Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка програмного додатку «Органайзер Healthy Habit»

Студентки 4 курсу, 6 групи,
спеціальності 121 «Інженерія
програмного забезпечення» _____ Солдатенко Юлії
підпис студента Олегівни

Науковий керівник
кандидат технічних наук,
доцент _____ Рзаєва Світлана
підпис керівника Леонідівна

Гарант освітньої програми
кандидат технічних наук,
доцент _____ Цензура Микола
підпис керівника Олександрович

КИЇВ – 2021

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь бакалавр
Спеціальність 121 Інженерія програмного забезпечення

Затверджую
Зав. кафедри інженерії
програмного забезпечення
та кібербезпеки
Криворучко О. В.
"20" жовтня 2020 р.

Завдання
на випускна кваліфікаційна робота студентові

Солдатенко Юлії Олегівни
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи «Розробка програмного додатку «Органайзер Healthy Habit»

Затверджена наказом ректора від "30" жовтня 2020 р. № 3225

2. Строк здачі студентом закінченої роботи

3. Цільова установка та вихідні дані до роботи

Мета роботи аналіз особливостей формування звичок, проектування та розробка програмного додатку «Органайзер Healthy Habit», що полегшить впровадження звичок у повсякденність користувачів завдяки простому інтерфейсу та інтерактивній частині додатку

Об'єкт дослідження додаток «Органайзер Healthy Habit»

Предмет дослідження методи та технології розробки програмного додатку для формування звичок

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ОСОБЛИВОСТЕЙ ФОРМУВАННЯ ЗВИЧОК

1.1. Опис процесу формування звичок

1.2. Теоретичні відомості про впровадження звичок

1.3. Технічне завдання

1.4. Висновки до розділу 1

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ ОРГАНАЙЗЕРУ «HEALTHY HABIT»

2.1 Визначення параметрів та характеристики додатку

2.2 Розробка структури додатку

2.2.1 Взаємодія клієнта з додатком

2.2.2 Структура класів

2.2.3 Діаграма діяльності

2.2.4 Діаграма станів

2.3 Проектування бази даних

2.3.1 Концептуальна модель

2.3.2 Логічна та фізична моделі

2.3.3 Створення таблиць

2.4 Висновки до розділу 2

РОЗДІЛ 3. РОЗРОБКА ДОДАТКУ ОРГАНАЙЗЕРУ «HEALTHY HABIT»

3.1 Розробка додатку та підключення бази даних

3.2 Опис функціональних характеристик додатку «Healthy Habit»

3.3 Висновки до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	21.09.2020	21.09.2020
2.	*Вступ та перелік літературних джерел*	14.12.2020	14.12.2020
3.	*Розділ 1. Аналіз особливостей формування звичок*	19.02.2021	19.02.2021
4.	*Розділ 2. Проектування додатку органайзеру «Healthy Habit»*	05.03.2021	05.03.2021
5.	*Розділ 3. Розробка додатку органайзеру «Healthy Habit»*	10.04.2021	10.04.2021
6.	*Висновки*	24.04.2021	24.04.2021
7.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	14.05.2021	14.05.2021
8.	*Підготовка автореферату та презентації доповіді*	17.05.2021	17.05.2021
9.	*Попередній захист випускної кваліфікаційної роботи*	18.05-2021 –	21.05.2021
10.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	01.06.2021	01.06.2021
11.	*Здача прошитого випускної кваліфікаційної роботи на кафедру*	02.06.2021	02.06.2021
12.	*Публічний захист випускної кваліфікаційної роботи*		

7. Дата видачі завдання «20» жовтня 2020 р.

8. Науковий керівник випускної кваліфікаційної роботи
Рзаєва С. Л.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Цензура М. О.
(прізвище, ініціали, підпис)

10. Завдання прийняла до виконання студентка Солдатенко Ю. О.
(прізвище, ініціали, підпис)

11. Відгук керівника випускної кваліфікаційної роботи

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Науковий керівник випускної кваліфікаційної роботи _____
(підпис, дата)

Відмітка про попередній захист __Рзаєва С. Л._____
(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну робота

Випускна кваліфікаційна робота студентки _____Солдатенко Ю. О._____
(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Цензура М. О._____
(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О.В._____
(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Дипломна робота на тему «Розробка програмного додатку «Органайзер Healthy Habit» містить 45 сторінок, 29 рисунків, 1 формулу та 1 додаток. Перелік посилань налічує 20 найменувань.

Мета дослідження: аналіз особливостей формування звичок, проектування та розробка програмного додатку «Органайзер Healthy Habit», що полегшить впровадження звичок у повсякденність користувачів завдяки простому інтерфейсу та інтерактивній частині додатку.

Об'єкт дослідження: додаток «Органайзер Healthy Habit».

Предметом дослідження є методи та технології розробки програмного додатку для формування звичок.

У першому розділі було проведено аналіз предметної області і особливостей процесу формування звичок, характеристики циклу звички, їх значення у житті людей; складено технічне завдання, де визначено мету, призначення та передумови створення додатку; описано вимоги до додатку.

У другому розділі було спроектовано структуру даних додатку та створено необхідні UML діаграми; досліджено предметну область для розробки концептуальної та логічної моделі даних, на основі яких було створено базу даних використовуючи підхід Code-First у структурі Entity Framework.

У третьому розділі було описано процес реалізації додатку із використанням програмного комплексу Microsoft Visual Studio та мови C#, що відповідає всім встановленим вимогам.

ANNOTATION

The thesis on the topic "Development of the software application «Healthy Habit Organizer»" contains 45 pages, 29 drawings, 1 formula and 1 appendix. The list of links includes 20 sources.

The purpose of the study: analysis of the characteristics of habit formation, design and development of the software application "Organizer Healthy Habit", which will facilitate the implementation of habits in everyday life of users through a simple interface and interactive part of the application.

Research subject: Healthy Habit Organizer application.

The study's subject is the methods and technologies of software development for the habits formation.

The first section analyzes the subject and features of process of habits formation, characteristics of a cycle of a habit, their value in a life of people; the technical task which defines the purpose and preconditions of creation of the application; the requirements for the application are described.

The second section analyzes the data structure of the application, required UML diagrams were created; the subject for the development of a conceptual and logical data model, on the basis of which was created a database using the Code-First approach in the Entity Framework technology.

The third section describes the process of implementing the application using Microsoft Visual Studio software and C # language, which meets all the established requirements.

ЗМІСТ

ВСТУП

Актуальність. Звички – це безперечно потужна частина життя кожної людини. Вони є невід'ємною частиною поведінкової психології, яка формує напрямок життя. Насправді стан і якість життя є прямим відображенням щоденних звичок людини. Покинути шкідливі звички і замінити їх корисними далеко не є простим завданням. Для цього потрібні відданість, сила волі та непохитне бажання подолати природні тенденції думати, відчувати, говорити та діяти певним чином.

Для того щоб досягти успіху у формуванні звичок, необхідно організувати систему, що буде забезпечувати результативність процесу. Трекери звичок допомагають зберігати увагу та відповідальність за поставлені цілі. Багато людей використовує для цього традиційні методи – тобто ручку та папір, але цей метод має багато недоліків.

Кожного дня людство все більше розраховує на сучасні технології та програми через їх простоту, доступність та економію часу. А спростити та зробити цікавим процес формування звичок можна саме завдяки створенню простого у використанні інструменту, який може допомогти користувачам легко впроваджувати корисні звички у своє життя без зайвих зусиль.

Саме тому темою даної випускної кваліфікаційної роботи є Розробка програмного додатку «Органайзер Healthy Habit», що буде слугувати користувачам надійним інструментом для впровадження звичок у повсякдення.

Метою дослідження є аналіз особливостей формування звичок, проектування та розробка програмного додатку «Органайзер Healthy Habit», що полегшить впровадження звичок у повсякденність користувачів завдяки простому інтерфейсу та інтерактивній частині додатку.

Об'єкт дослідження – додаток «Органайзер Healthy Habit».

					КНТЕУ 121 06-21.БР		
Зм	Аркуш	№ докум	Підпис	Дата			
Зав.кафедри		Криворучко О.В		14.12.20	Розробка програмного додатку «Органайзер Healthy Habit»	Стадія	Аркуш	Аркушів
Керівник		Рзаєва С.Л.		14.12.20		В	4	45
Гарант		Цензура М.О.		14.12.20		Факультет інформаційних технологій, 4 курс, 6 група		
Розроб.		Солдатенко Ю.О.		14.12.20	Вступ			

Предмет дослідження – методи та технології розробки програмного додатку для формування звичок.

Завдання дослідження:

- дослідити процес формування звичок, визначити параметри та характеристики додатку;
- спроектувати структуру додатку;
- розробити базу даних;
- із використанням програмного комплексу Microsoft Visual Studio та мови C# реалізувати працюючий додаток.

РОЗДІЛ 1
АНАЛІЗ ОСОБЛИВОСТЕЙ ФОРМУВАННЯ ЗВИЧОК

1.1 Опис процесу формування звичок

Питання здорового способу життя не втрачає актуальності серед людства. Кожен день створюються нові способи бути здоровіше, коли насправді одним із найдієвіших способів є впровадження здорових звичок у повсякденність людини.

Звичка – це поведінка, яка виконується автоматично, тому що її часто виконували в минулому. Подібне повторення створює психологічний зв'язок між ситуацією і дією (поведінкою), що означає, що при виникненні ситуації, дія виконується автоматично. Автоматичність має ряд компонентів, одним з яких є відсутність думок.

Звички відіграють важливу роль у спрощенні життя і зменшенні кількості сенсорних стимулів, які потрібно обробляти людині. Звички рятують енергію, оскільки за своєю природою вони є автоматичними і вимагають мало фізичних і психологічних сил. Тож впровадження корисних звичок є запорукою здорового способу життя та основою для самодисципліни. 45% щоденних звичок людини є автоматичними. Це означає, що одну з кожних двох щоденних звичок, що має людина, вона виконує не замислюючись. [1]

Щоб зрозуміти, як формуються звички, перше, на що слід звернути увагу, - це нейронні шляхи. Нейронний шлях - це серія нейронів, які зв’язані між собою для передачі сигналу від однієї частини мозку до іншої. Щоразу, коли людина дізнається щось нове, створюється новий нервовий шлях. А щоразу, коли вона використовуєте нову інформацію, до нейронного ланцюга зв’язків додається більше нейронів. Це робить нервовий шлях міцнішим, отже, легшим для доступу. Так формуються і звички за допомогою циклу звичок.

Цикл звичок - це модель формування звичок, яку можна застосувати до будь-якої звички. Цей процес у мозку є триступеневим циклом. По-перше, є

ситуація, тобто тригер, який повідомляє мозку перейти в автоматичний режим і яку звичку використовувати. По-друге є рутина, яка може бути фізичною, розумовою чи емоційною. Останньою є винагорода, яка допомагає мозку зрозуміти, чи варто цей цикл запам'ятовувати. З часом цей цикл стає все більш і більше автоматичним і формується звичка.

Тобто модель циклу звички має три основних компоненти:

1. Ситуація/тригер - те, що спричиняє виконання звички. Це може бути ранкова рутина, що починається після спрацювання будильнику. Почувши будильник, людина прокидається, встає з ліжка та переходить до своєї ранкової рутини. В наведеному прикладі тригером є будильник.

Усі звички мають певну форму тригера, незалежно від того, чи має тригер очевидні фізичні ознаки, чи неочевидні психічні ознаки. Іноді необхідно проаналізувати звичку більш детально, щоб мати змогу визначити менш очевидні тригери.

При формуванні звички необхідно спочатку усвідомити тригер, щоб систематизувати процес впровадження надійним чином.

2. Рутина – це і є основа звички, а саме повторення поведінкової дії. Прийняття рішення про виконання рутини визначається наступним важливим компонентом звички – нагородою, або результатом.

3. За виконанням рутини слідує нагорода. Нагорода може проявлятися як просте задоволення після виконання рутини. Наприклад, при виконанні повсякденної рутини чищення зубів вранці, нагородою (або результатом) є відчуття комфорту та свіжості і білі зуби.

