Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Коваленка Олександра Сергійовича*
	(ПІБ)
академічної групи	*121М-22-1*
	(шифр)
спеціальності	*121 Інженерія програмного забезпечення*
	(код і назва спеціальності)
освітньої програми	*«121 Інженерія програмного забезпечення»*
	(назва освітньої програми)
на тему:	*Розробка та дослідження ефективності впровадження програмного забезпечення розрахунку раціону харчування для хворих на цукровий діабет другого типу.*

_____*О.С. Коваленко*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Бердник М.Г.*			
Рецензент	*Проф. Гнатушенко В.В.*			
Нормоконтролер	*Проф. Лактіонов І.С.*			

Дніпро
2023

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)
_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)
« » _____ 20 23 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121м-22-1* *Коваленку Олександру Сергійовичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка та дослідження ефективності впровадження програмного забезпечення розрахунку раціону харчування для хворих на цукровий діабет другого типу.*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 09.10.2023 р. № 1227-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес розробки та впровадження програмного забезпечення для розрахунку раціону харчування для хворих на цукровий діабет другого типу.

Предмет досліджень – математичні моделі та програмне забезпечення, для розрахунку оптимального раціону харчування у хворих на цукровий діабет другого типу.

Мета роботи – розробка програмного забезпечення, призначеного для покращення оптимізації раціону харчування та контролю рівня цукру в крові хворих на цукровий діабет другого типу.

Вихідні дані для проведення роботи – теоретичні та експериментальні дослідження, методи оптимізації задач із пошуку оптимального рішення.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна – запропоновано метод автоматизації розрахунку оптимального раціону харчування хворих на цукровий діабет другого типу в залежності від індивідуальних даних хворих та із врахуванням максимальної можливої суми витрат на продукти харчування.

Практична цінність полягає в тому, що результат дослідження можна використовувати як у побутових цілях так і у медичних закладах задля формування оптимального раціону харчування із врахуванням бажаної максимальної суми витрат.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання запропонованих методів. В результаті роботи повинен бути розроблений програмний продукт для розрахунку раціону харчування хворих на цукровий діабет другого типу.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі.	16.09.2023-29.09.2023
Дослідження існуючих методів вирішення оптимізаційних задач.	30.09.2023-20.10.2023
Створення програмного забезпечення для розрахунку раціону харчування хворих на цукровий діабет другого типу.	21.11.2023-08.12.23

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки розробці алгоритму та програмного забезпечення для розрахунку раціону харчування для хворих на цукровий діабет другого типу.

Соціальний ефект від реалізації результатів роботи очікується позитивним, завдяки полегшенню процесу створення індивідуального раціону хворих на цукровий діабет другого типу.

Завдання видав _____ *Бердник М. Г.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Коваленко О.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *15.09.2023 р.*

Термін подання дипломного проекту до ЕК *12.12.2023*

РЕФЕРАТ

Пояснювальна записка: 106 сторінок, 18 таблиць, 52 рисунки, 3 додатки, 32 джерела.

Об’єкт дослідження: процес розробки та впровадження програмного забезпечення для розрахунку раціону харчування для хворих на цукровий діабет другого типу.

Предмет дослідження: математичні моделі та програмне забезпечення, для розрахунку оптимального раціону харчування у хворих на цукровий діабет другого типу.

Мета магістерської роботи: розробка програмного забезпечення, призначеного для покращення оптимізації раціону харчування та контролю рівня цукру в крові хворих на цукровий діабет другого типу.

Методи дослідження: методи дослідження базуються на основних принципах теорії багатовимірних моделей даних, теорії баз даних. Використано методи математичного моделювання.

Наукова новизна даної роботи полягає в обґрунтуванні та розв’язанні проблеми розрахунку оптимального раціону харчування хворих на цукровий діабет другого типу та створення відповідного ПЗ.

Практична цінність полягає в здатності ефективно вплинути на процес створення та керування оптимізованим раціоном харчування управління для користувачів, хворих на цукровий діабет другого типу. Програмне забезпечення надає можливість слідкувати за рівнем цукру у крові та порівнювати його із наявними медичними нормами.

Область застосування: результат кваліфікаційної роботи зосереджено на вдосконаленні процесу автоматизації розрахунку оптимального з точки зору споживання вуглеводів раціону харчування, а тому він буде найбільш корисним для побутового використання.

Значення роботи та висновки: автоматизація та полегшення підходу до управління харчуванням людей з діабетом 2 типу. Програмне забезпечення здатне розширити можливості пацієнтів зі слідкування за раціоном, що призведе до покращення дотримання дієтичних рекомендацій та покращення загальних результатів здоров’я. Це дослідження спрямоване на усунення критичної прогалини в сучасній практиці лікування діабету, пропонуючи ефективний інструмент, який допоможе пацієнтам ефективніше слідкувати за своїм раціоном та дотриманням медичних рекомендацій.

Прогнози щодо розвитку досліджень: подальші дослідження – додаткове вивчення та розробка нових методів та підходів для автоматизації розрахунку раціону та регуляції рівня цукру в крові. Впровадження інновацій: використання нових технологій для покращення функціональності та ефективності автоматизації. Співпраця з експертами в галузі охорони здоров’я: спільне вдосконалення створеного алгоритму розрахунку раціону пацієнтів, збільшуючи кількість оброблюваних даних і, як наслідок, рівень персоналізації результатів.

Список ключових слів: раціон харчування, БД, MVVM, рівень цукру, додаток, оптимізація, продукти харчування.

ABSTRACT

Explanatory note: 106 pages, 18 tables, 52 figures, 3 appendices, 32 sources.

Object of research: the process of developing and implementing software for calculating the diet for patients with type two diabetes mellitus.

Subject of research: mathematical models and software for calculating the optimal diet for patients with type two diabetes mellitus.

The purpose of the master's thesis: to develop software designed to improve diet optimization and blood sugar control in patients with type two diabetes mellitus.

Research methods: the research methods are based on the basic principles of the theory of multidimensional data models, database theory. Methods of mathematical modeling were used.

The scientific novelty of this work is to substantiate and solve the problem of calculating the optimal diet for patients with type two diabetes mellitus and to create appropriate software.

Practical value of the results consists of the ability to effectively influence the process of creating and managing an optimized diet management for users with type two diabetes mellitus. The software provides the ability to monitor blood sugar levels and compare them with existing medical standards.

Scope of application: the result of the qualification work is focused on improving the process of automating the calculation of the optimal diet in terms of carbohydrate consumption, and therefore it will be most useful for domestic use.

The value of the work and conclusions: automation and facilitation of the approach to nutrition management for people with type two diabetes. The software has the potential to empower patients to monitor their diets, leading to improved adherence to dietary recommendations and better overall health outcomes. This study aims to address a critical gap in current diabetes care by offering an effective tool to help patients more effectively monitor their diet and adherence to medical recommendations.

Research forecast and development: additional study and development of new methods and approaches to automate diet calculation and blood sugar regulation. Implementation of innovations: use of new technologies to improve the functionality and efficiency of automation. Collaboration with healthcare experts: joint improvement of the created algorithm for calculating patients' diets, increasing the amount of data processed and, as a result, the level of personalization of results.

Keywords: diet, database, MVVM, sugar level, application, optimization, food.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – application programming interface;

CDC – центри контролю та профілактики захворювань;

EF / EF Core – Entity Framework / Entity Framework Core;

LINQ – Language-Integrated Query;

MVC –Model-View-Controller;

MVP – Model-View-Presenter;

MVVM –Model-View-ViewModel;

UI –User Interface;

WPF – Windows Presentation Foundation;

XAML – eXtensible Application Markup Language;

XML – EXtensible Markup Language;

БД – база даних;

МДФ –Міжнародна діабетична федерація;

ОРСКБД / ORDBMS – об’єктно-реляційна система керування базами даних / object–relational database management system;

ПЗ – Програмне забезпечення;

РСКБД / RDBMS – реляційна система керування базами даних / relational database management system;

СКБД – система керування базами даних;

ХО – хлібна одиниця.

ЗМІСТ

ВСТУП

Діабет, особливо діабет другого типу, є поширеною проблемою охорони здоров'я в усьому світі, яка визначається складною взаємодією спадкових, поведінкових та екологічних змінних. Раціон харчування має вирішальне значення для належного лікування діабету другого типу, оскільки він допомагає підтримувати контроль рівня глюкози та запобігати ускладненням.

Актуальність дослідження полягає в розробці та оцінці нового програмного продукту, придатного для ефективного розрахунку оптимального споживання поживних речовин, насамперед вуглеводів, для пацієнтів з цукровим діабетом другого типу, оскільки це трудомісткий процес, що вимагає складних інженерних розрахунків, які враховують багато параметрів, таких як вік, стать, вага, кількість хлібних одиниць у прийомах їжі, тощо. Багато людей, які змушені самостійно контролювати рівень цукру в крові, стикаються з труднощами при визначенні вмісту кошика в супермаркеті.

Одним з ключових аспектів актуальності є економічний. Розроблюваний програмний продукт виходить за рамки простого розрахунку кількості спожитих вуглеводів. Він включає в себе функцію скорочення витрат, що підвищує його практичну корисність для кінцевого користувача. На основі введених даних алгоритм враховує фінансові обмеження, з якими може зіткнутися користувач, і намагається зменшити загальну вартість раціону. Цей метод враховує не лише ефективність харчування, але й реальні обмеження та економічні фактори, які можуть мати суттєвий вплив на здатність людини дотримуватися дієтичних рекомендацій.

Відсутність подібних інструментів у відкритому доступі в українській частині індустрії програмного забезпечення дозволяє цьому продукту заповнити суттєві прогалини в існуючому ландшафті лікування діабету і справити позитивний соціальний ефект.

Загалом, дослідження програмного забезпечення для розрахунку раціону харчування для пацієнтів з діабетом другого типу надає нові можливості до

підвищення якості сфери охорони здоров’я, оскільки воно може задовольнити унікальні потреби, які не можуть бути задоволені існуючими рішеннями.

Мета дослідження полягає в розробці програмного забезпечення, призначеного для покращення оптимізації раціону харчування та контролю рівня цукру в крові хворих на цукровий діабет другого типу.

Об’єкт дослідження: процес розробки та впровадження програмного забезпечення для розрахунку раціону харчування для хворих на цукровий діабет другого типу.

Предмет дослідження: математичні моделі та програмне забезпечення для розрахунку оптимального раціону харчування у хворих на цукровий діабет другого типу.

Методи дослідження: методи дослідження базуються на основних принципах теорії багатовимірних моделей даних, теорії баз даних. Використано методи математичного моделювання.

Наукова новизна: програмне забезпечення для розрахунку оптимального раціону харчування для хворих на цукровий діабет другого типу із можливістю моніторингу рівня глюкози у крові.

Практична цінність полягає в можливості успішно впливати на процес створення та дотримання оптимального раціону харчування для хворих на цукровий діабет 2 типу. Додаток дозволяє перевіряти рівень цукру в крові та порівнювати його з медичними нормами.

Особистий внесок автора:

1 Створення теоретичної частини роботи, яка досліджує та систематизує знання про цукровий діабет як сучасну проблему охорони здоров'я, вплив вуглеводів на рівень глюкози в крові та сучасні рекомендації щодо щоденного споживання вуглеводів.

2 Розробка математичної моделі, що відповідає поставленому завданню.

3 Дослідження результатів різних методів оптимізації, використаних на розробленій математичній моделі.

4 Вибір методів дослідження, оптимізації та технологій реалізації.

5 Розробка програмного забезпечення для розрахунку оптимального раціону для хворих на цукровий діабет 2 типу.

6 Аналіз та оцінка отриманих результатів.

Структура і обсяг роботи. Робота складається з вступу, трьох розділів, висновків, списку використаних джерел і двох додатків. Загальний обсяг роботи становить 106 сторінок, із них 61 сторінка основного тексту, в тому числі 52 рисунки, 18 таблиць і список використаних джерел із 32 найменування на 5 сторінках.

РОЗДІЛ 1. АНАЛІЗ ТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Проблема цукрового діабету

Цукровий діабет – це хронічне порушення обміну речовин, що характеризується високим рівнем глюкози (або цукру в крові), яке з часом призводить до катастрофічного пошкодження серця, судин, очей, нирок і нервів [2].

На діабет хворіють близько 422 мільйонів людей у всьому світі, більшість з яких проживає в країнах з низьким і середнім рівнем доходу, і діабет є безпосередньою причиною 1,5 мільйона смертей на рік. За останні кілька десятиліть кількість випадків діабету та його поширеність поступово зростали як у світі, так і в Україні, як наведено на рис. 1.1 [3].

Рис. 1.1. Кількість хворих на цукровий діабет за даними Всесвітньої організації охорони здоров'я та Міжнародної діабетичної федерації

Згідно з Атласом діабету Міжнародної діабетичної федерації (МДФ) станом на 2021 рік, 10,5% дорослого населення (20-79 років) хворіють на діабет, причому майже половина з них не знають про свою хворобу [4].

Так званий стан інсулін-резистентності виникає, коли клітини в організмі людини не реагують належним чином на гормон інсулін. Підшлункова залоза

виробляє додатковий інсулін, намагаючись стимулювати реакцію клітин. Коли підшлункова залоза більше не встигає, рівень цукру в крові підвищується, закладаючи основу для перед діабетичного стану та виникнення діабету 2 типу [5].

Діабет 2 типу вражає понад 90% людей, хворих на діабет, і це зумовлено соціально-економічними, демографічними, екологічними та генетичними факторами. Незважаючи на те, що певну кількість чинників виникнення цукрового діабету 2 типу можна контролювати – більшість з них, такі як старіння, належність до жіночої статі, генетику тощо, контролювати, наразі, неможливо [6].

1.2. Вплив вуглеводів на рівень цукру в крові

Незважаючи на те, що більшість факторів впливу на можливість розвитку цукрового діабету контролювати неможливо, певні дії у якості профілактики та навіть лікування, вжити можна. Наприклад, людина може слідкувати за своїм раціоном харчування та рівнем фізичних навантажень – відмовитися від споживання шкідливих продуктів, вживати більше води, а не солодких газованих напоїв та приділити увагу заняттям фізичними вправами. Все це дозволить контролювати рівень ожиріння, яке вважається однією з причин виникнення цукрового діабету другого типу.

