UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SESSION 2023/2024 SEM 2
SECJ1023 Programming
Technique II

PROJECT TITLE:
SLEEP CYCLE ANALYZER SYSTEM

TASK:
PROJECT DELIVERABLE 4
PROJECT FINALE

LECTURER:
Dr. Lizawati binti Mi Yusuf

Group Members:

Name	Matrix No
NUR SYAKIRAH ADILAH BINTI AZRI	A23CS0159
YEO WERN MIN	A23CS0285
VIBHUSHA A/P SAMPASIVA RAO	A23CS0194
JELIZA JUSTINE A/P SEBASTIN	A21EC0034

Section A:

Introduction

The Sleep Analyzer System is an advanced platform designed to track, monitor, analyze, and enhance users' sleep patterns with the awareness of the importance of sleep for overall health and well-being, the system offers a comprehensive solution. It aims to provide individuals with deep insights into their sleep habits, facilitating informed decision to improve sleep quality. By leveraging data-driven insights, the Sleep Analyzer System aims to provide personalized recommendations and actionable plans tailored to each user's unique sleep profile. This report outlines the functionalities and benefits of the system, emphasizing its role in empowering users to achieve optimal sleep health.

Purpose of the System

The primary purpose of the Sleep Analyzer System is to assist users in understanding, managing, and enhancing their sleep quality. By offering detailed analysis and personalized recommendations based on individual sleep data, the system aims to:

- Track and monitor sleep patterns.
- Provide insights into sleep stages and overall sleep quality.
- Enable users to make informed decisions regarding sleep habits and routines.
- Offer actionable plans for improving sleep health based on data-driven insights.

The Sleep Analyzer System aims to empower users to take proactive steps towards achieving better sleep quality through these functionalities. By having excellent sleep quality, it will contribute to their overall health and quality of life.

