***SMESTER PROJECT***

***GROUP MEMBERS:***

***RAO ALI AHMAD (223710)***

***M SAJJAD (223631)***

***PROJECT NAME***



***SUBMITTED TO:***

***PROF, HAFIZ MUHAMMAD ANAS ALI***

#include <iostream>

#include <string>

#include <vector>

#include <conio.h> // For getch()

using namespace std;

class Post {

private:

string name, address, street, description, date;

double quantity, price, availableUnits;

string accountNumber, ownerName, bankName;

bool paymentMade;

public:

// Constructor

Post(string name, string address, string street, string description, string date,

double quantity, double price, string accountNumber, string ownerName, string bankName)

: name(name), address(address), street(street), description(description),

date(date), quantity(quantity), price(price), availableUnits(quantity),

accountNumber(accountNumber), ownerName(ownerName), bankName(bankName),

paymentMade(false) {}

// Getters and Setters

string getName() { return name; }

string getAddress() { return address; }

string getDescription() { return description; }

double getAvailableUnits() { return availableUnits; }

double getPrice() { return price; }

string getOwnerName() { return ownerName; }

string getBankName() { return bankName; }

string getAccountNumber() { return accountNumber; }

void subtractUnits(double units) {

if (units <= availableUnits) {

availableUnits -= units;

} else {

cout << "Insufficient units available." << endl;

}

}

void makePayment(double requestedUnits) {

double totalPayment = requestedUnits \* price;

cout << "Payment for " << requestedUnits << " units successful!" << endl;

cout << "Total Amount Paid: " << totalPayment << endl;

cout << "Bank: " << bankName << ", Account: " << accountNumber << endl;

paymentMade = true;

}

void printDetails() {

cout << "Name: " << name << endl

<< "Address: " << address << endl

<< "Street: " << street << endl

<< "Description: " << description << endl

<< "Date: " << date << endl

<< "Quantity: " << quantity << endl

<< "Price per Unit: " << price << endl

<< "Available Units: " << availableUnits << endl

<< "Owner Name: " << ownerName << endl

<< "Bank: " << bankName << ", Account: " << accountNumber << endl;

}

};

class User {

private:

vector<string> requestHistory;

vector<double> paymentHistory;

public:

void viewAvailablePosts(vector<Post>& posts) {

if (posts.empty()) {

cout << "No posts available." << endl;

return;

}

for (size\_t i = 0; i < posts.size(); ++i) {

cout << "Post #" << i + 1 << ":" << endl;

posts[i].printDetails();

cout << "--------------------------" << endl;

}

}

void requestUnits(vector<Post>& posts) {

if (posts.empty()) {

cout << "No posts available to request units from." << endl;

return;

}

int postIndex;

double requestedUnits;

cout << "Enter the post number you want to request units from: ";

cin >> postIndex;

if (postIndex < 1 || postIndex > posts.size()) {

cout << "Invalid post number." << endl;

return;

}

cout << "Enter the number of units you want to request: ";

cin >> requestedUnits;

Post& selectedPost = posts[postIndex - 1];

if (requestedUnits <= selectedPost.getAvailableUnits()) {

selectedPost.subtractUnits(requestedUnits);

cout << "Request successful! You requested " << requestedUnits << " units." << endl;

cout << "Total Payment: " << requestedUnits \* selectedPost.getPrice() << endl;

requestHistory.push\_back(selectedPost.getName());

paymentHistory.push\_back(requestedUnits \* selectedPost.getPrice());

} else {

cout << "Insufficient units available." << endl;

}

}

void makePayment(vector<Post>& posts) {

if (posts.empty()) {

cout << "No posts available for payment." << endl;

return;

}

int postIndex;

double requestedUnits;

cout << "Enter the post number for payment: ";

cin >> postIndex;

if (postIndex < 1 || postIndex > posts.size()) {

cout << "Invalid post number." << endl;

return;

}

cout << "Enter the number of units you want to pay for: ";

cin >> requestedUnits;

Post& selectedPost = posts[postIndex - 1];

selectedPost.makePayment(requestedUnits);

}

};

class Owner:public User {

private:

vector<Post> posts;

public:

void addPost() {

string name, address, street, description, date, accountNumber, ownerName, bankName;

double quantity, price;

cout << "Enter post details:" << endl;

cin.ignore();

cout << "Name: ";

getline(cin, name);

cout << "Address: ";

getline(cin, address);

cout << "Street: ";

getline(cin, street);

cout << "Description: ";

getline(cin, description);

cout << "Date: ";

getline(cin, date);

cout << "Quantity: ";

cin >> quantity;

cout << "Price per Unit: ";

cin >> price;

cin.ignore();

cout << "Account Number: ";

getline(cin, accountNumber);

cout << "Owner Name: ";

getline(cin, ownerName);

cout << "Bank Name: ";

getline(cin, bankName);

