

**UNIVERSIDAD NACIONAL DE SAN AGUSTÍN
FACULTAD DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE CIENCIA DE LA
COMPUTACIÓN**



“DOCUMENTACIÓN DEL APLICATIVO”

**ASIGNATURA:
CIENCIA DE LA COMPUTACIÓN II**

**SEMESTRE:
III**

ESTUDIANTES:

Roque Sosa, Owen Haniel
Zhong Callasi, Linghai Joaquin
Ruiz Mamani, Eduardo German
Velásquez Montoya Juan José
Castillo Sancho Sergio Ahmed

Arequipa - Perú

2022

DOCUMENTACIÓN DEL PROYECTO

Título del proyecto	Programa para la agendación de citas de centros médicos
Código del proyecto	Enlace a repositorio en Github: https://github.com/Sommerfield3/-CCII-proyecto
Tipo de documento adjunto	Informe del Proyecto (PDF)
Organización/Usuario para quien se redacta el proyecto	Universidad Nacional de San Agustín
Autor (es)	<ul style="list-style-type: none"> • Castillo Sancho, Sergio Ahmed • Roque Sosa, Owen Haziel • Ruiz Mamani, Eduardo German • Velásquez Montoya, Juan José • Zhong Callasi, Linghai Joaquín
Duración estimada en jornadas:	Fecha de inicio: 1 de Junio de 2022 Fecha de Entrega (III Parcial): 19 de agosto de 2022

Introducción:

Se presenta un sistema de registro de citas médicas que se divide principalmente en tres partes: médico, paciente y administrador. El médico y el paciente tienen acceso a un calendario de citas que se actualiza y visualiza de acuerdo a las opciones que elija el usuario. Es importante mencionar que los usuarios se ordenan automáticamente con cada ingreso de nuevos usuarios.

Asimismo, el administrador tiene la facultad de crear nuevos pacientes, médicos y agregar citas.

En el proyecto se implementaron smart pointers para reducir el número de memory leaks. Asimismo, se implementaron functores (function objects) para realizar el conteo de las operaciones realizadas en cada sesión.

En las clases newmed y newpac donde anteriormente se generaban instancias para el ingreso de datos, se decidió utilizar los tipo de punteros QScopedPointers (análogos a los Unique Pointers) y los QSharedPointers (análogos a los Shared Pointers).

Es importante mencionar que el proyecto se realizó en Qt, que es una aplicación multiplataforma y herramientas de widget que utiliza C++ estándar.

Esquema del proyecto:

1. Base de datos:

Se elaboró en MySQL con el nombre de cc_proyecto_final_22.

Se utilizaron las siguientes tablas: administrador, citas, médico, paciente y usuarios.

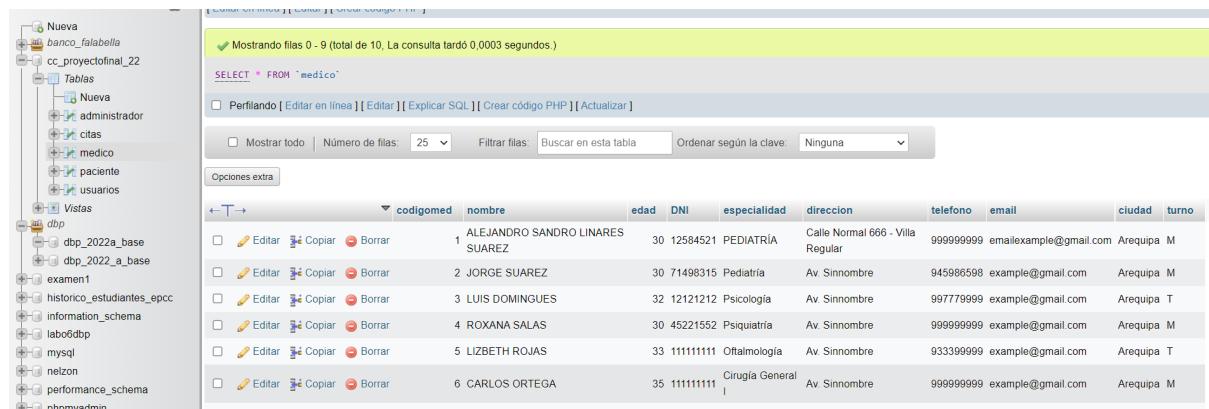
Tabla de administradores:



The screenshot shows the MySQL Workbench interface. On the left, the database schema is visible with the 'administrador' table selected under the 'cc_proyecto_final_22' database. The main pane displays the 'administrador' table with two rows of data:

	codigoadm	nombre	edad	DNI	direccion	telefono	email	ciudad
1	ALVARO HENRY MAMANI ALIAGA	25	12345678	Av TodosAprobados 2020 - Villa Bondad	987654321	eimejorprofe77@gmail.com	Esperanza	
2	ENZO EDIR VELASQUEZ LOBATON	24	87654321	Calle TheBest 2020 - Urbanización Misericordia	123456789	evelasquez@unsa.edu.pe	Victoria	

Tabla de médicos:



The screenshot shows the MySQL Workbench interface. On the left, the database schema is visible with the 'medico' table selected under the 'cc_proyecto_final_22' database. The main pane displays the 'medico' table with six rows of data:

	codigomed	nombre	edad	DNI	especialidad	direccion	telefono	email	ciudad	turno
1	ALEJANDRO SANDRO LINARES SUAREZ	30	12584521	PEDIATRÍA Regular	Calle Normal 666 - Villa	999999999	email@example@gmail.com	Arequipa	M	
2	JORGE SUAREZ	30	71498315	Pediatria	Av. Sinnombre	945986598	example@gmail.com	Arequipa	M	
3	LUIS DOMINGUES	32	12121212	Psicología	Av. Sinnombre	997779999	example@gmail.com	Arequipa	T	
4	ROXANA SALAS	30	45221552	Psiquiatría	Av. Sinnombre	999999999	example@gmail.com	Arequipa	M	
5	LIZBETH ROJAS	33	1111111111	Oftalmología	Av. Sinnombre	933399999	example@gmail.com	Arequipa	T	
6	CARLOS ORTEGA	35	1111111111	Cirugía General I	Av. Sinnombre	999999999	example@gmail.com	Arequipa	M	

Tabla de citas:

Tree view of database structure:

- tablas
 - Nueva
 - administrador
 - citas
 - medico
 - paciente
 - usuarios
- Vistas
 - Nueva
 - medicosview
 - pacientesview
- dbp
 - dbp_2022a_base
 - dbp_2022_a_base
- examen1
- historico_estudiantes_epcc
- information_schema
- labo6dbp
- mysql
- nelzon
- performance_schema
- phpmyadmin
- sistemaasistencia

Table data for 'citas' (Turno M = mañana, T = tarde, N = Noche):

idcita	codigopac	codigomed	turno	fecha	hora	estado
2	4	1	M	2022-08-01	08:00:00	1
3	1	4	M	2022-08-17	08:30:00	2
4	2	4	M	2022-08-01	09:00:00	2
5	3	4	M	2022-08-01	10:00:00	2
8	5	2	T	2022-08-15	12:30:00	2
9	6	2	T	2022-08-15	13:00:00	2
10	7	10	N	2022-08-17	20:00:00	2
11	8	1	T	2022-08-01	14:00:00	1
12	9	1	N	2022-08-01	18:00:00	1
13	1	1	M	2022-08-18	09:00:00	2
14	3	1	T	2022-08-18	15:00:00	2

Tabla de pacientes:

Tree view of database structure:

- banco_falabella
- cc_proyectofinal_22
 - Tablas
 - Nueva
 - administrador
 - citas
 - medico
 - paciente
 - usuarios
 - Vistas
- dbp
 - dbp_2022a_base
 - dbp_2022_a_base
- examen1
- historico_estudiantes_epcc
- information_schema
- labo6dbp
- mysql
- nelzon

Table data for 'paciente':

codigopac	nombre	edad	DNI	direccion	telefono	email	ciudad
1	SERGIO ESCOBEDO	27	12345678	Av.SinNombre	999999999	example@gmail.com	Arequipa
2	SOLEDAD RAMOS	20	12312312	Av.SinNombre	944499999	example@gmail.com	Arequipa
3	JUANA RODRIGUEZ	35	12312365	Av.SinNombre	999777999	example@gmail.com	Arequipa
4	LEANDRO SANTANA	36	12345645	Av.SinNombre	777777777	example@gmail.com	Arequipa

Tabla de usuarios:

Tree view of database structure:

- banco_falabella
- cc_proyectofinal_22
 - Tablas
 - Nueva
 - administrador
 - citas
 - medico
 - paciente
 - usuarios
 - Vistas
- dbp
 - dbp_2022a_base
 - dbp_2022_a_base
- examen1

Table data for 'usuarios':

codigo	cargo	usuario	clave	idreferencia
1	P	juanaR	202cb962ac59075b964b07152d234b70	3
2	M	roxanaS	202cb962ac5977777777152d234b70	4
3	A	admEnzo	202cb962ac59075b964b07152d234b70	2
5	M	alejo	202cb962ac59075b964b07152d234b70	1