Нагорода може бути більш складною, так як деякі результати важко відстежити або виміряти. Якщо взяти за приклад звичку правильно харчуватися, складно визначити як саме різна їжа впливає на здоров'я безпосередньо в процесі. В цьому випадку здоровий результат важко оцінити, так як на нього впливають багато факторів, таких як психологічний стан, емоційний добробут та фізичне самопочуття.

В таких випадках необхідно уважно аналізувати довгостроковий результат та перспективу впливу звички на стан життя в цілому. Також важливо розуміти результат виконання рутини як щось вимірюване, для легшого відстежування ефективності нової звички та щоб не губити мотивацію.

Саме за рахунок нагороди, цикл звички посилюється. Цикл – це самозміцнюючий механізм, який з часом стає автоматичним.

Рис. 1.1. Цикл звички.

Надана модель впровадження звичок була створена Чарльзом Дюїггом і вважається найбільш ефективною. [2]

1.2 Теоретичні відомості про впровадження звичок

Для того щоб зрозуміти, як часто треба виконувати нову звичку для того, щоб вона стала автоматичною, необхідно звернутися до наукових досліджень.

В перші дні космічної програми NASA розробило експеримент, щоб визначити фізіологічні та психологічні наслідки просторової дезорієнтації, яку астронавти зазнають у невагомому космічному середовищі.

В ході експерименту вчені NASA обладнали кожного з астронавтів парою опуклих окулярів, які перевернули все в полі зору на 180 градусів. Астронавтам доводилося носити окуляри цілодобово, сім днів на тиждень, навіть коли вони спали. Спочатку вони відчували фізичні симптоми тривоги та стресу - підвищений артеріальний тиск, дихання та інші життєво важливі показники, але поступово адаптувались. На двадцять перший день експерименту декілька із астронавтів почали бачити світ звичним способом, навіть в окулярах.

Так вчені виявили те, що через 21-30 днів цього безперервного потоку нових надходжень мозок астронавтів насправді створив нейронні шляхи, які переобладнали їх мозок, щоб знову бачити свій світ нормально. Тобто мозку потрібно приблизно тридцять безперервних днів для формування нових нервових зв'язків – тобто для формування нових звичок. [3]

Окрім цього, існує дослідження пластичного хірурга Максвела Мальца. Він зауважив, що для того, щоб люди адаптувалися до нової поведінки або прийняли зміну свого іміджу, потрібно мінімум двадцять один день. Як зазначив сам Мальц, 21-денний часовий проміжок був мінімальним часом для зміни. [4]

Згідно наведеним вище даним, можна зробити висновок, що для формування мозком нової звички необхідно від двадцяти одного дня щоденної практики.

Також для того, щоб змінити звичку, надзвичайно важливо встановити системи, щоб забезпечити дотримання обраної нової поведінки.

Одним із важливих компонентів впровадження звички є відстеження прогресу. Дослідження показали, що люди, які відстежують свій прогрес у цілях впровадження нової звички, мають кращі показники ніж ті, хто цього не робить. Інше дослідження, що було проведено групою людей, показало, що ті, хто щоденно веде журнал харчування, втрачають удвічі більше ваги, ніж ті, хто цього не робив. [5]

Згідно наведеним дослідженням можна зробити висновок, що органайзер для відстеження звичок - це простий спосіб зареєструвати поведінку, а сам факт відстеження поведінки може допомогти викликати бажання змінити її.

1.3 Технічне завдання

1. Загальні відомості

1.1. Найменування системи

1.1.1. Повне найменування системи: Додаток для впровадження звичок «Органайзер Healthy Habit».

1.1.2. Скорочене найменування системи: «Healthy Habit».

1.2. Планові терміни початку та закінчення робіт:

Дата початку робіт: 21 вересня 2020 року.

Дата закінчення робіт: 30 березня 2017 року.

1.3. Головний бенефіціар та потенційні користувачі системи

Потенційними користувачами «Healthy Habit» є жінки та чоловіки віком від 18 років що зацікавлені в особистому розвитку в незалежності від проживання та загального виду зайнятості.

2. Мета та призначення створення системи

2.1. Призначення системи

Створення додатку має на меті спростити користувачу процес впровадження корисних звичок у повсякденне життя. Додаток має бути спроможним виконувати такі задачі:

- авторизування у систему;
- створення звички (введення даних про неї)
- зміна даних про звичку;
- позначення процесу виконання звички;
- видалення звички;
- перегляд інформації про звичку;
- забезпечення захищеного зберігання введених даних.

2.2. Мета створення системи

«Healthy Habit» створюється з метою:

- забезпечення користувачу можливість створювати та відстежувати корисні звички;
- надавати користувачу необхідну інформацію про звички;
- слугувати мотивацією для користувача завдяки інтерактивній частині додатку;
- підвищення продуктивності та покращення рівню життя.

3. Вимоги до системи

- реєстрація облікового запису користувача;
- вхід у обліковий запис користувача;
- редагування облікового запису користувача;
- створення звички (назва, опис, колір, частота, рослина);

- маніпулювання звичкою (зміна даних про звичку, переміщення звички, видалення);
- можливість відмічання звички;
- перегляд архіву звичок;
- перегляд графіку формування звички;
- перегляд інтерактивної частини;
- приємний та зрозумілий у використанні інтерфейс.

4. Вимоги до програмного забезпечення

- операційна система Windows 10

5. Вимоги до технічного забезпечення

- оперативна й зовнішня пам’ять, що будуть достатніми для нормальної роботи базових програмних засобів
- наявність 100 Мб зовнішньої пам’яті.

1.4 Висновки до розділу 1

З отриманої інформації можна інтерпретувати, що звичка - це поведінковий шаблон, який можна розвивати шляхом частого повторення. Крім того, формування звичок може використовуватися для покращення показників поведінкової моделі, яку людина повторює, що позитивно впливає на якість життя.

Згідно досліджень, проведеним автором, для повного формування встановленої звички необхідно щонайменше 21 день. Якщо звичка проста і її можна швидко повторити протягом короткого часу, впровадження звички може зайняти менше часу. В іншому випадку, якщо звичка є більш складною і вимагає тривалої роботи, це може зайняти більше двадцяти одного дня.

У наведеному розділі було досліджено, що люди, які мають успіх у впровадженні корисних звичок, часто вимірюють, кількісно оцінюють і відстежують свій прогрес різними способами. Тому одним із найкращих підходів для впровадження звичок є відстеження прогресу. Це можна робити за допомогою

трекеру звичок, що представить всю необхідну інформацію про виконання звичок та може слугувати мотивацією.

У результаті аналізу наведених даних, можна виявити що головною метою роботи є створення додатку органайзеру «Healthy Habit», що допомагає щоденно відстежувати процес впровадження корисних звичок у життя користувача, використовуючи інтерактивну частину додатку як головну нагороду і мотивацію для роботи.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ДОДАТКУ ОРГАНАЙЗЕРУ «HEALTHY HABIT»

2.1 Визначення параметрів та характеристики додатку

Програми, що можуть покращувати здоровий образ життя людей безумовно є об’ємною частиною цифрового світу. Приблизно 3,4 мільярда людей користуються сучасними технологіями у всьому світі. За оцінками програмної галузі, половина з них користується додатками, що допомагають покращувати здоров’я. [6]

Ринок мобільних додатків для здоров’я протягом останніх кількох років стабільно зростав, але 2020 рік був особливим. Пандемія коронавірусу та карантин змусили людство покладатися на цифрові технології більше, ніж коли-небудь раніше. Саме через це у соціальних мережах, відео-платформах та електронній комерції сильно зросло загальне використання. [7]

Людство почало шукати доступні способи покращити якість свого життя у просторах особистої оселі – деяким карантин змінив робочу рутину та завдав їм стресу, а інші не розуміли що їм робити з вільним часом. Саме тоді почав зростати попит на додатки та програми для здоров’я, що заохочують здоровий спосіб життя, та особливу увагу люди приділили необхідності та важливості корисних звичок та як вони впливають на повсякденність.

Навіть якщо говорити за рамками карантинних обмежень, у будь-який час програми для покращення здоров’я будуть корисними, вони допомагають людям покращувати здорову поведінку, мотивують на позитивні зміни для підвищення добробуту та навіть надають легший доступ до корисної інформації, пов'язаної зі здоров'ям. Тому можна з впевненістю сказати, що подібні програми продовжуватимуть впроваджувати нові інновації, які покращують якість життя на довгі роки.

Провівши аналіз деяких існуючих додатків, які націлені на допомогу у впровадженні користувачеві нових звичок, можна зазначити, що у всіх досліджених додатках відсутнє поняття мотивації користувача під час формування нових корисних звичок, що нажаль змушує багатьох користувачів залишати свій прогрес, і в наслідку це стає на шляху до мети впровадження здорового образу життя.

Необхідно також виділити велику нестачу додатків для персональних комп'ютерів на ринку, майже всі додатки створені для мобільних пристроїв.

Згідно наданим даним можна сформулювати необхідну інформацію для створення додатку органайзеру «Healthy Habit», що допомагає щоденно відстежувати процес впровадження корисних звичок у життя користувача, використовуючи інтерактивну частину додатку як головну нагороду і мотивацію для роботи.

Саме спираючись на дані, що були описані у розділі 1.1, можна зазначити що третій і головний етап циклу звичок це нагорода, яку людина отримує після виконання звички. Згідно дослідженням, багато звичок не впроваджуються мозком, так як людина не має достатній рівень мотивації для продовження, тому що не бачить результативності.

Спираючись на те, що мотивація є одною з базових складових процесу засвоєння звички, у процесі створення додатку «Healthy Habit» було прийнято рішення впровадити інтерактивну систему умовних нагород, що будуть слугувати мотивацією для користувача.

Інтерактивна частина є симуляцією саду з квітами. Дана методика дозволить користувачу візуалізувати свою звичку за допомогою автоматично створеною зі звичкою рослини – йому доведеться «поливати» її щодня, відмічаючи виконання звички, і кожен день спостерігати за тим, як вона росте. Це допоможе користувачу побачити той прогрес виконання звички, який неможливо побачити у реальному житті на перших етапах впровадження, що часто приводить до рішення зупинити процес. По досягненню цілі і успішного засвоєння звички користувач зможе

зберегти рослину і передивлятися статистику звичок, а зрештою створити свій власний сад.

Додаток органайзер «Healthy Habit» націлений на впровадження корисних звичок користувачу за допомогою функцій відстеження виконання звичок та мотивації через інтерактивну частину програми і буде містити в собі наступні можливості: реєстрація та логін у програмі, зміна даних особистого кабінету, створення нової звички, додання опису звички, переміщення (організація) звичок, відстеження виконання звичок кожного дня, видалення звички, створення рослини (частина інтерактивної програми) що репрезентує звичку, сповіщення про успішне або невдале формування звички, перегляд процесу виконання звичок у вигляді графіку, перегляд архіву звичок та рослин.

Робота з додатком буде виконуватись наступним чином: після запуску програми, новому користувачу необхідно створити обліковий запис, вказуючи свої дані для ідентифікації, так як прогрес прив'язується саме до облікового запису. Після успішного логіну, користувачу буде запропоновано створити нову звичку, для цього необхідно натиснути на відповідну кнопку та ввести дані (назва звички, опис звички, колір відображення звички, періодичність виконання звички, рослину для репрезентування звички), після цього створена звичка відобразиться на головному екрані у вигляді назви та полей, які користувач повинен відмічати згідно виконання звички кожного дня. Можливість відмічати виконання звички обмежена до поточної дати. Це зроблено для того, щоб користувач не мав змоги відмітити виконання звички за передню дату, але функція відмічання працює за минулі дати. Для відмітки про виконання звички необхідно просто натиснути на відповідному полі з поточною датою. При досягненні 21 дня безперервних відміток про виконання звички, вона вважається успішно впровадженою, але користувач має змогу продовжити відмічати свій прогрес. Якщо користувач пропустив день або більше, відповідна кількість днів додається до циклу формування звички, що збільшує необхідний час для впровадження. Якщо користувач не відмічав виконання звички 21 день поспіль, звичка вважається не сформованою, і користувачу пропонується почати спочатку або змінити умови

звички. Коли звичка успішно впроваджена, рослина в інтерактивній частині завершує свій зріст та зберігається у архів, де її можна буде переглянути у будь який час, а якщо вона не сформована, рослина навпаки зав'ядає. Протягом усього процесу можна у будь який момент переглянути етап росту рослини, який буде змінюватись згідно прогресу користувача. Також користувач може змінювати особисті дані в обліковому записі, переглядати архів звичок та переглядати графік звичок.

