EXPENSE TRACKER APP

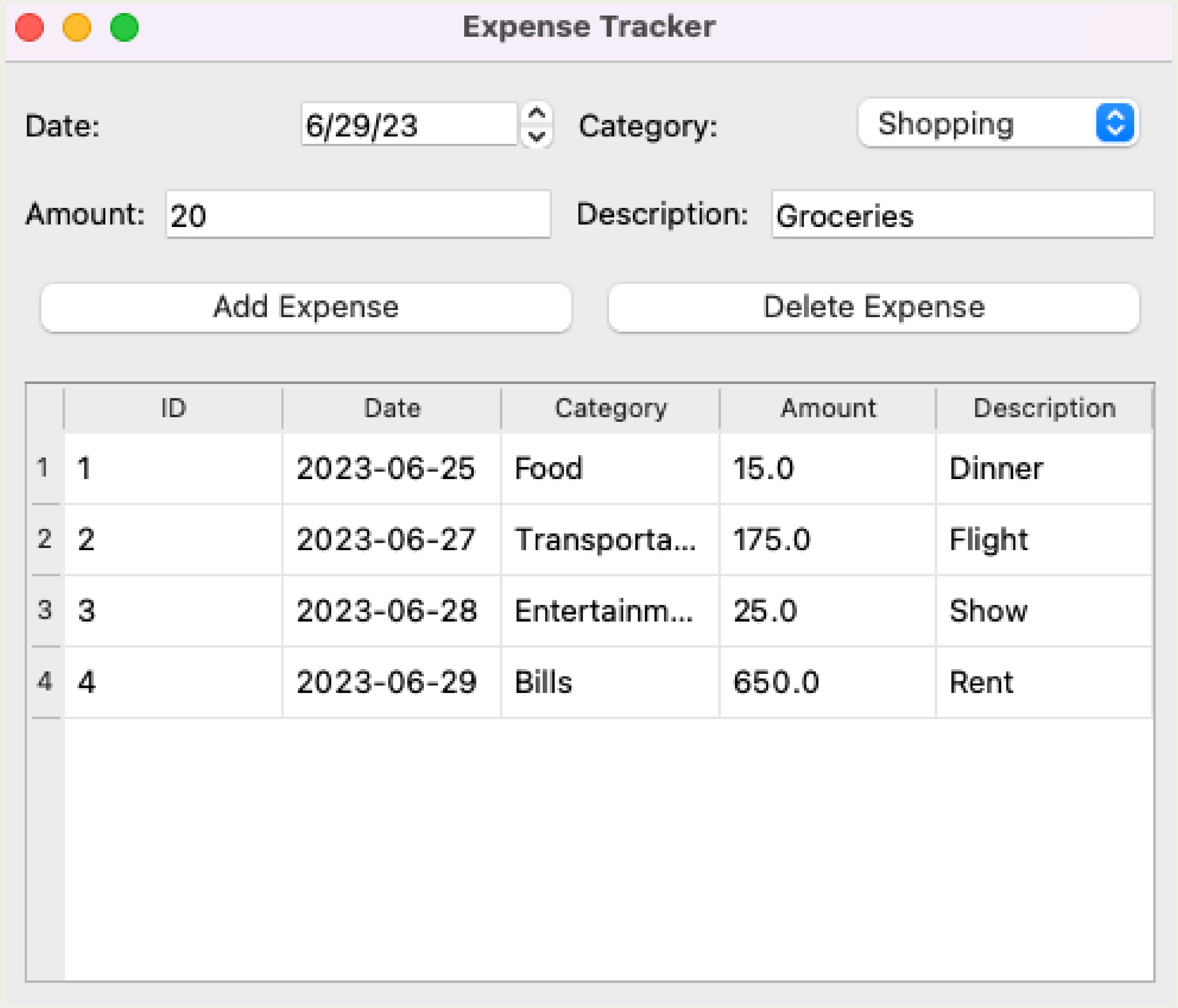
Adding a SQL Database to our App to track and maintain live data

ZERO TO KNOWING

App Overview:

What Widgets do you see?

ZERO TO KNOWING



Expense Tracker

Date: 6/29/23 Category: Shopping

Amount: 20 Description: Groceries

	ID	Date	Category	Amount	Description
1	1	2023-06-25	Food	15.0	Dinner
2	2	2023-06-27	Transporta...	175.0	Flight
3	3	2023-06-28	Entertainm...	25.0	Show
4	4	2023-06-29	Bills	650.0	Rent

Let's take a look at the App we will be building:

We can:

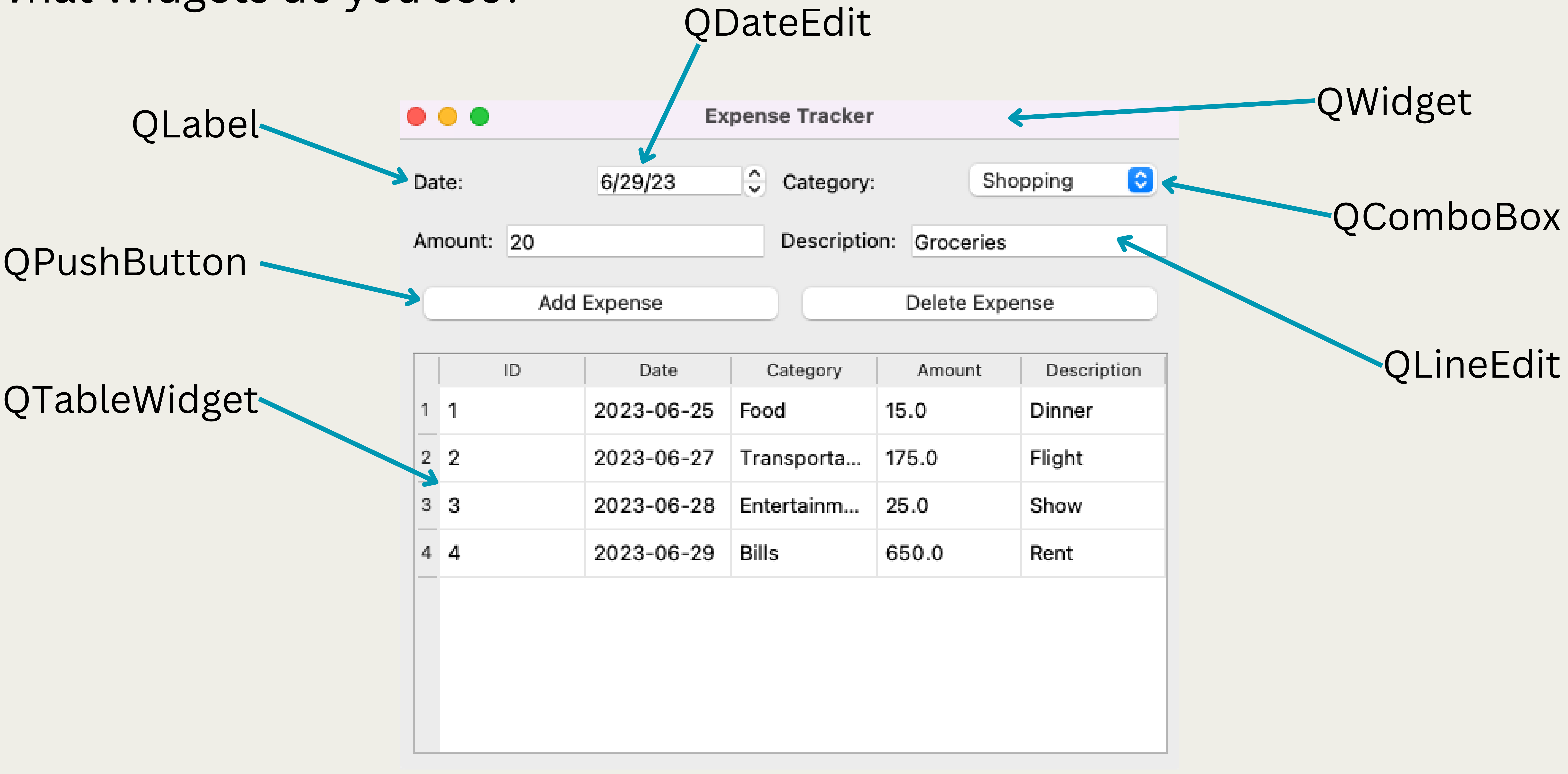
-**Add** Expenses

-**Delete** Expenses

-Save our data in a **Database**

App Overview:

What Widgets do you see?