2. Conexión a la base de datos:

```
● ● ●

1 #include <QString> //16 bit chars
2 #include <QSqlError>
3 #include <QSqlDatabase>
4 #include <QDebug>
5
6 class DBConexion{
7 private:
8     QString hostName = "localhost";
9     QString dbName = "cc_proyectofinal_22";
10    QString userName = "root";
11    QString password = "";
12
13 public:
14     QSqlDatabase mDatabase;
15     bool connOpen() {
16         mDatabase = QSqlDatabase::addDatabase("QMYSQL");
17         mDatabase.setHostName(hostName);
18         mDatabase.setPort(3306);
19         mDatabase.setDatabaseName(dbName);
20         mDatabase.setUserName(userName);
21         mDatabase.setPassword(password);
22         if (!mDatabase.open()){
23             QString log;
24             QSqlError e = mDatabase.lastError();
25             switch (e.type()) {
26                 case QSqlError::NoError: log += "No error : " + e.text(); break;
27                 case QSqlError::ConnectionError: log += "Connection error : " + e.text(); break;
28                 case QSqlError::StatementError: log += "Statement error : " + e.text(); break;
29                 case QSqlError::TransactionError: log += "Transaction error : " + e.text(); break;
30                 case QSqlError::UnknownError: log += "Unknown error : " + e.text(); break;
31             }
32             qDebug() << "QSqlError: " << log;
33             return false;
34         }
35         return true;
36     }
37 }
```

```
● ● ●

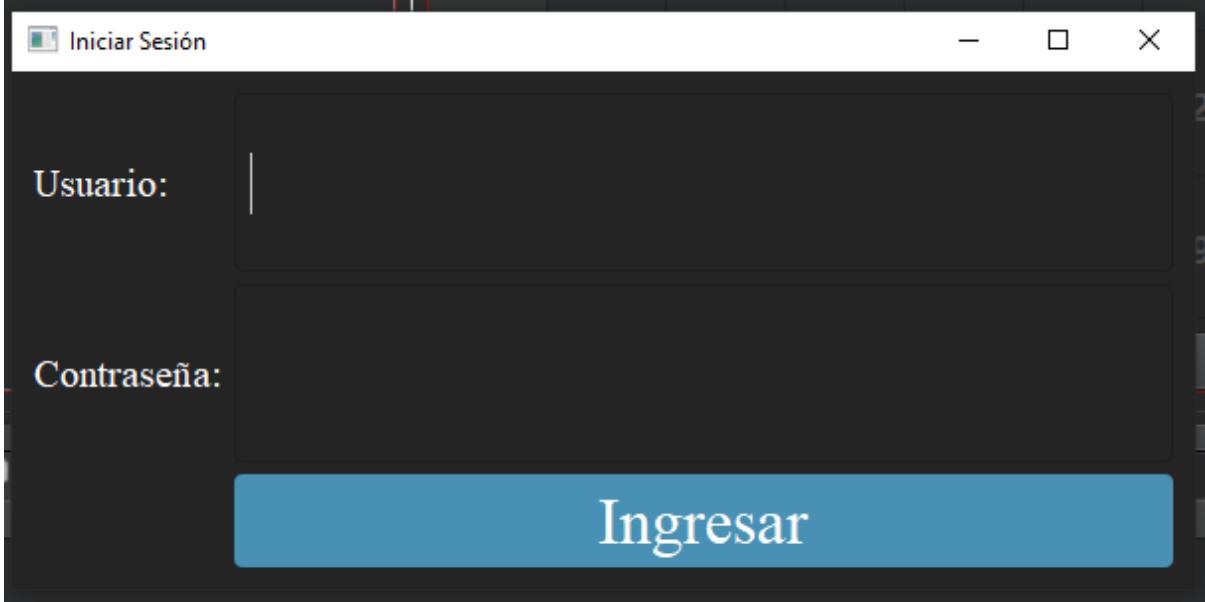
1 void connClose() {
2     mDatabase.close();
3     mDatabase.removeDatabase(QSqlDatabase::defaultConnection);
4     qDebug() << "Database Closed";
5 }
```

3. Main y Login en QT:

Main:

```
1 #include "widget.h"
2 #include "login.h"
3 #include "menudoctor.h"
4 #include "citascalendar.h"
5 #include <QApplication>
6 #include <QDebug>
7 #include <QFile>
8 #include <QString>
9 #include <QDir>
10
11 int main(int argc, char *argv[])
12 {
13     /* Se crea una instancia de la clase QApplication. */
14     QApplication a(argc, argv);
15     /* Se crea un objeto QFile y se establece el path. */
16     QFile stylesheetFile(QDir::currentPath() + "/Diffnes.qss");
17     /* Se abre el archivo en modo solo leer (read only mode). */
18     stylesheetFile.open(QFile::ReadOnly);
19 //    /*Se guarda el contenido del archivo en un QString. */
20     QString ss = QLatin1String(stylesheetFile.readAll());
21     /* Se establece el estilo de hoja para la aplicación. */
22     a.setStyleSheet(ss);
23     /* Se llama al Login para ser mostrado. */
24
25     Login l;
26     l.show();
27     return a.exec();
28 }
```

Vista del login:



Una función hash criptográfica es un algoritmo que toma una cantidad arbitraria de entrada de datos (una credencial) y produce una salida de tamaño fijo de texto cifrado llamado valor hash, o simplemente "hash".

Se incluyó una librería de hashing para encriptar las contraseñas.

```
● ● ●  
1 #include "login.h"  
2 #include "ui_login.h"  
3 #include "menudoctor.h"  
4 #include "menupaciente.h"  
5 #include "vistaadm.h"  
6 #include <QString>  
7 #include <QDebug>  
8 #include <QSqlQuery>  
9 #include <QMMessageBox>  
10 #include <QCryptographicHash>  
11  
12 Login::Login(QWidget *parent) : QWidget(parent), ui(new Ui::Login)  
13 {  
14     ui->setupUi(this);  
15 }  
16  
17 Login::~Login()  
18 {  
19     delete ui;  
20 }  
21
```

Constructor de la clase Login:

Crea el puntero ui.

Destructor de la clase Login:

Elimina el puntero ui.

Función on_btn_login_clicked:

Chequea si el usuario existe en la database, si existe, chequea el rol del usuario y abre la ventana correspondiente.

```
1 void Login::on_btn_login_clicked()
2 {
3     if (!conn.connOpen()) {
4         QMessageBox::warning(this, "Error", "QSqlError: REVISAR LOG");
5         QApplication::quit();
6         //return;
7     }
8
9     QString username = ui->input_username->text();
10    QString pass = QString("%1").arg(QString(QCryptographicHash::hash
11                                (ui->input_password->text().toUtf8(), QCryptographicHash::Md5).toHex()));
12
13    QSqlQuery qry;
14    qry.prepare("SELECT usuario, cargo,idreferencia FROM usuarios WHERE usuario= :usuario AND clave= :clave");
15    qry.bindValue(":usuario", username);
16    qry.bindValue(":clave", pass);
17    if (qry.exec()){
18        if (qry.size() > 0){
19            if(qry.first()){
20                //QString name = qry.value(0).toString();
21                QString role = qry.value(1).toString();
22                QString idreferencia = qry.value(2).toString();
23                conn.connClose();
24                if (role == "M") {
25                    menuDoctor *wdg_mDoctor = new menuDoctor(idreferencia, nullptr);
26                    wdg_mDoctor->setModal(true);
27                    wdg_mDoctor->exec();
28                    this->close();
29                    conn.connClose();
30                } else if (role == "A") {
31                    vistaadm *ventanaadm = new vistaadm();
32                    ventanaadm->show();
33                    this->close();
34                    conn.connClose();
35                } else if (role == "P") {
36                    menupaciente *wdg_mpaciente = new menupaciente(idreferencia, nullptr);
37                    wdg_mpaciente->setModal(true);
38                    wdg_mpaciente->exec();
39                    this->close();
40                    conn.connClose();
41                }
42            }
43        } else {
44            QMessageBox::warning(this, "Error", "No coinciden los credenciales ingresados con"
45                                " ningún registro en la BD");
46        }
47    } else {
48        QMessageBox::information(this, "ERROR", "error en qry.exec()");
49    }
50 }
```

En la línea 31, se añadió la implementación de un Smart Pointer: El puntero QShared pasa entre las diferentes vistas de nuestro proyecto.

```
● ● ●  
1 } else if (role == "A") {  
2         /* Creating a shared pointer to a functorOperaciones object. */  
3         QSharedPointer<functorOperaciones> cont(new functorOperaciones);  
4         vistaadm *ventanaadm = new vistaadm(cont, nullptr);  
5         ventanaadm->show();  
6         this->close();  
7         conn.connClose();  
8 }
```

4. Clase Functor:

Nos permite evitar crear una función para cada clase utilizada en el proyecto sin tener que escribirla en cada clase individualmente.