При контролі раціону харчування для профілактики цукрового діабету 2 типу необхідно особливу увагу звертати на продукти з високим вмістом вуглеводів, оскільки, серед усіх макропоживних речовин, саме вони найбільше впливають на рівень цукру у крові [7].

Згідно з дослідженнями, низько вуглеводна дієта зменшує або усуває потребу в ліках і значно знижує рівень цукру в крові. Крім того, на відміну від фармацевтичних препаратів, дослідники стверджують, що ця дієта не має негативних наслідків [8].

У довгостроковій перспективі дієта з обмеженням вуглеводів є корисною. Протягом шести місяців люди з діабетом 2 типу дотримувалися низько

вуглеводної дієти. За цей час вони знизили вагу і підтримували стабільний рівень глюкози. Крім того, двоє учасників, які дотримувалися цієї дієти протягом трьох і двох років відповідно, скинули 20 кг і позбулися всіх симптомів діабету [9].

За даними Центрів контролю та профілактики захворювань (CDC), діабетик повинен споживати приблизно половину своїх калорій за рахунок вуглеводів.

Отже, якщо людина споживає 1800 калорій на день, щоб підтримувати здорову вагу, вуглеводи можуть становити 800-900 з цих калорій. Візуальне зображення такого співвідношення наведено на рисунку 1.2.

Рис. 1.2. Співвідношення кількості вуглеводів та інших поживних речовин для споживання 1800 калорій на день

За даними CDC, це означає 200-225 грамів вуглеводів на день [10].

Варто зазначити, що серед макроелементів, які позначаються загальним поняттям «вуглеводи» не всі з них негативно впливають на рівень цукру в крові, підвищуючи його. Наприклад, моносахариди (глюкоза) та дисахариди (сахароза) мають дуже високий вплив на рівень цукру в крові, якщо містяться у продуктах, які пройшли недостатню кулінарну обробку (наприклад, солодкі газовані напої). В той же час, якщо вжити, наприклад, персик – отримана з нього сахароза не

матиме згубного впливу на рівень цукру у крові, оскільки, окрім неї, організм також споживає клітковину, вітаміни та мінерали [11].

Для контролю рівня цукру в крові лікарями була створено відповідності значення рівня цукру у крові для здорових людей та хворих на цукровий діабет, як наведено у табл. 1.1 [12].

Таблиця 1.1

Таблиця відповідності рівня цукру у крові для здорової людини та хворого на цукровий діабет

	Вік	Значення рівня цукру у крові натще	Значення рівня цукру у крові через 1-2 години після їжі
Норма рівню цукру у крові здорової людини	$x < 14$ років	2,8-5,0 ммоль/л<100 мг/дл	6,5-7,8 ммоль/л120-140 мг/дл
	$14 \leq x < 18$ років	3,3-5,5 ммоль/л<100 мг/дл	6,5-7,8 ммоль/л120-140 мг/дл
	$18 \leq x < 50$ років	3,3-5,5 ммоль/л80-130 мг/л	7,8 ммоль/л90-140 мг/л
	$51 \leq x < 60$ років	3,8-5,9 ммоль/л80-130 мг/л	7,8 ммоль/л90-140 мг/л
	$61 \leq x < 90$ років	4,2-6,2 ммоль/л100-140 мг/л	7,8 ммоль/л100-140 мг/л
	$x > 90$ років	4,6-6,9 ммоль/л100-140 мг/л	7,8 ммоль/л100-140 мг/л
Норма рівню цукру у крові людини, хворої на цукровий діабет		4,4-7,2 ммоль/л80-130 мг/дл	10 ммоль/л<180 мг/дл

Оскільки слідкувати за кількістю вуглеводів, при споживанні багатих на них продуктів, може ставати важко – можливо, доведеться рахувати їх сотнями

– вчені ввели одиницю виміру «Хлібна одиниця», таку, що задовольняє рівність (1.1) [13].

$$1\ \text{ХО} \approx 10 - 12\ \text{г. споживаних вуглеводів} \tag{1.1}$$

Згідно із наявними рекомендаціями, норми споживання вуглеводів за хлібними одиницями можна виділити так, як наведено у табл. 1.2 – 1.4.

Таблиця 1.2

Обмеження кількості споживаних хлібних одиниць залежно від індексу маси тіла

Індекс маси тіла	Кількість ХО
$x \leq 18{,}49$	18-30
$35 < x \leq 40$	10
$x > 40$	6-8

Таблиця 1.3

Обмеження кількості споживаних хлібних одиниць залежно від віку та статі людини

Вік, років	Кількість ХО
4-6	12-13
7-10	15-16
11-14	18-20 (хлопчики)
	16-17 (дівчата)
15-18	19-21 (хлопчики)
	17-18 (дівчата)
Дорослі	20-22

Таблиця 1.4

Обмеження кількості споживаних хлібних одиниць залежно від рівня фізичної активності

Рівень фізичного навантаження	Стать	Кількість ХО
1	2	3
Високий	Чоловіки та жінки	23-28
Середній	Чоловіки	19-22
	Жінки	16-19

Продовження таблиці 1.4

1	2	3
Низький	Чоловіки	14-16
	Жінки	10-14

1.3. Постановка задачі

Розробити математичну модель, метод розв’язку та програмне забезпечення мовою програмування C# для розрахунку раціону харчування для хворих на цукровий діабет другого типу.

Для вирішення поставленої задачі необхідно:

– дослідити програмні та математичні методи, які дозволяють розраховувати раціон харчування;

– розробити програмне забезпечення для розрахунку раціону харчування для хворих на цукровий діабет другого типу.

Програмне забезпечення повинно відповідати наступним функціональним вимогам:

– наявність зрозумілого та дружнього графічного користувальницького інтерфейсу;

– наявність можливості ведення даних про користувачів та продукти харчування для проведення подальших обчислень;

– наявність можливості аналізувати введені дані рівню цукру у крові впродовж заданого проміжку часу та будувати графік змін;

– наявність можливості обчислення раціону харчування на основі вказаних вимог (максимальної допустимої суми витрат, наявності або відсутності певних продуктів харчування, рекомендацій щодо споживання вуглеводів).

Додаток використовує дані користувача (вік, стать, ступінь фізичної активності), дані про продукти харчування (ціна, вага, кількість продуктів) та максимальну суму витрат як вхідні дані.

Усі введені дані слід перевіряти на точність.

Дані, які були введені та розраховані, повинні бути збережені в базі даних.

1.4. Висновки

У даному розділі було проведено аналіз предметної області, були визначені поняття та властивості захворювання на цукровий діабет другого типу та статистика захворюваності. Також, було визначено залежність між споживанням вуглеводів та підвищенням рівня цукру у крові, наведено рекомендації щодо обчислення та щоденного споживання вуглеводів хворими на цукровий діабет другого типу.

Отримана інформація надає можливість зрозуміти та побудувати математичну модель для розрахунку раціону хворих на цукровий діабет другого типу. Проблема полягає у створенні відповідного програмного забезпечення.

Актуальність розробки полягає у створенні програмного забезпечення, яке дозволить автоматизувати та спросити процес розрахунку раціону харчування хворих на цукровий діабет другого типу.

РОЗДІЛ 2. РОЗРОБКА МАТЕМАТИЧНОЇ МОДЕЛІ І МЕТОДУ ОПТИМІЗАЦІЇ

2.1. Математична модель

Якщо людині, хворій на цукровий діабет другого типу, необхідно споживати певну кількість вуглеводів (вимірюється в хлібних одиницях) щодня, вона повинна вживати певну кількість їжі, з якої ці вуглеводи (хлібні одиниці) будуть отримані.

Згідно з дослідженнями, кожен продукт харчування має різний рівень вмісту вуглеводів через природу свого походження, що призводить до ситуацій, коли споживання одного продукту призводить до недотримання норми споживання хлібних одиниць, тоді як споживання іншого – до перевищення цієї норми.

Як наслідок, необхідно розробити обмеження математичної моделі, які б гарантували, що загальна кількість хлібних одиниць у спожитій їжі знаходиться в діапазоні від мінімального до максимального запропонованого значення, залежно від стану конкретного пацієнта. Математично, такі обмеження можна записати у вигляді нерівності (2.1):

$$Crh_1 < \sum_{i=0}^{n-1} x_i * BU_i \ < Crh_2, \qquad (2.1)$$

де Crh_1 – мінімальна денна рекомендована до споживання кількість хлібних одиниць;

Crh_2 – максимальна денна рекомендована до споживання кількість хлібних одиниць;

x_i – і-й продовольчий продукт;

BU_i – вміст хлібних одиниць в і-му продукті;

$i = 0,1,2 \dots n-1$ – номер обраного продукту харчування;

n – загальна кількість обраних продуктів харчування.

На додаток до різного вмісту хлібних одиниць, кожен продукт має свою ціну, яка може коливатися в залежності від різних змінних, таких як неврожай, військові дії, неможливість імпорту або експорту тощо. Враховуючи необхідність розподілу матеріальних ресурсів та складність їх використання виключно для придбання продуктів харчування, варто додати до математичної моделі обмеження, щоб загальна кількість придбаних продуктів харчування не перевищувала суму, яку людина виділила б на бюджетні покупки. Математично, це можна представити у вигляді співвідношення (2.2).

$$\sum_{i=0}^{n-1} p_i x_i = h, h > 0, \quad (2.2)$$

де x_i – кількість придбаного і-го продукту харчування;

p_i – вартість придбаного і-го продукту харчування;

h – загальна сума, виділена на закупки продовольчих товарів.

Важливо також обмежити максимальне добове споживання кожного товару, який братиме участь в обчисленнях, щоб не виникали сценарії з нелогічними або навіть безглуздими значеннями, наприклад, 1 кілограм хліба на день. Формула (2.3) є тим обмеженням, яке дозволить це регулювати.

$$0 \leq x_i \leq Am_{max}, \quad (2.3)$$

де x_i – кількість придбаного і-го продукту харчування в грамах;

Am_{max} – максимальна рекомендована кількість щоденного споживання продукту.

Слід зазначити, що максимальна кількість щоденного споживання в грамах залежить від виду продукту. Наприклад, 300 г для хліба, 450 г для картоплі, 1,5 г для солі та цукру і так далі.

Оскільки сума, витрачена на харчування, яке обирає пацієнт, не повинна становити основну частину бюджету, доречно мінімізувати цю суму, отримавши функцію (2.4).

$$h \to min \quad (2.4)$$

Отримана функція може слугувати цільовою для створюваної математичної моделі.

2.2. Вибір методу оптимізації

Для вирішення цієї проблеми було розглянуто декілька методів оптимізації, включаючи метод найменших квадратів, лінійну оптимізацію, метод Пауелла та метод Нелдера-Міда.

Створено набір даних для порівняння результатів обраних методів оптимізації, див. табл. 2.1.

Таблиця 2.1

Тестовий набір даних для перевірки результатів оптимізації

№ продукту	Ціна продукту, грн	Вага / об'єм продукту	Кількість ХО, що міститься у продукті	Максимальна сума витрат, грн	Мінімальна кількість ХО до споживання	Максимальна кількість ХО до споживання
1	33,63	240 г	1,21	1000	23	28
2	28,40	1 кг	0,20			
3	105,90	1 кг	0,56			
4	46,20	1 л	0,36			

На рисунку 2.1 зображено результати оптимізації з використанням підходів, описаних вище. При оптимізації максимальна кількість продуктів була вказана 250 г.

```
Лінійна оптимізація:
Кількість кожного продукту, гр: 250.0, 250.0, 250.0, 250.0
Загальна вартість, грн: 80.156
Загальна кількість хлібних одиниць: 1.54

МНК з обмеженнями:
Кількість кожного продукту, гр: 239.153, 250.0, 250.0, 250.0
Загальна вартість, грн: 78.636
Загальна кількість хлібних одиниць: 1.486

Нелдер-Мід:
Кількість кожного продукту, гр: 206.612, 1250.0, 446.429, 694.444
Загальна вартість, грн: 143.812
Загальна кількість хлібних одиниць: 1.792

Пауелл:
Кількість кожного продукту, гр: 206.612, 1250.0, 446.429, 694.444
Загальна вартість, грн: 143.812
Загальна кількість хлібних одиниць: 1.792
```

Рис. 2.1. Отримані результати оптимізації методами найменших квадратів, лінійної оптимізації, Пауелла та Нелдера-Міда

Як можна побачити, методи Пауелла та Нелдера-Міда виявилися більш ефективними з точки зору загальної кількості хлібних одиниць в отриманому раціоні, хоча їхні результати перевищують максимально рекомендовану добову норму споживання різних продуктів, що змушує їх відкинути у якості методів оптимізації для поставленого завдання.

Порівняно з методом найменших квадратів, лінійна оптимізація дає дещо кращі результати, але з вищою загальною вартістю закупівель, що не відповідає обмеженням цільової задачі.

В результаті, для оптимізації рішення цієї задачі було обрано метод найменших квадратів, який використовується, щоб вирішити проблему визначення значень змінної x таким чином, щоб кожне з них на першій ітерації $i = 1, \dots m$ було якомога ближче до заданого значення y_i. Математично, це описується виразом (2.5).

$$\sum_i e_i^2 = \sum_i (y_i - f_i(x))^2 \to \min_x \tag{2.5}$$

При розв’язанні задачі лінійного програмування виду (2.6)

$$\min_{x} f(x), \tag{2.6}$$

з обмеженнями (2.7) і (2.8),

$$b(x) \geq 0; \tag{2.7}$$

$$c(x) = 0, \tag{2.8}$$

метод найменших квадратів шукає значення функції Лагранжа типу (2.9).

$$L(x, \lambda, \sigma) = f(x) - \lambda^{T} b(x) - \sigma^{T} c(x), \tag{2.9}$$

де, λ та σ – значення множників Лагранжа.