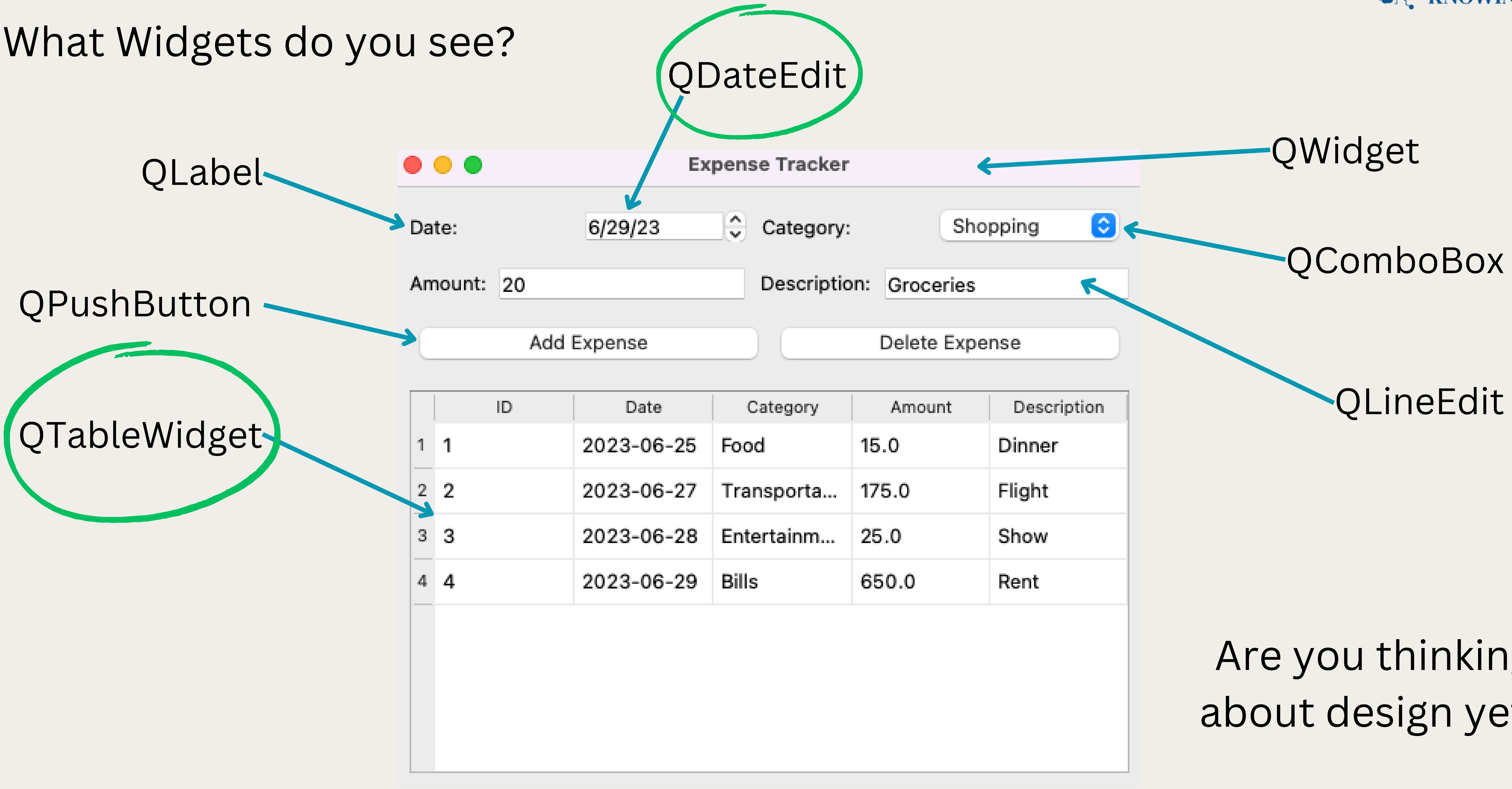
Date: 6/29/23 Category: Shopping

Amount: 20 Description: Groceries

	ID	Date	Category	Amount	Description
1	1	2023-06-25	Food	15.0	Dinner
2	2	2023-06-27	Transporta...	175.0	Flight
3	3	2023-06-28	Entertainm...	25.0	Show
4	4	2023-06-29	Bills	650.0	Rent

App Overview:

What Widgets do you see?