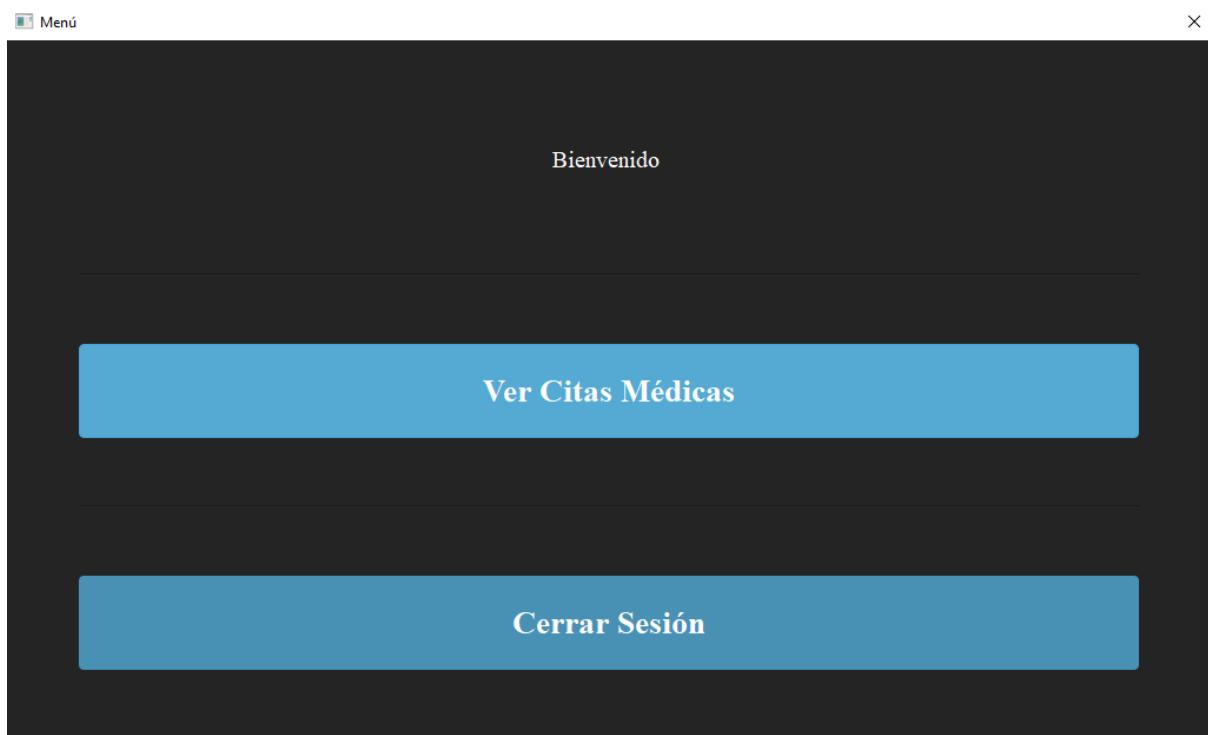
```
● ● ●  
1 class functorOperaciones {  
2 private:  
3     int x=0;  
4 public:  
5     ~functorOperaciones();  
6     void operator()();  
7     int RetornarNumeroOperacionesSesion();  
8 };  
9 
```

```
● ● ●

1 #include "functorOperaciones.h"
2 #include <QDebug>
3
4 functorOperaciones::~functorOperaciones() {
5     qDebug() << "Borrando puntero functor";
6 }
7 void functorOperaciones::operator()(){
8     x++;
9 }
10 int functorOperaciones::RetornarNumeroOperacionesSesion(){
11     return x;
12 }
```

5. Menú Doctor:

Vista del menú del doctor:



Incluye las librerías QDialog, que sirve para tareas de corta duración y breves comunicaciones con el usuario.

Por otro lado, QString guarda un string de 16-bit QChars. ¿Qué es un Qchar? Es un carácter Unicode de 16-bits. En Qt, los caracteres Unicode no tienen marcado ni estructura.

Asimismo, se incluye el header de citascalendar, que trataremos en el siguiente apartado.



Se crea el constructor con el identificador (id) y el objeto de la clase QWidget, que es la clase base de los objetos de interfaz.

```
1  namespace Ui {
2  class menuDoctor;
3  }
4
5  class menuDoctor : public QDialog
6  {
7      Q_OBJECT
8
9  public:
10     explicit menuDoctor(QString idreferencia, QWidget *parent = nullptr);
11     // , const QString& n = "", const QString& r = ""
12     ~menuDoctor();
13
14 private slots:
15     void on_btn_logout_clicked();
16     void on_btn_citasCalendar_clicked();
17
18 private:
19     Ui::menuDoctor *ui;
20     //Usuario *usuario;
21     QString idReferencia;
22     citasCalendar *wdg_citasCal = nullptr;
23 };
24
```

6. Calendario de citas (Archivo citascalendar):

Vista del calendario de citas:

Form

Nombre Medico:	ALEJANDRO SANDRO LINARES SUAREZ			
Especialidad:	PEDIATRÍA			
Hora de la Cita	15:00	Turno:	Tarde	
Estado: En espera				
ID	Nombre del Medic	Especialidad	Hora	Turno
1 14	ALEJANDR...	PEDIATRÍA	15:00:00	Tarde
2 16	MARIA ...	Pediatria	00:00:00	Tarde

← Agosto 2022 →

	dom.	lun.	mar.	mié.	jue.	vie.	sáb.
31	31	1	2	3	4	5	6
32	7	8	9	10	11	12	13
33	14	15	16	17	18	19	20
34	21	22	23	24	25	26	27
35	28	29	30	31	1	2	3
36	4	5	6	7	8	9	10

[Regresar al Menú](#)

Citascalendar.h: Esta clase nos va a permitir ver el calendario de citas de cada médico dependiendo de la fecha seleccionada.

```
1 #include "DBConexion.h"
2 #include <QModelIndex>
3 #include <QWidget>
4 #include <QDate>
5
6 namespace Ui {
7 class citasCalendar;
8 }
9
10 class citasCalendar : public QWidget
11 {
12     Q_OBJECT
13
14 public:
15     explicit citasCalendar(QString idreferencia,QWidget *parent = nullptr);
16     ~citasCalendar();
17
18 public slots:
19     void updateDataCalendar(const QDate &d);
20
21 private slots:
22     void on_pushButton_clicked();
23     void on_tableView_clicked(const QModelIndex &index);
24
25 private:
26     Ui::citasCalendar *ui;
27     QString idReferencia;
28     DBConexion conn;
29 };
30
31 #endif // CITASCALENDAR_H
32
```

Archivo citascalendar.cpp:

La clase QSqlQueryModel proporciona un modelo de datos de solo lectura para conjuntos de resultados de SQL.



```
● ● ●
1 #include "citascalendar.h"
2 #include "ui_citascalendar.h"
3 #include "menudoctor.h"
4 #include <QMessageBox>
5 #include <QSqlQuery>
6 #include <QSqlQueryModel>
7
```

Constructor citasCalendar:

Permite construir las citas con el id, nombre, DNI, teléfono, edad, hora, turno, estado de los pacientes con un médico (con código y fecha que coincidan).

Se crea la interfaz del usuario para la clase.

Se crean nuevas instancias de QSqlQueryModel y QSqlQuery.

Se unen los valores de la tabla con las variables de la clase.

Con model->setQuery(*qry), se establece la query en el modelo.

Se conecta la señal activada (const QDate&) del QCalendarWidget ui->calendarCitas a la ranura updateDataCalendar(const QDate) de la clase citasCalendar.

```
1  citasCalendar::citasCalendar(QString idreferencia,QWidget *parent) :
2      QWidget(parent),
3      ui(new Ui::citasCalendar)
4  {
5      ui->setupUi(this);
6      idReferencia = idreferencia;
7      if (!conn.connOpen()) {
8          QMessageBox::warning(this,"Error", "QSqlError: REVISAR LOG");
9          QApplication::quit();
10         //return;
11     }
12     //ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
13     QSqlQueryModel* model = new QSqlQueryModel;
14     QSqlQuery * qry = new QSqlQuery(conn.mDatabase);
15
16     qry->prepare("SELECT idc, nombre, DNI, telefono, edad, hora, turno, estado "
17                   "FROM pacientesview WHERE codigomed=:codmedico AND fecha=:fecha");
18     qry->bindValue(":codmedico", idReferencia); /* completar despues medico activo */
19     qry->bindValue(":fecha", QDate::currentDate());
20     qry->exec();
21     model->setQuery(*qry);
22     model->setHeaderData(0, Qt::Horizontal, tr("ID"));
23     model->setHeaderData(1, Qt::Horizontal, tr("Nombre del Paciente"));
24     model->setHeaderData(2, Qt::Horizontal, tr("DNI"));
25     model->setHeaderData(3, Qt::Horizontal, tr("Telefono"));
26     model->setHeaderData(4, Qt::Horizontal, tr("Edad"));
27     model->setHeaderData(5, Qt::Horizontal, tr("Hora"));
28     model->setHeaderData(6, Qt::Horizontal, tr("Turno"));
29     model->setHeaderData(7, Qt::Horizontal, tr("Estado"));
30     ui->tableView->setModel(model);
31
32     connect(ui->calendarCitas, SIGNAL(activated(const QDate&)),
33             this, SLOT(updateDataCalendar(const QDate)));
34 }
35
```

De esta manera generamos la vista en pantalla para las citas correspondientes al día indicado según la señal enviada del calendario. Se pasa el dato de la fecha y se procesa en el slot para generar el cuadro de citas.