Застосування додатку органайзеру «Healthy Habit» дозволить користувачам заохочувати здоровий спосіб життя та покращувати рівень успішності за допомогою полегшення та організації процесу формування корисних звичок, використовуючи інтерактивну частину як нагороду та підкріплення мотивації.

На основі озвучених даних, додаток органайзер «Healthy Habit» повинен відповідати наступним вимогам:

- Реєстрація облікового запису користувача;
- Вхід у обліковий запис користувача;
- Редагування облікового запису користувача;
- Створення звички (назва, опис, колір, частота, рослина);
- Маніпулювання звичкою (зміна даних про звичку, переміщення звички, видалення);
- Можливість відмічання звички;
- Перегляд архіву звичок;
- Перегляд графіку формування звички;
- Перегляд рослин звичок (інтерактивна частина);
- Приємний та зрозумілий у використанні інтерфейс.

Додаток, що відповідає даним функціональним вимогам, забезпечує виконання визначених цілей дипломної роботи.

2.2 Розробка структури додатку

Оскільки стратегічна цінність програмного забезпечення зростає для багатьох компаній, галузь шукає методи для автоматизації виробництва

програмного забезпечення та покращення якості, зменшення вартості та часу виходу на ринок. Ці методи включають технологію компонентів, візуальне програмування, шаблони та рамки. Підприємства також шукають методи управління складністю систем у міру збільшення їх масштабу та масштабу. Зокрема, вони визнають необхідність вирішення повторюваних архітектурних проблем, таких як фізичний розподіл, паралельність, тиражування, безпека, балансування навантаження та стійкість до відмов. Крім того, розвиток Всесвітньої павутини, хоча спрощує деякі речі, посилює ці архітектурні проблеми. Уніфікована мова моделювання (UML) була розроблена з урахуванням цих потреб.

UML (Unified Modeling Language) - це стандартизована мова моделювання, що складається з інтегрованого набору діаграм, розробленого для допомоги розробникам систем та програмного забезпечення визначення, візуалізації, побудови та документування артефактів програмних систем, а також для бізнес-моделювання. UML є дуже важливою частиною розробки об'єктно-орієнтованого програмного забезпечення та розробки іншого програмного забезпечення. UML використовує переважно графічні позначення для вираження дизайну програмних проектів. Використання UML допомагає командам проектів спілкуватися, досліджувати потенційні проекти та перевіряти архітектурний дизайн програмного забезпечення.

Метою UML є створення стандартних позначень, які можуть використовуватись усіма об'єктно-орієнтованими методами, а також вибір та інтеграція найкращих елементів позначень попередників. UML розроблено для широкого кола програм. Отже, вона забезпечує конструкції для широкого кола систем та видів діяльності (наприклад, розподілених систем, аналізу, проектування та розгортання системи). [8]

У UML існує безліч варіантів різних діаграм (моделей). Причиною цьому є те, що на систему можна поглянути з багатьох різних точок зору. Так як у розробці програмного забезпечення зазвичай бере участь декілька зацікавлених сторін, їх цікавлять різні аспекти системи, і кожна з них вимагає різного рівня

деталізації. Наприклад, програміст повинен розуміти дизайн системи і мати можливість перетворити дизайн на код низького рівня. В той же час, технічний автор цікавиться поведінкою системи в цілому і повинен розуміти, як функціонує продукт. UML надає необхідні інструменти, завдяки яким всі зацікавлені сторони можуть скористатися принаймні однією діаграмою UML.

Рис. 2.1. Види UML діаграм

Структурні діаграми показують статичну структуру системи та її частин на різних рівнях абстракції та реалізації і те, як вони пов’язані між собою. Елементи на структурній діаграмі представляють значущі поняття системи і можуть включати абстрактні, реальні концепції та концепції реалізації. Існує сім типів структурної діаграми: Діаграма класів, діаграма компонентів, діаграма розгортання, діаграма об’єктів, пакетна діаграма, діаграма складеної структури, діаграма профілю. [9]

Діаграми поведінки показують динамічну поведінку об'єктів у системі, що можна описати як серію змін у системі з часом. Існує сім типів діаграм поведінки: діаграма станів, діаграма діяльності, діаграма прецедентів, діаграма послідовності, діаграма кооперації, діаграма взаємодії, діаграма синхронизації.

Протягом останніх кількох років способи проектування діаграм стали дуже складними; будь-яка складна система потребує інструменту для побудови. Таким чином, вибираючи найкращий інструмент, потрібно знати найважливіші його особливості. Сьогодні UML розглядається як фактичний стандарт при розробці

програмного забезпечення і використовується в багатьох сферах, починаючи від наукового моделювання та моделювання бізнесу. [10]

Провівши аналіз інструментів для створення діаграм, було обрано StarUML. Це інструмент програмного забезпечення з відкритим кодом, який підтримує структуру UML для моделювання системи та програмного забезпечення. Він заснований на UML версії 1.4, забезпечує одинадцять різних типів діаграм та приймає позначення UML 2.0. StarUML це простий у використанні інструмент, який підтримує всі необхідні типи діаграм що необхідно створити для додатку «Healthy Habit». [11]

В контексті розробки додатку органайзеру «Healthy Habit» необхідно розробити діаграму класів, діаграму послідовностей, діаграму діяльності та діаграму станів.

2.2.1 Взаємодія клієнта з додатком

Модель використання (Use Case) описує функціональні вимоги системи з точки зору випадків використання. Це модель передбачуваної функціональності системи (випадки використання) та її середовища (актори). Приклади використання дозволяють співвідносити те, що потрібно від системи, з тим, як система задовольняє ці потреби.

Оскільки це дуже потужний інструмент планування, модель використання зазвичай використовується на всіх фазах циклу розробки усіма членами команди.

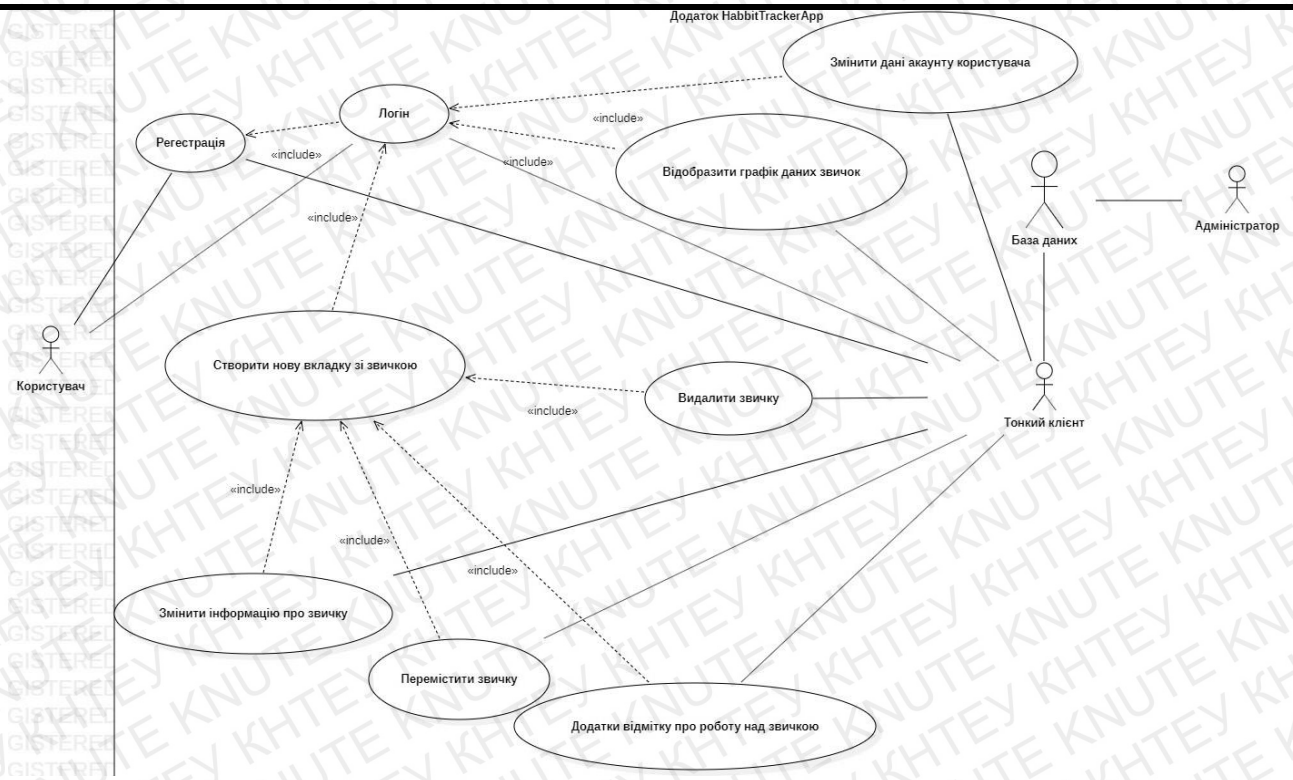

Рис. 2.2. Діаграма використання додатку «Healthy Habit»

На рисунку 2.2. представлено розроблену діаграму використання для додатку «Healthy Habit».

На діаграмі надано процес можливої взаємодії користувача з функціоналом додатку, який представлено через тонкий клієнт встановленого додатку. Тонкий клієнт передає отримані дані для запису у базу даних, та навпаки. Управління базою даних здійснює адміністратор.

2.2.2 Структура класів

Діаграма класів описує систему, ілюструючи атрибути, операції та взаємозв'язки між класами. Вона працює за принципами об'єктно-орієнтованих методів, що описують взаємодію об’єктів між собою.

Клас - це термін, що використовується для опису колективної структури та комбінованої поведінки об'єктів (класифікація). Іншими словами: об’єкти включаються до класу, якщо вони мають подібні властивості - тобто взаємні властивості в конкретних об’єктах згруповані в один клас.