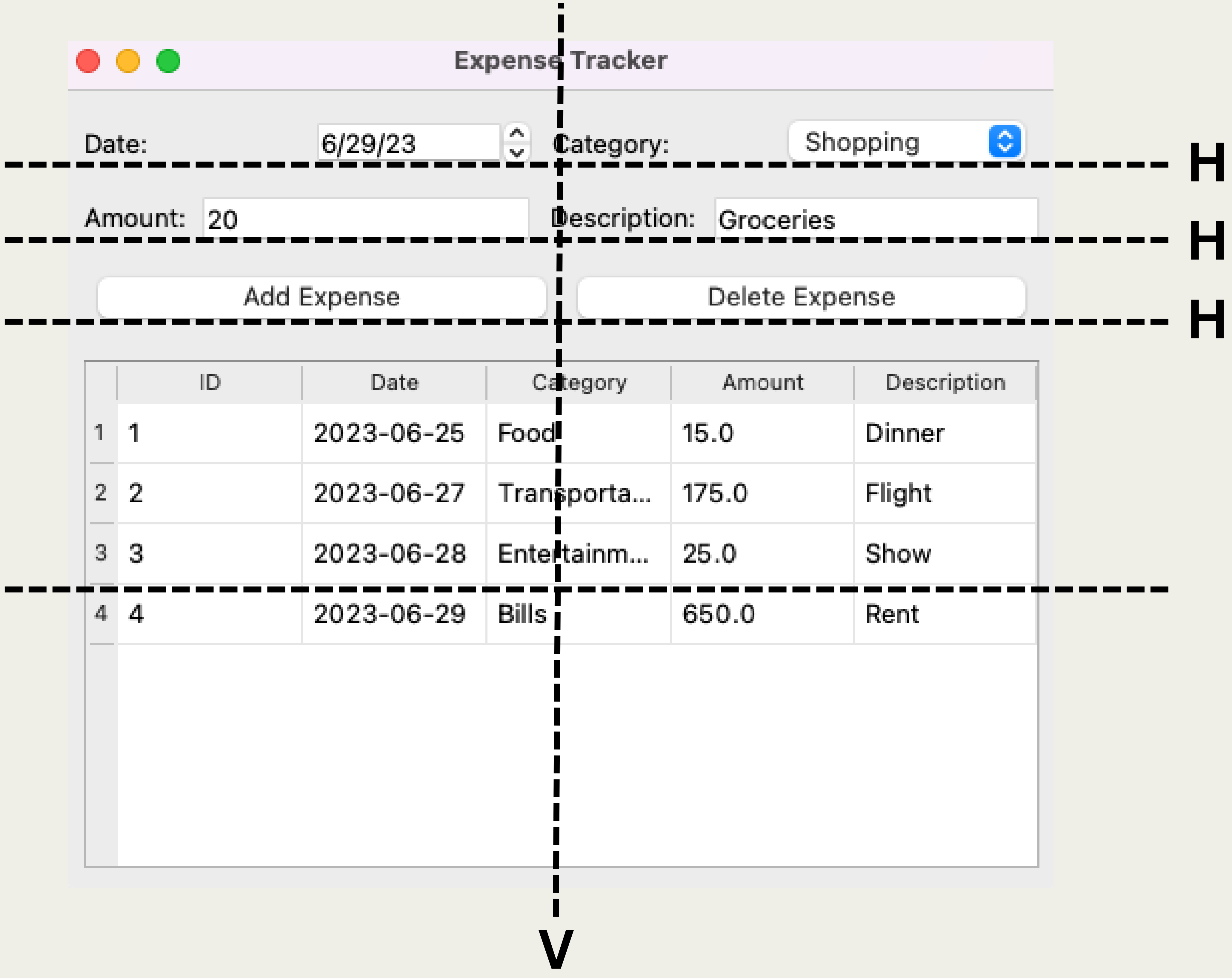
Date: 6/29/23 Category: Shopping

Amount: 20 Description: Groceries

	ID	Date	Category	Amount	Description
1	1	2023-06-25	Food	15.0	Dinner
2	2	2023-06-27	Transporta...	175.0	Flight
3	3	2023-06-28	Entertainm...	25.0	Show
4	4	2023-06-29	Bills	650.0	Rent

Are you thinking about design yet?

App Design



	ID	Date	Category	Amount	Description
1	1	2023-06-25	Food	15.0	Dinner
2	2	2023-06-27	Transporta...	175.0	Flight
3	3	2023-06-28	Entertainm...	25.0	Show
4	4	2023-06-29	Bills	650.0	Rent

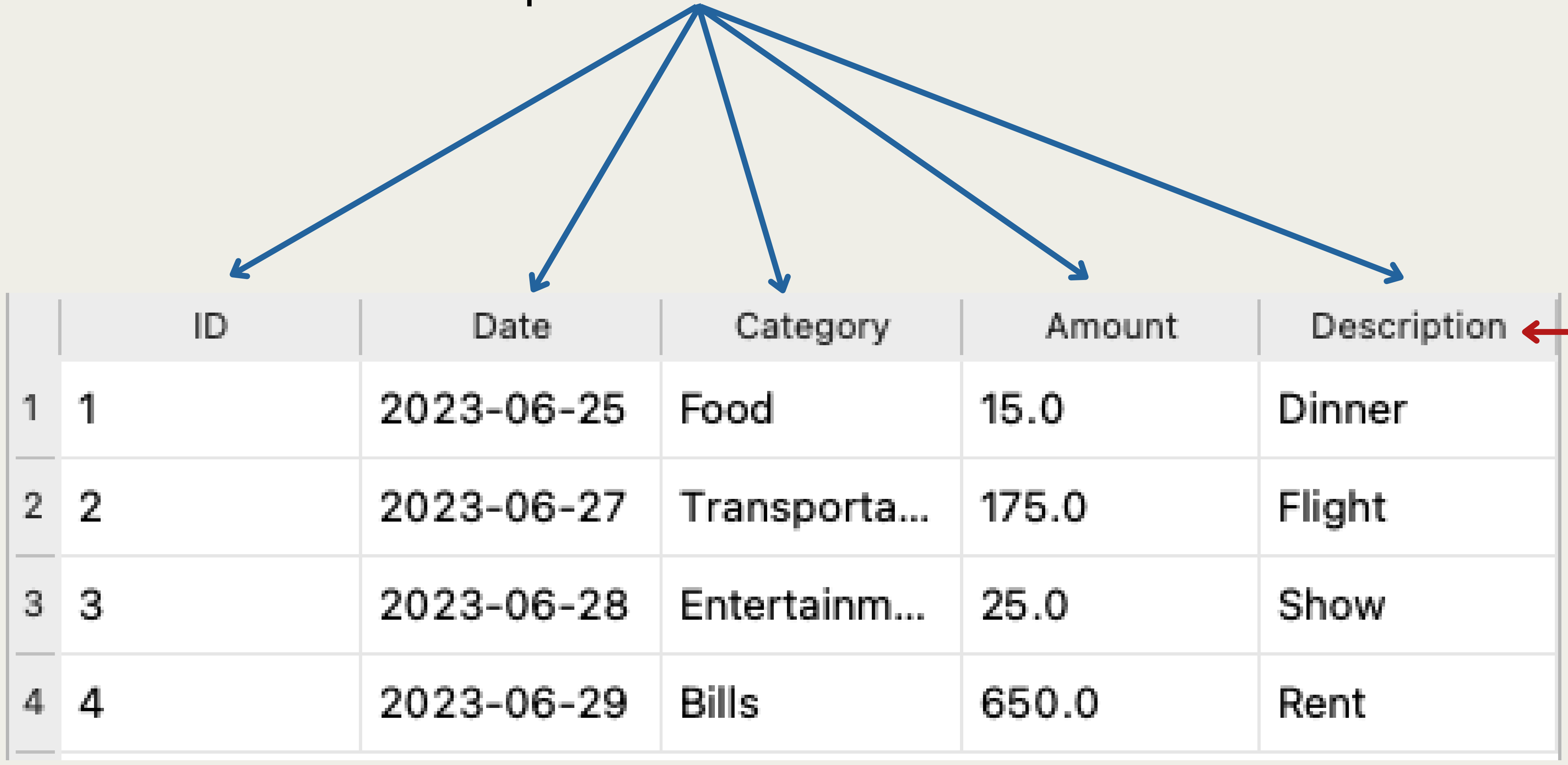
Design can be done with the trusty **QVBoxLayout** and **QHBoxLayout**