Función updateDataCalendar:

Actualiza las citas.

```
1 void citasCalendar::updateDataCalendar(const QDate &d) {
2     ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
3     QSqlQueryModel* model = new QSqlQueryModel;
4     QSqlQuery *qry = new QSqlQuery(conn.mDatabase);
5     qry->prepare("SELECT idc, nombre, DNI, telefono, edad, hora, turno, estado "
6                   "FROM pacientesview WHERE codigomed=:codmedico and fecha=:fecha");
7     qry->bindValue(":codmedico", idReferencia); /* completar despues medico activo */
8     qry->bindValue(":fecha", d.toString("yyyy-MM-dd")); //ui->calendar->selectedDate());
9     qry->exec();
10
11    model->setQuery(*qry);
12    model->setHeaderData(0, Qt::Horizontal, tr("ID"));
13    model->setHeaderData(1, Qt::Horizontal, tr("Nombre del Paciente"));
14    model->setHeaderData(2, Qt::Horizontal, tr("DNI"));
15    model->setHeaderData(3, Qt::Horizontal, tr("Telefono"));
16    model->setHeaderData(4, Qt::Horizontal, tr("Edad"));
17    model->setHeaderData(5, Qt::Horizontal, tr("Hora"));
18    model->setHeaderData(6, Qt::Horizontal, tr("Turno"));
19    model->setHeaderData(7, Qt::Horizontal, tr("Estado"));
20
21    ui->tableView->setModel(model);
22    ui->tableView->show();
23
24 }
25
```

Función on_pushButton_clicked:

Regresa al menú del doctor.

```
1 void citasCalendar::on_pushButton_clicked()
2 {
3     this->close();
4     conn.connClose();
5     menuDoctor * wdg_mDocket = new menuDoctor(idReferencia);
6     wdg_mDocket->setModal(true);
7     wdg_mDocket->exec();
8 }
9
```

Función `on_tableView_clicked`:

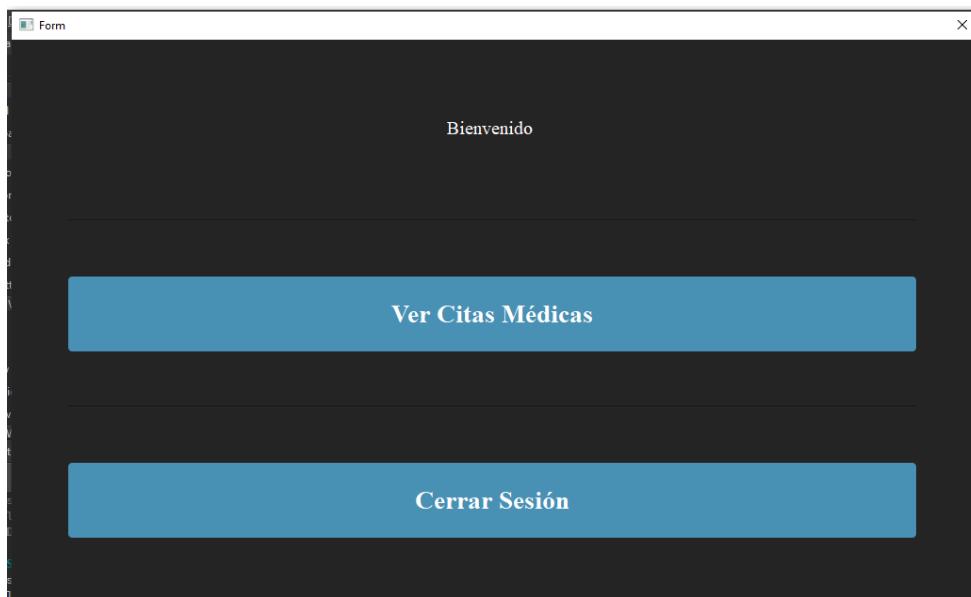
Genera la tabla con la información de la cita: nombre, dni, telefono, edad, hora. Cuando se hace clic en una fila, los datos de la fila se almacenan en una lista y luego se muestran en los cuadros de texto.



```
1 void citasCalendar::on_tableView_clicked(const QModelIndex &index)
2 {
3     int nfila = index.row();
4     QList<QString> data;
5     for (int i = 1; i < 8; i++){
6         QString val = ui->tableView->model()->data(
7             ui->tableView->model()->index(nfila, i)
8             ).toString();
9         data.append(val);
10    }
11    ui->txt_nombrePac->setText(data[0]);
12    ui->txt_dni->setText(data[1]);
13    ui->txt_telef->setText(data[2]);
14    ui->txt_edad->setText(data[3]);
15    QTime t = QTime::fromString(data[4]);
16    ui->time_horaCita->setTime(t);
17    ui->txt_turno->setText(data[5]);
18    ui->txt_estado->setText(data[6]);
19 }
20
```

7. Menú del paciente:

Vista del menú paciente:



El menú del paciente permite la generación de citas para el beneficio del paciente.

Menu paciente.h:

```
1 #include <QDialog>
2 #include <QString>
3 #include "citascalendarpac.h"
4
```

```
1 namespace Ui {
2 class menupaciente;
3 }
4
5 class menupaciente : public QDialog
6 {
7     Q_OBJECT
8
9 public:
10     explicit menupaciente(QString idreferencia, QWidget *parent = nullptr);
11     ~menupaciente();
12
13 private slots:
14     void on_btn_logout_clicked();
15     void on_btn_citasCalendar_clicked();
16
17 private:
18     Ui::menupaciente *ui;
19     //Usuario *usuario;
20     QString idReferencia;
21     citascalendarpac *wdg_citascalpac = nullptr;
22 };
23
```

Menu paciente.cpp:

QDebug es bastante similar a std::cout en la biblioteca estándar, pero la ventaja de usar qDebug es que, dado que es parte de Qt, admite clases Qt listas para usar y puede generar su valor sin necesidad de cualquier conversión.



```
1 #include "menupaciente.h"
2 #include "ui_menupaciente.h"
3 #include "login.h"
4 #include "citascalendarpac.h"
5 #include <QFont>
6 #include <QDebug>
7 #include <QMessageBox>
8 #include <QSqlError>
9 #include <QSharedPointer>
10
```

¿Por qué usamos QWidget *parent en el constructor?

La organización en forma de árbol de QWidgets (y de hecho de cualquier QObject) es parte de la estrategia de administración de memoria utilizada dentro de Qt-Framework. Asignar un QObject a un parent significa que la propiedad del objeto secundario se transfiere al objeto principal. Si se elimina el parent, también se eliminan todos sus hijos.

```
1
2 menupaciente::menupaciente(QString idreferencia, QWidget *parent) :
3     QDialog(parent),
4     ui(new Ui::menupaciente)
5 {
6     ui->setupUi(this);
7     ui->txt_bienvenida->setText("Bienvenido ");
8     //usuario->getNombre()
9     idReferencia = idreferencia;
10    connect(ui->btn_citasCalendar,SIGNAL(clicked()),
11             this, SLOT(on_btn_citasCalendar_clicked()));
12 }
13
```

Destructor:

```
1 menupaciente::~menupaciente()
2 {
3     delete ui;
4     delete wdg_citascalpac;
5 }
6
```

Función `btn_citasCalendar_clicked` que dirige al menú calendario de citas:

Crea un objeto `citascalendarpac`, el cual permite visualizar las citas del paciente individualmente.

```

1 void menupaciente::on_btn_citasCalendar_clicked() {
2     if(wdg_citascalpac==nullptr)
3         wdg_citascalpac = new citascalendarpac(idReferencia); /* aqui se llama 2 veces */
4     wdg_citascalpac->show();
5     this->close();
6 }

```

Función que nos permite salir del menú:

```

1 void menupaciente::on_btn_logout_clicked() {
2     QMessageBox::StandardButton out = QMessageBox::question(this, "Exit", "Cerrando Sesión...", QMessageBox::Cancel | QMessageBox::Ok);
3     if (out == QMessageBox::Ok) {
4         this->close();
5         QApplication::quit();
6         //return;
7     } else {}
8 }
9
10

```

8. Archivo para mostrar las citas del paciente (citascalendarpac.h y citascalendarpac.cpp): Vista del calendario de citas del paciente:

Calendario de Citas

Nombre Paciente:		JUANA RODRIGUEZ					
DNI:	11111111	Telefono:	999999999	Edad:	35		
Hora de la Cita	10:00	Turno:	Diurno				
Estado: En Espera							
ID	Nombre del Paciente	DNI	Telefono	Edad			
1 4	SOLEDAD ...	11111111	999999999	20			
2 5	JUANA ...	11111111	999999999	35			