Flowchart

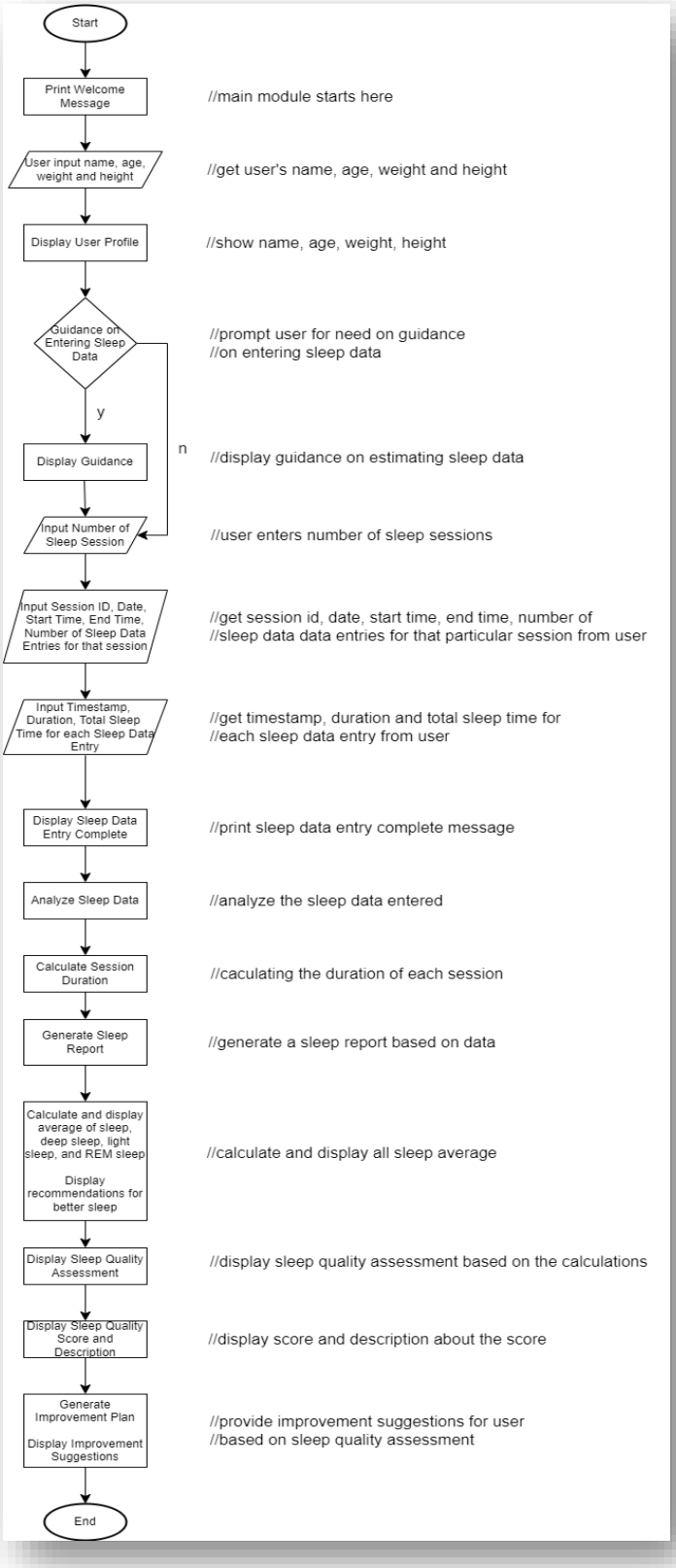

Figure 1: Flowchart of Sleep Analyzer System

The flowchart begins with the “Start” symbol, prompting the user to input personal details like name, age, weight, and height. Once entered, the system retrieves this information. Next, the user enters sleep data, including session ID, date, start time, end time, and the number of sleep data entries for each session. The loop continues until all sessions are recorded. The system calculates session duration and total sleep time. Finally, it generates a report with average sleep duration and quality scores based on the data entered.

After analyzing the sleep data, the system assesses sleep quality. It displays the assessment and provides improvement suggestions. These recommendations aim to enhance sleep quality by addressing factors like sleep duration, deep sleep, and REM sleep. The flowchart concludes with the “End” symbol, signifying the completion of the process.