Does the **Table** need a row if it already takes up the whole screen?

Methods for our Table

QTableWidget ->This **creates a Table** within our App

.setColumnCount(#) -> This method **sets the number of columns** in our table. It accepts a **number**



	ID	Date	Category	Amount	Description
1	1	2023-06-25	Food	15.0	Dinner
2	2	2023-06-27	Transporta...	175.0	Flight
3	3	2023-06-28	Entertainm...	25.0	Show
4	4	2023-06-29	Bills	650.0	Rent

.setHorizontalHeaderLabels([]) -> This method **sets the Name of each Column** in our table. It accepts a **List**

Intro to SQL

Working with SQLite with QtSql in PyQt

ZERO TO KNOWING

SQL Basics and Syntax

SQL - Structured Query Language

Used to **manage data stored in relational databases**

Relational Databases **store structured data in tables**

Basic Syntax

SELECT - **Column** you want to look at

FROM - **Table** where the data lives

WHERE - A specific **condition is True**

Example

Table name: users
Columns: username & password
Condition: Look for "mario123"

SQL Basics and Syntax

SQL - Structured Query Language

Used to **manage data stored in relational databases**

Relational Databases **store structured data in tables**

Basic Syntax

SELECT - **Column** you want to look at

FROM - **Table** where the data lives

WHERE - A specific **condition is True**

Example
Table name: users
Columns: username & password
Condition: Look for "mario123"

SELECT *	**Select everything**
FROM users	**From** the **Table** named **users**

SELECT username	Select the Column username
FROM users	From the Table named users
WHERE user="mario123"	Where the username is "mario123"

