Solution: Railway Management System Problem

Problem: Railway Management System

You are tasked with developing a PyQt5 application to manage a railway system. The application should allow users to view train schedules, book tickets, and manage train routes. The system must handle real-time updates and provide a user-friendly interface.

Requirements

1. Train Schedule Management (10 Marks)

- Implement a ‘QTableWidget‘ to display train schedules (columns: Train ID, Departure Time, Arrival Time, Route). - Allow users to add, edit, and delete train schedules. - Use ‘QTimer‘ to update the schedule in real-time (e.g., simulate delays).

Solution:

```
from PyQt5.QtWidgets import QTableWidget, QTableWidgetItem, QVBoxLayout, QWidget
from PyQt5.QtCore import QTimer

class TrainSchedule(QWidget):
    def __init__(self):
        super().__init__()
        self.table = QTableWidget(self)
        self.table.setColumnCount(4)
        self.table.setHorizontalHeaderLabels(["Train ID", "Departure Time", "Arrival Time", "Route"]
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_schedule)
        self.timer.start(5000)  # Update every 5 seconds
        layout = QVBoxLayout()
        layout.addWidget(self.table)
        self.setLayout(layout)

    def update_schedule(self):
        # Simulate real-time updates
        pass
```

—

2. Ticket Booking System (10 Marks)

- Implement a form for users to book tickets (fields: Train ID, Passenger Name, Seat Number). - Validate user input and display error messages for invalid data. - Use ‘QMessageBox‘ to confirm successful bookings.

Solution:

```
from PyQt5.QtWidgets import QLineEdit, QPushButton, QMessageBox, QFormLayout, QWidget

class TicketBooking(QWidget):
    def __init__(self):
        super().__init__()
```

```
        self.train_id = QLineEdit(self)
        self.passenger_name = QLineEdit(self)
        self.seat_number = QLineEdit(self)
        self.book_button = QPushButton("Book Ticket", self)
        self.book_button.clicked.connect(self.book_ticket)
        layout = QFormLayout()
        layout.addRow("Train ID:", self.train_id)
        layout.addRow("Passenger Name:", self.passenger_name)
        layout.addRow("Seat Number:", self.seat_number)
        layout.addRow(self.book_button)
        self.setLayout(layout)

    def book_ticket(self):
        if not self.train_id.text() or not self.passenger_name.text() or not self.seat_number.text()
            QMessageBox.warning(self, "Error", "All fields are required!")
        else:
            QMessageBox.information(self, "Success", "Ticket booked successfully!")
```

—

3. Route Management (10 Marks)

- Implement a ‘QTreeWidget‘ to display train routes hierarchically (e.g., Station A → Station B → Station C). - Allow users to add, edit, and delete routes. - Use ‘QGraphicsView‘ to visualize the route map.

Solution:

```
from PyQt5.QtWidgets import QTreeWidget, QTreeWidgetItem, QGraphicsView, QGraphicsScene, QVBoxLayout

class RouteManagement(QWidget):
    def __init__(self):
        super().__init__()
        self.tree = QTreeWidget(self)
        self.tree.setHeaderLabels(["Route"])
        self.scene = QGraphicsScene(self)
        self.view = QGraphicsView(self.scene, self)
        layout = QVBoxLayout()
        layout.addWidget(self.tree)
        layout.addWidget(self.view)
        self.setLayout(layout)
```

—

4. Real-Time Updates (5 Marks)

- Use ‘QThread‘ to simulate real-time updates (e.g., train delays, cancellations). - Display notifications in a ‘QStatusBar‘.

Solution:

```
from PyQt5.QtCore import QThread, pyqtSignal

class UpdateThread(QThread):
    update_signal = pyqtSignal(str)

    def run(self):
        while True:
            self.update_signal.emit("Train delayed by 10 minutes.")
            self.msleep(10000)  # Simulate delay
```

—

5. User Authentication (5 Marks)

- Implement a login system using ‘QDialog‘. - Store user credentials securely using ‘QSettings‘.

Solution:

```
from PyQt5.QtWidgets import QDialog, QLineEdit, QPushButton, QFormLayout
from PyQt5.QtCore import QSettings

class LoginDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.username = QLineEdit(self)
        self.password = QLineEdit(self)
        self.login_button = QPushButton("Login", self)
        self.login_button.clicked.connect(self.login)
        layout = QFormLayout()
        layout.addRow("Username:", self.username)
        layout.addRow("Password:", self.password)
        layout.addRow(self.login_button)
        self.setLayout(layout)

    def login(self):
        settings = QSettings("MyCompany", "RailwayApp")
        if (self.username.text() == settings.value("username") and
            self.password.text() == settings.value("password")):
            self.accept()
        else:
            self.reject()
```

—