UML class diagram

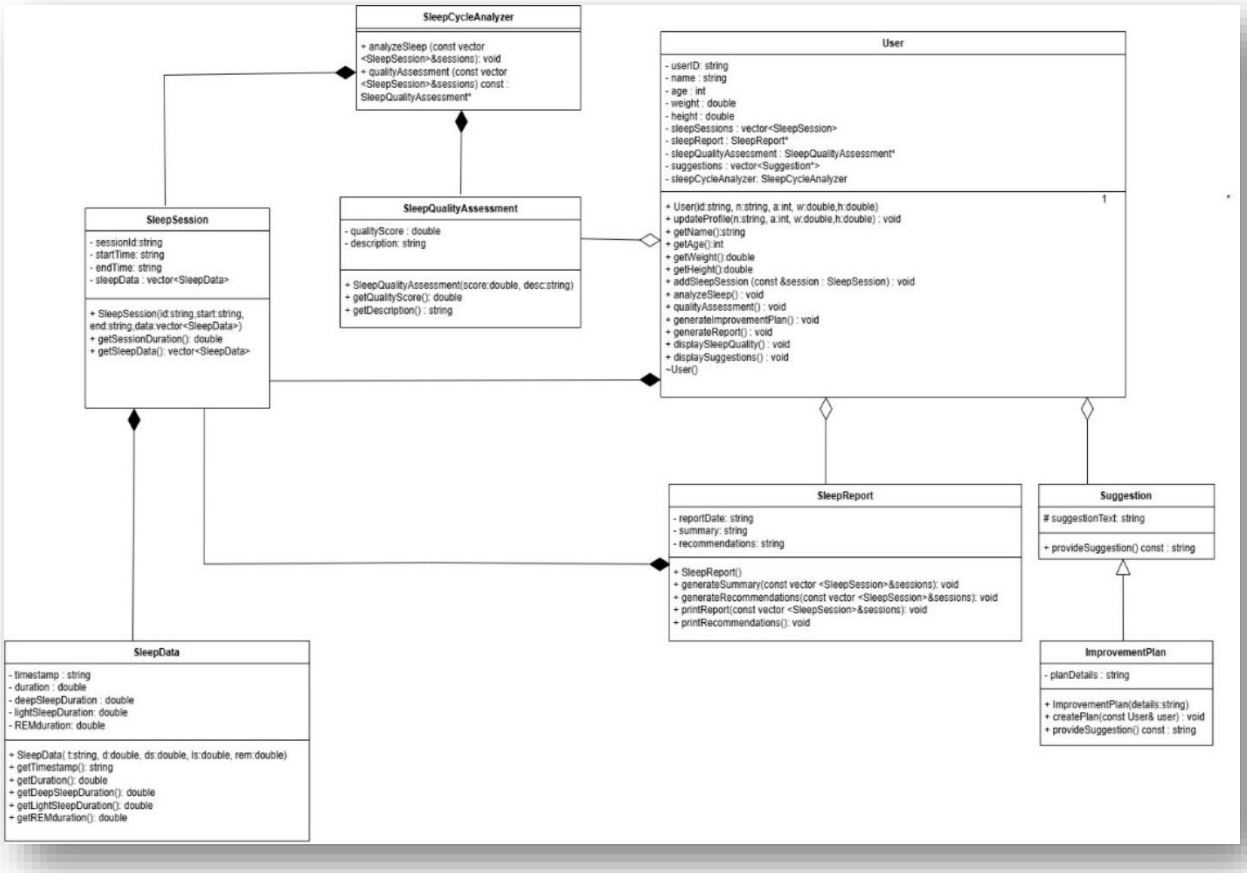

Figure 2: UML Diagram of Sleep Cycle Analyzer system

Section B: Implementation of the Concepts

Module 1: Sleep Data

1.Encapsulation

SleepData class encapsulates sleep-related data with private member variables such as such as timestamp, duration, deepSleepDuration,lightSleepDuration and REMduration and provides public methods including a constructor like SleepData and methods like getTimestamp, getDuration, getDeepSleepDuration, getLightSleepDuration, getREMduration to access these variables, ensuring that the internal state of the object can only be modified through its methods.

```
//Module 1 :  Sleep Data
// Sleep Data Class
class SleepData {
private:
    string timestamp;
    double duration;
    double deepSleepDuration;
    double lightSleepDuration;
    double REMduration;

public:
    SleepData(string t, double d, double ds, double ls, double rem)
        : timestamp(t), duration(d), deepSleepDuration(ds), lightSleepDuration(ls), REMduration(rem) {}

    string getTimestamp() const { return timestamp; }
    double getDuration() const { return duration; }
    double getDeepSleepDuration() const { return deepSleepDuration; }
    double getLightSleepDuration() const { return lightSleepDuration; }
    double getREMduration() const { return REMduration; }
};
```

Figure 3: class SleepData

2. Composition

The SleepSession class contains a vector of SleepData objects that spans multiple segments within a single sleep period. This indicates that a sleep session is composed of multiple sleep data entries, but SleepData objects can exist independently of SleepSession. For example, methods in this class such as getSessionDuration will operate on the contained SleepData objects to calculate the total duration of the sleep session.

```cpp
// Sleep Session Class
class SleepSession {
private:
    string sessionId;
    string startTime;
    string endTime;
    vector<SleepData> sleepData;

public:
    SleepSession(string id, string start, string end, vector<SleepData> data)
        : sessionId(id), startTime(start), endTime(end), sleepData(data) {}

    double getSessionDuration() const {
        double totalDuration = 0;
        for (const auto& data : sleepData) {
            totalDuration += data.getDuration();
        }
        return totalDuration;
    }

    vector<SleepData> getSleepData() const { return sleepData; }
};
```