2.3. Висновки

У даному розділі було побудовано математичну модель поставленої задачі, яка спростить пошук оптимального її вирішення. Отримана модель є актуальною, оскільки дозволяє здійснювати пошук оптимального рішення за актуальними, з точки зору здоров’я хворої на цукровий діабет другого типу людини, критеріями кількості споживання хлібних одиниць. Розроблена математична модель також може бути корисною при вирішенні задач, спрямованих на оптимізацію закупок при обмеженій кількості матеріальних засобів.

Отримані результати оптимізації тестового набору даних кількома методами оптимізації показали, що з попередньо обраних для оптимізації методів підходящим виявився метод найменших квадратів, який, в подальшому, було обрано для створення програмної реалізації алгоритму пошуку оптимального рішення поставленої задачі.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РОЗРАХУНКУ РАЦІОНУ ХАРЧУВАННЯ ДЛЯ ХВОРИХ НА ЦУКРОВИЙ ДІАБЕТ ДРУГОГО ТИПУ

3.1. Опис структури бази даних

Для виконання поставленої задачі було спроектовано реляційну базу даних зі структурою, зображеною на рисунку 3.1.

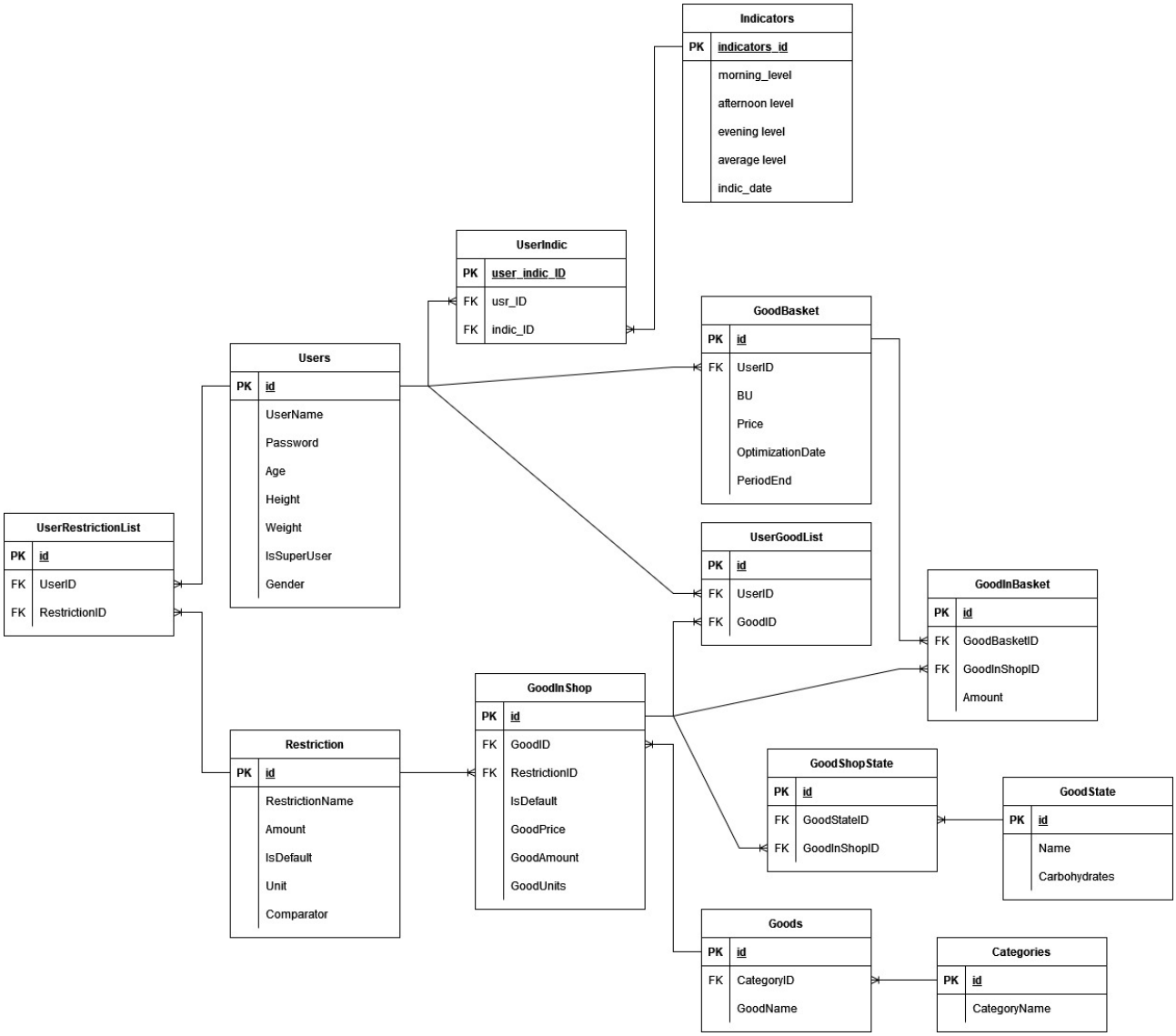


Рис. 3.1. Структура бази даних

Як можна побачити, спроектована база даних складається з тринадцяти таблиць:

– Users – зберігає дані про користувачів системи;

– UserRestrictionList – зберігає дані про користувальницький список обмежень споживання продуктів;

– UserIndic – забезпечує зв’язок між користувачем та показниками рівня цукру в крові;

– UserGoodList – зберігає список користувальницьких продуктів харчування;

– Indicators – зберігає показники рівня цукру у крові користувача;

– Restriction – зберігає список обмежень;

– GoodInShop – містить дані про продукт у крамниці;

– Goods – містить дані про продукти;

– Categories – містить дані про категорії продуктів;

– GoodShopState – зберігає дані про стан продукту в крамниці;

– GoodState – містить дані про стан продукту;

– GoodInBasket – зберігає список продуктів в кошику;

– GoodBasket – містить дані про отриманий кошик.

При аналізі предметної області було з’ясовано, що дані про користувача повинні включати інформацію про його / її вік, стать, вагу, зріст та рівень фізичного навантаження. Окрім того, оскільки необхідно, також, передбачити функціонал із контролю рівню цукру у крові за наявними рекомендаціями [14-16], хворі на цукровий діабет другого типу повинні робити заміри вранці натще, вдень та ввечері, а отже, система повинна, також, зберігати і цю інформацію. З програмної точки зору, дані про користувача повинні, також містити його / її логін та пароль для авторизації та автентифікації. За цими міркуваннями було спроектовано структури даних, див. табл. 3.1 – 3.2.

Таблиця 3.1

Структура таблиці Users

Назва поля	Призначення поля	Тип даних	Ключ
id	Унікальний ідентифікатор	integer	Primary key
UserName	Логін	text	
Password	Пароль	text	
Age	Вік	integer	
Height	Зріст	double	
Weight	Вага	double	
Gender	Стать	text	
IsSuperUser	Показник, чи є користувач адміністратором	boolean	

Таблиця 3.2

Структура таблиці Indicators

Назва поля	Призначення поля	Тип даних	Ключ
1	2	3	4
indicators_id	Унікальний ідентифікатор	integer	Primary key
morning_level	Показник рівня цукру в крові вранці	double	
afternoon_level	Показник рівня цукру в крові вдень	double	

Продовження таблиці 3.2

1	2	3	4
evening_level	Показник рівня цукру в крові ввечері	double	
average_level	Середнє значення рівня цукру в крові	double	
indic_date	Дата проведення замірів	date	

Для забезпечення зв’язку між користувачем та його показниками рівня цукру в крові було створено структуру даних, див. табл. 3.3.

Таблиця 3.3

Структура таблиці UserIndic

Назва поля	Призначення поля	Тип даних	Ключ
user_indic_id	Унікальний ідентифікатор	integer	Primary key
usr_ID	Покажчик на користувача	integer	Foreign key
indic_ID	Покажчик на заміри	integer	Foreign key

Користувач може сам приймати рішення щодо споживання тих чи інших продуктів харчування, керуватися рекомендаціями дієтологів, нутриціологів або ендокринологів чи, наприклад, релігійними або культурними заборонами. Зважаючи на це, доцільно зберігати інформацію про обмеження споживання продуктів харчування – як загально прийняті, так і користувацькі, наприклад,

заборону на споживання свинини, непереносимість молочних продуктів тощо. Це зумовлює створення структур даних, див. табл. 3.4 – 3.5.

Таблиця 3.4

Структура таблиці Restriction

Назва поля	Призначення поля	Тип даних	Ключ
id	Унікальний ідентифікатор	integer	Primary key
RestrictionName	Назва	text	
Amount	Кількість продукту	double	
IsDefault	Показник, чи є обмеження стандартним	boolean	
Comparator	Тип обмеження	text	
Unit	Одиниці виміру	text	

Таблиця 3.5

Структура таблиці UserRestrictionList

Назва поля	Призначення поля	Тип даних	Ключ
1	2	3	4
id	Унікальний ідентифікатор	integer	Primary key
RestrictionID	Покажчик на обмеження	integer	Foreign key
UserID	Покажчик на користувача	integer	Foreign key

Для зручності обліку продуктів та для полегшення навігації користувача списком продуктів було спроектовано структуру даних для збереження інформації про категорію товарів, до яких відноситься той чи інший продукт, див. табл. 3.6.

Таблиця 3.6

Структура таблиці Categories

Назва поля	Призначення поля	Тип даних	Ключ
id	Унікальний ідентифікатор	integer	Primary key
CategoryName	Назва	text	

Виходячи з постановки задачі, про продукти харчування необхідно зберігати дані щодо їх ціни та кількості хлібних одиниць (або вуглеводів). Також не зайвим буде включити до цього списку назву товару, його кількість та стан, оскільки в залежності від того, чи ви вживаєте, наприклад, свіжу капусту чи тушковану, відсоткове співвідношення поживних речовин, зокрема, вуглеводів, в ній змінюється. На основі даної інформації було сформовано відповідні структури, див. табл. 3.7 – 3.8.

Таблиця 3.7

Структура таблиці Goods

Назва поля	Призначення поля	Тип даних	Ключ
id	Унікальний ідентифікатор	integer	Primary key
GoodName	Назва	text	
CategoryID	Ідентифікатор категорії	integer	Foreign key

Таблиця 3.8

Структура таблиці GoodState

Назва поля	Призначення поля	Тип даних	Ключ
id	Унікальний ідентифікатор	integer	Primary key
Name	Назва стану	text	
Carbohydrates	Кількість хлібних одиниць	double	

Окрім того, в крамниці (зауважимо, що поняття «крамниця» є відносним і може стосуватися як офіційної крамниці, так і ринку, знайомих виробників тощо) може не знайтися, наприклад, тушкованої капусти, а лише свіжа, або квашена. Це призводить до необхідності зберігати інформацію про те, в якому стані товар присутній в крамниці. Також, не в усіх крамницях є однакові товари, тому бажано вказати до збереження інформацію про те, який товар доступний в крамниці, як наведено в таблицях 3.9 – 3.10.

Таблиця 3.9

Структура таблиці GoodInShop

Назва поля	Призначення поля	Тип даних	Ключ
1	2	3	4
id	Унікальний ідентифікатор	integer	Primary key
GoodId	Покажчик на продукт	integer	Foreign key
IsDefault	Показник, чи є продукт стандартним	boolean	

Продовження таблиці 3.9

1	2	3	4
GoodPrice	Ціна	double	
GoodAmount	Кількість	double	
GoodUnits	Одиниці виміру	text	
RestrictionID	Покажчик на обмеження	integer	Foreign key

Таблиця 3.10

Структура таблиці GoodShopState

Назва поля	Призначення поля	Тип даних	Ключ
id	Унікальний ідентифікатор	integer	Primary key
GoodStateID	Покажчик на стан продукту	integer	Foreign key
GoodInShopID	Покажчик на продукт у крамниці	integer	Foreign key

Як було зазначено вище, поняття «крамниця» є дуже відносним, і користувач може мати бажання додати до списку наявних продуктів, наприклад, відбірні курячі яйця, які купує у своєї сусідки за привабливішою ціною. Задля забезпечення цієї можливості було спроектовано таблицю 3.11.

Таблиця 3.11

Структура таблиці GoodShopState

Назва поля	Призначення поля	Тип даних	Ключ
id	Унікальний ідентифікатор	integer	Primary key
GoodInShopID	Покажчик на продукт у крамниці	integer	Foreign key
UserID	Покажчик на користувача	integer	Foreign key

Наостанок, оскільки метою розробки програмного забезпечення є розрахунок оптимального з точки зору складу та ціни продуктового кошику, варто зберігати інформацію про, власне, продуктовий кошик. До цієї інформації відноситься загальна його вартість, загальна кількість хлібних одиниць продуктів, які до нього входять, данні про те, які саме продукти знаходяться у цьому кошику та в якій кількості. Також, сюди відноситься інформація про те, коли користувач виконував ці обчислення, та, на який проміжок часу, оскільки це впливає на значення норм споживання продуктів харчування. Усе вищезазначене зумовлює створення таблиць 3.12 та 3.13.

Таблиця 3.12

Структура таблиці GoodBasket

Назва поля	Призначення поля	Тип даних	Ключ
1	2	3	4
id	Унікальний ідентифікатор	integer	Primary key

Продовження таблиці 3.9

1	2	3	4
UserID	Покажчик на користувача	integer	Foreign key
BU	Загальна кількість хлібних одиниць	double	
Price	Загальна ціна кошику	double	
OptimizationDate	Дата створення кошику	date	
PeriodEnd	Дата завершення періоду оптимізації	date	

Таблиця 3.13

Структура таблиці GoodInBasket

Назва поля	Призначення поля	Тип даних	Ключ
id	Унікальний ідентифікатор	integer	Primary key
GoodInShopID	Покажчик на продукт у крамниці	integer	Foreign key
UserID	Покажчик на користувача	integer	Foreign key

3.2. Застосовані технології та методи розробки програмного забезпечення

Програмний додаток реалізовано за шаблоном проектування Model-View-ViewModel (MVVM), який є умовним нащадком шаблонів MVC та MVP, основна перевага якого полягає у розмежуванні візуальної частини від логіки його роботи [17]. Основний принцип роботи MVVM наведено на рис. 3.2.

