BSc (Hons) in Information Technology

Object Oriented Concepts – IT1050

Assignment 2

Topic	:	Online Customer Support System
Group no	:	MLB_04.02_07
Campus	:	Malabe
Submission Date	:	20/10/2024

We declare that this is our own work and this Assignment does not incorporate without acknowledgment any material previously submitted by anyone else in SLIIT or any other university/Institute. And we declare that each one of us equally contributed to the completion of this Assignment.

Registration No	Name	Contact Number
IT23631410	B G S H Dilshara	0763777579
IT23775442	T G C Theekshana	0716984376
IT23637832	D T Samaranayaka	0719999214
IT23588714	K H S Dinsara	0703287271
IT23656574	G H C R Saumya	0729563036

BSc (Hons) in Information Technology

Object Oriented Concepts – IT1050

Assignment 2

Contents

BSc (Hons) in Information Technology

Object Oriented Concepts – IT1050

Assignment 2

01.Description of the requirements

- The website can be entered without the need to create an account.
- It is possible to see the stock of cars with basic information.
- It is possible to allow partial access to features.
- Guests log in to become users.
- The login is secure and can use any of social media authentication or username/password.
- After-login features are also introduced.
- User's personalized dashboard for viewing saved cars and managing preferences
- Real-time availability, financing alternatives, full information on cars.
- The ability to use support features
- There is immediate access to an automated chat service for brief questions.
- Provide details of the car, warranty, expected date of delivery, etc.
- The chatbot will be integrated with either an AI model or knowledge base for specific answers in a very short amount of time.
- The user can call a representative to get customized assistance.
- Users get help with comprehensive questions and guide them through the purchase process.
- Calls are tracked and recorded for quality control and future use.
- Administrator's dashboard to monitor and regulate the events taking place on the website.
- The security tools for the platform, updating car listings, and handling the users' accounts
- Tracking in real time each of the Users' actions and the performance of the platform.
- The discovery of valued clients is done with the help of interactions and purchase data.
- Creating and launching customized marketing campaigns with automation.
- Data Analytics tool for observing consumer preference and behavior.
- Secure management of user information and transactions.
- Respect for legislation on the protection

02.Identified classes

- Guest
- User
- Vehicle
- ChatAssistant
- CallAssistant
- Admin
- MarketingManager
- AuthenticationService
- PaymentProcessor
- NotificationService
- inquiry

03.CRC cards

Guest	
Responsibilities	**Collaborations**
Browse the vehicle inventory with limited information	Vehicle
Register to become a user	User
View basic details about available vehicles	

User	
Responsibilities	**Collaborations**
Log in to access personalized features	AuthenticationService
Explore detailed vehicle information	Vehicle
Save vehicles for future reference	Vehicle
Initiate chat or call assistance	ChatAssistant, CallAssistant
Complete vehicle purchase and manage transactions	PaymentProcessor
Receive notifications for offers and updates	NotificationService

Vehicle	
Responsibilities	**Collaborations**
Store and manage detailed information on vehicles	Admin
Display vehicle information to users and guests	
Update availability based on purchases	PaymentProcessor
Provide data to chat and call assistance services	ChatAssistant, CallAssistant

Chat_Assistant	
Responsibilities	**Collaborations**
Respond to user queries instantly	
Retrieve vehicle information to assist users	Vehicle
Provide warranty and delivery details	
Escalate queries to call assistance when needed	CallAssistant

Call_Assistant	
Responsibilities	**Collaborations**
Facilitate live calls and provide personalized support	User
Guide users through the purchase process	
Record and log call details for quality assurance	

Admin	
Responsibilities	**Collaborations**
Monitor and manage website operations	
Update vehicle listings and availability	Vehicle
Maintain platform security and manage accounts	AuthenticationService
Track and record user activity	
Collaborate with marketing to identify high-value users	MarketingManager

Marketing_Manager	
Responsibilities	**Collaborations**
Identify high-value customers based on purchase data	
Send tailored offers and manage campaigns	NotificationService
Analyze customer behavior and segment users	
Manage loyalty programs and maintenance offers	Vehicle, User

Authentication_Service	
Responsibilities	**Collaborations**
Authenticate users during login and registration	User, Admin
Manage secure handling of user credentials	Admin
Log authentication attempts for security monitoring	Admin

Payment_Processor	
Responsibilities	**Collaborations**
Handle payments for vehicle purchases	
Manage financing options	Vehicle
Validate and process payment details	Admin
Record transaction history	