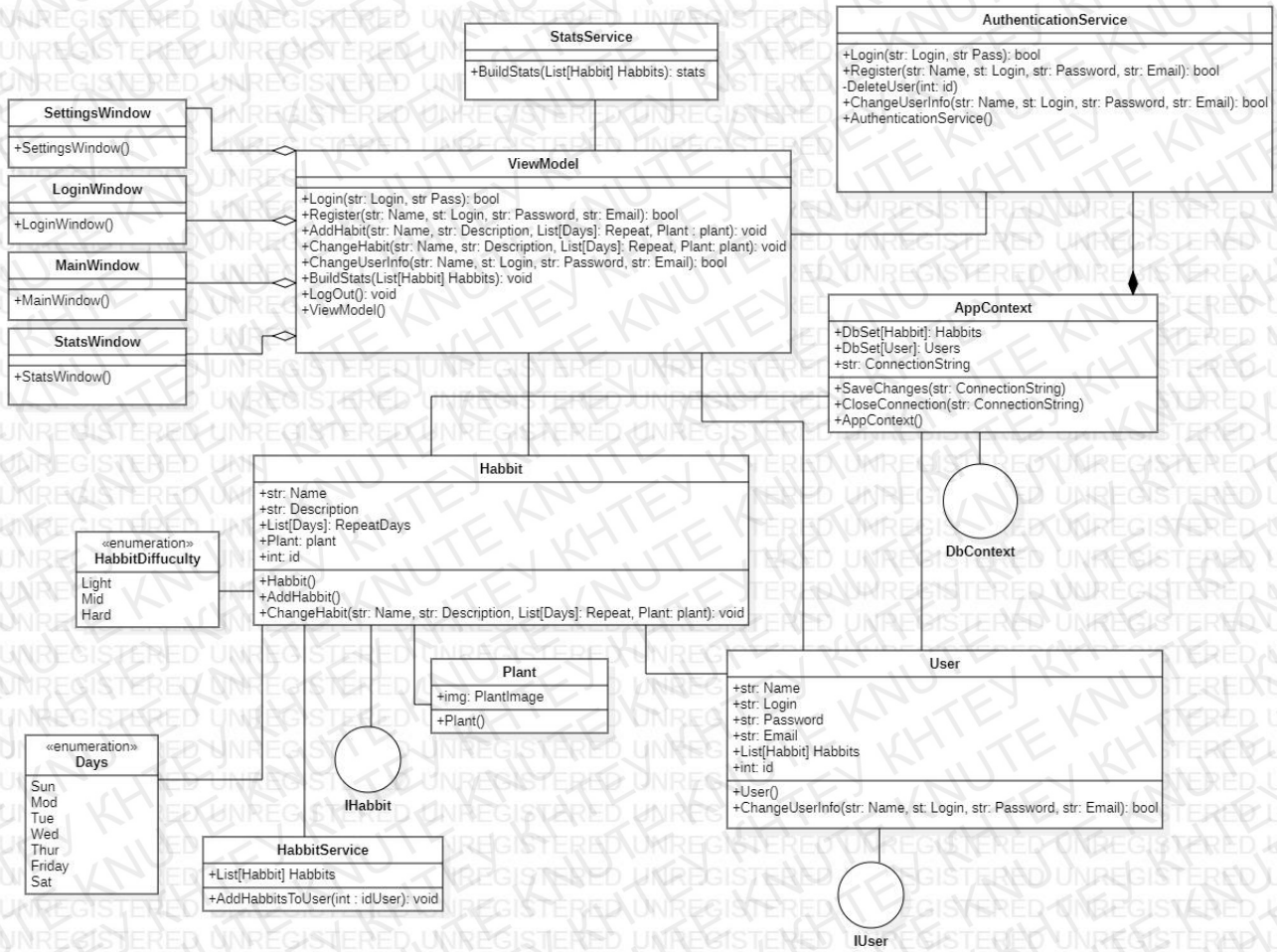

Рис. 2.3. Структура класів додатку «Healthy Habit»

На діаграмі класів додатку «Healthy Habit» (рис. 2.3.) детально проілюстровано структуру системи, показано всі її атрибути, операції, а також взаємозв'язки. Діаграма класів візуалізує шляхи між класами у вигляді агрегатів та асоціацій, а також шляхом передачі властивостей та поведінки між класами.

2.2.3 Діаграма діяльності

Діаграма діяльності – це важлива поведінкова діаграма в UML для опису динамічних аспектів системи. Це, по суті, вдосконалена версія блок-схеми, що моделює перехід від однієї діяльності до іншої.

Діаграма діяльності - це графічне зображення робочих процесів поетапних дій та дій із підтримкою вибору, ітерації та паралельності. Віна описує потік управління цільовою системою, такий як вивчення складних бізнес-правил та операцій, описуючи варіант використання, а також бізнес-процес. В уніфікованій

мові моделювання діаграми діяльності призначені для моделювання як обчислювальних, так і організаційних процесів (тобто робочих процесів).

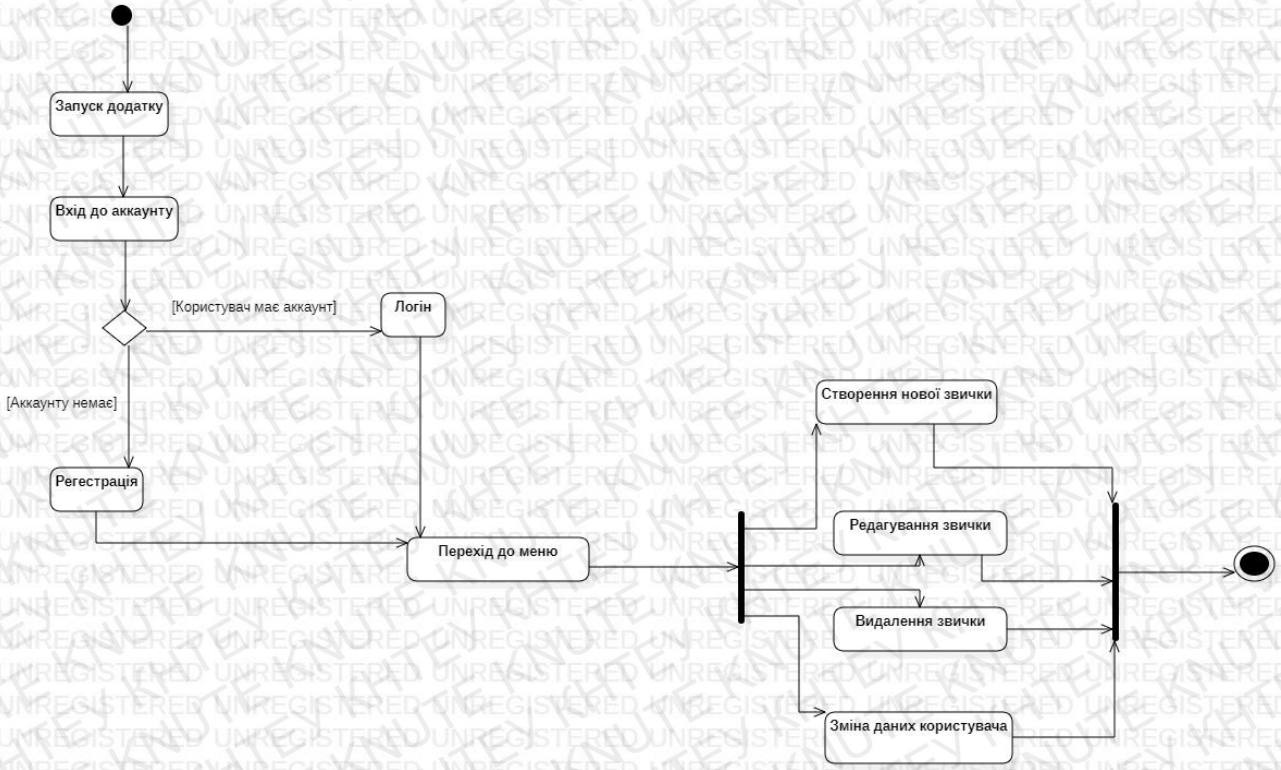

Рис. 2.4. Процедура координації-діяльності додатку «Healthy Habit»

Діаграма діяльності представлена на рис. 2.4., це координація діяльності здійснюється з метою отримання інформації на різних рівнях абстракції.

2.2.4 Діаграма станів

Діаграма станів - це тип діаграми, що використовується в UML для опису поведінки систем, що базується на концепції діаграм стану Девіда Харела. Діаграма станів зображує дозволені стани та переходи, а також події, що впливають на ці переходи. Це допомагає візуалізувати весь життєвий цикл об’єктів і, таким чином, допомагає краще зрозуміти стани системи.

Поведінка сутності є не лише прямим наслідком її вкладених даних, вона також залежить від попереднього стану. Попередню історію сутності можна найкраще змоделювати за допомогою схеми станів сутностей.

Діаграма станів зазвичай використовується для опису залежної від стану поведінки об'єкта. Об'єкт по-різному реагує на одну і ту ж подію в залежності від

того, в якому стані він знаходиться. Діаграми станів, як правило, застосовуються до об'єктів, але можуть застосовуватися до будь-якого елемента, який має поведінку щодо інших об'єктів, таких як: актори, випадки використання, методи, системи підсистем та тощо, і вони, як правило, використовуються разом із діаграмами взаємодії.

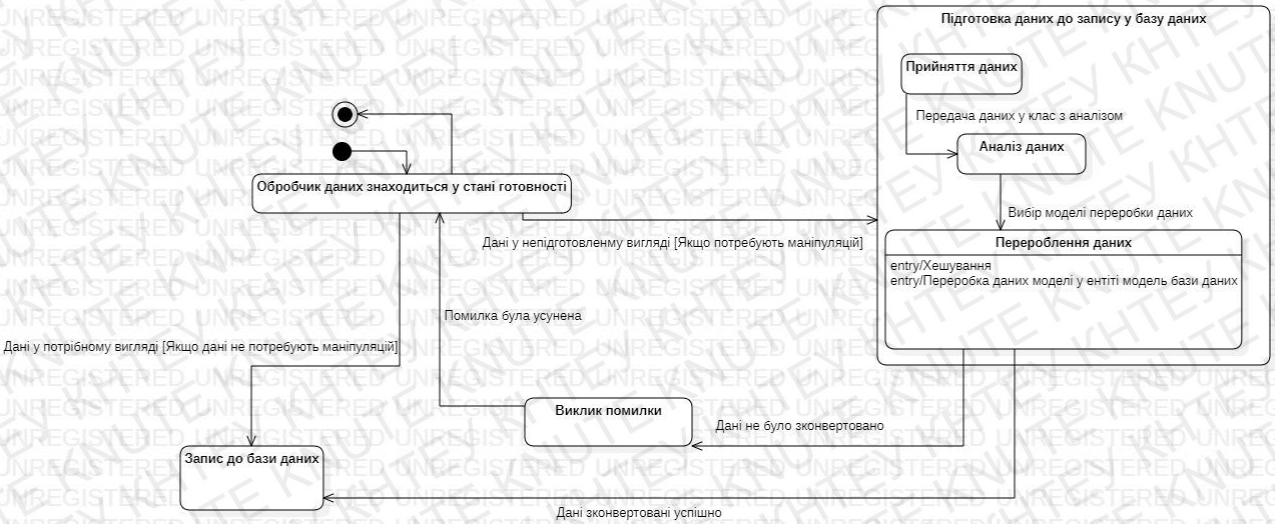

Рис. 2.5. Схема станів додатку «Healthy Habit»

На рис. 2.5. представлено діаграму станів додатку «Healthy Habit» . Діаграма стану показує різні стани сутності, а також як саме сутність реагує на різні події, змінюючи один стан на інший. Діаграма стану використовується для моделювання динамічного характеру системи.

2.3 Проектування бази даних

Наступним кроком проектування системи додатку стає створення бази даних.

Майже кожен сучасний веб-додаток взаємодіє з базою даних. База даних являє собою систематичний збір даних. Вони підтримують електронне зберігання та маніпулювання даними. Бази даних полегшують управління даними. Існує декілька видів баз даних, але у даному випадку використовується реляційна база даних. [12]

Реляційна база даних - це сукупність взаємопов'язаних таблиць, кожна з яких містить інформацію про об'єкти певного типу. Рядок таблиці містить дані

про один об'єкт, а стовпці таблиці описують різні характеристики цих об'єктів - атрибутів. Записи, тобто рядки таблиці, мають однакову структуру - вони складаються з полів, що зберігають атрибути об'єкта. Кожне поле, тобто стовпець, описує тільки одну характеристику об'єкта і має певний тип даних.

У реляційній базі даних кожна таблиця повинна мати первинний ключ - поле або комбінацію полів, які єдиним чином ідентифікують кожен рядок таблиці. Таблиці реляційної бази даних повинні відповідати вимогам нормалізації відносин. [13]

Entity Framework - це структура ORM із відкритим кодом для програм .NET, що підтримуються корпорацією Майкрософт. Ця структура дозволяє розробникам працювати з даними, використовуючи об'єкти класів конкретного домену, не фокусуючись на основних таблицях та стовпцях бази даних, де ці дані зберігаються. За допомогою Entity Framework розробники можуть працювати на вищому рівні абстракції, коли мають справу з даними, а також можуть створювати та підтримувати орієнтовані на дані програми з меншим кодом порівняно з традиційними програмами. [14]

Саме структура Entity Framework представила підхід Code-First. У підході Code-First процес роботи зосереджено на домені створюваної програми, який починається саме зі створення класів сутностей домену, а не з проектування бази даних та створення класів що відповідають дизайну бази даних, як це робиться зазвичай. З використанням даного підходу, Entity Framework створює базу даних на основі створених класів предметної області та конфігурацій додатку. Це означає, що процес розробки починається саме з кодування на мові програмування C # або VB.NET, а після цього Entity Framework створить базу даних відповідно коду.

