GreenFleet System Implementation
Software Architecture and Design

Name and Student Identification: Arfaz Hossain, V00984826

1 Introduction

This document presents the implementation of the GreenFleet ride-sharing platform's critical components using the Builder Pattern for notifications and the Singleton Pattern for real-time updates. The implementation is done in Python, focusing on scalability, adaptability, and reliability requirements.

A video with explanation can be found here: https://youtu.be/UOyq_dCpXGQ

2 Design Patterns Overview

2.1 Class Diagrams

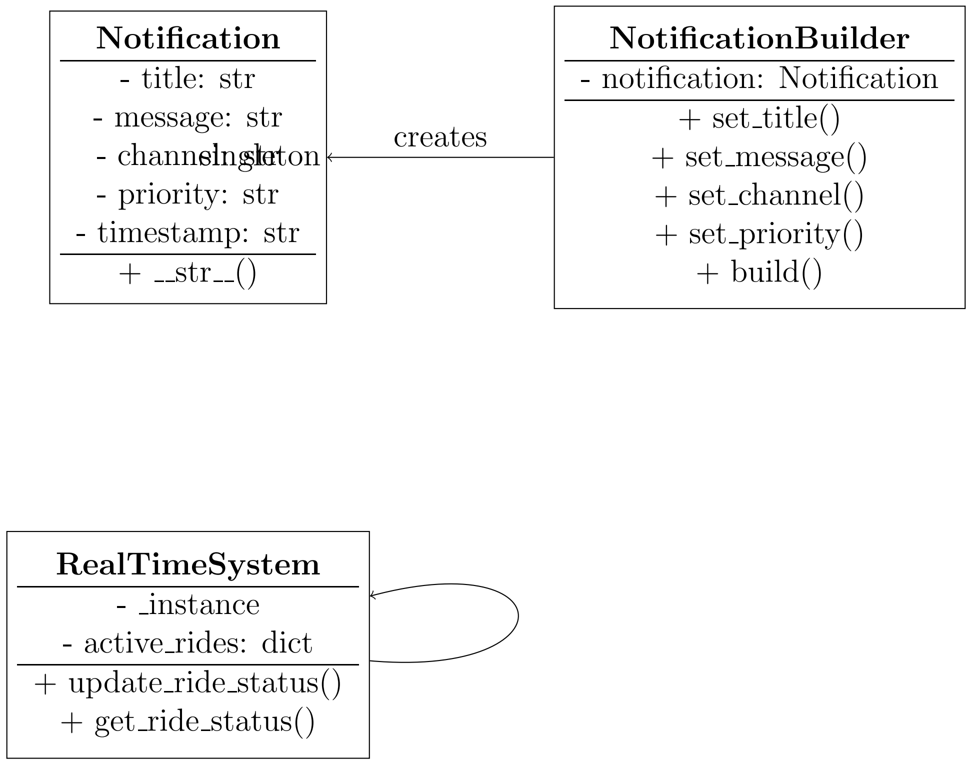

Figure 1: System Architecture Class Diagram

3 Implementation Details

3.1 Notification System - Builder Pattern

```python
class Notification:
    def __init__(self):
        self.title = ""
        self.message = ""
        self.channel = ""
        self.priority = ""
        self.timestamp = ""

    def __str__(self):
        return f"""
[Notification Details]
Title: {self.title}
Message: {self.message}
Channel: {self.channel}
Priority: {self.priority}
Timestamp: {self.timestamp}
"""

class NotificationBuilder:
    def __init__(self):
        self.notification = Notification()

    def set_title(self, title: str):
        print(f"[Builder] Setting notification title: {title}")
        self.notification.title = title
        return self

    def set_message(self, message: str):
        print(f"[Builder] Setting notification message: {message}")
        self.notification.message = message
        return self

    def set_channel(self, channel: str):
        print(f"[Builder] Setting notification channel: {channel}")
        self.notification.channel = channel
        return self

    def set_priority(self, priority: str):
        print(f"[Builder] Setting notification priority: {priority}")
        self.notification.priority = priority
        return self

    def build(self):
        self.notification.timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        print("[Builder] Building notification...")
        return self.notification
```

Listing 1: Notification and Builder Implementation

3.2 Real-Time System - Singleton Pattern

```python
class RealTimeSystem:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            print("[System] Creating new RealTimeSystem instance")
            cls._instance = super(RealTimeSystem, cls).__new__(cls)
            cls._instance.active_rides = {}
        return cls._instance

    def update_ride_status(self, ride_id: str, status: str):
        print(f"[RealTime] Updating ride {ride_id} status to: {status}")
        self.active_rides[ride_id] = status
        print(f"[RealTime] Current active rides: {self.active_rides}")
    def get_ride_status(self, ride_id: str) -> str:
        status = self.active_rides.get(ride_id, "Not Found")
        print(f"[RealTime] Retrieved status for ride {ride_id}: {status}")
        return status
```

Listing 2: RealTimeSystem Singleton Implementation

4 Execution Results

This section presents the results of executing the GreenFleet system's implemented code. These results demonstrate the functionality and interaction of the Builder and Singleton patterns in the context of the GreenFleet system.

4.1 Real-Time System (Singleton)

```
[Test] Testing Real-time System (Singleton):
[System] Creating new RealTimeSystem instance
[Test] Are instances the same? True
[RealTime] Updating ride RIDE001 status to: Driver En Route
[RealTime] Current active rides: {'RIDE001': 'Driver En Route'}
[RealTime] Updating ride RIDE002 status to: Ride in Progress
[RealTime] Current active rides: {'RIDE001': 'Driver En Route', 'RIDE002': 'Ride in Prog
[RealTime] Retrieved status for ride RIDE001: Driver En Route
[Test] Getting status: Driver En Route
```

4.2 Notification Builder

```
[Test] Testing Notification Builder:
[Builder] Setting notification title: Driver Arriving Soon
[Builder] Setting notification message: Your driver John will arrive in 3 minutes
[Builder] Setting notification channel: SMS
[Builder] Setting notification priority: High
[Builder] Building notification...

[Notification Details]
Title: Driver Arriving Soon
Message: Your driver John will arrive in 3 minutes
Channel: SMS
Priority: High
Timestamp: 2024-12-03 20:59:15

[Builder] Setting notification title: New Ride Request
[Builder] Setting notification message: Passenger waiting at Downtown Station
[Builder] Setting notification channel: Push
[Builder] Setting notification priority: Medium
```

```
[Builder] Building notification...

[Notification Details]
Title: New Ride Request
Message: Passenger waiting at Downtown Station
Channel: Push
Priority: Medium
Timestamp: 2024-12-03 20:59:15
```

5 Pattern Benefits and Justification

5.1 Builder Pattern Benefits

- **Step-by-Step Construction**: Allows creation of notifications with mandatory and optional parameters
- **Method Chaining**: Enables fluent interface for notification creation
- **Immutability**: Final notification object is immutable once built
- **Flexibility**: Easy to add new notification attributes or channels

5.2 Singleton Pattern Benefits

- **Single Instance**: Ensures only one RealTimeSystem instance exists
- **Global State**: Maintains consistent ride status across the system
- **Resource Management**: Prevents multiple instances of status tracking
- **Thread Safety**: Built-in Python GIL helps manage concurrent access

6 Usage Example

The system can be tested using the provided main function, which demonstrates:

- Creating different types of notifications using the Builder pattern
- Verifying Singleton pattern behavior for the RealTimeSystem
- Updating and retrieving ride statuses
- Interaction between notification and real-time systems