VISTAS

El **Modelo – Vista – Controlador** (**MVC**) es un patrón arquitectónico utilizado para desarrollar interfaces de usuario que divide una aplicación en tres partes interconectadas permitiendo separar la representación interna de los datos de cómo se presenta y acepta la información del usuario.

El patrón de diseño MVC tiene tres componentes principales:

- El **modelo** contiene la estructura de datos con la que está trabajando la aplicación.
- La **vista** es cualquier representación de información que se muestra al usuario, ya sea gráfica o de tablas. Se permiten múltiples vistas del mismo modelo de datos.
- El **controlador** acepta la entrada del usuario, transformándola en comandos para modificar el modelo o la vista.

Las dos partes son esencialmente responsables de:

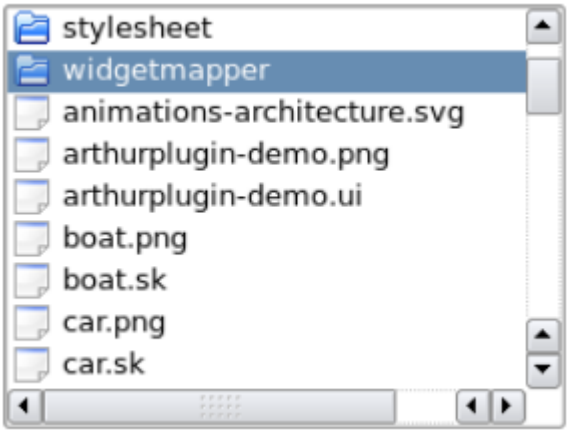
- El **modelo** almacena los datos, o una referencia a ellos, y devuelve registros de rangos individuales, rangos individuales y metadatos asociados o instrucciones de visualización.
- La **vista** solicita datos del modelo y muestra lo que se devuelve en el widget

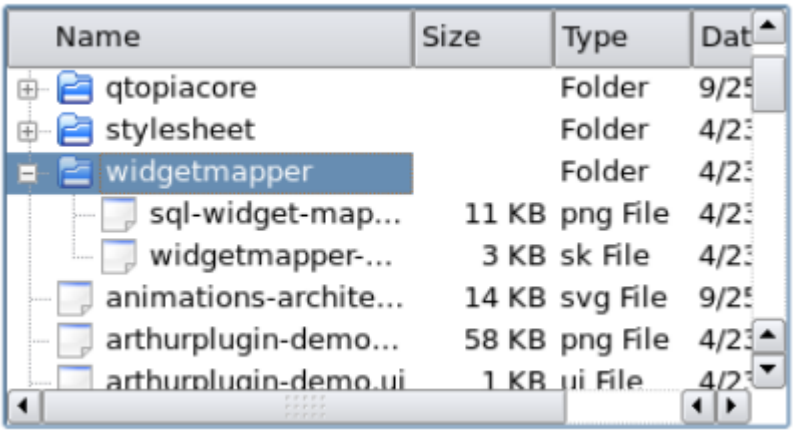
Es posible que la distinción entre la vista y el controlador se vuelva un poco difusa. PyQt acepta eventos de entrada del usuario (a través del sistema operativo) y los delega a los widgets que actúan como controladores para manejar dichos eventos además de manejar la presentación del estado actual al usuario, es decir, actuando en el modo Vista.