Post newPost(name, address, street, description, date, quantity, price, accountNumber, ownerName, bankName);

posts.push\_back(newPost);

cout << "Post added successfully!" << endl;

}

void viewPosts() {

if (posts.empty()) {

cout << "No posts available." << endl;

return;

}

for (size\_t i = 0; i < posts.size(); ++i) {

cout << "Post #" << i + 1 << ":" << endl;

posts[i].printDetails();

cout << "--------------------------" << endl;

}

}

vector<Post>& getPosts() { return posts; }

};

class SystemManager {

private:

Owner owner;

User user;

public:

void menu() {

int choice;

while (true) {

cout << "1. Owner Menu" << endl;

cout << "2. User Menu" << endl;

cout << "3. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1) {

ownerMenu();

} else if (choice == 2) {

userMenu();

} else if (choice == 3) {

cout << "Exiting..." << endl;

break;

} else {

cout << "Invalid choice. Try again." << endl;

}

}

}

void ownerMenu() {

int choice;

while (true) {

cout << "1. Add Post" << endl;

cout << "2. View Posts" << endl;

cout << "3. Back to Main Menu" << endl;

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1) {

owner.addPost();

} else if (choice == 2) {

owner.viewPosts();

} else if (choice == 3) {

break;

} else {

cout << "Invalid choice. Try again." << endl;

}

}

}

void userMenu() {

int choice;

while (true) {

cout << "1. View Available Posts" << endl;

cout << "2. Request Units" << endl;

cout << "3. Make Payment" << endl;

cout << "4. Back to Main Menu" << endl;

cout << "Enter your choice: ";

cin >> choice;

vector<Post>& posts = owner.getPosts();

if (choice == 1) {

user.viewAvailablePosts(posts);

} else if (choice == 2) {

user.requestUnits(posts);

} else if (choice == 3) {

user.makePayment(posts);

} else if (choice == 4) {

break;

} else {

cout << "Invalid choice. Try again." << endl;

}

}

}

};

int main() {

SystemManager systemManager;

systemManager.menu();

return 0;

}

1. ***Goal-Question-Metric (GQM) Approach:***

**Goal 1: Enhance User Experience**

* **Questions**

**1.** How intuitive and user-friendly is the GreenShare app for both solar energy producers and consumers?

**2.** To what extent does the app meet the needs and expectations of its target audience?

* **Metrics:**
  1. **User Satisfaction Score (NPS):** Measure overall user satisfaction and identify areas for improvement.
  2. **App Store Ratings and Reviews:** Track user feedback and sentiment expressed through app store ratings and reviews.
  3. **Time to Task Completion:** Measure the time it takes users to complete key tasks within the app (e.g., posting energy for sale, requesting energy, making payments).

**Goal 2: Maximize Energy Sharing Transactions**

* **Question**

**1.** How effectively does the app facilitate successful peer-to-peer energy trading between users?

**2.** What are the key factors influencing user participation in energy sharing transactions?

* **Metrics:**
  1. **Number of successful energy sharing transactions:** Track the total number of successful transactions completed within a given timeframe.
  2. **Transaction volume:** Measure the total amount of energy shared through the platform.
  3. **User participation rate:** Track the percentage of registered users actively participating in energy sharing transactions.

**Goal 3: Ensure System Reliability and Security**

* **Questions**

**1.** How reliable and secure is the GreenShare app and its underlying infrastructure?

**2.** How effectively does the app protect user data and prevent unauthorized access?

* **Metrics:**
* **System uptime:** Track the availability and uptime of the app and its backend systems.
* **Number of security incidents:** Monitor the number of reported security breaches, data breaches, or unauthorized access attempts.
* **User trust and confidence:** Gauge user trust and confidence in the app's security measures through surveys and feedback.