Наступний рисунок (Рис. 2.3.1) ілюструє робочий процес розробки бази даних за допомогою підходу Code-First:

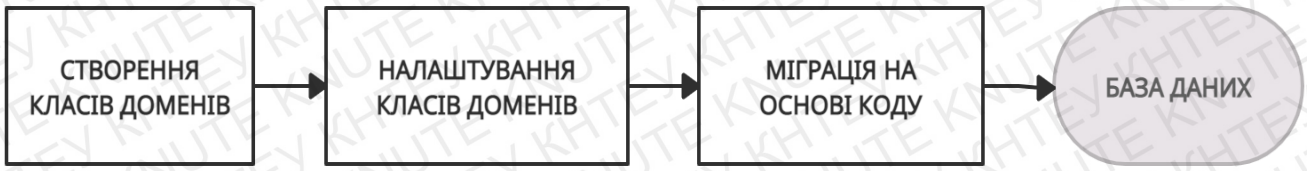

Рис. 2.6. Робочий процес Code-First.

Робочий процес розробки в підході Code-First буде таким:

1. Створення або зміна класів доменів.
2. Налаштування цих класів доменів за допомогою Fluent-API або атрибутів анотації даних.
3. Створення або оновлення схеми бази даних за допомогою автоматизованої міграції або міграції на основі коду. [15]

Перш ніж почати створення бази даних за допомогою підходу Code-First, необхідно спроектувати концептуальну модель бази даних, логічну та фізичну моделі, і вже після цього переходити до створення таблиць.

2.3.1 Концептуальна модель

Концептуальна модель даних являє собою організований вид загальних атрибутів баз даних і їх відносин. Мета створення концептуальної моделі даних полягає в створенні об'єктів, їх атрибутів і відносин. На цьому рівні моделювання даних не важливі деталі фактичної структури бази даних. Концептуальна модель розробляється незалежно від специфікацій, таких як ємність зберігання даних, розташування або специфікації програмного забезпечення та технологій. [16]

Після проведеного аналізу предметної області можна відзначити, що додаток органайзер «Healthy Habit» потребую невелику базу даних для зберігання всієї необхідної інформації.

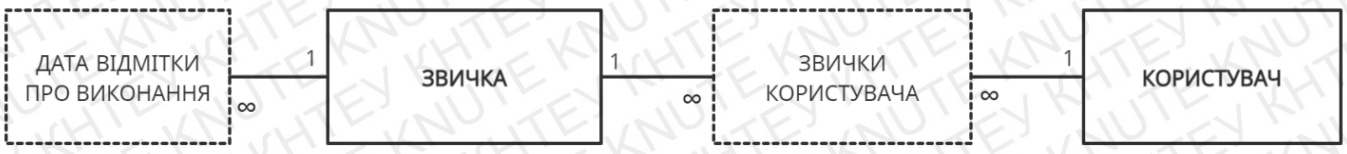

Рис. 2.7. Концептуальна модель бази даних.

В концептуальній моделі (Рис. 2.7.) існують наступні сутності: звичка, користувач, звички користувача та дата відмітки про виконання. В ній використовуються типи відносин «один-до-багатьох» в цілях нормалізації бази даних.

Нормалізація представляє собою розбиття таблиці на дві або більше з метою отримання такого проекту БД, в якому кожна частина інформації з'являється лише у одному місці, і таким чином виключено надмірність інформації.

В моделі наведено дві основі сутності: звичка та користувач. В цілях нормалізації відносин та правильного зберігання даних, було створено дві додаткові сутності – дата відмітки про виконання та звички користувача (відмічені пунктирними лініями).

Після проектування концептуальної моделі бази даних, можна переходити до логічного та фізичного проектування.

2.3.2 Логічна та фізична моделі

Логічна модель це перетворена концептуальна модель, що описує поняття предметної області, їх взаємозв'язок, а також обмеження на дані, що накладаються предметною областю. Логічна модель містить уявлення сутностей та атрибутів, відношень, унікальних ідентифікаторів, підтипів та супертипів та обмежень між зв'язками.

Фізична модель це конкретний опис реалізації бази даних. Вона визначає спосіб розміщення даних в середовищі зберігання і способи доступу до цих даних, які підтримуються на фізичному рівні. [16]

В деяких випадках немає необхідності у створенні окремих моделей, так як логічна модель є основою для побудови фізичної. В таких випадках зазвичай будується лише одна модель.

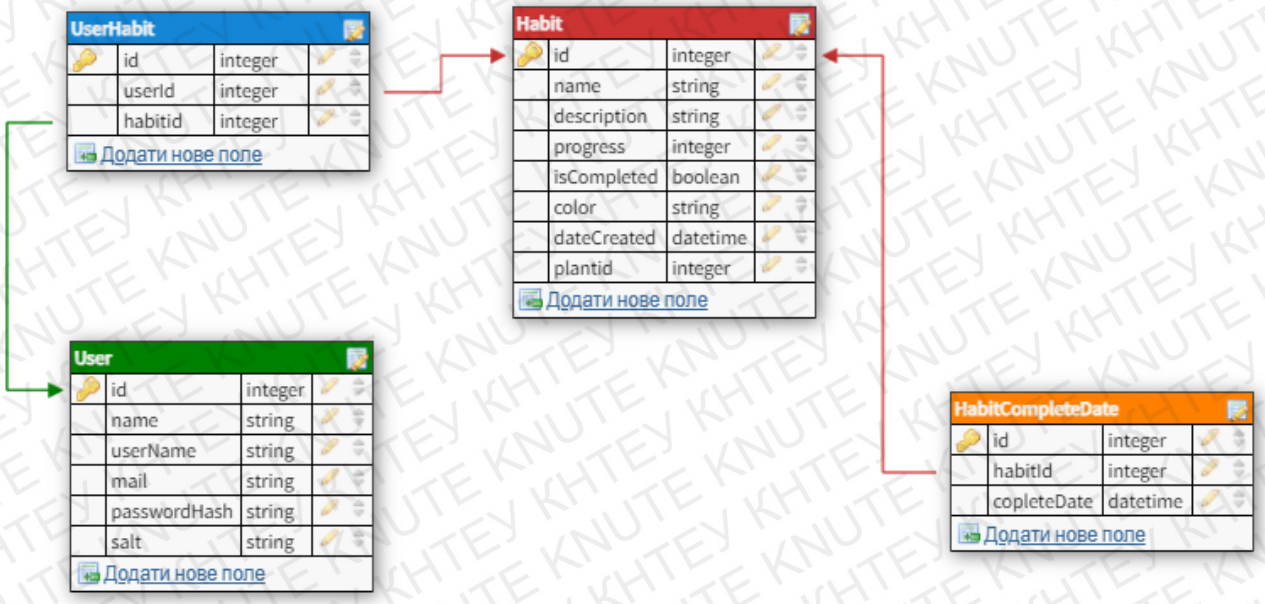

Рис. 2.8 Логічна модель бази даних.

Створена логічна модель бази даних для додатку «Healthy Habit» (Рис. 2.8) містить у собі всі необхідні дані для створення самої бази даних: найменування таблиць і стовпців, типи даних, встановлені первинні і зовнішні ключі та відносини між таблицями. Наступним етапом є написання коду для створення бази даних та таблиць використовуючи підхід Code-First.

2.3.3 Створення таблиць

Використовуючи підхід Code-First у структурі Entity Framework, можна переходити безпосередньо до створення бази даних та її таблиць згідно даним, що наведені у логічній моделі (підрозділ 2.3.2) використовуючи мову програмування C#.

На рис. 2.9. представлено контекст даних, що створює базу даних та пов’язує її із програмою.

```
public class SystemContextSQL : DbContext
    {
        public DbSet<Habit> Habit { get; set; }
        public DbSet<HabitCompleteDate> HabitCompleteDate { get; set; }
        public DbSet<User> User { get; set; }
        public DbSet<UserHabit> UserHabit { get; set; }
        public SystemContextSQL(DbContextOptions<SystemContextSQL> options) : base(options)
        {
            Database.EnsureCreated();
        }
    }
```

Рис. 2.9. Створення бази даних

Створення таблиці проходить наступним чином: після назви таблиці через двокрапку вказується клас «ModelTemplate», що містить у собі ідентифікатор таблиці та дозволяє всім рядкам мати поле ID, що автоматично визначається як первинний ключ. Після вказання модифікатору доступу, з нової строки коду вказується тип даних та назва поля. Процес створення усіх таблиць представлено на рисунках 2.10 - 2.13.

```
public class User : ModelTemplate
    {
        public string Name { get; set; }
        public string UserName { get; set; }
        public string Mail { get; set; }
        public string PasswordHash { get; set; }
        public string Salt { get; set; }
    }
```

Рис. 2.10. Створення таблиці «Користувач».

```
public class Habit : ModelTemplate
{
    public string Name { get; set; }
    public string Description { get; set; }
    public int  Progress { get; set; }
    public bool IsCompleted { get; set; }
    public string Color { get; set; }
    public DateTime DateCreated { get; set; }
    public int PlantId { get; set; }
}
```

Рис. 2.11. Створення таблиці «Звичка».

```
public class HabitCompleteDate : ModelTemplate
{
    public Habit Habit { get; set; }
    public int? HabitId { get; set; }
    public DateTime CompleteDate { get; set; }
}
```

Рис. 2.12. Створення таблиці «Дата відмітки про виконання».

```
public class UserHabit : ModelTemplate
{
    public User User { get; set; }
    public int? UserId { get; set; }
    public Habit Habit { get; set; }
    public int? HabitId { get; set; }
}
```

Рис. 2.13. Створення таблиці «Звички користувача».

На Рис. 2.14 представлено схему взаємозв’язків таблиць бази даних.

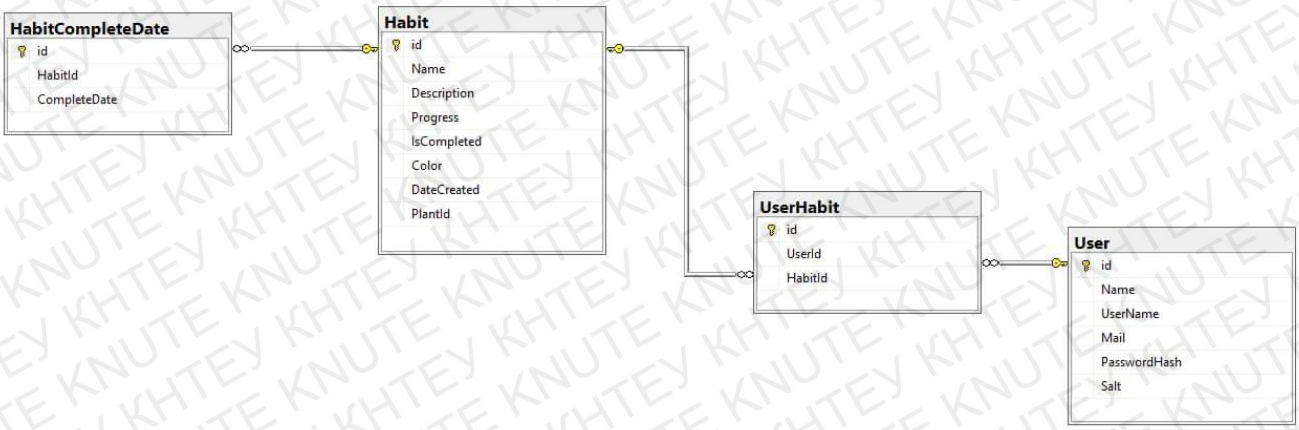

Рис. 2.14. Схема взаємозв’язків таблиць бази даних.