Notification_Service	
Responsibilities	**Collaborations**
Send purchase confirmation and offer notifications	User, MarketingManager
Manage user notification preferences	
Integrate with marketing for personalized messaging	MarketingManager
Alert admins of platform activity and security issues	

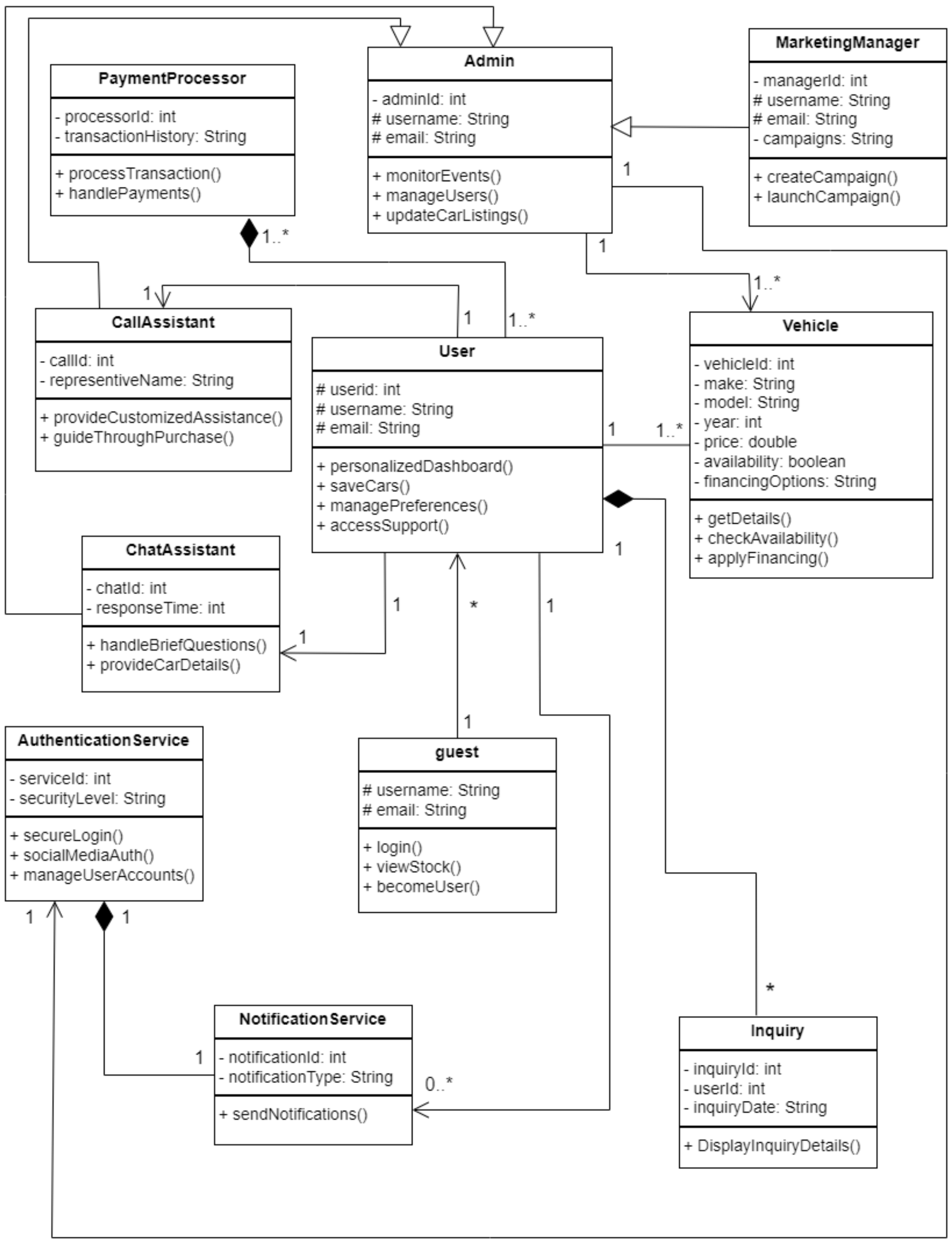
Inquiry	
Responsibilities	**Collaborations**
Track Inquiry Information	
Display Inquiry Details	NotificationService
Notify Relevant Parties	