SQL Setup in PyQt

ZERO TO KNOWING

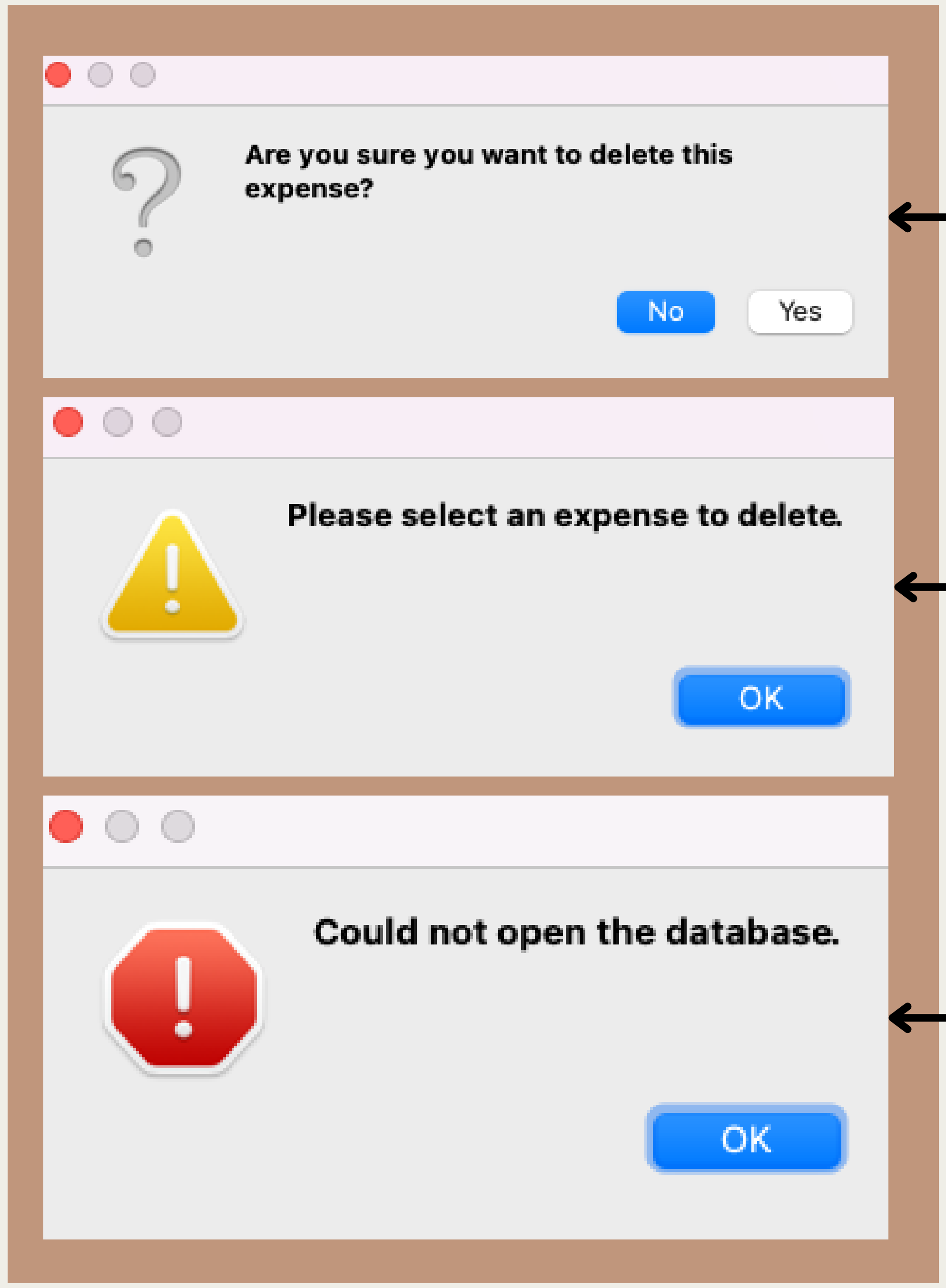
We want to create a connection with a SQLite Database

```
database = QSqlDatabase.addDatabase("QSQLITE")
database.setDatabaseName("app_database.db")
if not database.open():
    QMessageBox.critical(None, "Error", "Could not open the database")
    sys.exit(1)
```

QtSql Class Methods	What they do
.addDatabase()	Establishing a Connection to a SQLite Database
.setDatabaseName()	Set the name of your new Database
open()	Python open method to open our Database

QMessageBox in PyQt

This is like an **alert pop-up window**. This can trigger **different alerts**



Are you sure you want to delete this expense?
No
Yes
Please select an expense to delete.
OK
Could not open the database.
OK

```
QMessageBox.question("Delete", "Are you sure you want to delete this expense?", QMessageBox.Yes | QMessageBox.No )
```