На схемі даних можна побачити що у базі даних забезпечено цілісність на рівні таблиць за допомогою первинних ключів кожної таблиці, а на рівні зв’язків – посилальною цілісністю.

Наповнення таблиць бази даних проходить автоматично, використовуючи безпосередньо додаток органайзер «Healthy Habit».

2.4 Висновки до розділу 2

Першим етапом проектування додатку органайзеру «Healthy Habit» було визначення всіх необхідних параметрів предметної області.

Після опису всіх необхідних умов створення додатку та установлення процесу його роботи, було визначено основні вимоги, яким повинен відповідати додаток для виконання визначеної мети ВКР.

Після визначення вимог було досліджено необхідність UML діаграм та види моделей, що важливо створити для додатку. Визначившись із інструментом для створення діаграм, а саме StarUML, в процесі проектування було створено схему моделі взаємодії клієнта с додатком, структуру класів додатку, процедуру координації-діяльності додатку та схему станів додатку.

Наступним етапом стає створення бази даних. Для цього було проведено аналіз можливих способів створення бази даних для найбільш ефективного результату, і один з них це використання структури Entity Framework, а саме

підхід Code-First. Цей підхід дозволяє створювати базу даних безпосередньо за допомогою кодування мовою C#.

Перед створенням бази даних було спроектовано концептуальну модель бази даних, для того щоб визначити основні атрибути та відношення. На основі концептуальної моделі було створено логічну модель, яка представляє собою детальний опис сутностей, атрибутів та відносин між ними.

На основі отриманих даних, використовуючи підхід Code-First у структурі Entity Framework мовою C# було створено базу даних та контекст даних, що пов’язує базу даних із програмою. Після цього було створено всі необхідні таблиці, опираючись на логічну модель бази даних. Всі відношення таблиць є цілісними та нормалізованими. Наповнення таблиць проходить автоматично через додаток органайзер «Healthy Habit».

РОЗДІЛ 3

РОЗРОБКА ДОДАТКУ ОРГАНАЙЗЕРУ «HEALTHY HABIT»

3.1 Розробка додатку та підключення бази даних

Для розробки додатку «Healthy Habit» було обрано об’єктно-орієнтовану мову програмування C#, а саме модульну платформу для розробки програмного забезпечення .NET, що має відкритий вихідний код [17].

Для проектування форм додатку було обрано систему для створення інтерфейсу що має назву WPF (Windows Presentation Foundation) та є складовою підсистемою у складі .NET [18]. WPF використовує мову XAML (extensible application markup language), що представляє собою мову розмітки декларованого опису інтерфейсу. [19]

Після створення проекту у Microsoft Visual Studio 2019, першим кроком розробки додатку є розбиття логічних складових проекту на збірки (assembly) та вистроювання залежностей і зв’язків між цими збірками. Для подальшої роботи, у збірці шару доступу до даних (data access layer) необхідне встановлення технології для доступу та взаємодії баз даних, Entity Framework Core, та інших залежних пакетів.

Наступним кроком є розробка класу контексту даних для Entity Framework і розробка моделей (у майбутньому – таблиць бази даних) у збірці «Models», а надалі безпосереднє додавання моделей у клас контексту даних (більш детально цей процес описано у розділі 2.3.3).

Після розробки інтерфейсів сервісів програми необхідно розробити класи, що реалізують інтерфейси, та налаштувати точки входу у програму. Для цього необхідно створити файл «appsettings.json», куди додається інформація (наведено у додатку А), що означає новий розділ у файлу налаштувань, в якому прописані все шляхи для підключення до бази даних [20]. Після реєстрації усіх файлів

налаштувань та впровадження залежностей, реєструються всі вікна та сервіси (код наведено у додатку А).

Останнім етапом розробки є створення майбутніх вікон програми, в кожному з яких додається XAML розмітка. Після цього залишається лише прописати логіку взаємодії вікон з бізнес логікою програми. Подальша розробка програми супроводжується тестуванням.

3.2 Опис функціональних характеристик додатку «Healthy Habit»

Після запуску створеного додатку першим чином відкривається головне вікно програми, яке представляє вікно авторизації (Рис. 3.1), що дозволяє користувачу увійти у систему. Для цього необхідно мати обліковий запис, ввести ім’я користувача та пароль у відповідних полях, та натиснути кнопку «Увійти».

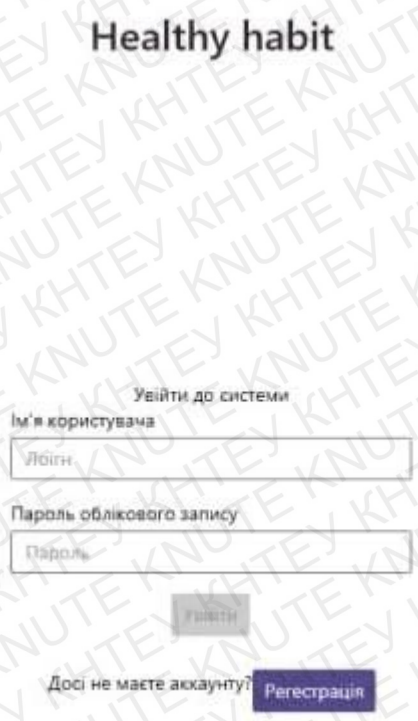

Рис. 3.1. Вікно авторизації додатку «Healthy Habit»

Якщо ім’я або пароль користувача введено невірно, програма покаже помилку та не дозволить увійти у обліковий запис (Рис. 3.2).

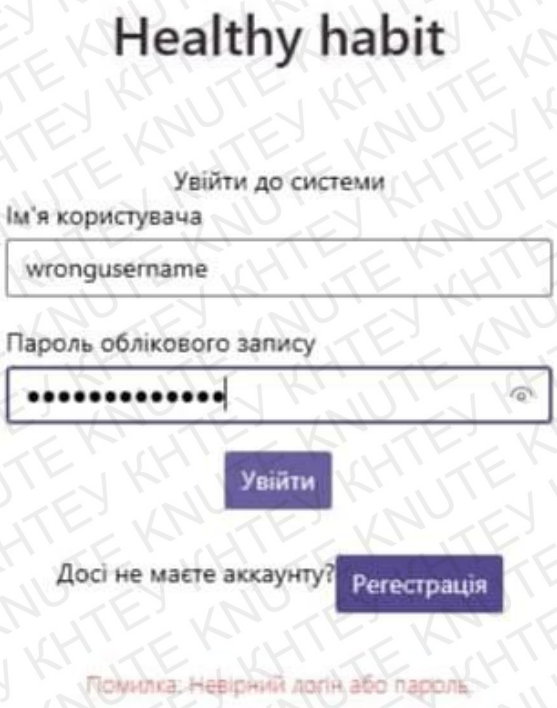

Рис. 3.2. Помилка при невірному вводі даних при авторизації

У випадку, коли користувач ще не має облікового запису, він може натиснути кнопку «Реєстрація» що знаходиться нижче на вікні авторизації, та відкриється вікно реєстрації (Рис. 3.3).

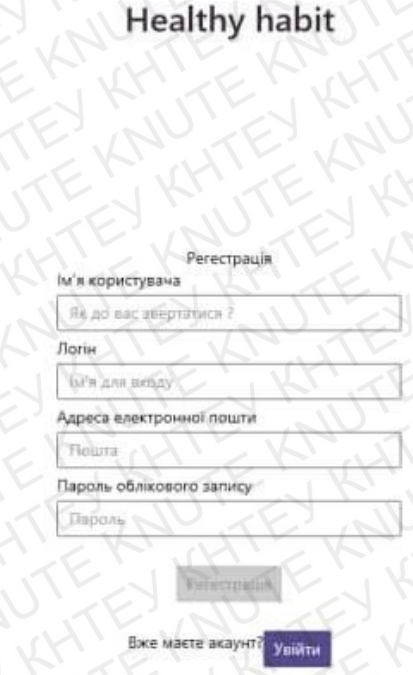

Рис. 3.3. Вікно реєстрації нового користувача

У вікні реєстрації облікового запису можна побачити наступні поля: ім’я користувача, логін (тобто ім’я для входу в аккаунт), адреса електронної пошти та пароль облікового запису. Їх необхідно заповнити та натиснути кнопку «Реєстрація» для створення нового облікового запису. При некоректному вводі даних, додаток покаже помилку та обліковий запис не буде створено. (Рис 3.4).

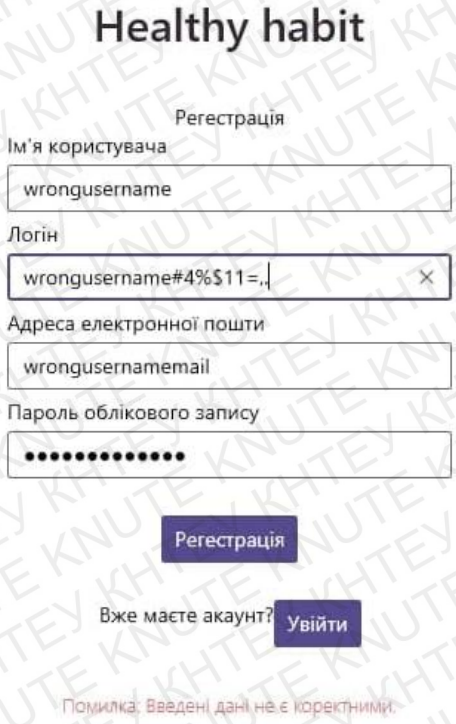

Рис. 3.4. Помилка при створенні облікового запису

Щоб повернутися на вікно авторизації, необхідно натиснути кнопку «Увійти», що знаходиться нижче у вікні.

Після успішної авторизації у додаток, відкривається головна сторінка. Дана форма складається з декількох елементів (Рис. 3.5). З лівої сторони вікна зверху розташовано привітання, яке змінюється згідно часу доби, ім’я користувача, день тижня та поточна дата. Для початку роботи з додатком необхідно натиснути кнопку «Додати звичку».

Рис. 3.5. Елементи головного вікна. Початок роботи.

По натисненню кнопки відкриється нова форма, яка представляє собою вікно створення нової звички (Рис. 3.6.). Для створення звички, необхідно заповнити наступні поля: назва звички, опис звички (не обов'язково), частота виконання звички (наприклад кожен день, або кожні два дні), колір та рослинка, яка буде відповідати звичці (інтерактивна частина додатку).

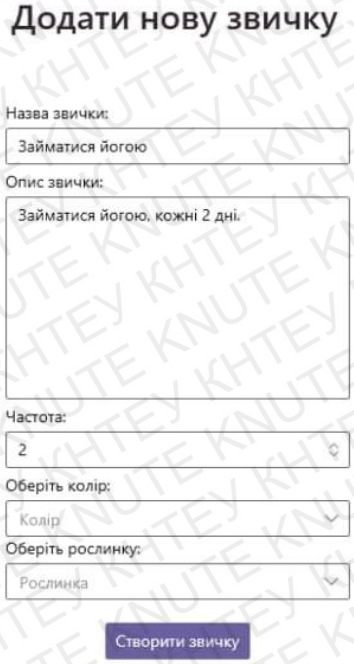

Рис. 3.6. Створення нової звички

Після заповнення всіх необхідних даних, треба натиснути кнопку «Створити звичку», і вона з'явиться на головному вікні (Рис. 3.7.).

Рис. 3.7. Створена звичка на головному вікні.

Звичка складається з наступних елементів: налаштування (три крапки з лівої сторони), завдяки яким можна редагувати або видалити звичку (Рис. 3.8.); іконка рослинки, по натисненню якої можна побачити рослину, що відповідає звичці, та стадію її росту відповідно до прогресу виконання звички; назву звички; поля для відмітки виконання звички.

Рис. 3.8. Налаштування звички.

На головному вікні відображається сім останніх днів виконання звички з відповідними датами. Для того, щоб помітити виконання звички, необхідно лише натиснути на поле, що відповідає дню виконання, і звичка зарахується виконаною.