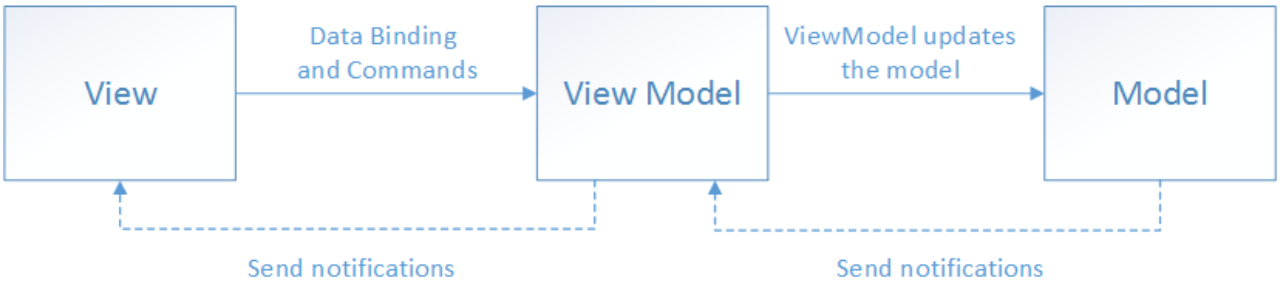


Рис. 3.2. Логіка роботи шаблону проектування MVVM

MVVM має наступні рівні коду:

1 Модель – відповідає за абстрагування джерел даних. Model і ViewModel співпрацюють для отримання та зберігання даних. Моделі – це класи, які представляють предметну область програми. Моделі містять дані та бізнес-логіку програми. Це бізнес-об’єкти, які не мають нічого спільного з візуальним виглядом програми [18].

2 View – повідомляє ViewModel про дії користувача. Цей рівень лише контролює ViewModel і не містить коду програми. Вся взаємодія з користувачем відбувається в межах View, який відповідає за виявлення введених даних (клацання мишею, введення з клавіатури тощо) і надсилання їх до ViewModel за допомогою зв’язування даних. Прив’язка даних, яка може бути здійснена за допомогою зворотного виклику або властивостей, є фізичним зв’язком між View та ViewModel [18].

3 ViewModel – забезпечує потоки даних, які є важливими для Представлення. Він також діє як з’єднувач між моделлю та представленням. ViewModel реалізує атрибути та інструкції, які можуть бути прив’язані до View.

Ці атрибути та дії визначають функціональність, яку View може надати користувачеві, але відображення цієї функціональності повністю залежить від View. ViewModel також відповідає за надсилання даних з класів Model до View, які View може споживати. Для цього ViewModel може безпосередньо надавати класам Model доступ до View, у цьому випадку клас Model повинен увімкнути зв’язування даних та події сповіщення про зміни [18].

Шаблон MVVM має ряд переваг, порівняно з іншими шаблонами проектування [19]:

1 Розподіл обов’язків.

2 Легкість тестування.

3 Повторне використання коду.

4 Легкість підтримки.

Розподіл обов’язків є ключовим поняттям у розробці програмного забезпечення. Воно дозволяє зосередитися на одному компоненті проблеми за раз, що робить складні системи більш зрозумілими для міркувань. Відокремлюючи код інтерфейсу користувача від коду бізнес-логіки, MVVM дозволяє вам розділяти завдання. Наприклад, команда дизайнерів може зосередитися на користувацькому інтерфейсі (UI), в той час як розробники працюють над логікою (ViewModel і Model).

За допомогою MVVM ви можете відокремити представлення від моделі даних. Іншими словами, можна використовувати одностороннє зв’язування даних, щоб підтримувати їх синхронізацію. Оскільки MVVM має модульну структуру і слабкі зв’язки, це полегшує тестування коду.

Інтерфейс користувача (UI) є одним з найскладніших аспектів програми для тестування. Оскільки ViewModel і Model повністю відокремлені від View, розробники можуть створювати тести для обох, не звертаючись до View. Окрім цього, існує можливість змінити один рядок коду, не впливаючи на інші. Наприклад, якщо ви хочете змінити логіку підрахунку балів користувача в грі, вам не потрібно турбуватися про зміну способу відображення даних на екрані.

Оскільки кожен компонент можна тестувати окремо, MVVM спрощує розробку модульних тестів. Ви можете використовувати ін’єкцію залежностей для введення фейкових об’єктів у ваш код, усуваючи необхідність тестування зовнішніх залежностей, таких як мережа або база даних.

MVVM дозволяє розробляти код для багаторазового використання, розбиваючи проект на менші, простіші в обслуговуванні частини. Ви також можете створювати шаблони для цих частин, щоб використовувати їх в інших проектах. MVVM спрощує розробку багаторазового коду. Ви можете повторно використовувати ту саму модель представлення і код презентера у багатьох проектах, наприклад, якщо вам потрібно відобразити список об’єктів на екрані, а потім дозволити користувачам вибрати один з них для отримання додаткової інформації.

Розділення багатьох компонентів програми також спрощує та очищає код. Як результат, код програми значно легше зрозуміти, а отже, і підтримувати. Легко зрозуміти, де необхідно розмістити нові функції і як вони вписуються в існуючий шаблон. Також за допомогою MVVM легко інтегрувати інші архітектурні шаблони (інверсія залежностей, сервіси тощо).

Windows Presentation Foundation (WPF) – це фреймворк Windows для розробки десктопних додатків, який підтримує парадигму MVVM. Він має можливість зв’язування даних для з’єднання шарів View і ViewModel, а також команди для управління взаємодією з користувачем. Саме за допомогою цього фреймворку було розроблено ПЗ для вирішення поставленої задачі.

WPF складається з трьох частин [20]:

– XAML (eXtensible Application Markup Language) – декларативна мова розмітки, яка використовується для побудови користувацьких інтерфейсів в WPF додатках;

– .NET Framework – фреймворк для створення потужних додатків на таких мовах, як C# та Visual Basic;

– DirectX – набір API, який використовується в WPF-програмах для створення візуальних ефектів.

Переваги WPF у порівнянні з WinForms [21]:

– WPF є більш адаптованою, що дозволяє вам досягти більшого без необхідності розробляти чи купувати нові елементи керування;

– XAML спрощує побудову та модифікацію графічного інтерфейсу і дозволяє розділити роботу між дизайнером та програмістом;

– прив’язка даних дозволяє більш чітко розділити дані та макет;

– для підвищення ефективності графічний інтерфейс промальовується з використанням апаратного прискорення;

– дозволяє створювати користувацькі інтерфейси як для Windows, так і для веб-додатків (Silverlight/XBAP).

Вищезгаданий механізм прив’язки даних значно спрощує розробку та подальшу підтримку і модифікацію програмного забезпечення із використанням фреймворку WPF. Принцип роботи цього механізму наведено на рис. 3.3.

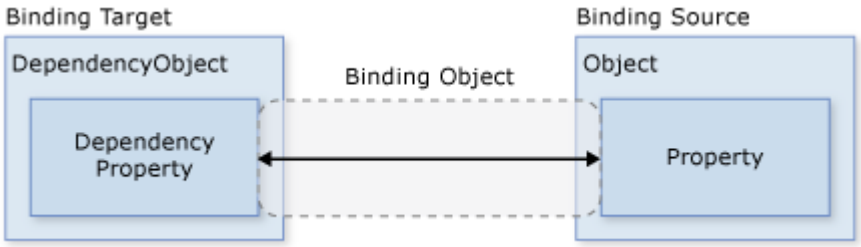


Рис. 3.3. Принцип роботи механізму прив’язки у WPF

За допомогою прив’язок можна зручно налаштовувати взаємодію View із ViewModel, наприклад, вказувати куди повинні зберігатися дані, які користувач вводить до елементів графічного інтерфейсу, та які команди повинні будуть бути виконані, коли відповідний елемент графічного інтерфейсу буде активовано. Приклад таких прив’язок наведено на рис. 3.4 – 3.5.

```
<MenuItem Header="Цукор у крові">
    <MenuItem Command="{Binding BloodSugarCalculationCommand}" Header="Ввести показники рівню цукру в крові" >
        <MenuItem.Icon>
            <Image Source="../Images/BloodDrop.png" Stretch="UniformToFill" />
        </MenuItem.Icon>
    </MenuItem>
    <MenuItem.Icon>
        <Image Source="../Images/BloodDrop.png" Stretch="UniformToFill" />
    </MenuItem.Icon>
</MenuItem>
```

Рис. 3.4. Прив’язка команди

```
<TextBox
    Grid.Row="1"
    MinWidth="100"
    Margin="0,5,0,0"
    Text="{Binding Login, UpdateSourceTrigger=PropertyChanged}" />
```

Рис. 3.5. Прив’язка даних

Для того, щоб зв’язати створене ПЗ із створеною базою даних було обрано і використано технологію Entity Framework Core. Entity Framework – це фреймворк для об’єктно-реляційного відображення (O/RM). Це розширення ADO.NET, яке надає розробникам автоматизовану техніку доступу та збереження даних у базах даних [22].

EF Core пропонує два підходи до розробки:

1 Спочатку код.

2 Спочатку база даних.

Оскільки візуальний конструктор або майстер для моделі БД не підтримується EF Core, він в першу чергу орієнтований на метод «спочатку код» і надає обмежену допомогу для підходу «спочатку база даних».

У методі «спочатку код» EF Core API використовує міграцію для створення бази даних і таблиць в залежності від угод і налаштувань, зазначених у ваших доменних класах. Цей метод корисний при доменно-орієнтованому проектуванні (DDD).

EF Core API створює доменні та контекстні класи на основі поточної бази даних, використовуючи команди EF Core в методі «спочатку база даних». Оскільки він не підтримує візуальний конструктор або майстер, цей метод має обмежену підтримку в EF Core.

Схематичний принцип роботи вищезазначених підходів наведено на рис. 3.6.

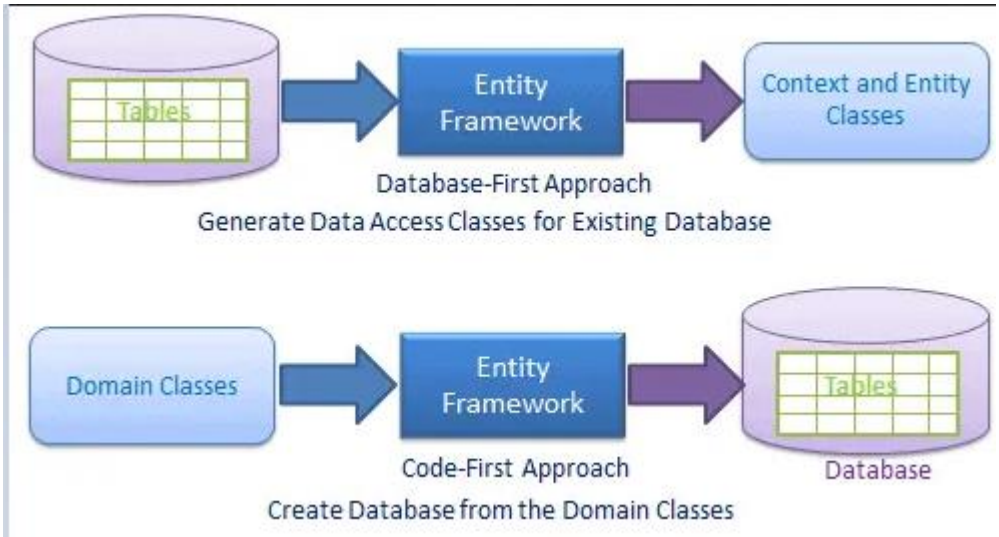


Рис. 3.6. Візуалізація підходів до розробки EF Core

Доступ до даних в EF Core здійснюється за допомогою моделі. Модель складається з класів сутностей і контекстного об’єкта, який представляє сеанс роботи з базою даних. Контекстний об’єкт дозволяє запитувати та зберігати дані [23].

EF підтримує наступні методології розробки моделей:

– створення моделі на основі існуючої бази даних;

– створення моделі вручну відповідно до бази даних;

– використання EF Migrations для створення бази даних на основі створеної моделі. Міграції дозволяють базі даних розвиватися разом зі зміною моделі.

Приклад використання методів EF Core для створення моделі у створеному ПЗ наведено на рис. 3.7.

```
public class GoodBasket
{
    //ідентифікатор
    Ссылок: 10
    public int id { get; set; }
    //ідентифікатор користувача
    Ссылок: 4
    public int UserID { get; set; }
    //покажчик на модель "користувач"
    Ссылок: 0
    public User User { get; set; }
    //загальна кількість хлібних одиниць у продуктовому кошику
    Ссылок: 4
    public double BU { get; set; }
    //загальна ціна продуктового кошику
    Ссылок: 4
    public double Price { get; set; }
    //продукти у продуктовому кошику
    Ссылок: 5
    public ObservableCollection<GoodInBasket> GoodInBasket { get; set; } = new ObservableCollection<GoodInBasket>();
    //дата розрахунку раціону харчування
    Ссылок: 4
    public DateTime OptimizationDate { get; set; }
    //дата закінчення часового проміжку розрахунку раціону харчування
    Ссылок: 4
    public DateTime PeriodEnd { get; set; }
    //конструктор з параметрами
    Ссылок: 2
    public GoodBasket(int userID, double bU, double price, DateTime optimizationDate, DateTime periodEnd)
    {
        UserID = userID;
        BU = bU;
        Price = price;
        OptimizationDate = optimizationDate;
        PeriodEnd = periodEnd;
    }
}
```

Рис. 3.7. Приклад створення моделі EF Core

Для отримання необхідних значень з бази даних, до якої підключено EF Core існує інструментарій LINQ [24].

Інтегровані в мову запити (Language-Integrated Query) відносяться до ряду технологій, які інтегрують можливості запитів безпосередньо в мову програмування C#. Традиційно запити до даних представляються у вигляді звичайних рядків без перевірки типів або допомоги IntelliSense під час складання. Крім того, для кожного типу джерела даних (бази даних SQL, XML-документи, численні веб-сервіси тощо) ви повинні освоїти нову мову запитів. Запит, як і класи, методи та події, є першокласною мовною конструкцією у LINQ.