BSc (Hons) in Information Technology

Object Oriented Concepts – IT1050

Assignment 2

04.Class diagram



05.Coding for the classes

Main.cpp

```
#include <iostream>
#include "Admin.h"
#include "User.h"
#include "MarketingManager.h"
#include "CallAssistant.h"
#include "ChatAssistant.h"
#include "PaymentProcessor.h"
#include "Vehicle.h"
#include "Guest.h"
#include "Inquiry.h"
#include "NotificationService.h"
#include "AuthenticationService.h"

using namespace std;

int main() {
    // Create objects
    Admin* admin1 = new Admin("Admin001", "Saman Kumara", "admin@example.com");

    User* user1 = new User("User001", "Namal Perera", "user@example.com");

    MarketingManager* marketingManager = new MarketingManager("Mngr001", "mngr123",
"marketing@example.com");

    CallAssistant* callAssistant = new CallAssistant("Call001", "Alice Smith");

    ChatAssistant* chatAssistant = new ChatAssistant("Chat001", 2.5);

    PaymentProcessor* paymentProcessor = new PaymentProcessor("PayProc001", "mmmmmmmm");

    Vehicle* vehicle1 = new Vehicle("V001", "India", "Toyota Camry", 2022, 20000000,
"Available", "jrryrtyt");

    Guest* guest1 = new Guest("Guest", "guest@example.com");    );

    Inquiry* inquiry1 = new Inquiry("Inquiry001", "User001", "20024/11/02");

    NotificationService* notificationService = new NotificationService("Notification001",
"Info");

    AuthenticationService* authService = new AuthenticationService("Auth001", "High");


    // Call methods on the objects
    admin1->manageUsers();
    marketingManager->createCampaign();
    callAssistant->provideCustomizedAssistance();
    chatAssistant->handleBriefQuestions();
    paymentProcessor->processPayment(100.0);
    vehicle1->displayVehicleDetails();
    guest1->askInquiry(inquiry1->getInquiryDetails());
    notificationService->sendNotification("New vehicles are available!");
    authService->secureLogin("user@example.com", "password");


    // Clean memory
    delete admin1;
    delete user1;
    delete marketingManager;
    delete callAssistant;
    delete chatAssistant;
    delete paymentProcessor;
    delete vehicle1;
    delete guest1;
    delete inquiry1;
    delete notificationService;
    delete authService;


    return 0;

}
```

Admin.h

```
#pragma once
#include <iostream>
#include <cstring>
#include "Vehicle.h"
#include "AuthenticationService.h"

class Admin {
protected:
    char adminId[10];
    char username[50];
    char email[50];

    // Uni-directional association with Vehicle and AuthenticationService
    Vehicle* vehicle;
    AuthenticationService* authService;

public:
    // Constructors
    Admin();
    Admin(const char pAdminId[], const char pUsername[], const char pEmail[], Vehicle*
pVehicle, AuthenticationService* pAuthService);

    void monitorEvent();
    void manageUsers();
    void updateCarListing();

    // Destructor
    ~Admin();
};
```

Admin.cpp

```
#include "Admin.h"

// Default constructor
Admin::Admin() {
    strcpy_s(adminId, "");
    strcpy_s(username, "");
    strcpy_s(email, "");
    vehicle = nullptr;
    authService = nullptr;
}

// Constructor with parameters
Admin::Admin(const char pAdminId[], const char pUsername[], const char pEmail[], Vehicle* 
pVehicle, AuthenticationService* pAuthService) {
    strcpy_s(adminId, pAdminId);
    strcpy_s(username, pUsername);
    strcpy_s(email, pEmail);

    vehicle = pVehicle; // Associate with a Vehicle object
    authService = pAuthService; // Associate with an AuthenticationService object
}

void Admin::monitorEvent() {

}
void Admin::manageUsers() {

}
void Admin::updateCarListing() {

}

// Destructor
Admin::~Admin() {
    // Clean up if needed
}
```

BSc (Hons) in Information Technology

Object Oriented Concepts – IT1050

Assignment 2

MarketingManager.h

```
#pragma once
#include "Admin.h"  // Inheritance from Admin class
#include <vector>
#include <string>

class MarketingManager : public Admin {
private:
    char managerId[10];
    std::vector<std::string> campaigns;

public:
    // Constructors
    MarketingManager();
    MarketingManager(const char pAdminId[], const char pUsername[], const char pEmail[], 
const char pManagerId[]);

    // Public methods
    void createCampaign(const std::string& campaignName);
    void launchCampaign();

    // Destructor
    ~MarketingManager();
};
```