Для повного впровадження звички необхідно виконувати її щодня (або відповідно до графіку, що було встановлено при створенні звички) протягом 21 дня. Після цього звичка вважається успішно впровадженою, рослинка в інтерактивній частині повністю виростає, а дані зберігаються у архіві, який можна переглянути у особистому кабінеті. Якщо користувач пропускає один або більше днів, необхідна кількість днів для впровадження змінюється, і вираховується по наступній формулі (3.1):

$$0 \leq (n-1) \leq 21 \quad (3.1)$$

Де n – порядковий номер дня виконання звички.

Завдяки цій формулі при кожній відмітці про виконанні звички перевіряється етап формування звички. По досягненню числа 21 звичка вважається сформованою. Якщо день або більше пропущено, від числа звички віднімається пропущена кількість днів, що змінює довжину циклу формування звички. Формула також перевіряє, щоб число звички не було від'ємним, тож у випадку якщо всі можливі дні пропущені, звичка вважається не сформованою та видаляється.

Інтерактивна частина звички представляє собою рослину, стадія росту якої відповідає стадії виконання звички, та після повного формування звички рослина повністю виростає. Дана методика розроблена для того, щоб користувач завжди мав наочне представлення свого прогресу та не губив мотивацію.

Кожна рослина має 7 можливих стадій росту які відповідають різним стадіям впровадження звички.

Головне вікно додатку із створеною звичкою виглядає наступним чином (Рис. 3.11), і для того щоб відмітити виконання усіх звичок необхідно лише натиснути на відповідному полі. Для прибирання відмітки необхідно натиснути ще раз.

Рис. 3.9. Головне вікно додатку зі створеною звичкою.

Щоб переглянути особистий кабінет, необхідно натиснути відповідну кнопку на головному вікні додатку. Після цього відкриється нове вікно (рис. 3.12.), де можна переглянути дані користувача, які було введено при реєстрації. Також тут можна змінити ім’я, електронну скриньку або пароль користувача. Тут же зберігається архів звичок, який можна переглянути натиснувши на відповідну кнопку.

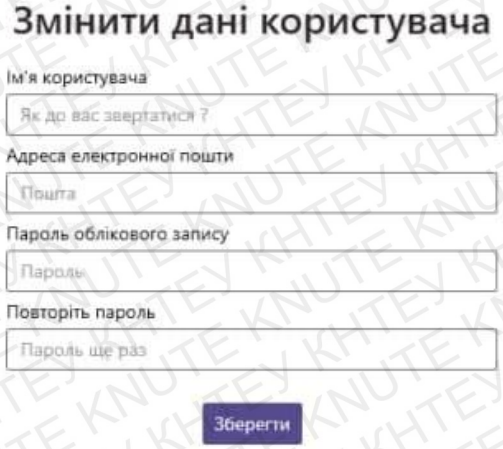

Рис. 3.10. Редагування облікового запису.

По завершенню роботи з додатком, необхідно просто закрити його. Всі дані, що вводить користувач, автоматично записується у базу даних та прив’язується

до особистого облікового запису. Приклад даних, що було автоматично записано у базу даних після маніпуляцій користувача, наведено на рис 3.13 – 3.14.

Рис. 3.11. Записи у базі даних користувача.

Рис. 3.12. Записи у базі даних звички.

3.3 Висновки до розділу 3

На основі попередньо спроектованого додатку було розроблено просту у використанні програму для впровадження корисних звичок «Healthy Habit», яка виконує всі поставлені передодні задачі та функції.

Додаток було розроблено у середовищі програмування Microsoft Visual Studio 2019, використовуючи модульну платформу для розробки програмного забезпечення .NET, мовою програмування C#. Для створення інтерфейсу додатку було обрано систему WPF що використовує мову розмітки XAML.

Для роботи із базами даних було використано технологію Entity Framework Core що дозволило створити базу даних та всі необхідні компоненти безпосередньо у програмі користуючись методом Code-First.

Після створення всіх необхідних вікон для програми було прописано всю необхідну логіку взаємодії вікон із бізнес логікою програми.

У результаті було отримано працюючий додаток «Healthy Habit» який відповідає всім вимогам ВКР.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У даній випускній кваліфікаційній роботі було представлено повний процес створення додатку органайзеру «Healthy Habit» що допомагає користувачу у формуванні корисних звичок.

На першому етапі роботи було проведено докладний аналіз предметної області, а саме особливостей процесу формування звичок, їх значення у житті людей та основні характеристики циклу звички. Проаналізувавши необхідну літературу, на цьому етапі було визначено, що для того щоб сформувати бажану звичку, необхідно від двадцяти одного дня постійного виконання звички.

Наступним етапом було проектування компонентів додатку органайзеру «Healthy Habit». Для цього було визначено всі необхідні параметри та характеристики, який повинен містити у собі додаток, процес роботи з додатком та відповідно було встановлено вимоги до готової програми.

Для коректного розуміння вимог додатку та спрощення розробки програми засобами UML було спроектовано необхідні діаграми, а саме: діаграму використання додатку, яка детально описує функціональні вимоги системи; схему класів додатку, яка описує структуру системи та взаємодію всіх її атрибутів, операцій та зв’язків; діаграму діяльності, що зображує робочі процеси додатку та описує динамічні аспекти системи; діаграму станів, яка описує поведінку системи, її можливі стани та переходи і події, які мають вплив на ці переходи.

Наступним кроком стало визначення робочого процесу бази даних, а саме у використанні підходу Code-First у структурі Entity Framework, який допомагає створювати базу даних безпосередньо в процесі написання коду, та одразу інтегрувати її у додаток.

Проектування бази даних додатку почалося з розробки концептуальної моделі, де було визначено основні атрибути бази даних та відносини між ними. На основі концептуальної моделі було створено логічну модель бази даних, яка

представляє собою детальний опис таблиць і стовпців, первинних та зовнішніх ключів та відносин між таблицями. На основі моделей бази даних було створено контекст даних та класи, що відповідають таблицям бази даних.

Подальший розвиток додатку буде направлено на постановку нових задач для впровадження нових функцій, а саме: інтеграція соціальних мереж, інтерактивні дії між користувачами додатку (наприклад, функція додавання інших користувачів у друзі, що відкриває можливість переглядати прогрес один одного), впровадження нових видів рослин в інтерактивній частині додатку, налаштування системи нагадувань. У майбутньому буде розглядатися можливість створення додатку на мобільні пристрої та його веб-версію з повною синхронізацією між пристроями.

У результаті роботи над проектом було створено працюючий додаток органайзер «Healthy Habit» для впровадження звичок, що відповідає всім попередньо поставленим вимогам та завданням, отже випуску кваліфікаційну роботу завершено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Habits – A Repeat Performance by David T. Neal, Wendy Wood, and Jeffrey M., 2006 [Електронний ресурс]. – Режим доступу: https://dornsife.usc.edu/assets/sites/208/docs/Neal.Wood.Quinn.2006.pdf
2. Ч. Дахигг. Сила звички. Чому ми діємо так, а не інакше в житті та бізнесі / Ч. Дахиг. – Київ: Символ-плюс, 2016. - 432 с.
3. Contributions to workload of rotational optical transformations: dissertation by Atkinson, R. P., Harrington, T. L. [Електронний ресурс]. – Режим доступу: https://ntrs.nasa.gov/citations/19860011635
4. M. Maltz. Psycho-Cybernetics, A New Way to Get More Living Out of Life / M. Maltz : Mass Market Paperback, 1989. - 288 p.
5. J. Clear. Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones / J. Clear : Kindle Edition, 2018. - 319 p.
6. The Outlook For Digital Marketing In 2021 by Simon Kemp [Електронний ресурс]. – Режим доступу: https://datareportal.com
7. Mobile App Download and Usage Statistics, Ian Blair [Електронний ресурс]. – Режим доступу: https://buildfire.com/app-statistics
8. J. Rambo. The unified modeling language reference manual / J. Rambo, I. Jacobson, G. Booch. 2011 – 496 p.
9. Р. Мартін. Принципи, патерни і методики гнучкої розробки на мові C# / Р. Мартін, М. Мартін. - Київ : Символ-плюс, 2011 - 768 с.
10. Ф. Мартін. UML. Основи. Короткий посібник по стандартній мові об'єктного моделювання / Ф. Мартін. - Київ : Символ-плюс, 2018 - 192 с.
11. StarUML documentation. [Електронний ресурс]. – Режим доступу: https://docs.staruml.io/

					КНТЕУ 121 06-21.БР		
Зм	Аркуш	№ докум	Підпис	Дата			
Зав.кафедри		Криворучко О.В		24.04.21	Розробка програмного додатку «Органайзер Healthy Habit»	Стадія: СД	Аркуш: 44 / Аркушів: 45
Керівник		Рзаєва С.Л.		24.04.21			
Гарант		Цензура М.О.		24.04.21			
Розроб.		Солдатенко Ю.О.		24.04.21	Список використаних джерел	Факультет інформаційних технологій, 4 курс, 6 група	

12. Скотт В. Емблер. Рефакторінг баз даних. еволюційне проектування / Скотт В. Емблер, Прамодкумар Дж. Садаладж. - Київ : Вільямс, 2016. - 368 с.
13. С. Кузнєцов. Бази даних. Моделі і мови / С. Кузнєцов. - Київ : Біном-Пресс, 2008. - 720 с.
14. J. Lerman. Programming Entity Framework / J. Lerman - O'Reilly Media, Inc, 2010. - 832 p.
15. J. Lerman. Programming Entity Framework: Code First / J. Lerman, R. Miller - O'Reilly Media, Inc, 2011. - 194 p.
16. Федько, В. В. Організація баз даних та знань: навч.-практ. посіб. для самост. підготов. студ. / В. В. Федько, О. В. Тарасов, М. Ю. Лосєв. - Харків : ХНЕУ, 2013. - 198 с.
17. Ф. Джепікс. Мова програмування C # 7 і платформи .NET і .NET Core / Ф. Джепікс, Е. Троелсен - Київ : Вільямс, 2018. - 1328 с.
18. M. MacDonald. Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5 / M. MacDonald - Apress, 2012. - 1078 p.
19. A. Nathan. XAML Unleashed / Adam Nathan - Sams; 1st edition, 2014. - 498 p.
20. A. Nathan. Windows Presentation Foundation: Unleashed / Adam Nathan - Sams; 1st edition, 2006. - 638 p.