1. ***Halstead Metrics:***

 **n1:** Number of unique operators = 20

 **N1:** Total number of operators = 80

 n**2:** Number of unique operands = 30

 **N2:** Total number of operands = 100

**Calculated Metrics:**

* **Program Length (N):**

N=N1+N2

N=80+100

N=180

* **Program Vocabulary (n):**

n=n1+ n2

n=20+30

n=50

* **Volume (V):**

V = N \* log2(n)

V=180\* log2(50)

V= 1015.89

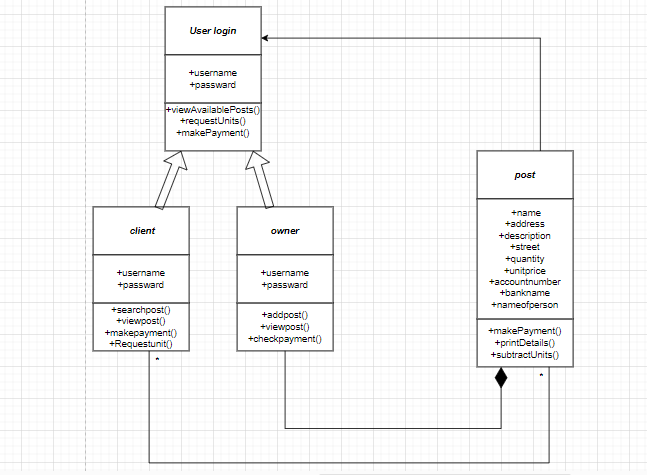
* **Difficulty (D):**

D = ( n1 / 2 ) \* ( N2 / n2 )D=(20/2)\*(100/30)

D=33.33

**Effort (E):** E= V\*D , E=33,863.14

1. ***Object-Oriented Metrics:***



**Weighted Methods per Class (WMC)**

WMC user login =3

WMC client =4

WMC owner =3

WMC post=3

**Average Weighted Methods per Class (WMC)**

AWMC***=***TOTAL CLASSES WMC / TOTAL CLASS

AWMC***=13/4***

AWMC***=3.25***

***Depth of Inheritance Tree (DIT)*** DIT of userlogin is =0

DIT of owner is = 1

DIT of client is =1

DIT of post is =0

***Number of Children (NoC)*** NoC of userlogin is =2

NoC of owner is = 0

NoC of client is =0

NoC of post is =0

***Coupling Between Objects (CBO)***

|  |  |  |
| --- | --- | --- |
| **Class** | **Coupled class** | **CBO** |
| **Post** | None | 0 |
| **User** | Post | 1 |
| **Owner** | Post | 1 |
| **SystemManager** | Owner,User | 2 |

***4. McCabe Cyclomatic Complexity:***

Calculate the complexity for key functions individually and aggregate the results.

1. **SystemManager::menu()**

* 1 while loop.
* 3 if-else if blocks.

CC = 1 (baseline) + 1 (while) + 3 (conditions) = 5

**2. SystemManager::ownerMenu()**

* 1 while loop.
* 3 if-else if blocks.  
  CC = 1 (baseline) + 1 (while) + 3 (conditions) = 5

**3. SystemManager::userMenu()**

* 1 while loop.
* 4 if-else if blocks.  
  CC = 1 (baseline) + 1 (while) + 4 (conditions) = 6

**4. User::viewAvailablePosts()**

* 1 if block to check for empty posts.
* 1 for loop.  
  CC = 1 (baseline) + 1 (if) + 1 (for) = 3

**5. User::requestUnits()**

* 1 if block to check for empty posts.
* 1 if block for invalid post numbers.
* 1 if block for sufficient units.  
  CC = 1 (baseline) + 3 (if) = 4

**6. User::makePayment()**

* 1 if block to check for empty posts.
* 1 if block for invalid post numbers.  
  CC = 1 (baseline) + 2 (if) = 3

**7. Owner::addPost()**

* No control structures (only input handling).  
  CC = 1

**8. Owner::viewPosts()**

* 1 if block to check for empty posts.
* 1 for loop.  
  CC = 1 (baseline) + 1 (if) + 1 (for) = 3

**9. Post::subtractUnits()**

* 1 if block to check available units.  
  CC = 1 (baseline) + 1 (if) = 2

**10. Post::makePayment()**

* No control structures.  
  CC = 1

|  |  |
| --- | --- |
| ***Function/Method*** | ***Cyclomatic Complexity*** |
| SystemManager::menu() | ***5*** |
| SystemManager::ownerMenu() | ***5*** |
| SystemManager::userMenu() | ***6*** |
| User::viewAvailablePosts() | ***3*** |
| User::requestUnits() | ***4*** |
| User::makePayment() | ***3*** |
| Owner::addPost() | ***1*** |
| Owner::viewPosts() | ***3*** |
| Post::subtractUnits() | ***2*** |
| Post::makePayment() | ***1*** |

1. ***Traceability Metrics:***

**Identify Requirements**

R1: Users must be able to view available posts.