MarketingManager.cpp

```
#include "MarketingManager.h"
#include <iostream>
#include <cstring>
using namespace std;

// Default constructor
MarketingManager::MarketingManager() {
    strcpy_s(managerId, "");
}

// Constructor with parameters
MarketingManager::MarketingManager(const char pAdminId[], const char pUsername[], const char 
pEmail[], const char pManagerId[]){
    strcpy_s(managerId, pManagerId);


}

void MarketingManager::createCampaign(const std::string& campaignName) {

}
```

```
void MarketingManager::launchCampaign() {

}

// Destructor

MarketingManager::~MarketingManager() {

}
```

CallAssistant.h

```
#pragma once
#include "Admin.h"

class CallAssistant : public Admin {
private:
    char callId[10];
    char representativeName[50];

public:
    // Constructors
    CallAssistant();
    CallAssistant(const char pCallId[], const char pRepresentativeName[], const char
pAdminId[], const char pUsername[], const char pEmail[]);

    // Public methods
    void provideCustomizedAssistance();
    void guideThroughPurchase();

    // Destructor
    ~CallAssistant();
};
```

CallAssistant.cpp

```
#include "CallAssistant.h"
#include <iostream>
#include <cstring>

// Default constructor
CallAssistant::CallAssistant() {
    strcpy_s(callId, "");
    strcpy_s(representativeName, "");
}



// Constructor with parameters
CallAssistant::CallAssistant(const char pCallId[], const char pRepresentativeName[], const
char pAdminId[], const char pUsername[], const char pEmail[])
     {
```

```
    strcpy_s(callId, pCallId);
    strcpy_s(representativeName, pRepresentativeName);
}

void CallAssistant::provideCustomizedAssistance() {

}
void CallAssistant::guideThroughPurchase() {

}

// Destructor
CallAssistant::~CallAssistant() {

}
```

ChatAssistant.h

```
#pragma once
#include "Admin.h"

class ChatAssistant : public Admin {
private:
    char chatId[10];
    double responseTime;

public:
    // Constructors
    ChatAssistant();
    ChatAssistant(const char pChatId[], double pResponseTime, const char pAdminId[], const
char pUsername[], const char pEmail[]);


    void handleBriefQuestions();
    void provideCarDetails();

    // Destructor
    ~ChatAssistant();
};
```

ChatAssistant.cpp

```
#include "ChatAssistant.h"
#include <iostream>
#include <cstring>

// Default constructor
ChatAssistant::ChatAssistant() {
    strcpy_s(chatId, "");
    responseTime = 0.0;
}



// Constructor with parameters
ChatAssistant::ChatAssistant(const char pChatId[], double pResponseTime, const char
pAdminId[], const char pUsername[], const char pEmail[])
```

```
    {
    strcpy_s(chatId, pChatId);
    this->responseTime = pResponseTime;
}

void ChatAssistant::handleBriefQuestions() {

}
void ChatAssistant::provideCarDetails() {

}

// Destructor
ChatAssistant::~ChatAssistant() {

}
```

User.h

```
#pragma once
#include <string>
#include "Vehicle.h"
#include "ChatAssistant.h"
#include "NotificationService.h"
#include "CallAssistant.h"
#include "Inquiry.h"

#define size 2

class User {
private:
    std::string userId;
    std::string username;
    std::string email;

    // Bidirectional relationship with Vehicle
    Vehicle* vehicle[size];

    // Unidirectional relationships
    ChatAssistant* chatAssistant;
    NotificationService* notificationService;
    CallAssistant* callAssistant;

    // Composition relationship
    Inquiry* inquiry[size];



public:
    // Constructors
    User();
    User(const std::string& id, const std::string& name, const std::string& mail,
ChatAssistant* pChatAssistant, NotificationService* pNotificationService, CallAssistant*
pCallAssistant);

    void personalizedDashboard();
    void saveCar();
    void managePreference();
    void accessSupport();

    // Composition relationship
    void addInquiry(const std::string& inquiryId, const std::string& userId, const
std::string& inquirtDate);
    void displayInquiryDetails();

    // Destructor
    ~User();

};
```