Agosto 2022

	dom.	lun.	mar.	mié.	jue.	vie.	sáb.
31	31	1	2	3	4	5	6
32	7	8	9	10	11	12	13
33	14	15	16	17	18	19	20
34	21	22	23	24	25	26	27
35	28	29	30	31	1	2	3
36	4	5	6	7	8	9	10

Regresar al Menú

citascalendarpac.h:

```
1 #include "DBConexion.h"
2 #include <QModelIndex>
3 #include <QWidget>
4 #include <QDate>
5
6 namespace Ui {
7 class citascalendarpac;
8 }
9
10 class citascalendarpac : public QWidget
11 {
12     Q_OBJECT
13
14 public:
15     explicit citascalendarpac(QString idreferencia, QWidget *parent = nullptr);
16     ~citascalendarpac();
17
18 public slots:
19     void updateDataCalendar(const QDate &d);
20
21 private slots:
22     void on_pushButton_6_clicked();
23     void on_tableView_6_clicked(const QModelIndex &index);
24
25 private:
26     Ui::citascalendarpac *ui;
27     QString idReferencia;
28     DBConexion conn;
29 };
30
```

citascalendarpac.cpp:

Se incluyeron las siguientes librerías, son las mismas que se incluyeron en citascalendar.cpp:

```
1 #include "citascalendarpac.h"
2 #include "ui_citascalendarpac.h"
3 #include "menupaciente.h"
4 #include <QMessageBox>
5 #include <QSqlQuery>
6 #include <QSqlQueryModel>
7
```

```
1  citascalendarpac::citascalendarpac(QString idreferencia, QWidget *parent) :  
2      QWidget(parent),  
3      ui(new Ui::citascalendarpac)  
4  {  
5      ui->setupUi(this);  
6      idReferencia = idreferencia;  
7      if (!conn.connOpen()) {  
8          QMessageBox::warning(this,"Error", "QSqlError: REVISAR LOG");  
9          QApplication::quit();  
10     }  
11     QSqlQueryModel* model = new QSqlQueryModel;  
12     QSqlQuery * qry = new QSqlQuery(conn.mDatabase);  
13  
14     qry->prepare("SELECT idc, nombre, espec, hora, turno, estado "  
15                  "FROM medicosview WHERE codigopac=:codpaciente AND fecha=:fecha");  
16     qry->bindValue(":codpaciente", idReferencia); /* completar despues medico activo */  
17     qry->bindValue(":fecha", QDate::currentDate());  
18     qry->exec();  
19     model->setQuery(*qry);  
20     model->setHeaderData(0, Qt::Horizontal, tr("ID"));  
21     model->setHeaderData(1, Qt::Horizontal, tr("Nombre del Medico"));  
22     model->setHeaderData(2, Qt::Horizontal, tr("Especialidad"));  
23     model->setHeaderData(3, Qt::Horizontal, tr("Hora"));  
24     model->setHeaderData(4, Qt::Horizontal, tr("Turno"));  
25     model->setHeaderData(5, Qt::Horizontal, tr("Estado"));  
26     ui->tableView_6->setModel(model);  
27  
28     connect(ui->calendarCitas_6, SIGNAL(activated(const QDate&)),  
29               this, SLOT(updateDataCalendar(const QDate)));  
30 }  
31  
32 citascalendarpac::~citascalendarpac()  
33 {  
34     delete ui;  
35     conn.connClose();  
36 }  
37
```

Función para actualizar el calendario de citas del paciente:

Actualiza los datos en una vista de tabla según la fecha seleccionada en el widget de calendario.

```
1 void citascalendarpac::updateDataCalendar(const QDate &d) {
2     ui->tableView_6->setEditTriggers(QAbstractItemView::NoEditTriggers);
3     QSqlQueryModel* model = new QSqlQueryModel;
4     QSqlQuery *qry = new QSqlQuery(conn.mDatabase);
5     qry->prepare("SELECT idc, nombre, espec, hora, turno, estado "
6                 "FROM medicosview WHERE codigopac=:codpaciente and fecha=:fecha");
7     qry->bindValue(":codpaciente", idReferencia); /* completar despues medico activo */
8     qry->bindValue(":fecha", d.toString("yyyy-MM-dd")); //ui->calendar->selectedDate());
9     qry->exec();
10
11    model->setQuery(*qry);
12    model->setHeaderData(0, Qt::Horizontal, tr("ID"));
13    model->setHeaderData(1, Qt::Horizontal, tr("Nombre del Medico"));
14    model->setHeaderData(2, Qt::Horizontal, tr("Especialidad"));
15    model->setHeaderData(3, Qt::Horizontal, tr("Hora"));
16    model->setHeaderData(4, Qt::Horizontal, tr("Turno"));
17    model->setHeaderData(5, Qt::Horizontal, tr("Estado"));
18
19    ui->tableView_6->setModel(model);
20    ui->tableView_6->show();
21
22 }
23
```

Función para volver al menú del paciente:

```
1
2 void citascalendarpac::on_pushButton_6_clicked()
3 {
4     this->close();
5     conn.connClose();
6     menupaciente * wdg_mpaciente = new menupaciente(idReferencia);
7     wdg_mpaciente->setModal(true);
8     wdg_mpaciente->exec();
9 }
10
```

Función para mostrar la información del calendario de citas:

```

1 void citascalendarpac::on_tableView_6_clicked(const QModelIndex &index)
2 {
3     /* Getting the row number of the clicked item. */
4     int nfila = index.row();
5     /* A list of strings. */
6     QList<QString> data;
7     /* It's getting the data from the tableView_6 and putting it in a list of strings. */
8     for (int i = 1; i < 6; i++){
9         QString val = ui->tableView_6->model()->data(
10                 ui->tableView_6->model()->index(nfila, i)
11                 ).toString();
12         data.append(val);
13     }
14     ui->txt_nombremed_6->setText(data[0]);
15     ui->txt_especialidad_6->setText(data[1]);
16     QTime t = QTime::fromString(data[2]);
17     ui->time_horaCita_6->setTime(t);
18     ui->txt_turno_6->setText(data[3]);
19     ui->txt_estado_6->setText(data[4]);
20 }
21

```

9. Menú del administrador (vistaadm):

Vista del administrador:



Se implementó haciendo uso de functores y smart pointers.
vistaadm.h:

```
1 #include <QWidget>
2 #include <functorOperaciones.h>
3 #include <QSharedPointer>
4
5 namespace Ui {
6 class vistaadm;
7 }
8
9 class vistaadm : public QWidget
10 {
11     Q_OBJECT
12
13 public:
14     explicit vistaadm(QSharedPointer<functorOperaciones> c, QWidget *parent = nullptr);
15     ~vistaadm();
16 public slots:
17     void on_btn_agregarpac_clicked();
18     void on_btn_agregarmed_clicked();
19     void on_btn_agregarcita_clicked();
20     void on_btn_cerrar_clicked();
21
22 private:
23     Ui::vistaadm *ui;
24     QSharedPointer<functorOperaciones> cont;
25 };
```

vistaadm.cpp:

```
1 #include "vistaadm.h"
2 #include "ui_vistaadm.h"
3 #include "newmed.h"
4 #include "newpac.h"
5 #include "login.h"
6 #include "dialog_agregar_citas.h"
7 #include <QMessageBox>
8 #include <QScopedPointer>
9
10 vistaadm::vistaadm(QSharedPointer<functorOperaciones> c, QWidget *parent)
11     : QWidget(parent), cont(c),
12       ui(new Ui::vistaadm)
13 {
14     ui->setupUi(this);
15 }
16
17 vistaadm::~vistaadm()
18 {
19     delete ui;
20 }
21
22 /**
23  *Función
```

Funciones para crear un nuevo registro de médicos, pacientes o citas:

```
1 /**
2  *Función que cierra la vista del admin y abre la ventana de Nuevo Paciente
3  */
4
5 void vistaadm::on_btn_agregarpac_clicked()
6 {
7     newpac *ventanapac = new newpac(cont, nullptr);
8     ventanapac->show();
9     this->close();
10    delete this;
11 }
12 /**
13  *Función que cierra la vista del admin y abre la ventana de Nuevo Médico
14  */
15
16 void vistaadm::on_btn_agregarmed_clicked()
17 {
18     newmed *ventanamed = new newmed(cont, nullptr);
19     ventanamed->show();
20     this->close();
21     delete this;
22 }
```

```
 1  /**
 2   *Función que cierra la vista del admin y abre la ventana de Agregar Citas
 3   */
 4
 5 void vistaadm::on_btn_agregarcita_clicked()
 6 {
 7     Dialog_agregar_citas *ventanacita = new Dialog_agregar_citas(cont, nullptr);
 8     ventanacita->show();
 9     this->close();
10     delete this;
11 }
12
```

Función para cerrar el menú del admin (vistaadm):

```
 1 void vistaadm::on_btn_cerrar_clicked()
 2 {
 3     /* Un message box que pregunta al usuario si desea salir. */
 4     QMessageBox::StandardButton out =
 5     QMessageBox::question(this, "Exit", "Cerrando Sesión..." +
 6     "\nOperaciones realizadas: " +
 7     + QVariant(cont->RetornarNumeroOperacionesSesion()).toString(),
 8     QMessageBox::Cancel | QMessageBox::Ok);
 9
10     if (out == QMessageBox::Ok) {
11         Login* ventanalog = new Login();
12         ventanalog->show();
13         this->close();
14         delete this;
15     } else {}
16
17 }
```

10. Clase para establecer un nuevo paciente (newpac):

Es importante decir “establecer”, puesto que otras clases (mostradas a continuación) se encargan de crear desde cero al paciente y al médico.

```
1 #include <QWidget>
2 #include <QtSql>
3 #include "ui_newpac.h"
4 #include "DBConexion.h"
5 #include <functorOperaciones.h>
6 #include <QSharedPointer>
7
8 namespace Ui {
9 class newpac;
10 }
11
12 class newpac : public QWidget
13 {
14     Q_OBJECT
15
16 public:
17     explicit newpac(QSharedPointer<functorOperaciones> c, QWidget *parent = nullptr);
18     ~newpac();
19
20 private slots:
21     void on_btn_ingresar_clicked();
22     void on_btn_volver_clicked();
23
24 private:
25     Ui::newpac *ui;
26     DBConexion db;
27     QSharedPointer<functorOperaciones> cont;
28 }
```

Se utiliza un `QSharedPointer` en el constructor.