Figure 4: class SleepSession

Module 2: Sleep Analysis

1.Encapsulation

The SleepQualityAssessment class encapsulates the data related to sleep quality, namely qualityScore and description. These attributes are kept private, ensuring they cannot be directly accessed or modified from outside the class. Instead, the class provides public getter methods (getQualityScore and getDescription) to allow controlled access to these attributes. This encapsulation enforces data integrity and hides the internal representation of sleep quality, only exposing relevant information to the users of the class.

```
//Module 2 : Sleep Analysis
// Sleep Quality Assessment Class
class SleepQualityAssessment {
private:
    double qualityScore;
    string description;

public:
    SleepQualityAssessment(double score, string desc)
        : qualityScore(score), description(desc) {}

    double getQualityScore() const { return qualityScore; }
    string getDescription() const { return description; }
};
```

Figure 5: class SleepQualityAssessment

The SleepCycleAnalyzer class encapsulates the logic for analyzing sleep data and assessing sleep quality. It contains methods that process sleep sessions (analyzeSleep) and compute a quality assessment (qualityAssessment). By keeping these methods within the class, the implementation details are hidden from the user. Users interact with the class through its public methods, ensuring that the underlying processing logic remains hidden and protected.

```cpp
// Sleep Cycle Analyzer Class
class SleepCycleAnalyzer {
public:
    void analyzeSleep(const vector<SleepSession>& sessions) {
        cout << "Analyzing sleep data..." << endl;
        for (const auto& session : sessions) {
            cout << "Session Duration: " << fixed << setprecision(2) << session.getSessionDuration() << " hours" << endl;
        }
    }

    SleepQualityAssessment* qualityAssessment(const vector<SleepSession>& sessions) const {
        double totalScore = 0;
        int dataCount = 0;

        for (const auto& session : sessions) {
            for (const auto& data : session.getSleepData()) {
                double sessionQuality = 0;
                sessionQuality += data.getDeepSleepDuration() * 1.5;
                sessionQuality += data.getLightSleepDuration();
                sessionQuality += data.getREMduration() * 1.2;
                totalScore += sessionQuality;
                ++dataCount;
            }
        }

        double averageScore = (dataCount > 0) ? (totalScore / dataCount) : 0;
        string description;

        if (averageScore >= 8.0) {
            description = "Excellent";
        } else if (averageScore >= 6.0) {
            description = "Good";
        } else if (averageScore >= 4.0) {
            description = "Fair";
        } else {
            description = "Poor";
        }

        return new SleepQualityAssessment(averageScore, description);
    }
};
```

Figure 6: class SleepCycleAnalyzer

2. Composition

The SleepCycleAnalyzer class demonstrates composition by using the SleepQualityAssessment class within its qualityAssessment method. After analyzing the sleep data, it creates a new SleepQualityAssessment object to represent the results. This use of composition shows a "has-a" relationship, where the SleepCycleAnalyzer class includes and utilizes instances of the SleepQualityAssessment class to provide comprehensive functionality. This approach promotes modular design and code reuse, as the SleepQualityAssessment class can be used independently or as part of other classes.

Module 3 : Sleep Summary

1. Encapsulation

The SleepReport class encapsulates the report date, summary, and recommendations as private member variables and the public member functions getReportDate(), getSummary(), and getRecommendations() provide a controlled interface to access these private members. This helps to organize code into logical units, making it easier to understand and maintain.

```
class SleepReport {
private:
    string reportDate;
    string summary;
    vector<string> recommendations;

public:
    SleepReport(string date, string sum, vector<string> recs) {
        reportDate = date;
        summary = sum;
        recommendations = recs;
    }

    string getReportDate() { return reportDate; }
    string getSummary() { return summary; }
    vector<string> getRecommendations() { return recommendations; }
};
```