User.cpp

```
#include "User.h"
#include <iostream>

using namespace std;

// Default constructor
User::User() {
    userId = "";
    username = "";
    email = "";
}

// Parameterized constructor
User::User(const std::string& id, const std::string& name, const std::string& mail,
ChatAssistant* pChatAssistant, NotificationService* pNotificationService, CallAssistant*
pCallAssistant) {
    userId = id;
    username = name;
    email = mail;
    chatAssistant = pChatAssistant;
    notificationService = pNotificationService;
    callAssistant = pCallAssistant;
}

void User::personalizedDashboard() {

}
void User::saveCar() {

}


void User::managePreference() {

}
```

```
void User::accessSupport() {

}

//Composition Relationship with Inquiry

void User::addInquiry(const std::string& inquiryId, const std::string& userId, const
std::string& inquirtDate) {
    inquiry[0] = new Inquiry(inquiryId, userId, inquirtDate);
    inquiry[1] = new Inquiry(inquiryId, userId, inquirtDate);
}
void User::displayInquiryDetails() {

}

// Destructor
User::~User() {

}
```

PaymentProcessor.h

```
#pragma once
#include <string>
#include <vector>
#include "User.h"

#define size 2

class PaymentProcessor {
private:
    std::string processorId;
    std::vector<std::string> transactionHistory;

    // Composition relationship
    User* user[size];

public:
    // Constructors
    PaymentProcessor();
    PaymentProcessor(const std::string& pId);

    void processTransaction();
    void handlePayment();


    // Composition relationship
    void makePayment(const std::string& id, const std::string& name, const std::string&
mail);
    void displayPaymentDetails();


 // Destructor
    ~PaymentProcessor();
};
```

PaymentProcessor.cpp

```
#include "PaymentProcessor.h"
#include <iostream>

using namespace std;

// Default constructor
PaymentProcessor::PaymentProcessor() {
    processorId = "";
}

// Parameterized constructor
PaymentProcessor::PaymentProcessor(const std::string& pId){
    processorId = pId;
}

void PaymentProcessor::processTransaction() {

}


void PaymentProcessor::handlePayment() {

}

//Composition Relationship with Payment
void PaymentProcessor::makePayment(const std::string& id, const std::string& name, const
std::string& mail) {

    user[0] = new User(id,name,mail);
    user[1] = new User(id, name, mail);

}

// Destructor
PaymentProcessor::~PaymentProcessor() {

}
```

BSc (Hons) in Information Technology

Object Oriented Concepts – IT1050

Assignment 2

Vehicle.h

```cpp
#pragma once
#include <string>
#include "User.h"

#define size 2

class Vehicle {
private:
    std::string vehicleId;
    std::string make;
    std::string model;
    int year;
    double price;
    bool availability;
    std::string financingOptions;

    User* user[size]; // Bidirectional relationship with User

public:
    // Constructors
    Vehicle();
    Vehicle(const std::string& vId, const std::string& vMake, const std::string& vModel, int
vYear, double vPrice, bool vAvailability, const std::string& vFinancingOptions);

    // Public methods
    void getDetails();
    bool checkAvailability();
    void applyFinancing(const std::string& option);

    // Destructor
    ~Vehicle();
};
```

Vehicle.cpp

```cpp
#include "Vehicle.h"
#include <iostream>

using namespace std;

// Default constructor
Vehicle::Vehicle() {
    vehicleId = "";
    make = "";
    model = "";
    year = 0;
    price = 0.0;
    availability = false;
    financingOptions = "";
}

// Parameterized constructor
Vehicle::Vehicle(const std::string& vId, const std::string& vMake, const std::string&
vModel, int vYear, double vPrice, bool vAvailability, const std::string& vFinancingOptions)
{
    vehicleId = vId;
    make = vMake;
    model = vModel;
    year = vYear;
    price = vPrice;
    availability = vAvailability;
    financingOptions = vFinancingOptions;
}

// Method to display vehicle details
void Vehicle::getDetails() {

}
bool Vehicle::checkAvailability() {
    return availability;
}
void Vehicle::applyFinancing(const std::string& option) {

}

// Destructor

Vehicle::~Vehicle() {

}
```

Guest.h

```
#pragma once
#include <string>
#include "User.h"

class Guest {
private:
    std::string username;
    std::string email;

    //Uni-direction Relationship with User class
    User* user1;

public:
    // Constructors
    Guest();
    Guest(const std::string& uname, const std::string& mail, User* pUser1);
    void logIn();
    void viewStock();
    User* becomeUser();
// Destructor
    ~Guest();
};
```