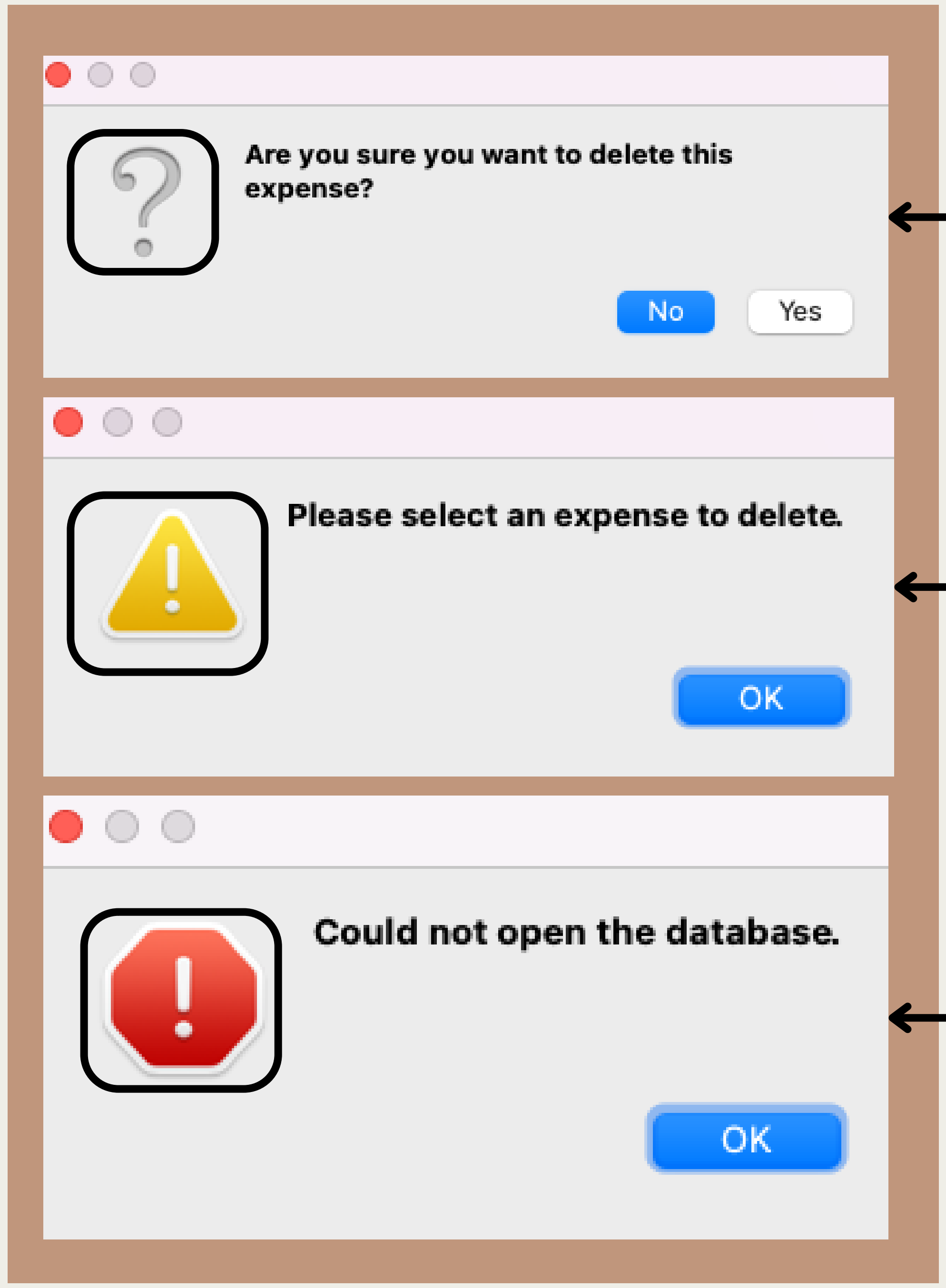
```
QMessageBox.warning("None Selected", "Please select an expense to delete" )
```

```
QMessageBox.critical( None, "Critical Error", "Could not open database" )
```

QMessageBox in PyQt

ZERO TO KNOWING

This is like an **alert pop-up window**. This can trigger **different alerts**


Are you sure you want to delete this expense?

No | Yes

Please select an expense to delete.

OK

Could not open the database.

OK

QMessageBox.question("Delete", "Are you sure you want to delete this expense?", QMessageBox.Yes | QMessageBox.No)

QMessageBox.warning("None Selected", "Please select an expense to delete")

QMessageBox.critical(None, "Critical Error", "Could not open database")

Creating a Query

ZERO TO KNOWING

We want to run a SQLite database **query to create a table** named "expenses", but **only if it doesn't already exist**

```
query = QSqlQuery()
query.exec_("""
    CREATE TABLE IF NOT EXISTS expenses (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT,
        category TEXT,
        amount REAL,
        description TEXT
    )
""")
```

Creates an Object of the QSqlQuery class, that's **used to execute SQL queries on our connected database**

The **exec_()** function provides a concise way to **execute queries without the need for additional boilerplate code**

Creating a Query

By using the IF NOT EXISTS clause, the query ensures that the table is only created if it doesn't already exist in the database

Executes the SQL query specified within the triple quotes

Creates a table named "expenses" **with** the following **columns**

```
query = QSqlQuery()
query.exec_("""
    CREATE TABLE IF NOT EXISTS expenses (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT,
        category TEXT,
        amount REAL,
        description TEXT
    )
""")
```

"id" - An INTEGER column serving as the primary key for the table

"date" - A TEXT column

"category" - A TEXT column

"amount"- A REAL column to store the amount

"description" - A TEXT column

Creating a Query

By using the IF NOT EXISTS clause, the query ensures that the table is only created if it doesn't already exist in the database

Executes the SQL query specified within the triple quotes

Creates a table named "expenses" **with** the following **columns**

```
query = QSqlQuery()
query.exec_("""
    CREATE TABLE IF NOT EXISTS expenses (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT,
        category TEXT,
        amount REAL,
        description TEXT
    )
""")
```