Se utiliza un functor Operaciones en el constructor.

Se crea un nuevo objeto de tipo `crearpac` y se le asigna a un puntero de tipo `crearcargo`.

```
 1 #include "newpac.h"
 2 #include <QString>
 3 #include <QDebug>
 4 #include <QMessageBox>
 5 #include <QSqlError>
 6 #include <QCryptographicHash>
 7 #include "vistaadm.h"
 8 #include "crearpac.h"
 9
10 newpac::newpac(functorOperaciones* c, QWidget *parent) : QWidget(parent), cont(c), ui(new Ui::newpac)
11 {
12     ui->setupUi(this);
13 }
14
15 newpac::~newpac()
16 {
17     delete ui;
18 }
19
20 void newpac::on_btn_ingresar_clicked()
21 {
22     crearcargo* nuevopac = new crearpac(cont,this);
23     nuevopac->set_nombre((ui->input_nombre->text()).toUpper());
24     nuevopac->set_edad(ui->input_edad->value());
25     nuevopac->set_dni(ui->input_DNI->text());
26     nuevopac->set_direccion(ui->input_dir->text());
27     nuevopac->set_telefono(ui->input_tel->text());
28     nuevopac->set_email(ui->input_email->text());
29     nuevopac->set_ciudad(ui->input_ciudad->text());
30     nuevopac->set_clave(ui->input_clave->text());
31
32     nuevopac->agregar();
33     delete nuevopac;
34 }
35
36 void newpac::on_btn_volver_clicked()
37 {
38     vistaadm *ventanaadm = new vistaadm(cont,nullptr);
39     ventanaadm->show();
40     this->close();
41 }
42
```

11. Clase crear cargo:

Esta clase es utilizada para la implementación del **patrón de diseño template** que nos permite simplificar la definición de los métodos usados tanto para generar tanto el constructor de médicos como de pacientes.

```
● ● ●

1  namespace std {
2  class crearcargo;
3  }
4
5  class crearcargo {
6  public:
7      void agregar() {
8          if (validacion() == false) {
9              qDebug() << "Aqui estoy";
10             return;
11         } else {}
12         insertar_en_cargo();
13         insertar_en_usuarios(obtener_codid());
14     }
15     void set_nombre(QString n) { nombre = n; }
16     void set_edad(int e) { edad = e; }
17     void set_dni(QString d) { dni = d; }
18     void set_direccion(QString dir) { direccion = dir; }
19     void set_telefono(QString tel) { telefono = tel; }
20     void set_email(QString em) { email = em; }
21     void set_ciudad(QString ciu) { ciudad = ciu; }
22     void set_especialidad(QString esp) { especialidad = esp; }
23     void set_turno(QChar t) { turno = t; }
24     void set_clave(QString cl) { clave = cl; }
25
26     virtual ~crearcargo() {}
27 protected:
28     QString nombre;
29     int edad;
30     QString dni;
31     QString direccion;
32     QString telefono;
33     QString email;
34     QString ciudad;
35     QString especialidad;
36     QChar turno;
37     QString clave;
38     DBConexion db;
39     functorOperaciones* cont;
40
41     virtual bool validacion() = 0;
42     virtual void insertar_en_cargo() = 0;
43     virtual void insertar_en_usuarios(QString) = 0;
44     virtual QString obtener_codid() = 0;
45 }
```

12. Clase crear médico (crearmed):

Crearmed.h:

```
● ● ●

1 #include "crearcargo.h"
2
3 class crearmed : public crearcargo
4 {
5 public:
6     crearmed(functorOperaciones*, QWidget* );
7     ~crearmed();
8 protected:
9     bool validacion() override;
10    void insertar_en_cargo() override;
11    void insertar_en_usuarios(QString) override;
12    QString obtener_codid() override;
13 private:
14     DBConexion db;
15     QWidget* ventana;
16 };
```

Crearmed.cpp:

```
● ● ●

1 #include "crearmed.h"
2
3 crearmed::crearmed(functorOperaciones* c, QWidget *parent) : ventana(parent) { cont = c; }
4
5 crearmed::~crearmed()
6 {
7     delete ventana;
8 }
```

Validación:

```
1 bool crearmed :: validacion() {
2     if (nombre.isEmpty() || edad == 0 || dni.isEmpty() || direccion.isEmpty() ||
3         especialidad.isEmpty() || turno.isNull() || telefono.isEmpty() || email.isEmpty()
4         || ciudad.isEmpty()) {
5         QMessageBox::warning(ventana,"Error", "Debe llenar todos los campos!");
6         return false;
7     }
8     return true;
9 }
10
```

Función para insertar médico en tabla de médicos:

```
1 void crearmed :: insertar_en_cargo() {
2     if (!db.connOpen()) {
3         QMessageBox::warning(ventana,"Error", "QSqlError: REVISAR LOG");
4         //QApplication::quit();
5     }
6     QSqlQuery qry;
7     qry.prepare("INSERT INTO `medico`(`nombre`, `edad`, `DNI`, `especialidad`, `direccion`, `telefono`, `email`, `ciudad`, `turno`) "
8                 "VALUES ('" + nombre + "','" +
9                     + QVariant(edad).toString() + "','" +
10                    + dni + "','" +
11                    + especialidad + "','" +
12                    + direccion + "','" +
13                    + telefono + "','" +
14                    + email + "','" +
15                    + ciudad + "','" +
16                    + turno + "')");
17
18     if (!qry.exec()) {
19         QMessageBox::information(ventana,"ERROR","error en qry.exec() insertar medico");
20         return;
21     }
22
23
24     db.connClose();
25 }
```

Función para obtener el código de un médico:

```

1  QString crearmed :: obtener_coid() {
2      if (!db.connOpen()) {
3          QMessageBox::warning(ventana,"Error", "QSqlError: REVISAR LOG");
4          //QApplication::quit();
5      }
6
7      QSqlQuery qry;
8
9      qry.prepare("SELECT codigomed FROM medico WHERE DNI= :dni AND especialidad= :especialidad");
10     qry.bindValue(":dni", dni);
11     qry.bindValue(":especialidad",especialidad);
12
13     QString codmed;
14     if (qry.exec()) {
15         if (qry.size() == 1) {
16             if(qry.first()){
17                 codmed = qry.value(0).toString();
18             }
19         } else {
20             QMessageBox::warning(ventana,"ERROR", "Datos duplicados");
21         }
22     }
23
24     db.connClose();
25     return codmed;
26 }
27

```

Función para insertar a un médico en los usuarios del sistema:

```

1
2  void crearmed :: insertar_en_usuarios(QString codmed) {
3      if (!db.connOpen()) {
4          QMessageBox::warning(ventana,"Error", "QSqlError: REVISAR LOG");
5          //QApplication::quit();
6      }
7
8      QString usuario = "med" + dni;
9
10     QSqlQuery qry;
11     qry.prepare("INSERT INTO `usuarios`(`cargo`, `usuario`, `clave`, `idreferencia`) "
12                "VALUES ('M','" + usuario + "','" + md5('' + clave + '') + "','" + codmed + ')");
13
14     if (qry.exec()) {
15         QMessageBox::information(ventana,"CORRECTO", "El medico ha sido ingresado\n"
16                                "USUARIO: " + usuario + "\nCLAVE: " + clave);
17         (*cont)();
18     } else {
19         QMessageBox::information(ventana,"ERROR", "Error al insertar usuario()");
20     }
21
22     db.connClose();
23 }

```

13. Clase crear paciente(crearpac):

Crearpac.h:

```
1 #include "crearcargo.h"
2
3 class crearpac : public crearcargo
4 {
5 public:
6     crearpac(functorOperaciones*, QWidget*);
7     ~crearpac();
8 protected:
9     bool validacion() override;
10    void insertar_en_cargo() override;
11    void insertar_en_usuarios(QString) override;
12    QString obtener_codid() override;
13 private:
14     DBConexion db;
15     QWidget* ventana;
16 };
```