Вираз запиту є найбільш очевидним «інтегрованим у мову» аспектом LINQ для розробника, який конструює запити. Для написання виразів запитів використовується декларативний синтаксис. Синтаксис запитів дозволяє виконувати дії фільтрації, сортування та групування даних у джерелах даних за допомогою невеликої кількості коду. Для запитів і перетворення даних у базах даних SQL, наборах даних ADO.NET, документах і потоках XML та колекціях

.NET ви використовуєте ті самі фундаментальні шаблони виразів запитів, що дозволяє зекономити час на вивченні аспектів і тонкощів виконання запитів у різних середовищах.

Приклад використання LINQ-запиту наведено на рис. 3.8.

```
var userIndics = context.UserIndic
    .Include(ui => ui.User)
    .Include(ui => ui.Indicators)
    .Where(ui => ui.User.id == userId)
    .ToList();
```

Рис. 3.7. Приклад створення моделі EF Core

3.3. Застосовані програмні засоби для реалізації програмного забезпечення

Для створення бази даних було використано PostgreSQL – надійну об’єктно-реляційну систему баз даних з відкритим вихідним кодом, яка використовує та розширює мову SQL і має кілька можливостей для належного зберігання та масштабування найскладніших вимог до даних [25–26]

Перевагами PostgreSQL є:

1 Наявність всіх необхідних типів даних, таких як документи, примітиви, геометрія, структури тощо.

2 Забезпечення цілісності даних шляхом встановлення обмежень та обмеження даних, які вносяться до БД.

3 Продуктивність. Розпаралелювання запитів на читання, потужні алгоритми індексування, контроль багатоверсійного паралелізму. Це лише деякі з багатьох удосконалень, які були впроваджені в PostgreSQL для покращення та оптимізації швидкості роботи з базою даних.

4 Відновлення після збоїв і надійність. PostgreSQL прагне забезпечити найкращий ступінь надійності даних. Дані повністю захищені завдяки інтелектуальним опціям реплікації. Крім того, завжди існує можливість створити резервну копію найважливіших даних.

5 Розширюваність. PostgreSQL не обмежує користувача певним типом документів у базі даних. База даних надає широкий спектр типів даних на вибір.

6 Postgres підтримує міжнародні набори символів для інтернаціоналізації та текстового пошуку. Він також підтримує повнотекстовий пошук для прискорення процесу пошуку і включає в себе нечутливі до регістру і наголосу зіставлення.

Базовий дизайн PostgreSQL є однією з найбільш фундаментальних відмінностей від більшості інших реляційних баз даних [27].

Реляційні системи керування базами даних (РСКБД / RDBMS) – найкращий спосіб описати більшість реляційних баз даних. СКБД – це програмне забезпечення, спеціально створене для управління реляційними базами даних, які зберігають дані у вигляді таблиць з визначеними стовпчиками і типами даних. Дані можна запитувати, змінювати та отримувати за допомогою методів реляційної алгебри, часто використовуючи мову структурованих запитів (SQL).

На відміну від SQL, PostgreSQL є об’єктно-реляційною системою керування базами даних (ОРСКБД / ORDBMS). Це означає, що вона має ті ж самі реляційні можливості, що й СКБД, але також має деякі об’єктно-орієнтовані характеристики.

На практиці це означає, що PostgreSQL дозволяє:

– створювати власні складні типи даних.

– перевантажувати функції для роботи з різними формами даних з параметрами.

– визначати зв’язки успадкування таблиць.

У якості мови програмування для розробки ПЗ було обрано C# – об’єктно-орієнтовану мову програмування, яка використовується для створення різноманітних програм і додатків. Хоча вона є досить адаптивною, найчастіше її використовують у трьох сферах [28–30]:

– розробка веб-додатків;

– розробка програм для Windows;

– розробка комп’ютерних ігор.

Переваги C#:

– коротший час розробки;

– низька крива навчання;

– висока масштабованість;

– кросплатформність;

– універсальність;

– підтримка методології об’єктно-орієнтованого програмування;

– наявність автоматичного функціоналу збору сміття для звільнення пам’яті, яка була виділена об’єктами, що більше не використовуються або недоступні;

– обробка виключень, яка дозволяє структурувати виявлення помилок та відновлення після їх виправлення;

– підтримка механізму використання лямбда-виразів;

– підтримка асинхронних операцій тощо.

Для побудови програмного додатку було використано середовище розробки Microsoft Visual Studio Community 2022. Перевагами цієї версії середовища розробки є:

– безкоштовність;

– підтримка мов програмування високого рівня, зокрема C#;

– підтримка технологій розробки WPF та .NET;

– підтримка технології IntelliSense для автозавершення вмісту рядків програми;

– підтримка внутрішньої структури програмного коду, що не впливає на зовнішню поведінку програми [31];

– наявність графічного дизайнера для проектів, створених за допомогою технологій WPF та .NET; можливість роботи з програмами, створеними за допомогою технологій WPF та .NET;

– наявність графічного дизайнера для проектів, створених за допомогою технологій WPF та .NET.

3.4. Опис структури програмного забезпечення

Програмний додаток має структуру декстопного багатовіконного програмного забезпечення, див. рис. 3.8.

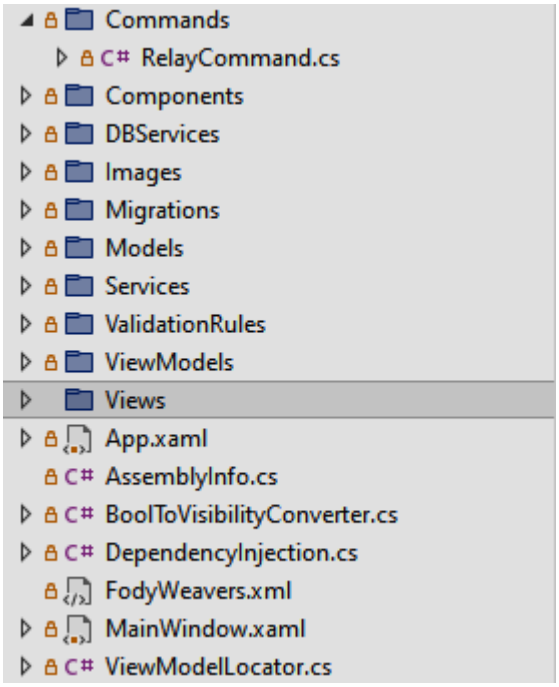


Рис. 3.8. Структура створеного ПЗ

Кожен з підрівнів, який представлений окремою текою, створений для покриття певної частини роботи програмного забезпечення:

– commands – містить клас RelayCommand, який відповідає за визначення основних властивостей команд для елементів графічного інтерфейсу;

– components – призначений для зберігання користувальницьких компонентів графічного інтерфейсу;

– dbServices – зберігає клас ApplicationContext, призначений для встановлення підключення до бази даних;

– images – зберігає графічні ресурси проекту;

– migrations – містить створені міграції бази даних;

– models – зберігає моделі системи;

– views – зберігає представлення системи;

– services – призначений для зберігання допоміжних сервісів програмного додатку;

– validationRules – визначає правила контролю вхідних даних;

– viewModels – зберігає елементи типу «представлення-модель» створеного програмного додатку.

Тека models містить моделі програмної системи, які відповідають таблицям створеної бази даних.

У папці views містяться наступні файли:

– adminMain.xaml – головне вікно програми для користувача-адміністратора, призначене для навігації системою;

– adminUserView.xaml – вікно, призначене для користувача-адміністратора для перегляду даних про користувачів програмної системи;

– basketList.xaml – вікно, призначене для перегляду списку продуктових кошиків, отриманих в результаті оптимізації та збережених користувачем;

– bloodSugarLevelDataEntry.xaml – вікно, яке відповідає за введення користувачем даних про заміряні рівні цукру у крові;

– bloodSugarLevelPlot.xaml – вікно, яке відображає введені користувачем дані про рівні цукру у крові протягом обраного проміжку часу;

– calculateDataInput.xaml – вікно, на якому розміщені елементи користувальницького графічного інтерфейсу для підтвердження користувачем даних, які необхідні для розрахунку раціону харчування;

– calculationPreparations.xaml – вікно, призначене для вибору користувачем продуктів харчування для наповнення продуктового кошику, введення максимальної суми витрат та періоду, на який проводяться обчислення;

– goodAddAdmin.xaml – вікно, призначене для додавання користувачем-адміністратором даних про нові продукти харчування із можливістю вказання їхнього типу «стандартний»;

– goodCustomUserView.xaml – вікно для відображення списку користувальницьких продуктів харчування (були додані безпосередньо користувачем через відповідне вікно);

– goodGuestView.xaml – вікно перегляду списку та властивостей продуктів харчування для неавторизованого в системі користувача;

– goodStandartUserView.xaml – вікно для відображення списку стандартних продуктів харчування (мають тип «стандартий»);

– goodsView.xaml – вікно для перегляду списку продуктів харчування із можливістю додати власний продукт;

– goodUserAdd.xaml – вікно, призначене для додавання користувачем продукту харчування із власноруч визначеними параметрами;

– goodUserEdit.xaml – вікно, призначене для редагування користувачем даних про власний користувальницький продукт харчування;

– goodViewAdmin.xaml – вікно для перегляду списку продуктів користувачем-адміністратором із можливістю додавання нового товару;

– guestGoodsBasket.xaml – вікно відображення результатів, на яке потрапляє неавторизований користувач після того, як виконає розрахунок раціону;

– guestMain.xaml – головне вікно для неавторизованого користувача в системі для виконання навігації системою;

– login.xaml – вікно для авторизації користувачів шляхом введення логіну та паролю із можливістю перейти до вікна реєстрації;

– promptToLogin.xaml – вікно, призначене для надання користувачеві можливості авторизуватися у програмній системі для отримання повного її функціоналу або продовжити як неавторизований користувач (гість) із наданням обмеженого функціоналу системи;

– registration.xaml – вікно для реєстрації нового користувача програмної системи та початкового збору даних про користувача;

– restrictionAddAdmin.xaml – вікно, призначене для користувача-адміністратора для додавання обмежень на споживання продуктів харчування із можливістю вказати тип обмеження «стандартне»;

– restrictionCustomUserView.xaml – вікно для перегляду користувальницьких обмежень на споживання продуктів харчування;

– restrictionEditAdmin.xaml – вікно для редагування обмежень на споживання продуктів харчування користувачем-адміністратором;

– restrictionGuestView.xaml – вікно для перегляду наявних стандартних обмежень на споживання продуктів харчування неавторизованим користувачем;

– restrictionStandartUserView.xaml – вікно для перегляду авторизованим користувачем стандартних обмежень на споживання харчових продуктів;

– restrictionUserAdd.xaml – вікно для додавання користувачем власних обмежень на споживання певних продуктів;

– restrictionUserEditAdd.xaml – вікно для редагування користувачем власних обмежень на споживання певних продуктів;

– restrictionViewAdmin.xaml – вікно для перегляду обмежень на споживання харчових продуктів для користувача-адміністратора із можливістю додавання нових обмежень;

– userEditUser.xaml – вікно, призначене для редагування персональних даних авторизованим користувачем;

– userGoodBasket.xaml – вікно надання результатів розрахунку раціону харчування для авторизованого користувача із можливістю зберегти результати;

– userMain.xaml – головне вікно програми для авторизованого користувача, призначене для навігації системою;

– userProfile.xaml – вікно особистого кабінету авторизованого користувача із можливістю внесення змін до персональних даних.

У теці viewmodels зберігаються елементи, які забезпечують взаємодію між вказаними представленнями користувальницького графічного інтерфейсу та створеними моделями.

3.5. Опис роботи розробленого програмного забезпечення

Створення облікового запису та авторизація

Після запуску програми перед користувачем відкривається вікно, зображене на рис. 3.9, яке містить пропозицію авторизуватися в системі чи продовжити її використання у якості гостя.

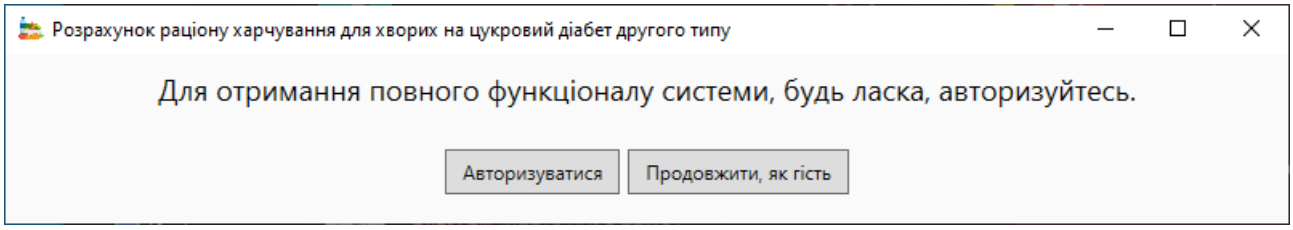


Рис. 3.9. Запит на вибір виду подальшої роботи із застосунком

Якщо користувач вибере опцію «Продовжити як гість», він отримає доступ до обмежених функцій системи, таких як перегляд стандартних продуктів, встановлення дієтичних обмежень та розрахунок свого раціону.

Якщо користувач натисне кнопку «Увійти», його буде перенаправлено на сторінку, зображену на рис. 3.10.