"id" - An INTEGER column serving as the primary key for the table

"date" - A TEXT column

"category" - A TEXT column

"amount"- A REAL column to store the amount

"description" - A TEXT column

ZERO TO KNOWING

Creating our Methods

Working with SQLite to Add and Delete Expenses

Load Expense Table

1. **Clear** the current Table - **.**setRowCount()
2. Create a query to **SELECT everything FROM** our **table** - QSqlQuery
3. Create a **Loop** to run as long as there are more **rows in the table** - **.**next()
4. **Retrieve the value from each column** in the table - **.**value(#)
5. **Insert** the collected **data** from #4 into a new row - **.**insertRow(#)
6. **Increase** row **counter**

Load Expense Table

ZERO TO KNOWING

```
while query.next():
        expense_id = query.value(0)
        date = query.value(1)
        category = query.value(2)
        amount = query.value(3)
        description = query.value(4)

        self.expense_table.insertRow(row)
        self.expense_table.setItem(row, 0, QTableWidgetItem( str( expense_id)))
        self.expense_table.setItem(row, 1, QTableWidgetItem( date))
        self.expense_table.setItem(row, 2, QTableWidgetItem( category))
        self.expense_table.setItem(row, 3, QTableWidgetItem( str( amount)))
        self.expense_table.setItem(row, 4, QTableWidgetItem( description))
```

3. Create a Loop to run as long as there are more rows in the table

4. Retrieve the value from each column in the table

5. Insert the collected data from #4 into a new row

Add New Expense

1. **Gather the info**rmation **entered in** the **input** boxes
2. **Insert the expense** into the database
3. **Clear** the **input fields** for next expense
4. **Load** in the updated **database**

.currentText() .prepare()

.toString()

.date() .text() .clear() .addBindValue()

.exec_() .currentDate() .setDate()

Add New Expense

Methods	What it does
.prepare()	**Checks** the provided SQL **query string** to ensure it is **valid**
.addBindValue()	**Put the information into a column** in our database
.exec_()	**Execute queries** and adds them to our Database
.toString()	**Converts an object** into its **string** representation
.date()	**Converts an inpu**t in our case, into a **date**
.setDate()	This will **set the date** or update the date
.currentDate()	Get the **live** current **date**
.currentText()	**Get** the currently selected **text from a dropdown list**

Add New Expense

```
date = self.date_edit.date().toString('yyyy-MM-dd')
category = self.category_combo.currentText()
amount = self.amount_edit.text()
description = self.description_edit.text()

query = QSqlQuery()
query.prepare("""
    INSERT INTO expenses (date, category, amount, description)
    VALUES (?, ?, ?, ?)
""")
query.addBindValue(date)
query.addBindValue(category)
query.addBindValue(amount)
query.addBindValue(description)
query.exec_()
```

Collecting the input field information

Creating and Preparing a new Query to be added to our Database

Adding and Sending our info to our Database

ZERO TO KNOWING

Final Stages

Adding the Final Touches onto our App

Delete an Expense

1. **Get the row** we **click** from our table - .currentRow()
2. Check to ensure we did indeed **choose a row**
3. Create a **variable** that gets the **ID of the selected row**
4. Create a **Question Pop-up** asking to Delete, **Yes or No** - .question()
5. **if** yes, **prepare a query** - DELETE FROM table WHERE id equals the value
6. **Load** our new table with the **updated database**

Delete an Expense

ZERO TO KNOWING

```
selected_row = self.expense_table.currentRow()

expense_id = int(self.expense_table.item(selected_row, 0).text())

query = QSqlQuery()
query.prepare("DELETE FROM expenses WHERE id = ?")
query.addBindValue(expense_id)
query.exec_()

self.load_expenses()
```

Getting the row we clicked on

Getting the ID from the selected row

Preparing a Query, adding the ID, executing the query

Literal Translation - **Delete from** my table named **expenses**, but only **where** the **id matches the one I give you**

Load our new table with the **updated database**

ZERO TO KNOWING

Intro to Styling

How to style PyQt apps with CSS

Cascade Styling Sheet - CSS

ZERO TO KNOWING

```
h1 {
   font-size: 32px;
   font-family: gothic;
}

.main_class  {
   border: 2px solid;
}

#chart_id  {
   background-color: #fff;
}
```

What would this look like in Python with PyQt?

CSS is used to style websites in **HTML**, we can use it to style in Python as well

We start be **target**ing a **parent elements** such as **main elements**, **classes** and **id's** (CSS terms)

Within a set of **{curly braces}** we **add our styles**

Each style **ends**/breaks with a **semi-colon;**

Cascade Styling Sheet - CSS

```
self.setStyleSheet(
    """
     FinanceApp{
         background-color: #222222;
     }

     QLabel, QLineEdit, QPushButton {
         background-color: #333333;
         color: #eeeeee;
     }
     QTreeView {
         background-color: #444444;
         color: #eeeeee;
     }
    """)
```

We can style the same way in PyQt by using the **.setStyleSheet Method**

The only difference is it is **all a string**

We start be **target**ing a **parent elements** such as **main elements**, **classes** and **id's** (CSS terms)

ZERO TO KNOWING

Congrats!

Our Finalized App!