Crearpac.cpp:

Constructor, destructor y validación del paciente:

```

1 #include "crearpac.h"
2
3 crearpac::crearpac(functorOperaciones* c, QWidget *parent) : ventana(parent) { cont = c; }
4
5 crearpac::~crearpac()
6 {
7     delete ventana;
8 }
9
10 bool crearpac :: validacion() {
11     if (nombre.isEmpty() || edad == 0 || dni.isEmpty() || direccion.isEmpty() ||
12         telefono.isEmpty() || email.isEmpty() || ciudad.isEmpty()) {
13         QMessageBox::warning(ventana,"Error", "Debe llenar todos los campos!");
14         return false;
15     }
16     return true;
17 }
18

```

Insertar al paciente en la tabla de pacientes:

```

1
2 void crearpac :: insertar_en_cargo() {
3     if (!db.connOpen()) {
4         QMessageBox::warning(ventana,"Error", "QSqlError: REVISAR LOG");
5         //QApplication::quit();
6     }
7
8     QSqlQuery qry;
9     qry.prepare("INSERT INTO `paciente`(`nombre`, `edad`, `DNI`, `direccion`,
10                 `telefono`, `email`, `ciudad`) "
11                 "VALUES ('" + nombre + "','" +
12                     + QVariant(edad).toString() + "','" +
13                     + dni + "','" +
14                     + direccion + "','" +
15                     + telefono + "','" +
16                     + email + "','" +
17                     + ciudad + "')");
18
19     if (!qry.exec()) {
20         QMessageBox::information(ventana,"ERROR","error en qry.exec() insertar paciente");
21         //QApplication::quit();
22     }
23
24     db.connClose();
25 }
26

```

Obtener el código del paciente:

```
1  QString crearpac :: obtener_codid() {
2      if (!db.connOpen()) {
3          QMessageBox::warning(ventana,"Error", "QSqlError: REVISAR LOG");
4          //QApplication::quit();
5      }
6
7
8      QSqlQuery qry;
9      qry.prepare("SELECT codigopac FROM paciente WHERE DNI= :dni");
10     qry.bindValue(":dni", dni);
11
12     QString codpac;
13     if (qry.exec()) {
14         if (qry.size() == 1) {
15             if(qry.first()){
16                 codpac = qry.value(0).toString();
17             }
18         } else {
19             QMessageBox::warning(ventana,"ERROR", "Datos duplicados");
20             //QApplication::quit();
21         }
22     }
23
24     db.connClose();
25     return codpac;
26 }
27
```

Insertar al paciente en la tabla de usuarios:

```

1
2 void clearpac :: insertar_en_usuarios(QString codpac) {
3     if (!db.connOpen()) {
4         QMessageBox::warning(ventana,"Error", "QSqlError: REVISAR LOG");
5         QApplication::quit();
6     }
7
8     QString usuario = "pac" + dni;
9
10    QSqlQuery qry;
11    qry.prepare("INSERT INTO `usuarios`(`cargo`, `usuario`, `clave`, `idreferencia`) "
12                "VALUES ('P','" + usuario + "','" + md5('" + clave + "') + "','" + codpac + "')");
13
14    if (qry.exec()) {
15        QMessageBox::information(ventana,"CORRECTO","El paciente ha sido ingresado\n"
16                               "USUARIO: " + usuario + "\nCLAVE: " + clave);
17        (*cont)();
18    } else {
19        QMessageBox::information(ventana,"ERROR","Error al insertar usuario()");
20    }
21
22    db.connClose();
23 }
24

```

Así como el menú del administrador (vistaadm) nos permite crear un nuevo paciente o médico, también podemos agregar una cita:

14. Menú para agregar citas:

Dialog_agregar_citas

X

AGREGAR CITA

USUARIO DEL PACIENTE:

(Empty input field)

ESPECIALIDAD:

(Empty input field)

TURNO:

- Mañana
- Tarde

FECHA:

1/01/2000

VOLVER **AGREGAR**

Archivo dialog_agregar_citas.cpp:

```
1
2 #include <QDialog>
3 #include <QtSql>
4 #include <QSqlDatabase>
5 #include <QDate>
6 #include "ui_dialog_agregar_citas.h"
7 #include "DBConexion.h"
8 #include "functorOperaciones.h"
9 #include <QSharedPointer>
10
11 namespace Ui {
12 class Dialog_agregar_citas;
13 }
14
15 class Dialog_agregar_citas : public QDialog
16 {
17     Q_OBJECT
18
19 public:
20     Dialog_agregar_citas(QSharedPointer<functorOperaciones> c, QWidget *parent = nullptr);
21     ~Dialog_agregar_citas();
22
23 private slots:
24     void on_volver_clicked();
25     void on_agregar_clicked();
26
27 private:
28     Ui::Dialog_agregar_citas *ui;
29     QSqlDatabase mDatabase;
30     DBConexion db;
31     QSharedPointer<functorOperaciones> cont;
32 };
33
```

Constructor Dialog_agregar_citas Destructor Dialog_agregar_citas

Función on_agregar_clicked:

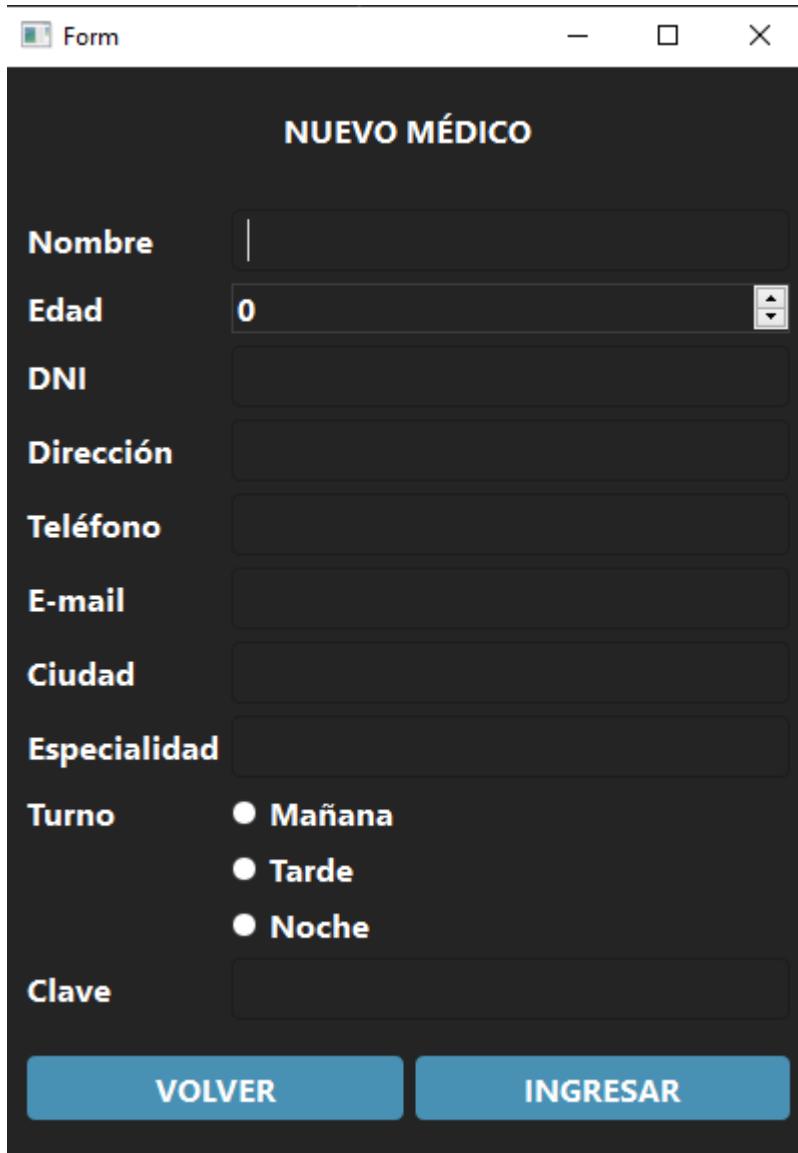
Toma el input del usuario, chequea si es válido. Si lo es, lo agrega a la base de datos.

Función on_volver:clicked:

Cuando el usuario hace click en el botón VOLVER, el programa crea un nuevo objeto-vistadm y lo muestra, luego cierra la ventana.

Archivo newmed.cpp

Vista:



The screenshot shows a Windows-style application window titled "NUEVO MÉDICO". The form contains the following fields:

- Nombre:** Text input field.
- Edad:** Text input field containing "0", with up/down arrow buttons to its right.
- DNI:** Text input field.
- Dirección:** Text input field.
- Teléfono:** Text input field.
- E-mail:** Text input field.
- Ciudad:** Text input field.
- Especialidad:** Text input field.
- Turno:** Radio button group with three options: **Mañana**, **Tarde**, and **Noche**. The **Noche** option is selected.
- Clave:** Text input field.

At the bottom are two buttons: **VOLVER** (left) and **INGRESAR** (right).

Constructor newmed:

Permite crear la ventana mediante la función `setupUi(this)`
Esta función es generada por QT Designer.