Figure 7 : Code snippet from class SleepReport

The use of auto also can be seen as a form of encapsulation. By using auto, we're encapsulating the type of the loop variable session within the compiler's type deduction mechanism. The type of session is not explicitly specified, which means the implementation details of the type are hidden from the rest of the code. Apart from that. the loop variable session is decoupled from the specific type of the container sleepSessions. Thus, this makes it easier to change the type of the container without affecting the loop logic.

```
1 for (auto& session : user->sleepSessions) {
```

Figure 8 : Code Snippet from class sleepSession

2. Inheritance

The ImprovementPlan class inherits from the Suggestion class, which means it inherits the provideSuggestion() method. The ImprovementPlan class overrides this method to provide its own implementation. This helps to establish a hierarchy of classes from suggestion to improvementPlan, making it easier to understand the relationships between them.

```
class Suggestion {
public:
    virtual string provideSuggestion() = 0;
};

class ImprovementPlan : public Suggestion {
private:
    string planDetails;

public:
    ImprovementPlan(string details) {
        planDetails = details;
    }

    string provideSuggestion() override {
        return "Improve your sleep with " + planDetails;
    }
};
```

Figure 9 : Code Snippet from Suggestion and ImprovementPLan class

3.Polymorphism

The Suggestion class provides a virtual method provideSuggestion(), which is overridden by the ImprovementPlan class. The User class has a method displaySuggestions() that takes a vector of Suggestion objects, which can be of type ImprovementPlan or any other subclass of Suggestion. In this case, Polymorphism helps to reduce coupling between classes, making it easier to change one class without affecting others.

```
class Suggestion {
public:
    virtual string provideSuggestion() = 0;
};

class ImprovementPlan : public Suggestion {
public:
    string provideSuggestion() override {
        return "Improve your sleep with a consistent schedule";
    }
};

class User {
public:
    void displaySuggestions(vector<Suggestion*> suggestions) {
        for (Suggestion* suggestion : suggestions) {
            cout << suggestion->provideSuggestion() << endl;
        }
    }
};

int main() {
    User user;
    vector<Suggestion*> suggestions;
    suggestions.push_back(new ImprovementPlan());

    user.displaySuggestions(suggestions);

    return 0;
```

Figure 10 : Code Snippet from Suggestion, ImprovementPlan, User, and Main class

4.Composition

Composition can be seen in the SleepReport class through its interaction with SleepSession objects. To generate detailed sleep report, the SleepReport class is using these objects. The generateSummary method iterates through a vector of SleepSession objects to calculate total and average sleep durations, including deep sleep, light sleep, and REM sleep. Similarly, the generateRecommendations method analyzes the average sleep duration from these SleepSession objects to generate personalized sleep recommendations. The printReport method calls both generateSummary and generateRecommendations to compile a comprehensive sleep report based on the data from SleepSession objects. This composition relationship shows that the SleepReport class "has" SleepSession objects, which means it relies on these objects to complete its responsibilities effectively.

```
void generateSummary(const vector<SleepSession>& sessions) {
    summary = "Summary of sleep report:\n";
    double totalDuration = 0;
    double totalDeepSleep = 0;
    double totalLightSleep = 0;
    double totalREM = 0;

    for (const auto& session : sessions) {
        for (const auto& data : session.getSleepData()) {
            totalDuration += data.getDuration();
            totalDeepSleep += data.getDeepSleepDuration();
            totalLightSleep += data.getLightSleepDuration();
            totalREM += data.getREMduration();
        }
```

```
void generateRecommendations(const vector<SleepSession>& sessions) {
 recommendations = "Recommendations for better sleep:\n";
```

```
void printReport(const vector<SleepSession>& sessions) {
    generateSummary(sessions);
    generateRecommendations(sessions);
    cout << "Sleep Report - " << reportDate << endl;
    cout << summary << endl;
    cout << recommendations << endl;
}
```