R2: Owners can add new posts.

R3: Users can request units of posts.

**Map Requirements to Design**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | ***Requirement ID*** |  |  | | --- | |  | | |  | | --- | | ***Requirement Description*** |  |  | | --- | |  | | | ***Design Element*** | | --- |  |  | | --- | |  | |
| ***R1*** | |  | | --- | | Users must view posts. |  |  | | --- | |  | | |  | | --- | | User::viewAvailablePosts() |  |  | | --- | |  | |
| ***R2*** | |  | | --- | | Owners can add posts. |  |  | | --- | |  | | Owner::addPost() |
| ***R3*** | |  | | --- | | *Users can request units of posts.* |  |  | | --- | |  | | User::requestUnits() |

***3.Map Design to Code***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Design Element** |  |  | | --- | |  | | | **Code Component** | | --- |  |  | | --- | |  | |
| |  | | --- | | User::viewAvailablePosts() |  |  | | --- | |  | | |  | | --- | | User::viewAvailablePosts() function |  |  | | --- | |  | |
| |  | | --- | | Owner::addPost() |  |  | | --- | |  | | |  | | --- | | Owner::addPost() function |  |  | | --- | |  | |

***4. Traceability Matrix***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | ***Requirement ID*** |  |  | | --- | |  | | |  | | --- | | ***Requirement Description*** |  |  | | --- | |  | | | ***Design Element*** | | --- |  |  | | --- | |  | | ***Code Component*** |
| ***R1*** | |  | | --- | | Users must view posts. |  |  | | --- | |  | | |  | | --- | | User::viewAvailablePosts() |  |  | | --- | |  | | User::viewAvailablePosts() function |
| ***R2*** | |  | | --- | | Owners can add posts. |  |  | | --- | |  | | Owner::addPost() | Owner::addPost() function |
| ***R3*** | |  | | --- | | *Users can request units of posts.* |  |  | | --- | |  | | User::requestUnits() | User::requestUnits() function |

1. ***Mean Time Between Failures (MTBF):***

As we are not using full working software so Mean Time Between Failures is not applicable in this.

1. ***COCOMO:***

COCOMO calculates **Effort (E)** in **person-months**, **Development Time (T)** in **months**, and estimates **Team Size** and **Cost** based on:

* **Size (KLOC)**: Thousand Lines of Code.
* **Mode**: Determines the project type (Organic, Semi-Detached, or Embedded).

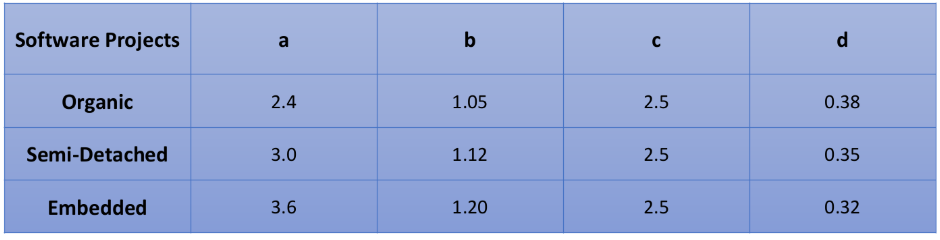
**Effort (E):**

E=a×(KLOC)b

**Development Time (T):**

T=c×(E)d

Constants a, b, c, and d vary based on the project mode:



**Estimating Key Inputs**

**Lines of Code (LOC):**

From the provided code, the approximate LOC is **250 lines**. Thus:

KLOC=250/100=0.25

**Project Type:**

The software is moderately complex with integration between multiple classes (Owner, User, Post). We will classify it as **Semi-Detached**.

**Calculations**

**Effort (E):**

Using Semi-Detached parameters:

E=3.0×(0.25)1.12

E=3.0×0.226=0.678 person-months

**Development Time (T):**

T=2.5×(0.678)0.35

T=2.5×0.855=2.14 months

**Team Size:**

Team Size=E/T

=0.678/2.14

=0.32

**Cost:**

Assume an average monthly developer salary of **$4000**. Then:

Cost=E×Monthly Salary

=0.678×4000

=2712 USD

|  |  |
| --- | --- |
| **Mertic** | **Value** |
| Efforts(Development) | 0.678 person-month |
| Development Time | 2.14 months |
| Team Size | 0.32 (single dev) |
| Development Cost | $2712 USDT |

**Interpretation**

* **Effort (0.678 PM)**: Small project, suitable for a single developer.
* **Cost ($2712)**: Affordable for small-scale projects.