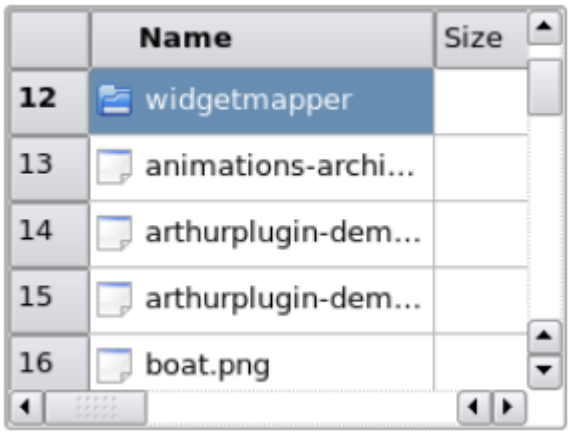
El modelo actúa como la interfaz entre el almacén de datos y el *ViewController.* El Modelo contiene los datos (o una referencia a ellos) y presenta estos datos a través de una API estandarizada que las Vistas luego toman y presentan al usuario. Las vistas múltiples pueden compartir los mismos datos, presentándolos de formas completamente diferentes. Se puede usar cualquier "almacén de datos" para su modelo, incluida, por ejemplo, una lista o diccionario estándar de Python, o una base de datos.

Todos los item models están basados en la clase *QabstractItemModel.* Esta clase proporciona una interfaz para datos que es lo suficientemente flexible como para manejar vistas que representan datos en forma de tablas, listas y árboles. Sin embargo, al implementar nuevos modelos para estructuras de datos tipo lista y tabla, las clases *QAbstractListModel y QAbstractTableMode*l son más útiles porque proporcionan implementaciones predeterminadas más legibles a través de los widgets *QListWidget, QTreeWidget y QTableWidget.*

En la siguiente imagen ejemplo podemos ver de arriba a abajo: *QlistView, QtreeView* y por último, el más común para la representación del contenido de una base de datos, el *QTableView.*







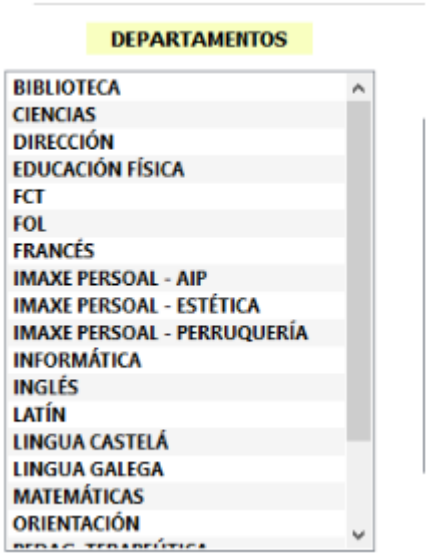
QListView

Es el más simple al contener solo una columna. Tiene múltpiles opciones como selección simple de datos, selección múltiple, carga de datos desde una BBDD.

En el siguiente ejemplo vemos como se carga un lista llamada lstDepar

```
def listarDepar(self):
    try:
        var.ui.lstDepart.clear()
        query = QtSql.QSqlQuery()
        query.prepare('select departamento from departamento')
        if query.exec_():
            while query.next():
                depar = query.value(0)
                var.ui.lstDepart.addItem(depar)
    except Exception as error:
        print('Erros en alta departamento', error)
```

La propiedad addItem permite la carga, el resultado sería algo así:



Para seleccionar debemos activar el siguiente evento ***itemClicked*** en el módulo principal ***main.py,*** para que se cargue al ejecutar el programa***:***

```
var.ui.lstDepart.itemClicked.connect(conexion.Conexion.selecDepar)
var.ui.lstConcepto.itemClicked.connect(conexion.Conexion.selecObjeto)
var.ui.tabAvisos.clicked.connect(events.Eventos.oneAviso)
```

```
def selecObjeto(item):
    try:
        item = item.text()
        var.carga = []

        if item == 'Copias B/N':
            var.carga.append('hojasbn')
        elif item == 'Copias Color':
            var.carga.append('hojascolor')
        elif item == 'Paquetes Folios':
            var.carga.append('paquetes')
        elif item == 'Rotuladores':
            var.carga.append('rotuladores')

    except Exception as error:
        print('Error selección objeto ', error)
```

Como podemos ver con hacer referencia a la propiedad ***.text()*** ya tenemos seleccionado el objeto o dato que deseemos.