****

**SUBJECT SDA:**

**SUBMITTED BY**

**NAME TAHIR ALI**

**REG NO: FA22-BSE-050**

**SECTION: 5A**

**FINAL LAB**

**SUBMITTED TO:**

**SIR MUKHTIAR ZAMIN**

**Principles Followed in the Car Booking System**

The Car Booking System uses the **Client-Server Architecture Style**, adhering to several software engineering principles for better design, scalability, and maintainability.

**1. Separation of Concerns (SoC)**

* **Definition**: This principle ensures that different parts of the system handle distinct responsibilities.
* **Implementation in the System**:
  + **Client**: Handles user interactions (e.g., displaying options, taking input, and sending requests to the server).
  + **Server**: Processes client requests, manages business logic, and responds with appropriate results.
  + **Car Controller**: Manages all logic related to car availability, booking, and cancellation.
  + **Car Model**: Encapsulates the attributes of cars, such as id, name, type, and price.

**2. Single Responsibility Principle (SRP)**

* **Definition**: Every class or module should have one specific responsibility.
* **Implementation in the System**:
  + The ClientApp class handles user input/output but does not include business logic.
  + The ServerApp class listens for client connections and forwards commands to appropriate controllers.
  + The CarController class focuses on car management tasks, like booking and cancellation.

**3. Encapsulation**

* **Definition**: Classes hide their internal state and only expose necessary functionality through methods.
* **Implementation in the System**:
  + The Car class encapsulates its properties (e.g., id, name, type, price) and provides controlled access via getter methods.
  + The client does not directly access or manipulate server-side data but communicates through a defined protocol.

**4. Loose Coupling**

* **Definition**: Components should have minimal dependencies on one another.
* **Implementation in the System**:
  + The client and server communicate using predefined commands (VIEW\_CARS, BOOK\_CAR, CANCEL\_CAR), ensuring the server can evolve independently of the client.
  + Business logic changes in the server do not impact the client.

**5. Scalability**

* **Definition**: The system should handle increased load gracefully.
* **Implementation in the System**:
  + The server listens on a specific port and is designed to handle multiple clients in the future (by adding threading or asynchronous handling).

**6. Maintainability**

* **Definition**: The system should be easy to update and extend.
* **Implementation in the System**:
  + Clear separation of functionality (client, server, controller, and model) ensures that updates (e.g., adding a database) can be made with minimal impact on other components.

**7. Reusability**

* **Definition**: Components should be reusable in different contexts.
* **Implementation in the System**:
  + The CarController logic is independent of the server-client communication, making it reusable in other applications.

**8. Modularity**

* **Definition**: The system should be divided into distinct modules for better organization.
* **Implementation in the System**:
  + Different packages are used for the client (package client;), server (package server;), controllers (package controllers;), and models (package models;).