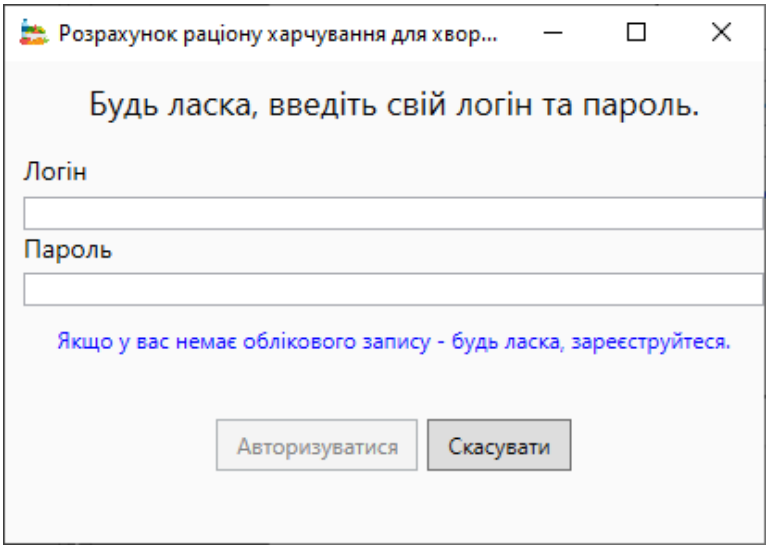


Рис. 3.10. Форма авторизації

Для доступу до системи користувачеві необхідно ввести своє ім’я користувача та пароль. Якщо програма запускається вперше і користувач не має облікового запису, але бажає користуватися повною функціональністю програмного забезпечення, він може зареєструвати обліковий запис, натиснувши на відповідне посилання. У цьому випадку відкриється сторінка, зображена на рисунку 3.11.

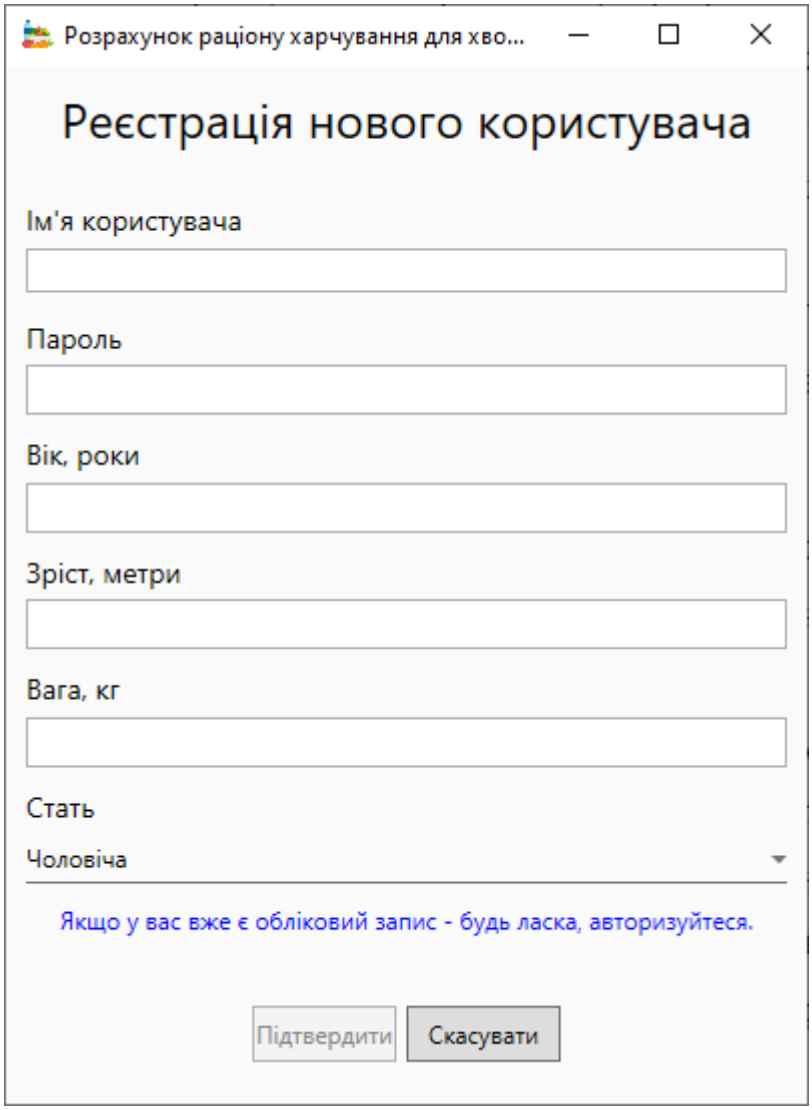


Рис. 3.11. Форма реєстрації

Користувач повинен заповнити реєстраційну форму з усією необхідною інформацією. Крім того, ці дані повинні відповідати визначеному формату:

– ім’я користувача має бути написане латинськими літерами і містити принаймні одну велику літеру;

– пароль має містити принаймні одну цифру, спеціальний символ і велику літеру;

– вік користувача має бути числовим значенням, що не перевищує 200 років і не дорівнює нулю;

– зріст користувача може бути цілим чи реальним значенням, що не перевищує 3 метри;

– вага користувача має бути цілим чи реальним значенням, що не перевищує 1000 кілограмів.

Якщо дані, що ввів користувач не відповідають вхідним критеріям, він отримає відповідне повідомлення, як показано на рис.3.12.

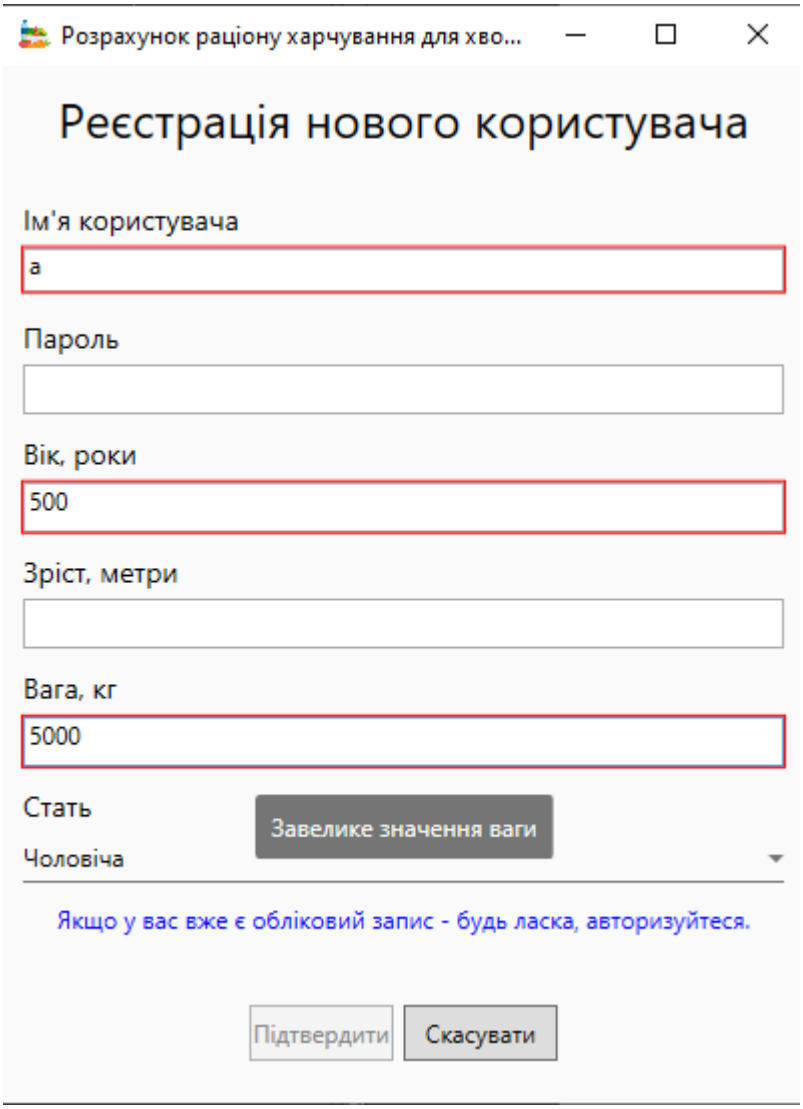


Рис. 3.12. Помилка при реєстрації

Якщо користувач ввів правильні реєстраційні дані, але введене ім’я користувача вже є в системі, користувач отримає повідомлення, показане на рис. 3.13.

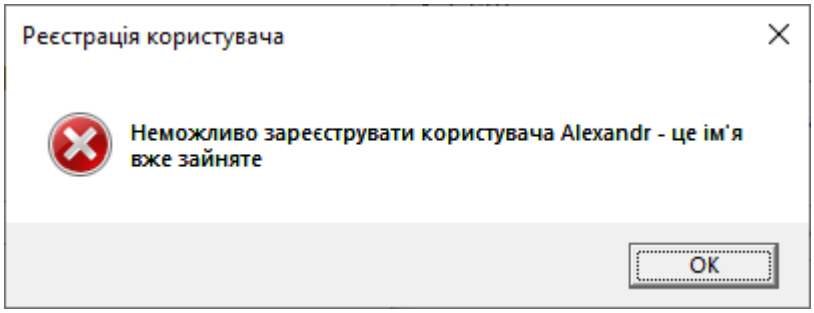


Рис. 3.13. Помилка реєстрації

Якщо під час реєстрації користувач натискає кнопку «Скасувати», з’являється вікно підтвердження, зображене на рис. 3.14.

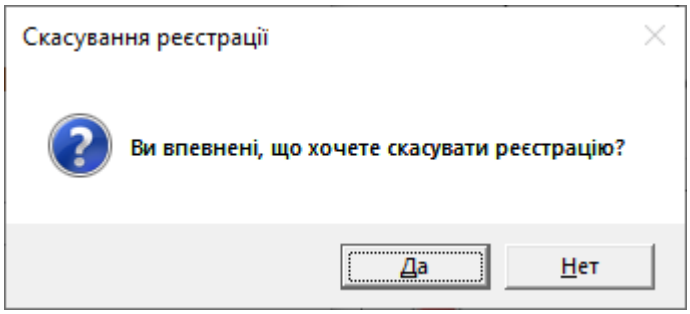


Рис. 3.14. Скасування реєстрації

Якщо користувач підтвердить свій намір, він буде повернутий на сторінку авторизації.

Коли користувач успішно зареєструє свій обліковий запис, він отримає повідомлення, показане на рис. 3.15, і буде перенаправлений на сторінку авторизації, де йому потрібно буде ввести своє ім’я та пароль.

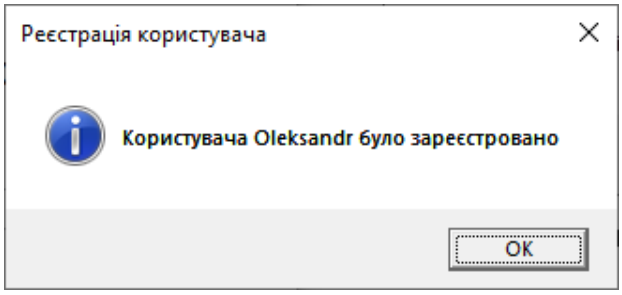


Рис. 3.15. Успішна реєстрація

Якщо користувач введе неправильні дані під час авторизації, з’явиться повідомлення, показане на рис. 3.16, і процес авторизації буде перервано.

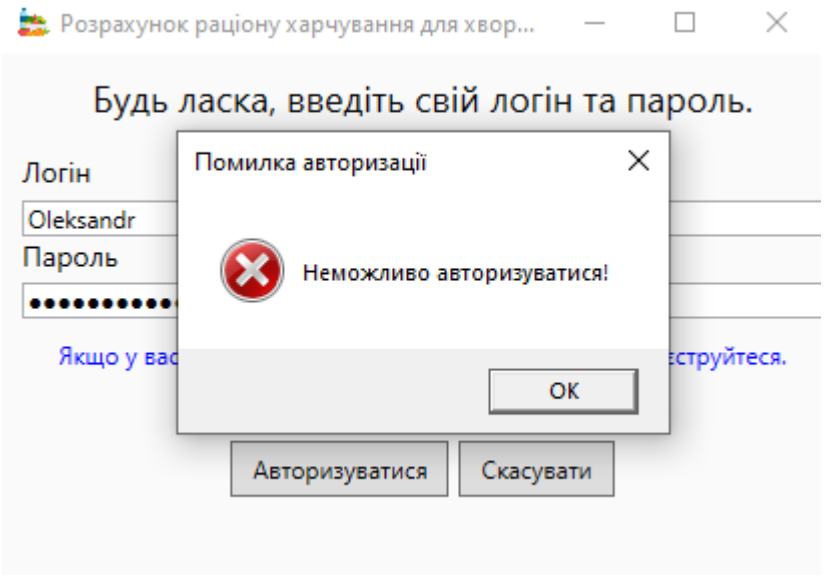


Рис. 3.16. Помилка авторизації

Після успішної авторизації користувач потрапляє на головну сторінку додатку, як показано на рис. 3.17, з якої він може взаємодіяти з функціоналом додатку, наприклад, переглядати та створювати продукти харчування, обмеження, продуктові кошики, переглядати та змінювати особисті дані, виходити з облікового запису або закривати додаток.

Рис. 3.17. Головна сторінка

Особистий кабінет

Щоб ознайомитися з персональними даними користувача, перейдіть до пункту «Обліковий запис» на головній сторінці програми, а потім до пункту "Профіль", див. рис. 3.18.

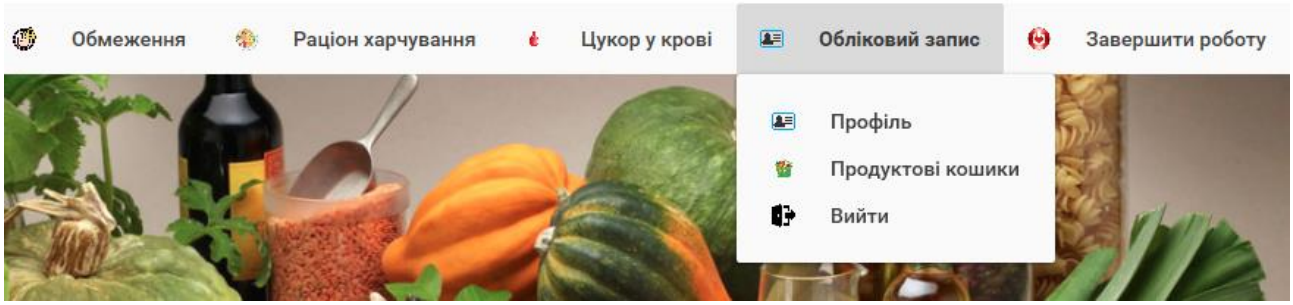


Рис. 3.18. Розташування особистого кабінету

Після цього користувач побачить сторінку зі своїми даними, як показано на рис. 3.19.

