningún puerto.";
80          }
81      }
82  }
83
84  #include <QElapsedTimer>
85  #include <QThread>
86
87  int SerialHandler::connectSerialPort(const QString &portName) {
88      if (serial->isOpen()) {
89          serial->close();
90          qDebug() << "Cerrando conexión en el puerto anterior.";
91      }
92
93      serial->setPortName(portName);
94      serial->setBaudRate(QSerialPort::Baud115200);
95      serial->setDataBits(QSerialPort::Data8);
96      serial->setParity(QSerialPort::NoParity);
97      serial->setStopBits(QSerialPort::OneStop);
98      serial->setFlowControl(QSerialPort::NoFlowControl);
99
100     if (serial->open(QIODevice::ReadWrite)) {
101         qDebug() << "Conexión serial abierta en" << portName;
102             serial->waitForReadyRead(4000);
103             QByteArray initResponse = serial->readAll();
104             sendToArduino("SAPEI_INIT");
105             if (serial->waitForReadyRead(3000)) {
106                 QByteArray connectedResponse = serial->readAll();
107                 if (connectedResponse.contains("CONNECTED")) {
108                     qDebug() << "Conexión verificada: 'CONNECTED' recibido.";
109                     connect(serial, &QSerialPort::readyRead, this, &SerialHandler::handleReadyRead);
110                     return 0;
111                 } else {
112                     qDebug() << "No se recibió respuesta 'CONNECTED'.";
113                     serial->close();
114                     return 1;
115                 }
116             } else {
117                 qDebug() << "Timeout: No se recibió 'CONNECTED'.";
118                 serial->close();
119                 return 1;
120             }
121     } else {
122         qDebug() << "No se pudo abrir la conexión serial en" << portName;
123         qDebug() << "Error:" << serial->errorString();
124         return 1;
125     }
126     return 1;
127 }
128
129 void SerialHandler::sendToArduino(const QString &message) {
130     if (serial->isOpen()) {
131         QByteArray byteArray = (message + "\n").toUtf8();
132         serial->write(byteArray);
133
```

```
134         if (!serial->waitForBytesWritten(1000)) {
135             qDebug() << "Error al enviar datos al Arduino:" << serial->errorString();
136         }
137     } else {
138         qDebug() << "El puerto serie no está abierto.";
139     }
140 }
141 
142 void SerialHandler::handleReadyRead() {
143     while (serial->canReadLine()) {
144         QString line = serial->readLine().trimmed();
145         emit idReceived(line);
146         qDebug() << "Datos leídos del puerto serie:" << line;
147     }
148 }
149 
150 
151 
152 bool SerialHandler::isConnected() const {
153     return serial->isOpen();
154 }
155 
```

18. vehiclelistdialog.h

```h
#ifndef VEHICLELISTDIALOG_H
#define VEHICLELISTDIALOG_H

#include <QDialog>
#include <QMessageBox>
#include <QString>
#include <QListWidgetItem>
#include "lib/SAPEICore/Client.h"
#include "lib/SAPEICore/DataBase.h"
#include "editvehicledialog.h"


namespace Ui {
    class VehicleListDialog;
}

class VehicleListDialog : public QDialog
{
    Q_OBJECT

public:
    explicit VehicleListDialog(DataBase* db, QWidget* parent = nullptr);
    ~VehicleListDialog();

private slots:
    void updateVehicleList();
    void onSearchTextChanged(const QString &text);
    void onEditVehicleButtonClicked();
    void onDeleteVehicleButtonClicked();
    void onVehicleDoubleClicked(QListWidgetItem *item);
private:
    Ui::VehicleListDialog *ui;
    DataBase *database;
};
#endif // VEHICLLISTDIALOG_H

```

19. vehiclelistdialog.cpp

```cpp
#include "vehiclelistdialog.h"
#include "ui_vehiclelistdialog.h"

VehicleListDialog::VehicleListDialog(DataBase* db, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::VehicleListDialog),
    database(db)
{
    ui->setupUi(this);

    updateVehicleList();

    // Conectar el cambio de texto del campo de búsqueda al slot de actualización de la lista
    connect(ui->searchLineEdit, &QLineEdit::textChanged, this,
    ↪  &VehicleListDialog::onSearchTextChanged);
    connect(ui->editVehicleButton, &QPushButton::clicked, this,
    ↪  &VehicleListDialog::onEditVehicleButtonClicked);
    connect(ui->deleteVehicleButton, &QPushButton::clicked, this,
    ↪  &VehicleListDialog::onDeleteVehicleButtonClicked);
    connect(ui->VehicleListWidget, &QListWidget::itemDoubleClicked, this,
    ↪  &VehicleListDialog::onVehicleDoubleClicked);
}

