International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Developing QML Remote Machine UI (RMUi) Application

Binoy Kurikaparambil Revi

Independent Researcher, USA
binoyrevi@live.com

Abstract:
It is often desirable to install lightweight, high-UI-rich applications that do not contain core technologies but simply expose data and UI controls on a remote machine. In many scenarios, it is advantageous to install lightweight applications on client systems that offer a rich user interface while not directly incorporating core functionalities. Instead, these applications focus on effectively presenting data from a remote machine. Developing a QML application using the QT framework as a Remote Machine User Interface(RMUi) provides such an advantage.

Keywords: Remote Application, Lightweight UI Client, QT, QML

Introduction:
Using a QML UI application with RESTful services[1] enables the creation of lightweight applications that utilize a minimal number of Qt packages while providing a rich user interface. This is particularly useful in client machines where configurations may vary, and security measures may not be fully implemented. This approach enhances security and reduces computing power requirements, allowing the client application to remain streamlined and efficient. The more resource-intensive tasks, such as data processing and storage, are handled on a separate Application Server. As a result, users benefit from an intuitive interface that quickly retrieves and displays the necessary information without the complexities of the underlying systems. Consequently, this can improve performance, provide a better user experience, and make maintenance easier.

Remote Machine User Interface(RMUi) Application Design:
The overall architecture of the QML Remote Machine User Interface (RMUi) application is outlined below. It primarily consists of three components: the RMUi Application, Network, and Application Server. A RESTful service is utilized for communication between the RMUi Application and the Application Server.

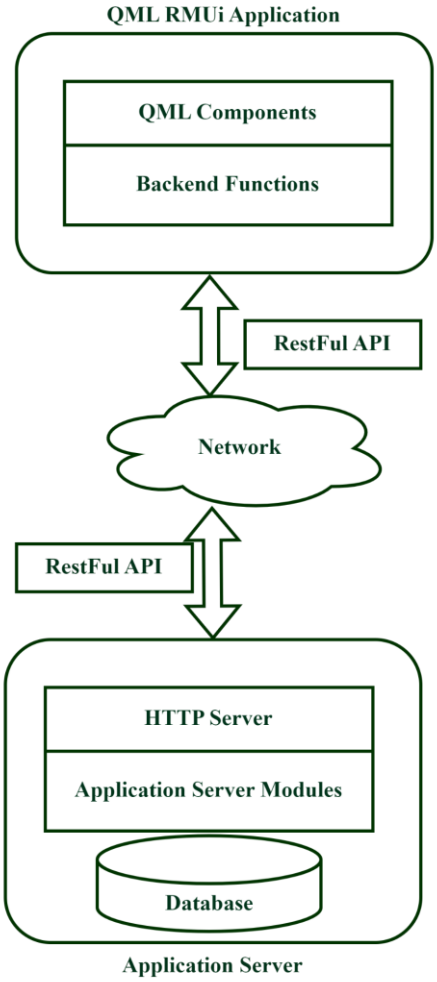

Figure 1: Overall Architecture of the System

A QML UI application can be designed and implemented using Qt Studio, Qt Creator, or VS Code. Qt Studio offers a visual toolkit that facilitates the rapid design of QML components. The application consists of two main parts: the QML side and the C++ backend. The `main.cpp` file serves as the entry point for the application, while `main.qml` is integrated into `main.cpp` as an object for the user interface implementation.

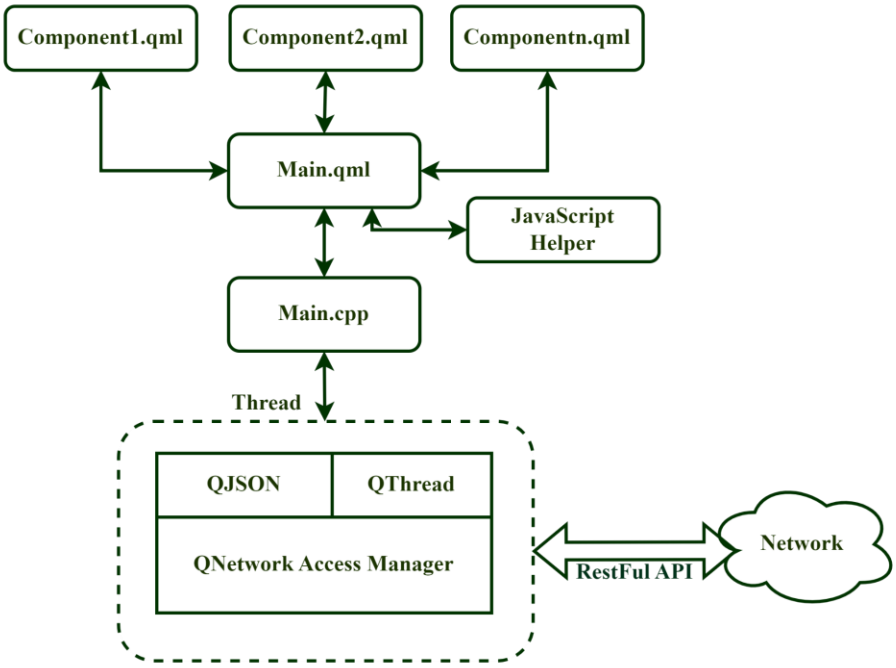

Figure 2: QML UI Application Design

QML Design: All user interface (UI) components and their subcomponents are derived from `main.qml`. It is essential to design these QML components and subcomponents in a hierarchical order to ensure a structured and well-defined data flow. All QML components interact with their subcomponents through methods, while subcomponents communicate with components using the Signal-Slot mechanism.
Main.cpp: This file is the entry point to the application. The UI component and the backend component communicate with each other through the main.cpp. Both these components are handled as objects. Mostly main.cpp initializes the backend module and starts the backend.
Communication Thread: This is a lightweight software thread initiated by the backend, primarily from the `main.cpp` file. It continuously monitors for RESTful service requests from the application. When it detects a request in the queue, it utilizes the `QNetworkAccessManager`[7] to send a RESTful request to the server. QJSON package is used to encode the request and decode the response. QThread package is used to create the threading framework.