Destructor newmed.

Función `on_button_ingresar_clicked:`

Inserta una nueva fila en la tabla médico y luego obtiene la clave primaria de esa fila y la inserta en la tabla usuarios con el usuario y la clave.

La consulta retorna el número de filas afectadas por la consulta(query).

Clase Nuevo paciente:

Vista:

Form - □ X

NUEVO PACIENTE

Nombre	<input type="text"/>
Edad	<input type="text" value="0"/> 
DNI	<input type="text"/>
Dirección	<input type="text"/>
Teléfono	<input type="text"/>
E-mail	<input type="text"/>
Ciudad	<input type="text"/>
Clave	<input type="text"/>

VOLVER **INGRESAR**

Archivo newpac.h (Nuevo paciente):

```
1 #include <QWidget>
2 #include <QtSql>
3 #include "ui_newpac.h"
4 #include "DBConexion.h"
5 #include <functorOperaciones.h>
6 #include <QSharedPointer>
7
8 namespace Ui {
9 class newpac;
10 }
11
12 class newpac : public QWidget
13 {
14     Q_OBJECT
15
16 public:
17     explicit newpac(QSharedPointer<functorOperaciones> c, QWidget *parent = nullptr);
18     ~newpac();
19
20 private slots:
21     void on_btn_ingresar_clicked();
22     void on_btn_volver_clicked();
23
24 private:
25     Ui::newpac *ui;
26     DBConexion db;
27     QSharedPointer<functorOperaciones> cont;
28 }
```

Constructor newpac:

Permite crear la ventana mediante la función setupUi(this)

Esta función es generada por QT Designer.

Destructor newpac.

Función on_button_ingresar:clicked:

Inserta una nueva fila en la tabla paciente y luego obtiene la clave primaria de esa fila (id).

Archivo newpac.cpp (Nuevo paciente):

```
1 #include "newpac.h"
2 #include <QString>
3 #include <QDebug>
4 #include <QMessageBox>
5 #include <QSqlError>
6 #include <QCryptographicHash>
7 #include "vistaadm.h"
8 #include "crearpac.h"
9 #include <QScopedPointer>
10
11 newpac::newpac(QSharedPointer<functorOperaciones> c, QWidget *parent) : QWidget(parent), cont(c), ui(new Ui::newpac)
12 {
13     ui->setupUi(this);
14 }
15
16 newpac::~newpac()
17 {
18     delete ui;
19 }
20
21 void newpac::on_btn_ingresar_clicked()
22 {
23     /* Creating a new object of type crearpac and passing it to the constructor of crearcargo. */
24     /* A smart pointer that deletes the object when it goes out of scope. */
25     QScopedPointer<crearcargo> nuevopac( new crearpac(cont,this) );
26     nuevopac->set_nombre(ui->input_nombre->text().toUpper());
27     nuevopac->set_edad(ui->input_edad->value());
28     nuevopac->set_dni(ui->input_DNI->text());
29     nuevopac->set_direccion(ui->input_dir->text());
30     nuevopac->set_telefono(ui->input_tel->text());
31     nuevopac->set_email(ui->input_email->text());
32     nuevopac->set_ciudad(ui->input_ciudad->text());
33     nuevopac->set_clave(ui->input_clave->text());
34     nuevopac->agregar();
35 }
36
37 void newpac::on_btn_volver_clicked()
38 {
39     vistaadm *ventanaadm = new vistaadm(cont,nullptr);
40     ventanaadm->show();
41     this->close();
42 }
```

Archivo vistaadm.cpp:

Constructor y destructor para generar la ventana.

Función on_btn_agregarpac_clicked:

Crea un objeto para el nuevo paciente.

Abre una nueva ventana y cierra la actual.

Función on_btn_agregarmed_clicked:

Crea un objeto para el nuevo médico.

Abre una nueva ventana y cierra la actual.

Función on:btn:agregarcita:clicked:

Crea un objeto para la nueva cita.

Abre una nueva ventana y cierra la actual.

Archivo widget.cpp:

Arma la base de datos.

Chequea si está abierta. Si no está abierta, retorna error.

Si está abierta, procede a crear una nueva instancia de la clase QSqlTableModel, seleccionar la data necesaria de la database y prepararla para la vista de la tabla (tableview).

Anexos:

Se ha visto esta estructura

Cada vez que quieres que "cuando haces algo, entonces sucede otra cosa", significa que necesitas una conexión.

```
● ● ●  
1 connect(ui->tableView->selectionModel(),  
2 &QItemSelectionModel::currentIndexChanged,  
3 [=](const QModelIndex& idx)->void  
4 {ui->tableView_show->model->setHeaderData(0,Qt::Horizontal, idx.data());});  
5
```

Cuando queremos mostrar una tabla de datos:

```
● ● ●  
1  
2 QSqlQueryModel *model = new QSqlQueryModel;  
3     model->setQuery("SELECT name, salary FROM employee");  
4     model->setHeaderData(0, Qt::Horizontal, tr("Name"));  
5     model->setHeaderData(1, Qt::Horizontal, tr("Salary"));  
6     QTableView *view = new QTableView;  
7     view->setModel(model);  
8     view->show();
```

Conceptos extras implementados:

Programación orientada a objetos:

Se utilizó el especificador **explicit**, que especifica que un constructor o una función de conversión (desde C++11) no permite conversiones implícitas ni inicialización de copias.

Explicit:

```
struct foo  
{  
    foo() {} //Cannot be called for implicit conversion  
    foo(int) {} //Can be called for implicit conversion
```

```
explicit foo(double) //Cannot be called for implicit
conversion
foo(int, int) {} //Not explicit, cannot be called for
implicit conversion
}

void bar(foo);

int main()
{
    bar(foo()); //legal
    bar(10); //Illegal, foo(int) is explicit
    bar(2.0); //Legal foo(double) is not explicit
}
```

Se utilizó polimorfismo en el proyecto. Por ejemplo, la clase **crear_cargo** contiene funciones virtuales que serán redefinidas por las clases hijas: **crear_paciente** y **crear_medico**. Una función virtual es una función miembro que se declara dentro de una clase base y se redefine (anula) por una clase derivada.

Anexos: Funciones originales elaboradas en C++ que sirvieron como base para la documentación:

Main:

Clase one:

Es una clase con las funciones virtuales puras get() y mostrar()

Clase infoDeCita:

Es una clase pública hija de la clase one.

Función get:

Es usada para tomar la información del paciente.

Función mostrar:

Muestra la información del paciente.

Clase citaMedico:

Hereda de la clase InfoDeCita.

Función get:

Escribe la data del objeto a1 hacia el archivo medico1.txt:

a1 -> medico1.txt

Función mostrar:

Si el documento está vacío, muestra un mensaje y regresa al menú.

De otro modo, lee el documento y muestra el contenido.

Clase medico:

Hereda de la clase pública one.

Se usa para guardar información sobre un médico.

Función get:

Guarda información del doctor en un archivo medico.txt

Función mostrar:

Muestra la información.

Clase paciente:

Hereda de la clase one.

Guarda la información del paciente.

Función get:

Guarda información del paciente en un archivo paciente.txt

Función mostrar:

Muestra la información.

Clase administrador:

Hereda de la clase one.

Guarda la información del paciente.

Función get:

Guarda información del administrador en un archivo administrador.txt

Función mostrar:

Muestra la información.

Función reservarCita:

Compara el input del usuario con los código de la clase medico (medico class). Si coinciden, se llama a la función get de la clase medico.

Función mostrarCita:

Compara el input del usuario con los código de la clase medico (medico class). Si coinciden,
se llama a la función mostrar

Función contrasena: //

Lee la contraseña de un archivo password .txt y la compara con el input.
Si coinciden, se dirige al menú. Si no, la función se llama a sí misma.

Función cambiarContrasena: // esto menos

Abre el archivo password.txt, consulta el ingreso de una nueva contraseña y escribe la nueva contraseña en el archivo.

Archivo menudoctor.cpp:

Es importante aclarar que al incluir <QShared Pointer>, se incluye un puntero inteligente que elimina el objeto cuando está fuera de rango.

Constructor menuDoctor:

Contiene una función que se llama para preparar la interfaz del usuario (ui).

Establece el texto de la etiqueta txt_bienvenida con el contenido “Bienvenido”.

Asigna el valor del parámetro idreferencia a la variable idReferencia.

Finalmente, conecta la señal clicked() del botón btn_citasCalendar a btn_citasCalendar_clicked() de la clase menuDoctor.

Destructor menuDoctor.

Función on_btn_citasCalendar_clicked:

Si no se ha creado el calendario de citas (es decir, wdg_citas==nullptr), se crea el objeto citasCalendar(idReferencia) y se muestra.

Si se ha creado, solo se muestra.

Función on_btn_logout_clicked:

Si el usuario hace clic en el botón de cierre de sesión, aparecerá un cuadro de mensaje que le preguntará si está seguro de que desea cerrar sesión. Si el usuario hace clic en 'Aceptar', el programa se cerrará. Si el usuario hace clic en 'Cancelar', el programa seguirá funcionando.