**Section 1: Management**

This section would contain a dashboard for the management team. The Qt framework provides multiple widgets and tools that can be used to create a dashboard. This could include:

* **QTableView** and **QStandardItemModel**: For displaying and managing data in a tabular format. The model would store the work orders, broken and fixed counts, cost of parts, etc. The view would display this data.
* **QChart**: For displaying graphical data like spending per day/month/year.
* **QBoxLayout** or **QGridLayout**: For arranging the widgets in the dashboard.
* **QSpinBox**, **QDoubleSpinBox**, **QLineEdit**, etc.: For inputting data.

**Section 2: Inventory**

This section would handle the inventory of parts. It could include:

* **QLineEdit** and **QSpinBox** widgets: For entering new stock data.
* **QPushButton**: For confirming the entry of new stock data.
* **QTableView** and **QStandardItemModel**: For displaying and managing stock data.
* **QMessageBox**: For displaying messages when stock is low or out.

**Section 3: Maintenance**

This section would manage the work orders. It could include:

* **QLineEdit**, **QSpinBox**, **QDateTimeEdit**, etc.: For entering new work order data.
* **QPushButton**: For confirming the entry of new work order data and for signing off completed work orders.
* **QTableView** and **QStandardItemModel**: For displaying and managing work order data.
* **QMessageBox**: For displaying messages when work orders are completed or parts are used.

All data could be stored in a relational database using the **QtSql** module. This would allow for easy querying and manipulation of the data. SQLite could be a good choice for a local database, but you could also use PostgreSQL, MySQL, or any other database supported by QtSql.

Remember to implement separation of concerns in your architecture: each section should be encapsulated in its own class or group of classes, and data handling (especially database operations) should be separated from the UI code. This will make your code more maintainable and testable.

This is a high-level overview of the project. The exact implementation would depend on the specific requirements and constraints of the project. I hope this helps!

If you want to separate the database from the UI and use a network-based approach, you'll need to make some changes to the architecture of your application.

Firstly, you'll need a central database that can be accessed over the network. MySQL is a good option for this. You will need to install MySQL server on the machine that will host the database, and set up the necessary users, permissions, and tables.

The architecture of your application could then look something like this:

1. **Database Server**: This would be the MySQL server that stores all your data. It should be configured to accept connections over the network from the machines running your application.
2. **Application Server**: This would be a separate application that handles communication between the UI clients and the database. It would connect to the MySQL server using the QtSql module, perform queries and updates on the database, and send the results back to the UI clients. This application could be implemented using the QtNetwork module, which provides tools for network communication.
3. **UI Clients**: These would be the actual UIs that the users interact with. They would connect to the application server using the QtNetwork module, send requests to the server (like "get all work orders", "add new stock", "sign off work order", etc.), and display the results received from the server.

Here's how the components of the Qt framework would be used in this architecture:

* **QtNetwork**: This module would be used in both the application server and the UI clients for network communication. The **QTcpServer** and **QTcpSocket** classes could be used for this purpose.
* **QtSql**: This module would be used in the application server to connect to the MySQL database and perform queries and updates. The **QSqlDatabase**, **QSqlQuery**, and related classes would be used for this.
* **QtWidgets**: This module would be used in the UI clients to create the user interface. The widgets mentioned in the previous response (QTableView, QStandardItemModel, QChart, QBoxLayout, QGridLayout, QSpinBox, QDoubleSpinBox, QLineEdit, QPushButton, QMessageBox) would be used for this.

Remember to implement good security practices in your application, like encrypting the network communication, protecting the database with a strong password, and sanitizing any user input before using it in a database query.

This approach allows you to separate the database and UI, and have multiple users connect to a central database. The UI clients can be customized to show different interfaces to different users based on their roles and permissions.