Application Server Design:

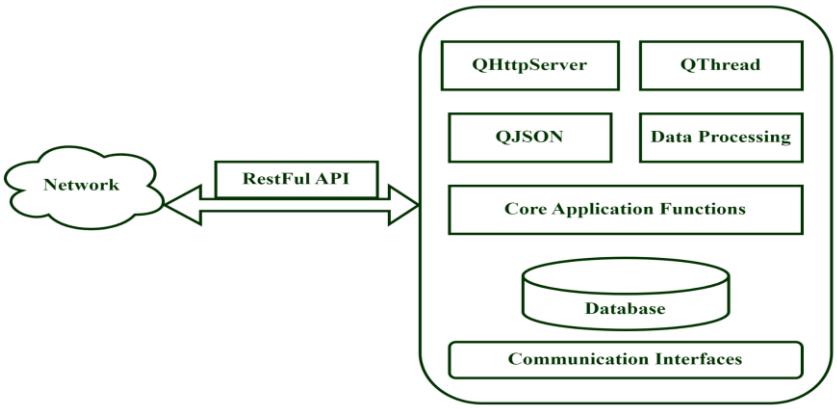

Figure 3: Application Server Design

The Application Server[3] is designed to provide secure access to accept and respond to RESTful requests[1]. In addition to this, it performs significant tasks related to application functionality, data management, and cybersecurity. A typical simple Application Server may include the following components, assuming it uses a database for data storage, supports communication interfaces to interact using various protocols and services RESTful requests:

1. QHTTPServer[6]: This package can be utilized to implement the server application functionality that accepts RESTful service requests and responds with data[2]. The QHTTPServer employs QT implementation techniques that make it event-driven.
2. QThread[9]: This package allows you to implement threading functionality within the server. It can be applied to the server itself or communication interfaces. Essentially, it handles any asynchronous job that needs to be executed in parallel.
3. QJSON[8]: The QJSON package is extremely useful for processing JSON data when dealing with HTTP requests.
4. Core Functionality and Data Processing: This is the core of the server application and is entirely dependent on the architecture and design of the system.
5. Database[5]: Depending on the application's needs, various database options can be used, such as in-memory databases like Redis or on-disk databases like MongoDB or SQL.
6. Communication Interfaces: This feature is optional and depends on the application's requirements and design.

This clear structure highlights each component's role in the Application Server's functionality[3].

Conclusion:

In the modern world of technology, cybersecurity threats are proliferating at a pace that parallels the rapid advancement of technology. The deployment of applications on systems characterized by cybersecurity uncertainties and insufficient hardware resources underscores the necessity of developing the 'Remote Machine UI Application' (RMUi Application).

QML provides a robust solution for developing Remote Machine User Interface (RMUi) applications that do not store proprietary technologies or sensitive data on the system. Instead, it simply displays data and offers an engaging user interface for interaction with the system[4]. This approach also ensures data security both at rest and during transfer. All core functionalities are managed by a remote application server, which should be situated in a secure and controlled environment to prevent unauthorized access and cybersecurity threats. The QT framework provides essential packages for easily developing RMUi applications and offers a lightweight UI option. On the application server side, the QT framework supplies all the necessary packages to develop server modules, but it is not limited to QT alone.

References:

1. Silvia Schreier. 2011. Modeling RESTful applications. In Proceedings of the Second International Workshop on RESTful Design (WS-REST '11). Association for Computing Machinery, New York, NY, USA, 15–21. https://doi.org/10.1145/1967428.1967434
2. Adamczyk, P., Smith, P.H., Johnson, R.E. and Hafiz, M., 2011. Rest and web services: In theory and in practice. REST: from research to practice, pp.35-57.
3. I. Kistijantoro, G. Morgan, S. K. Shrivastava and M. C. Little, "Enhancing an Application Server to Support Available Components," in IEEE Transactions on Software Engineering, vol. 34, no. 4, pp.

531-545, July-Aug. 2008, doi: 10.1109/TSE.2008.38.
4. M. Garriga, K. Rozas, D. Anabalon, A. Flores and A. Cechich, "RESTful mobile architecture for social security services: A case study," 2016 XLII Latin American Computing Conference (CLEI), Valparaiso, Chile, 2016, pp. 1-11, doi: 10.1109/CLEI.2016.7833367.
5. F. Orellana and M. Niinimaki, "Distributed Computing with RESTful Web Services," 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Victoria, BC, Canada, 2012, pp. 103-110, doi: 10.1109/3PGCIC.2012.30.
6. QT HTTP Server : https://doc.qt.io/qt-6/qthttpserver-index.html
7. QT Network Libraries: https://doc.qt.io/qt-6/qnetworkaccessmanager.html
8. QT JSON Library: https://doc.qt.io/qt-6/qjsonobject.html
9. QT Thread Library: https://doc.qt.io/qt-6/qthread.html