ДОДАТКИ

Додаток А

Інтерфейс IAccountHolder

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HealthyHabit.BL.Abstract
{
    public interface IAccountHolder<User>
    where User : class
    {
        void SetUser(User user);
        User GetUser();
    }
}
```

Інтерфейс IAuthenticationService

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HealthyHabit.BL.Abstract
{
    public interface IAuthenticationService <Datacontext>
    where Datacontext : class
    {
        void Login(Datacontext datacontext, string username, string password);
        void Register(Datacontext datacontext, string name, string username, string mail, string password);
    }
}
```

Інтерфейс IColorService

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HealthyHabit.BL.Abstract
{
    public interface IColorService <Datacontext, TColor>
    where Datacontext : class
    where TColor : class
    {
        void Create(Datacontext datacontext, string HexCode, string ColorName);
        void Remove(Datacontext datacontext, TColor habit);
        void Change(Datacontext datacontext, string HexCode, string ColorName);
    }
}
```

Інтерфейс IHabitService

```
using System;
using System.Collections.Generic;
using System.Text;
namespace HealthyHabit.BL.Abstract
{
    public interface IHabitService<Datacontext, TUser, THabit, TColor, TPlant>
    where Datacontext : class
    where TUser : class
    where THabit : class
    where TColor : class
    where TPlant : class

    {
        void Create(Datacontext datacontext, TUser user, string name, string desciption, int progress, int frequency, bool iscompleted,
TColor color, DateTime datecreated, TPlant plant);
        void Change(Datacontext datacontext, TUser user, string name, string desciption, int progress, int frequency, bool iscompleted,
TColor color, DateTime datecreated, TPlant plant);
        void Remove(Datacontext datacontext, THabit habit);
```

```
        void AddProgress(Datacontext datacontext, THabit habit);
        bool IsCompleted(Datacontext datacontext, THabit habit);
        void HabitCheker(Datacontext datacontext, THabit habit);
    }
}
```

Інтерфейс IPlantService

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HealthyHabit.BL.Abstract
{
    public interface IPlantService <Datacontext, TPlant>
    where Datacontext : class
    where TPlant : class
    {
        void Create(Datacontext datacontext, string previewpath, string stage0path, string stage1path, string stage2path, string stage3path,
string stage4path, string stage5path, string stage6path, string stage7path, string name);
        void Remove(Datacontext datacontext, TPlant Plant);
        void Change(Datacontext datacontext, string previewpath, string stage0path, string stage1path, string stage2path, string stage3path,
string stage4path, string stage5path, string stage6path, string stage7path, string name);
    }
}
```

Інтерфейс ISaltService

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HealthyHabit.BL.Abstract
{
    public interface ISaltService
    {
        string Generate();
    }
}
```

Інтерфейс IUserHabitService

```
using System;
using System.Collections.Generic;
using System.Text;
namespace HealthyHabit.BL.Abstract
{
    public interface IUserHabitService<Datacontext, TUser, THabit>
    where Datacontext : class
    where TUser : class
    where THabit : class
    {
        List<THabit> GetHabitsByUser(Datacontext datacontext, TUser user);
    }
}
```

Інтерфейс IUserService

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HealthyHabit.BL.Abstract
{
    public interface IUserService<Datacontext, TUser>
    where Datacontext : class
    where TUser : class
    {
        void Add(Datacontext datacontext, string name, string username, string mail, string passwordhash, string salt);
        void Remove(Datacontext datacontext, TUser user);
        void Change(Datacontext datacontext, string name, string username, string mail, string password);
        bool IsExists(Datacontext datacontext, string username);
        bool IsExists(Datacontext datacontext, TUser user);
```

```
    }
}
```

Клас AccountHolder

```
using System;
using System.Collections.Generic;
using System.Text;
using HealthyHabit.Models;
using HealthyHabit.DAL.Implementation;
using HealthyHabit.BL.Abstract;
namespace HealthyHabit.BL.Implementation
{
    public class AccountHolder : IAccountHolder<User>
    {
        private User Account { get; set; }
        public AccountHolder()
        {
        }
        public void SetUser(User user)
        {
            this.Account = user;
        }

        public User GetUser()
        {
            return this.Account;
        }
    }
}
```

Клас AuthenticationService

```
using System;
using System.Collections.Generic;
using System.Text;
using HealthyHabit.Models;
using HealthyHabit.DAL.Implementation;
using HealthyHabit.BL.Abstract;
using System.Linq;

namespace HealthyHabit.BL.Implementation
{
    public class AuthenticationService : IAuthenticationService<SystemContextSQL>
    {
        private ISaltService salt { get; set; }
        private IHashService hash { get; set; }
        private IUserService<SystemContextSQL, User> UserService { get; set; }
        private IAccountHolder<User> AccountHolder { get; set; }
        public AuthenticationService(ISaltService saltService, IHashService hashService, IUserService<SystemContextSQL, User>
userService, IAccountHolder<User> accountHolder)
        {
            salt = saltService;
            hash = hashService;
            UserService = userService;
            AccountHolder = accountHolder;
        }
        public void Login(SystemContextSQL datacontext, string username, string password)
        {
            if (datacontext.User.Any(user => user.UserName == username))
            {

                var tmpUser = datacontext.User.FirstOrDefault(user => user.UserName == username);
                if (hash.Hash(password + tmpUser.Salt).Equals(tmpUser.PasswordHash))
                {
                    AccountHolder.SetUser(tmpUser);
                }
                else
                {
                    throw new Exception("Wrong username or password");
                }
```

```
            }
            else
            {
                throw new Exception("User not found");
            }
        }

        public void Register(SystemContextSQL datacontext, string name, string username, string mail, string password)
        {
            string saltstr = salt.Generate();
            UserService.Add(datacontext, name, username, mail, hash.Hash(password+saltstr), saltstr);
        }
    }
}
```

Клас UserService

```
Using System;
using System.Collections.Generic;
using System.Text;
using HealthyHabit.Models;
using HealthyHabit.DAL.Implementation;
using HealthyHabit.BL.Abstract;
using System.Linq;
namespace HealthyHabit.BL.Implementation
{
    public class UserService : IUserService<SystemContextSQL, User>
    {
        public void Add(SystemContextSQL datacontext, string name, string username, string mail, string passwordhash, string salt)
        {
            if (!IsExists(datacontext, username))
            {
                datacontext.User.Add(new User(name, username, mail, passwordhash, salt));
                datacontext.SaveChanges();
            }
            else
            {
                throw new Exception("User is already exists");
            }
        }

        public void Change(SystemContextSQL datacontext, string name, string username, string mail, string password)
        {
            User tmpUser = datacontext.User.FirstOrDefault(user => user.UserName == username);
            tmpUser.Name = name;
            tmpUser.UserName = username;
            tmpUser.Mail = mail;
            datacontext.SaveChanges();
        }

        public bool IsExists(SystemContextSQL datacontext, string username)
        {
            return datacontext.User.Any(user => user.UserName == username);
        }

        public bool IsExists(SystemContextSQL datacontext, User user)
        {
            return datacontext.User.Any(tuser => tuser == user);
        }

        public void Remove(SystemContextSQL datacontext, User user)
        {
            datacontext.User.Remove(user);
            datacontext.SaveChanges();
        }
    }
}
```

Клас контексту даних SystemContextSQL

```
using HealthyHabit.Models;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace HealthyHabit.DAL.Implementation
{
   public class SystemContextSQL : DbContext
   {
      public DbSet<Habit> Habit { get; set; }
      public DbSet<HabitCompleteDate> HabitCompleteDate { get; set; }
      public DbSet<User> User { get; set; }
      public DbSet<UserHabit> UserHabit { get; set; }
      public DbSet<Color> Colors { get; set; }
      public DbSet<Plant> Plants { get; set; }
      private List<UserHabit> habits;
      public SystemContextSQL(DbContextOptions<SystemContextSQL> options) : base(options)
      {
         Database.EnsureCreated();
         this.AddColors();
         this.AddFlowers();
      }
      private void AddColors()
      {
         if (this.Colors.ToList().Count == 0)
         {
            this.Colors.Add(new Color("#574b90", "Фіолетовий"));
            this.Colors.Add(new Color("#e66767", "Рожевий"));
            this.Colors.Add(new Color("#63cdda", "Голубий"));
            this.Colors.Add(new Color("#f5cd79", "Жовтий"));
            this.Colors.Add(new Color("#f19066", "Помаранчевий"));
         }
         this.SaveChanges();
      }
      private void AddFlowers()
      {
         if (this.Plants.ToList().Count == 0)
         {
            this.Plants.Add(new Plant(
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_preview.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage0.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage1.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage2.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage3.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage4.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage5.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage6.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage7.png",
               "https://raw.githubusercontent.com/mynameischeezee/Healthy-
Habit/master/HealthyHabit.View/.FlowersImages/Sunflower_test/sunflower_stage1.png",
               "Sunflower"));
         }
         this.SaveChanges();
      }
   }
}
```

Клас моделі даних Color

```
using System;
using System.Collections.Generic;
using System.Text;
using HealthyHabit.DAL.Abstract;
```

```
namespace HealthyHabit.Models
{
    public class Color : ModelTemplate
    {
        public string HexCode { get; set; }
        public string ColorName { get; set; }
        public Color()
        {

        }
        public Color(string Hex, string Name)
        {
            this.HexCode = Hex;
            this.ColorName = Name;
        }
    }
}
```

Клас моделі даних Habit

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;
using HealthyHabit.DAL.Abstract;

namespace HealthyHabit.Models
{
    public class Habit : ModelTemplate
    {
        public string Name { get; set; }
        public string Description { get; set; }
        public int  Progress { get; set; }
        public int Frequency { get; set; }
        public bool IsCompleted { get; set; }
        [ForeignKey(nameof(ColorId))]
        public Color Color { get; set; }
        public int ColorId { get; set; }
        public DateTime DateCreated { get; set; }
        [ForeignKey(nameof(PlantId))]
        public Plant Plant { get; set; }
        public int PlantId { get; set; }
        public Habit()
        {

        }
        public Habit(string name, string desciption, int progress, int frequency, bool iscompleted, Color color, DateTime datecreated, Plant
plant)
        {
            this.Name = name;
            this.Description = desciption;
            this.Progress = progress;
            this.Frequency = frequency;
            this.IsCompleted = iscompleted;
            this.Color = color;
            this.DateCreated = datecreated;
            this.Plant = plant;
        }
    }
}
```

Клас моделі даних HabitCompleteDate

```
using HealthyHabit.DAL.Abstract;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;
namespace HealthyHabit.Models
{
    public class HabitCompleteDate : ModelTemplate
    {
```

```
        [ForeignKey(nameof(HabitId))]
        public Habit Habit { get; set; }
        public int? HabitId { get; set; }
        public DateTime CompleteDate { get; set; }
        public HabitCompleteDate()
        {

        }
        public HabitCompleteDate(Habit habit, DateTime completeDate)
        {
            this.Habit = habit;
            this.CompleteDate = completeDate;
        }
        public HabitCompleteDate(Habit habit, int Id, DateTime completeDate)
        {
            this.Habit = habit;
            this.HabitId = Id;
            this.CompleteDate = completeDate;
        }
    }
}
```

Клас моделі даних Plant

```
using System;
using System.Collections.Generic;
using System.Text;
using HealthyHabit.DAL.Abstract;
namespace HealthyHabit.Models
{
    public class Plant : ModelTemplate
    {
        public string PreviewPath { get; set; }
        public string Stage0Path { get; set; }
        public string Stage1Path { get; set; }
        public string Stage2Path { get; set; }
        public string Stage3Path { get; set; }
        public string Stage4Path { get; set; }
        public string Stage5Path { get; set; }
        public string Stage6Path { get; set; }
        public string Stage7Path { get; set; }
        public string CurrentStage { get; set; }
        public string Name { get; set; }
        public Plant()
        {

        }
        public Plant(string previewpath, string stage0path, string stage1path, string stage2path, string stage3path, string stage4path, string
stage5path, string stage6path, string stage7path, string currentStage, string name)
        {
            this.PreviewPath = previewpath;
            this.Stage0Path = stage0path;
            this.Stage1Path = stage1path;
            this.Stage2Path = stage2path;
            this.Stage3Path = stage3path;
            this.Stage4Path = stage4path;
            this.Stage5Path = stage5path;
            this.Stage6Path = stage6path;
            this.Stage7Path = stage7path;
            this.CurrentStage = currentStage;
            this.Name = name;
        }
    }
}
```

Клас моделі даних User

```
using HealthyHabit.DAL.Abstract;
using System;
using System.Collections.Generic;
using System.Text;
namespace HealthyHabit.Models
{
```

```
    public class User : ModelTemplate
    {
        public string Name { get; set; }
        public string UserName { get; set; }
        public string Mail { get; set; }
        public string PasswordHash { get; set; }
        public string Salt { get; set; }

        public User()
        {

        }
        public User(string name, string username, string mail, string passwordhash, string salt)
        {
            this.Name = name;
            this.UserName = username;
            this.Mail = mail;
            this.PasswordHash = passwordhash;
            this.Salt = salt;
        }

    }
}
```

Клас моделі даних UserHabit

```
using HealthyHabit.DAL.Abstract;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;
namespace HealthyHabit.Models
{
    public class UserHabit : ModelTemplate
    {
        [ForeignKey(nameof(UserId))]
        public User User { get; set; }
        public int? UserId { get; set; }
        [ForeignKey(nameof(HabitId))]
        public Habit Habit { get; set; }
        public int? HabitId { get; set; }
        public UserHabit()
        {

        }
        public UserHabit(User user, Habit habit)
        {
            this.User = user;
            this.Habit = habit;
        }
    }
}
```