Figure 11 : Code Snippet from SleepReport class

Module 4 : User

1. Encapsulation

The User class encapsulates user attributes which is userID, name, age, weight, and height and methods like updateProfile, addSleepSession, analyzeSleep, and generateReport to manipulate these attributes. The private attributes are accessed and modified through public methods, ensuring control over the data.

```
class User {
private:
    string userID;
    string name;
    int age;
    double weight;
    double height;
    vector<SleepSession> sleepSessions;
    SleepReport* sleepReport;
    SleepQualityAssessment* sleepQualityAssessment;
    vector<Suggestion*> suggestions;
    SleepCycleAnalyzer sleepCycleAnalyzer;

public:
    User(string id, string n, int a, double w, double h)
        : userID(id), name(n), age(a), weight(w), height(h), sleepQualityAssessment(nullptr), sleepReport(nullptr) {}

    void updateProfile(string n, int a, double w, double h) {
        name = n;
        age = a;
        weight = w;
        height = h;
    }

    string getName() const { return name; }
    int getAge() const { return age; }
    double getWeight() const { return weight; }
    double getHeight() const { return height; }
```

Figure 12 : Code Snippet from User class

2. Composition

The SleepSession is directly contains instances of SleepData object vector<SleepData>. This collection represents detailed data about the sleep session, including timestamps, durations of different sleep stages (deep sleep, light sleep, and REM). The SleepData instances are tightly bound to their respective SleepSession instances means that when a SleepSession is created, its associated SleepData instances are also initialized. Conversely, when the SleepSession is destroyed, all its contains SleepData instances are automatically destroyed as well. Reflecting a strong ownership relationship where the whole, SleepSession is composed of its parts, SleepData.

```
vector<SleepSession> sleepSessions;
```

Figure 13 : Code Snippet from User class

The SleepCycleAnalyzer is an instance of SleepCycleAnalyzer directly embedded within the User class. It performs analysis on the sleep sessions owned by the User.

```
SleepCycleAnalyzer sleepCycleAnalyzer;
```

Figure 14: Code Snippet from User class

3. Aggregation

The User class aggregated a SleepReport object pointer. This represents that a User has an associated SleepReport that contains summary information and recommendations.

```
SleepReport* sleepReport;
```

Figure 15: Code Snippet from User class

The User class aggregates a pointer to a SleepQualityAssessment object, indicating that the user's sleep quality can be assessed and updated independently.

```
SleepQualityAssessment* sleepQualityAssessment;
```

Figure 16 : Code Snippet from User class

The vector<Suggestion*>suggestion in the User class can be considered as aggregation. The User class contains Suggestion objects, but they can exist independently of the User class.

```
vector<Suggestion*> suggestions;
```

Figure 17 : Code Snippet from User class

4. Array of Objects

The User class contains a vector<SleepSession> which is an array of SleepSession objects. This allows the User to manage multiple sleep sessions.

```
class User {
private:
```

```
vector<SleepSession> sleepSessions;
```

```
void addSleepSession(const SleepSession& session) {
    sleepSessions.push_back(session);
}
```

Figure 18 : Code Snippet from User class

References

1. Yilmaz, D., Tanrikulu, F., & Dikmen, Y. (2017). Research on sleep quality and the factors affecting the sleep quality of the nursing students. PubMed, 43(1), 20–24. https://doi.org/10.12865/chsj.43.01.03
   - Taken for sleep quality score calculation in the SleepQualityAssessment class.

2. Ms, J. L. (2024, January 18). How much deep, light, and REM sleep do you need? Healthline. https://www.healthline.com/health/how-much-deep-sleep-do-you-need#stages-of-sleep
   - The article provides information on the recommended ranges for sleep duration, deep sleep, light sleep, and REM sleep. The information is used in the ImprovementPlan class in the createPlan method to provide suggestions for improving sleep quality.

3. Patel, A. K., Reddy, V., Shumway, K. R., & Araujo, J. F. (2024b, January 26). Physiology, sleep stages. StatPearls - NCBI Bookshelf. https://www.ncbi.nlm.nih.gov/books/NBK526132/
   - The article provides information on the different sleep stages and their typical duration. This information is used for sleep stages percentage calculation in the main function.