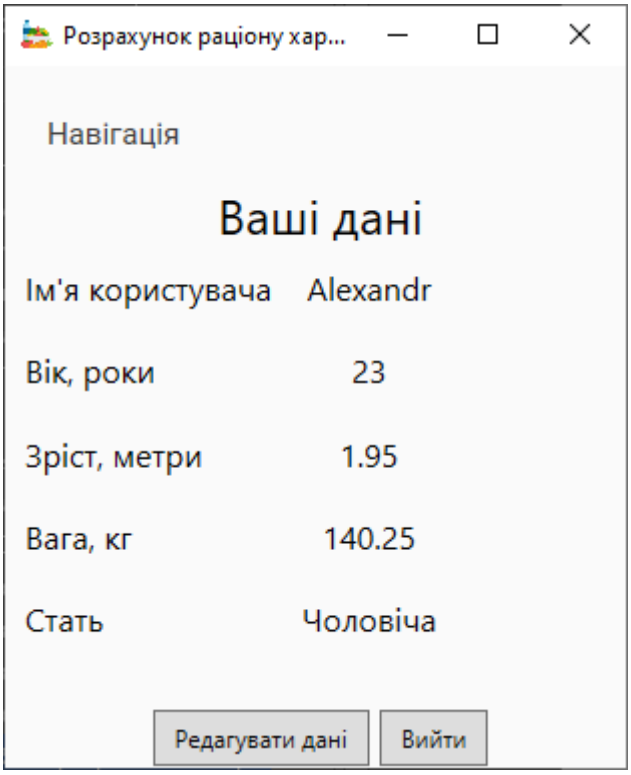


Рис. 3.19. Особистий кабінет

Якщо користувач хоче змінити свою особисту інформацію, він може отримати доступ до форми редагування (рис. 3.20), натиснувши на кнопку «Редагувати дані».

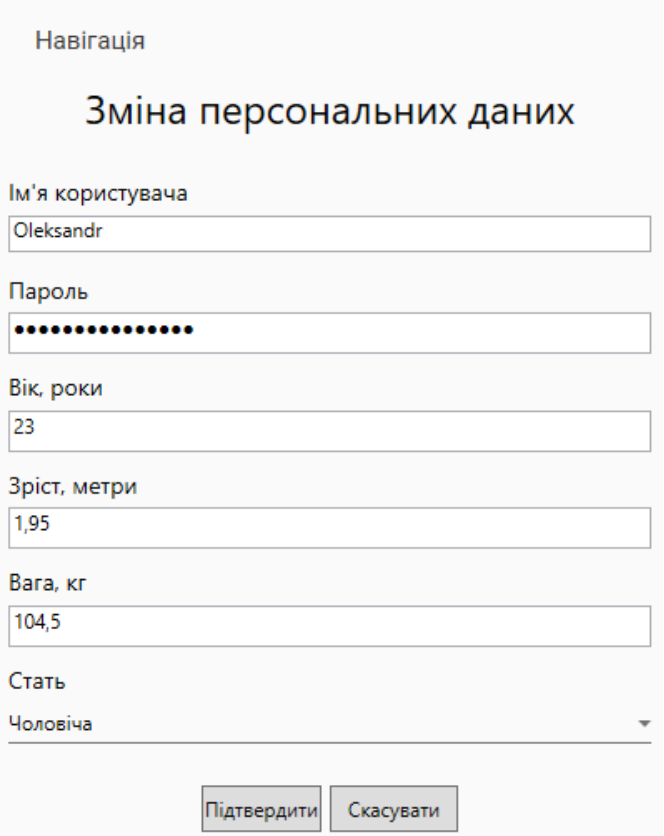


Рис. 3.20. Зміна особистих даних

Поля для введення інформації співставні з полями у формі реєстрації облікового запису і мають ті ж самі стандарти щодо змісту та формату інформації.

Коли користувач скасовує процес оновлення даних, його або її просять підтвердити дію. Якщо редагування буде успішним, користувач отримає відповідне повідомлення.

Розрахунок раціону

Щоб оптимізувати продуктовий кошик, зайдіть у головне меню і виберіть пункт «Раціон харчування», а потім підпункт «Оптимізувати раціон харчування», див. рис. 3.21.

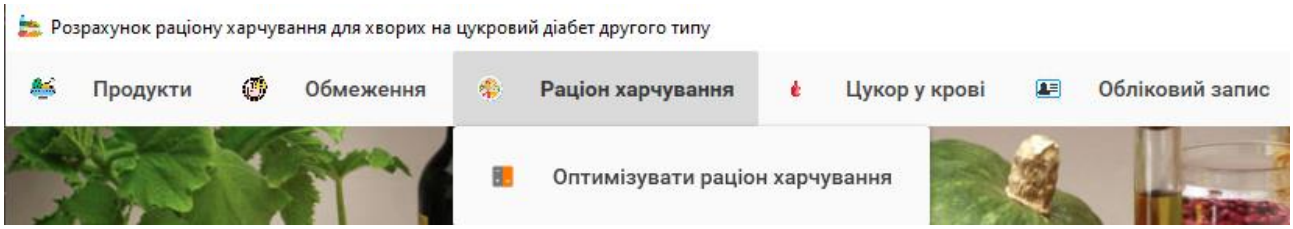


Рис. 3.21. Розташування розрахунку раціону

На сторінці, що з’явилася, користувач має підтвердити особисту інформацію та визначити кількість фізичної активності, див. рис. 3.22.

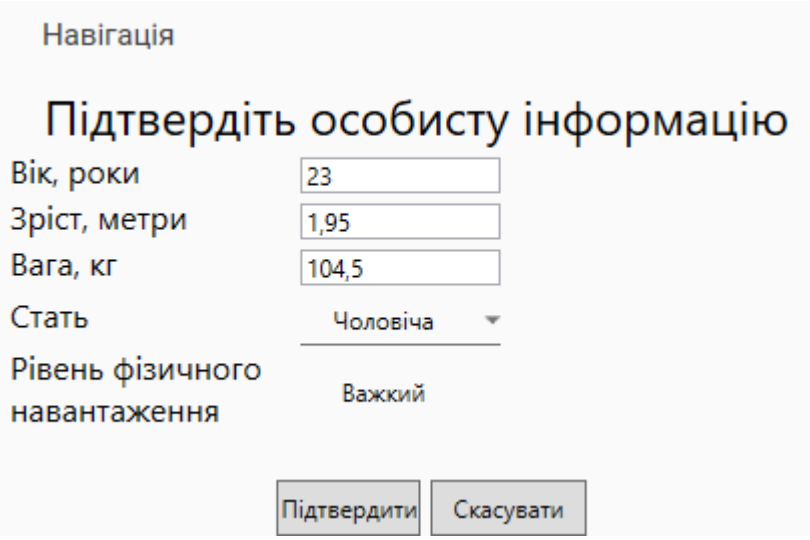


Рис. 3.22. Підготовка до розрахунків

На наступному екрані, як показано на рис. 3.23, користувач має обрати перелік продуктів, які бажає внести в свій раціон, а також максимальну ціну цього продуктового кошика та період часу, за який будуть виконані розрахунки.

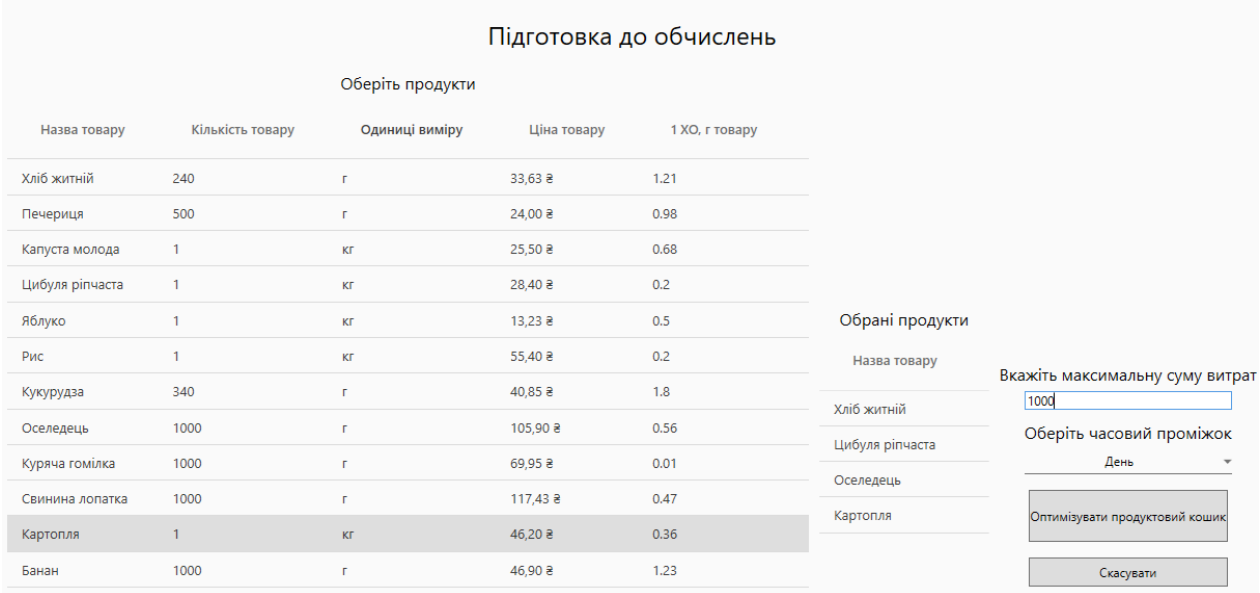


Рис. 3.23. Підготовка до обчислень

Після завершення розрахунків користувачеві буде представлено список позицій, які найбільше відповідають вказаним критеріям (рис. 3.24), включаючи кількість товарів, загальну кількість хлібних одиниць та обмеження.

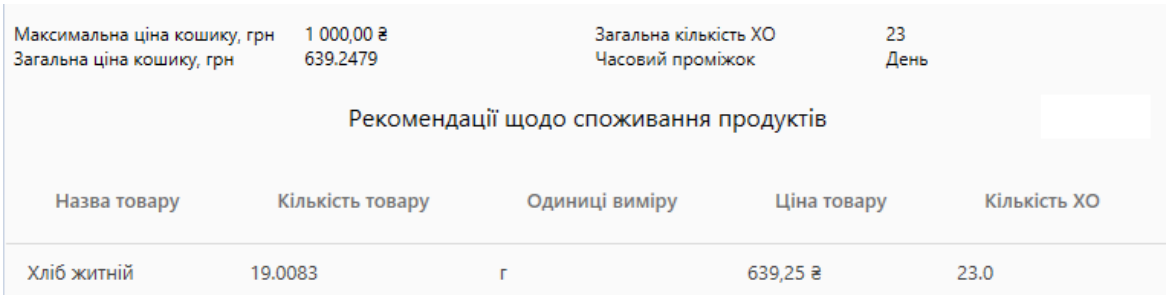


Рис. 3.24. Результат розрахунків

За бажанням користувач може зберегти результати розрахунків. Для цього необхідно перейти на форму перегляду результатів розрахунків і вибрати пункт меню «Кошик», а потім підпункт «Зберегти». Після цього, як показано на рис. 3.25, з’явиться список доступних результатів.

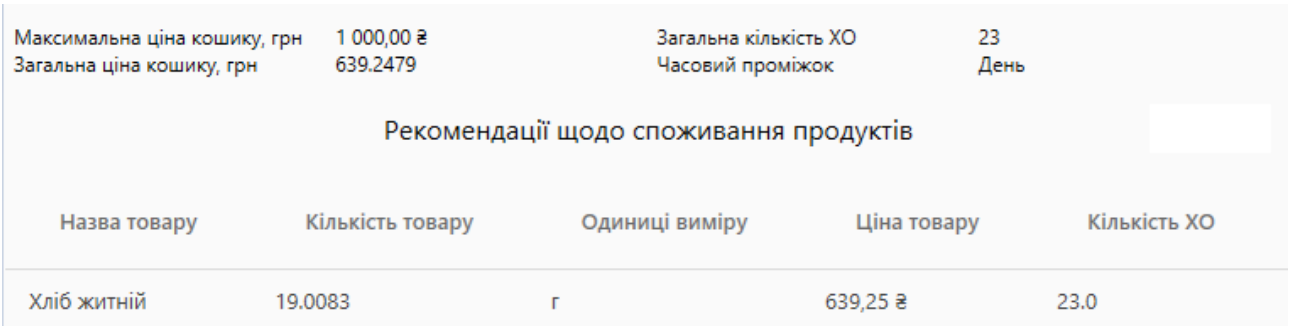


Рис. 3.25. Збережені результати

Список продуктів

Для того, щоб переглянути список продуктів харчування, користувач повинен спочатку вибрати пункт меню «Продукти» на головній формі (рис. 3.26), а потім вибрати, чи потрібно переглядати «Стандартні» або «Користувацькі» продукти.