VehicleListDialog::~VehicleListDialog()
{
    delete ui;
}

void VehicleListDialog::updateVehicleList()
{
```

```
27      ui->VehicleListWidget->clear();
28
29      std::vector<Vehicle> vehicles = database->getAllVehicles();
30      for (const Vehicle &vehicle : vehicles) {
31          QString displayText = QString::fromStdString(vehicle.getLicensePlate() + " - " +
32                                                       vehicle.getBrand() + " - " +
33                                                       vehicle.getModel());
34          QListWidgetItem *item = new QListWidgetItem(displayText);
35          ui->VehicleListWidget->addItem(item);
36      }
37  }
38
39  void VehicleListDialog::onSearchTextChanged(const QString &text)
40  {
41      ui->VehicleListWidget->clear();
42
43      std::vector<Vehicle> vehicles = database->getAllVehicles();
44      for (const Vehicle &vehicle : vehicles) {
45          QString displayText = QString::fromStdString(vehicle.getLicensePlate() + " - " +
46                                                       vehicle.getBrand() + " - " +
47                                                       vehicle.getModel());
48          if (displayText.contains(text, Qt::CaseInsensitive)) {
49              QListWidgetItem *item = new QListWidgetItem(displayText);
50              ui->VehicleListWidget->addItem(item);
51          }
52      }
53  }
54
55
56  void VehicleListDialog::onEditVehicleButtonClicked()
57  {
58      QListWidgetItem *selectedItem = ui->VehicleListWidget->currentItem();
59      if (!selectedItem) {
60          QMessageBox::warning(this, "Advertencia", "Por favor, selecciona un vehículo para editar.");
61          return;
62      }
63
64      QString itemText = selectedItem->text();
65      QStringList parts = itemText.split(" - ");
66      if (parts.isEmpty()) {
67          QMessageBox::warning(this, "Error", "Formato de texto inválido en la lista de vehículos.");
68          return;
69      }
70      std::string licensePlate = parts[0].toStdString();
71
72      Vehicle vehicle = database->getVehicleByPlate(licensePlate);
73      if (vehicle.isNull()) {
74          QMessageBox::warning(this, "Error", "El vehículo no existe en la base de datos.");
75          return;
76      }
77
78      unsigned long long clientId = vehicle.getClientId();
79
80      EditVehicleDialog editDialog(this);
81      editDialog.setLicensePlate(QString::fromStdString(vehicle.getLicensePlate()));
82      editDialog.setType(QString::fromStdString(vehicle.getType()));
83      editDialog.setColor(QString::fromStdString(vehicle.getColor()));
84      editDialog.setBrand(QString::fromStdString(vehicle.getBrand()));
85      editDialog.setModel(QString::fromStdString(vehicle.getModel()));
86
87      // Si se confirma la edición, actualizar los datos del vehículo
88      if (editDialog.exec() == QDialog::Accepted) {
89          Vehicle updatedVehicle(
90              editDialog.getLicensePlate().toStdString(),
91              editDialog.getType().toStdString(),
92              editDialog.getColor().toStdString(),
93              editDialog.getBrand().toStdString(),
94              editDialog.getModel().toStdString()
95          );
96
97          database->rmVehicle(licensePlate);
98          database->addVehicle(clientId, updatedVehicle);
99
100         updateVehicleList();
```

```
101     }
102 }
103
104
105 void VehicleListDialog::onDeleteVehicleButtonClicked(){
106     QListWidgetItem *selectedItem = ui->VehicleListWidget->currentItem();
107     if (!selectedItem) {
108         QMessageBox::warning(this, "Advertencia", "Por favor, selecciona un vehículo para
        ↪  eliminar.");
109         return;
110     }
111
112  QString itemText = selectedItem->text();
113     QStringList parts = itemText.split(" - ");
114     if (parts.isEmpty()) {
115         QMessageBox::warning(this, "Error", "Formato de texto inválido en la lista de vehículos.");
116         return;
117     }
118
119     QString licensePlateQString = parts[0];
120
121     QMessageBox::StandardButton reply;
122     reply = QMessageBox::question(this, "Confirmar eliminación",
123                                   QString("¿Estás seguro de que deseas eliminar el vehículo con
                                      ↪  placa '%1'?").arg(licensePlateQString),
124                                   QMessageBox::Yes | QMessageBox::No);
125     if (reply == QMessageBox::Yes) {
126         database->rmVehicle(licensePlateQString.toStdString());
127         updateVehicleList();
128     }
129 }
130
131 void VehicleListDialog::onVehicleDoubleClicked(QListWidgetItem *item)
132 {
133     if (!item) return;
134
135     QString itemText = item->text();
136     QStringList parts = itemText.split(" - ");
137     if (parts.isEmpty()) return;
138
139     std::string licensePlate = parts[0].toStdString();
140     Vehicle vehicle = database->getVehicleByPlate(licensePlate);
141     if (vehicle.isNull()) return;
142
143     Client client = database->getClientById(vehicle.getClientId());
144
145     ui->typeLabel->setText("Tipo: " + QString::fromStdString(vehicle.getType()));
146     ui->colorLabel->setText("Color: " + QString::fromStdString(vehicle.getColor()));
147     ui->brandLabel->setText("Marca: " + QString::fromStdString(vehicle.getBrand()));
148     ui->modelLabel->setText("Modelo: " + QString::fromStdString(vehicle.getModel()));
149     ui->clientInfoLabel->setText("Cliente: " + QString::fromStdString(client.getName()));
150     ui->licensePlateLabel->setText("Placa: " + QString::fromStdString(vehicle.getLicensePlate()));
151     ui->idLabel->setText("ID: " + QString::number(client.getId()));
152     ui->hexIdLabel->setText("ID: " + QString::number(client.getId(), 16).toUpper());
153 }
```