**Lab Exercise 13 – Building Dynamic UIs with QML**

**Objective:**

Learn how to **create dynamic and interactive UIs** using **QML and Python (PySide6)**. This lab focuses on:

* **Dynamically updating UI elements**
* **Using QML ListView to display Python data**
* **Implementing real-time UI updates**

**1. Learning Outcomes**

By completing this lab, you will:  
✅ **Dynamically generate UI elements** in QML.  
✅ **Use QML ListView to display lists of data from Python.**  
✅ **Implement interactive UI updates using signals and slots.**  
✅ **Bind QML elements to Python backend logic.**

**2. Project Setup & Requirements**

**Project Structure**

dynamic\_ui\_qml/

│── main.py # Python backend

│── main.qml # QML UI file

│── requirements.txt # Dependencies (PySide6)

**Install Dependencies**

Ensure you have **PySide6** installed:

pip install PySide6

**3. Creating the QML UI (main.qml)**

This **QML file** defines:

* **A ListView** displaying dynamic data from Python
* **A Button** to add new items to the list
* **A Label** displaying real-time count of items

import QtQuick 6.2  
import QtQuick.Controls 6.2  
  
ApplicationWindow {  
 visible: true  
 width: 400  
 height: 500  
 title: "Dynamic ListView"  
  
 Column {  
 anchors.fill: parent  
 spacing: 10  
 padding: 10  
  
 Button {  
 text: "Add Item"  
 onClicked: backend.addItem()  
 }  
  
 Button {  
 text: "Remove Last Item"  
 onClicked: backend.removeItem()  
 }  
  
 ListView {  
 id: listView  
 width: parent.width  
 height: 400  
 model: listModel // Ensure correct model reference  
  
 delegate: Rectangle {  
 width: listView.width  
 height: 50  
 color: "lightblue"  
 border.color: "black"  
  
 Text {  
 anchors.centerIn: parent  
 text: model.display // Correct reference to model data  
 font.pixelSize: 16  
 }  
 }  
 }  
 }  
}

**4. Loading and Handling QML in Python (main.py)**

This Python script:

* Loads the QML file using QQmlApplicationEngine
* Connects Python **backend logic** to QML UI
* Updates UI dynamically when items are added

import sys  
from PySide6.QtWidgets import QApplication  
from PySide6.QtCore import QStringListModel, QObject, Slot  
from PySide6.QtQml import QQmlApplicationEngine  
  
  
class Backend(QObject):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_model = QStringListModel(["Item 1", "Item 2", "Item 3"]) *# Initial list items* @Slot()  
 def addItem(self):  
 *"""Adds a new item to the list."""* current\_items = self.\_model.stringList()  
 current\_items.append(f"Item {len(current\_items) + 1}")  
 self.\_model.setStringList(current\_items)  
  
 @Slot()  
 def removeItem(self):  
 *"""Removes the last item from the list."""* current\_items = self.\_model.stringList()  
 if current\_items:  
 current\_items.pop()  
 self.\_model.setStringList(current\_items)  
  
 def get\_model(self):  
 *"""Expose the model to QML."""* return self.\_model  
  
  
def main():  
 app = QApplication(sys.argv)  
 engine = QQmlApplicationEngine()  
  
 backend = Backend()  
 engine.rootContext().setContextProperty("backend", backend)  
 engine.rootContext().setContextProperty("listModel", backend.get\_model()) *# Explicitly expose model* engine.load("main.qml")  
 if not engine.rootObjects():  
 sys.exit(-1)  
  
 sys.exit(app.exec())  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**5. Running the Application**

Run the Python script:

python main.py

✅ **Click "Add Item"** → A new item appears in the list, and the count updates dynamically.

**6. Key Features in This Exercise**

✅ **Dynamic ListView updates** in real-time.  
✅ **Python-generated data displayed in QML UI.**  
✅ **Signals and slots to handle UI interactions.**  
✅ **Efficient QML-Python communication using context properties.**

**7. Next Steps & Enhancements**

* ✅ Implement **item removal feature**
* ✅ Use **QML animations for smooth UI transitions**
* ✅ Enable **data persistence using JSON or a database**