Guest.cpp

```
#include "Guest.h"
#include <iostream>

using namespace std;

// Default constructor
Guest::Guest() {
    username = "";
    email = "";
}

// Parameterized constructor
Guest::Guest(const std::string& uname, const std::string& mail, User* pUser1) {
    username = uname;
    email = mail;
    user1 = pUser1;
}
void Guest::logIn() {

}
void Guest::viewStock() {

}
User* Guest::becomeUser() {

}

// Destructor
Guest::~Guest() {
}
```

Inquiry.h

```
#pragma once
#include <string>

class Inquiry {
private:
    std::string inquiryId;
    std::string userId;
    std::string inquiryDate;

public:
    // Constructors
    Inquiry();
    Inquiry(const std::string& inqId, const std::string& uId, const std::string& inqDate);
// Public method
    void displayInquiryDetails() const;

    // Destructor
    ~Inquiry();
};
```

Inquiry.cpp

```
#include "Inquiry.h"
#include <iostream>

using namespace std;

// Default constructor
Inquiry::Inquiry() {
    inquiryId = "";
    userId = "";
    inquiryDate = "";
}

// Parameterized constructor
Inquiry::Inquiry(const std::string& inqId, const std::string& uId, const std::string&
inqDate) {
    inquiryId = inqId;
    userId = uId;
    inquiryDate = inqDate;
}

void Inquiry::displayInquiryDetails() const {

}

// Destructor
Inquiry::~Inquiry() {

}
```

NotificationService.h

```
#pragma once
#include <string>
#include "AuthenticationService.h"

#define size 2

class NotificationService {
private:
    std::string notificationId;
    std::string notificationType;
    // Composition relationship
    AuthenticationService* authService[size];


public:
    // Constructors
    NotificationService();
    NotificationService(const std::string& nId, const std::string& nType);

    void sendNotification();


    // Composition relationship
    void authenticationService(const std::string& serviceId, const std::string&
sequrityLevel);

    // Destructor
    ~NotificationService();
};
```

NotificationService.cpp

```
#include "NotificationService.h"
#include <iostream>

using namespace std;

// Default constructor
NotificationService::NotificationService() {
    notificationId = "";
    notificationType = "";
}

// Parameterized constructor
NotificationService::NotificationService(const std::string& nId, const std::string& nType) {
    notificationId = nId;
    notificationType = nType;
}

void NotificationService::sendNotification() {

}

//Composition Relationship
void NotificationService::authenticationService(const std::string& serviceId, const
std::string& sequrityLevel) {

    authService[0] = new AuthenticationService(serviceId, sequrityLevel);
    authService[1] = new AuthenticationService(serviceId, sequrityLevel);

}


// Destructor
NotificationService::~NotificationService() {

}
```

BSc (Hons) in Information Technology

Object Oriented Concepts – IT1050

Assignment 2

AuthenticationService.h

```
#pragma once
#include <iostream>
#include "NotificationService.h"
#include <cstring>


#define size 2

class AuthenticationService {
private:
    char serviceId[10];
    char securityLevel[20];

    NotificationService* notificationService1;

public:
    // Constructors
    AuthenticationService();
    AuthenticationService(const char pServiceId[], const char pSecurityLevel[]);

    // Public methods
    void secureLogin();
    void socialMediaAuth();
    void manageUserAccounts();

    // Destructor
    ~AuthenticationService();
};
```

AuthenticationService.cpp

```
#include "AuthenticationService.h"

// Default constructor
AuthenticationService::AuthenticationService() {
    strcpy_s(serviceId, "");
    strcpy_s(securityLevel, "");
}

// Constructor with parameters
AuthenticationService::AuthenticationService(const char pServiceId[], const char
pSecurityLevel[]) {
    strcpy_s(serviceId, pServiceId);
    strcpy_s(securityLevel, pSecurityLevel);
}


void AuthenticationService::secureLogin() {
}

void AuthenticationService::socialMediaAuth() {

}
void AuthenticationService::manageUserAccounts() {

}

// Destructor
AuthenticationService::~AuthenticationService() {

}
```

Individual contribution

Registration No	Name	Contribution
IT23631410	B G S H Dilshara	Call Assistant class Admin class
IT23775442	T G C Theekshana	Inquiry class Guest class
IT23637832	D T Samaranayaka	Chat Assistant class Notification class
IT23588714	K H S Dinsara	Marketing Manager class Payment Process class User class
IT23656574	G H C R Saumya	Authentication class Vehicle class