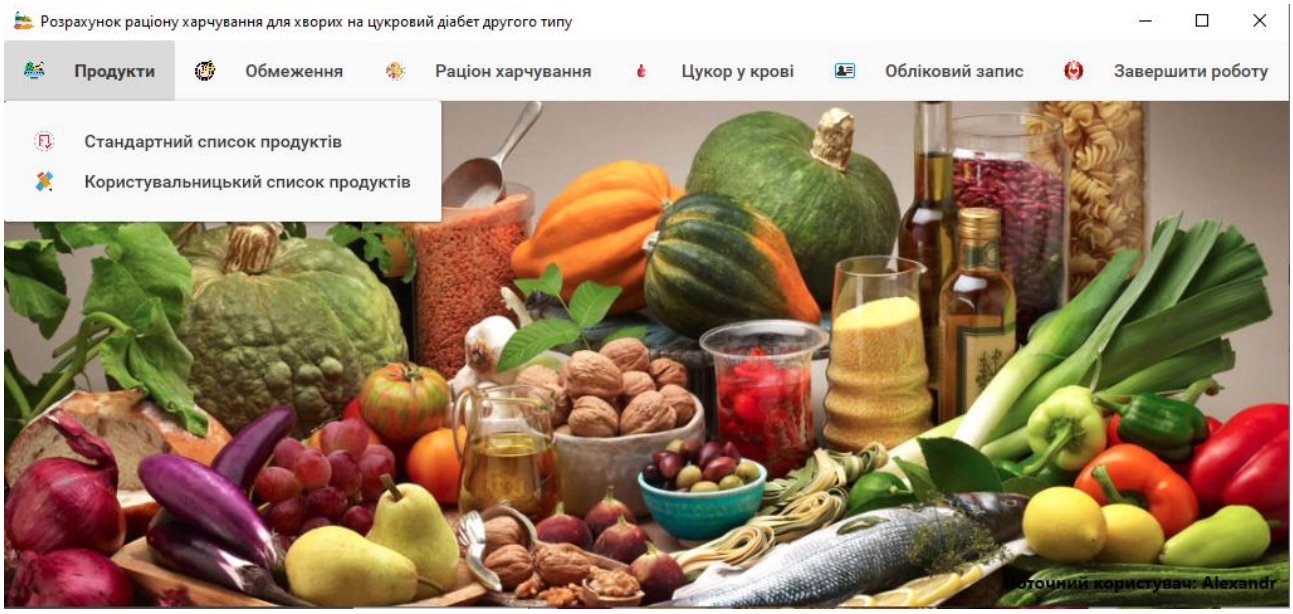
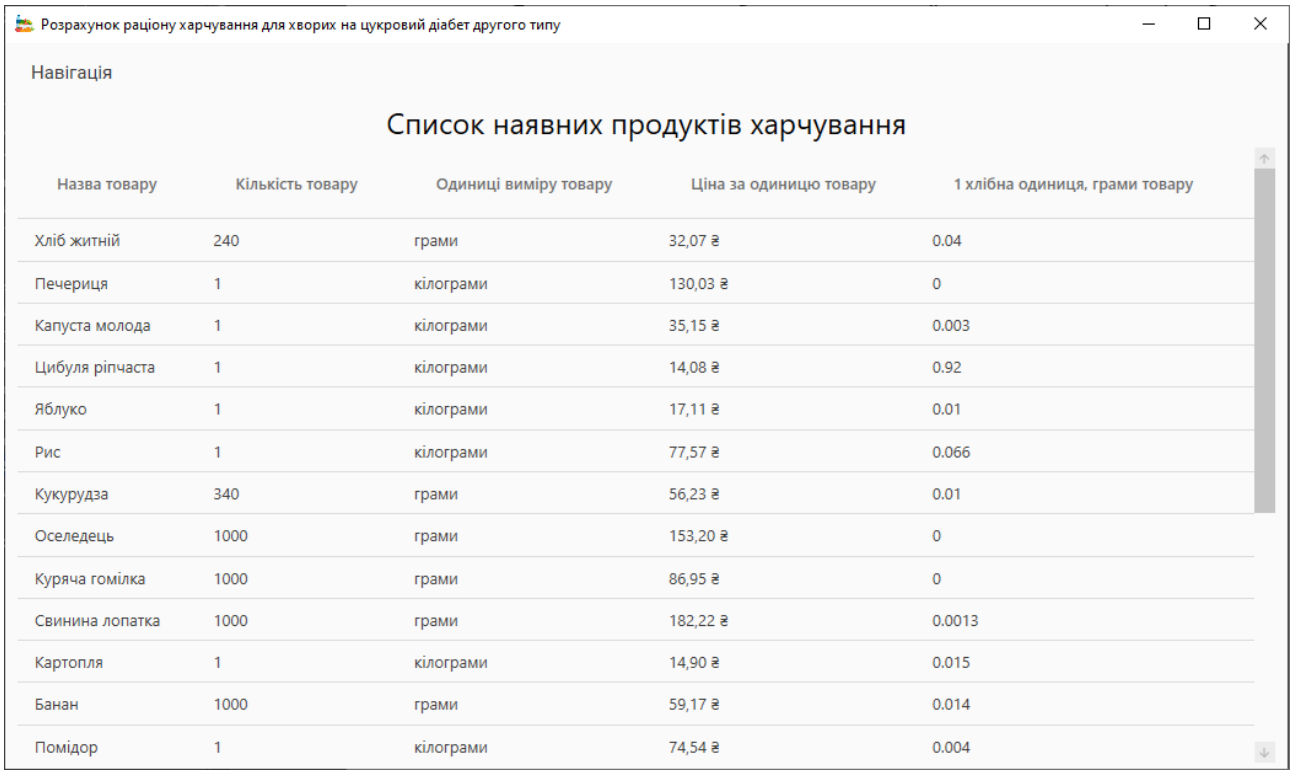


Рис. 3.26. Розташування продуктів

Якщо користувач обирає стандартний тип продуктів, він буде перенаправлений на сторінку, показану на рис. 3.27, яка пропонує список всіх можливих стандартних продуктів. Цей список не може бути змінений користувачем у будь-який спосіб.



Назва товару	Кількість товару	Одиниці виміру товару	Ціна за одиницю товару	1 хлібна одиниця, грами товару
Хліб житній	240	грами	32,07 ₴	0.04
Печериця	1	кілограми	130,03 ₴	0
Капуста молода	1	кілограми	35,15 ₴	0.003
Цибуля ріпчаста	1	кілограми	14,08 ₴	0.92
Яблуко	1	кілограми	17,11 ₴	0.01
Рис	1	кілограми	77,57 ₴	0.066
Кукурудза	340	грами	56,23 ₴	0.01
Оселедець	1000	грами	153,20 ₴	0
Куряча гомілка	1000	грами	86,95 ₴	0
Свинина лопатка	1000	грами	182,22 ₴	0.0013
Картопля	1	кілограми	14,90 ₴	0.015
Банан	1000	грами	59,17 ₴	0.014
Помідор	1	кілограми	74,54 ₴	0.004

Рис. 3.27. Стандартні продукти

Користувач може повернутися на головну сторінку, обравши пункт меню «Навігація».

Якщо користувач вирішить переглянути власний тип товару, він буде перенаправлений на екран, зображений на рис. 3.28.

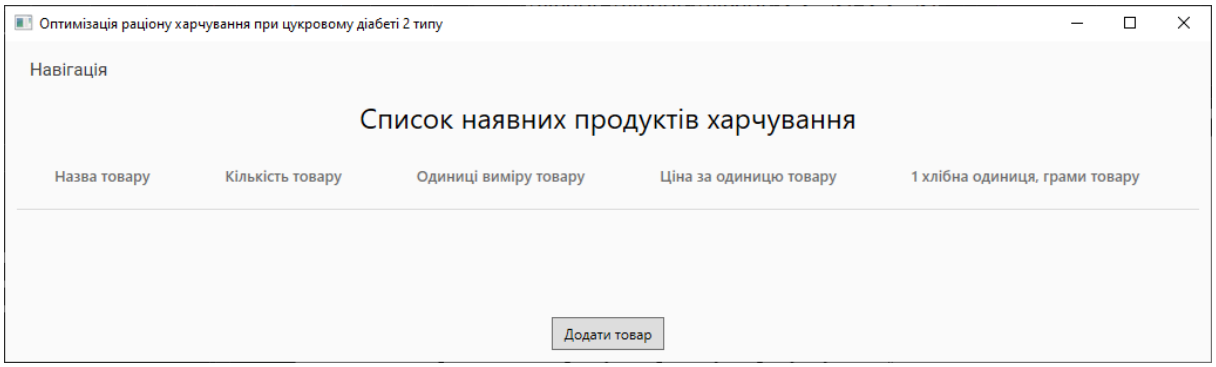


Рис. 3.28. Користувальницькі продукти

Натиснувши на кнопку «Додати продукт», користувач може повернутися на головну сторінку додатку або перейти на сторінку введення даних про новий харчовий продукт, див. рис. 3.29.

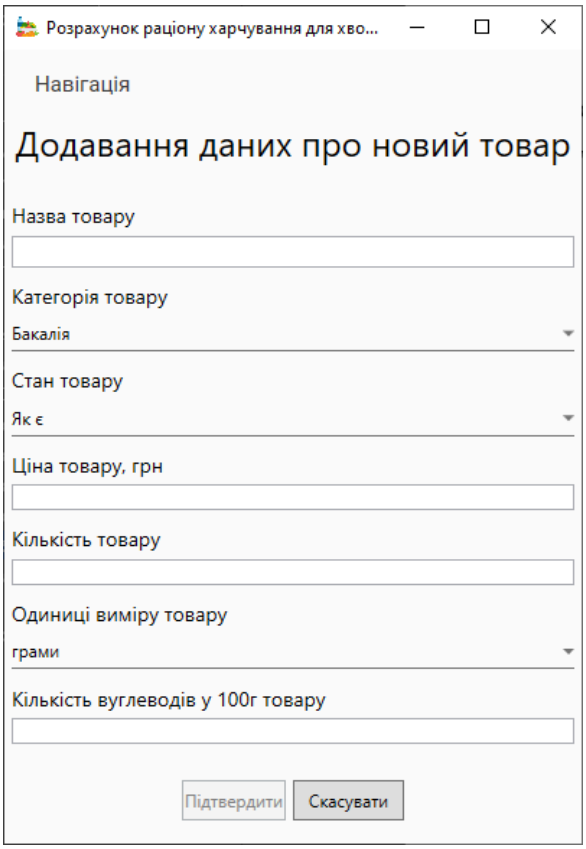


Рис. 3.29. Створення продукту

Дані, що вводяться користувачем, повинні відповідати наступним критеріям:

– назва продукту не може бути менше двох символів;

– ціна за одиницю не може дорівнювати нулю і повинна бути цілим або дійсним числом;

– кількість продукту не може дорівнювати нулю і повинна бути цілим або дійсним числом;

– кількість вуглеводів у 100 г продукту може бути цілим або дійсним числом.

Якщо користувач введе неточні дані, він отримає відповідне попередження. Після успішного додавання продукту з’явиться повідомлення з підтвердженням, як показано на рис. 3.30.

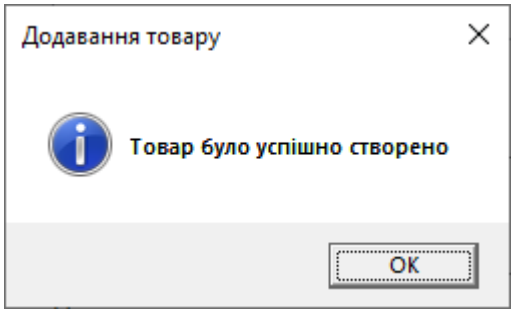


Рис. 3.30. Товар додано

Після додавання всіх необхідних позицій користувач може повернутися до форми для перегляду списку товарів, див. рис. 3.31.

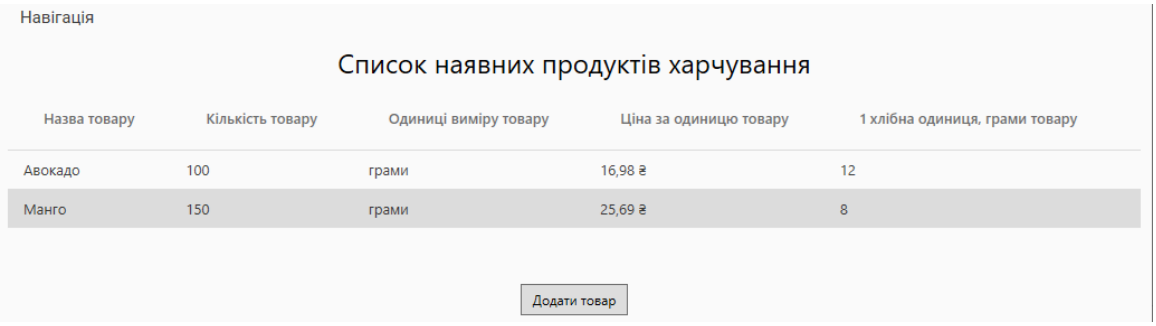


Рис. 3.31. Власні товари

Користувач може видалити товар зі списку товарів за допомогою контекстного меню. Як показано на рис. 3.32, у цьому сценарії з’явиться повідомлення з проханням підтвердити наміри.

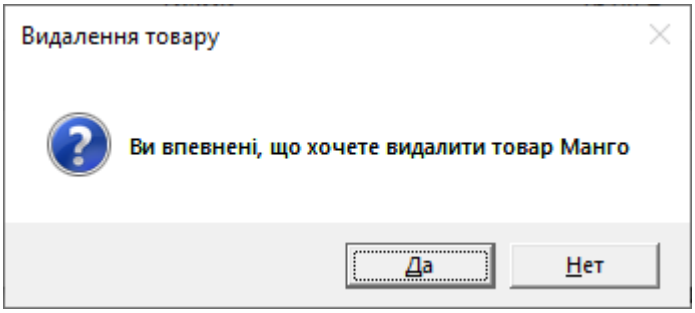


Рис. 3.32. Процес видалення товару

Щоб ознайомитися з обмеженнями споживання, перейдіть до пункту меню "Обмеження" на головній сторінці програми та оберіть підпункт "Стандартні обмеження". Ви потрапите на сторінку зі стандартними обмеженнями на продукти харчування, як показано на рис. 3.33.

Навігація

Список наявних обмежень для продуктів харчування

Назва товару	Назва обмеження	Тип обмеження	Кількість	Одиниці виміру
Хліб житній	Стандартне обмеження	Менше або дорівнює	250	г
Печериця	Стандартне обмеження	Менше або дорівнює	250	г
Капуста молода	Стандартне обмеження	Менше або дорівнює	250	г
Цибуля ріпчаста	Стандартне обмеження	Менше або дорівнює	250	г
Яблуко	Стандартне обмеження	Менше або дорівнює	250	г
Рис	Стандартне обмеження	Менше або дорівнює	250	г
Кукурудза	Стандартне обмеження	Менше або дорівнює	250	г
Молоко топлене	Обмеження 1	Менше ніж	1000	мл
Сир плавлений 55%	Стандартне обмеження	Менше або дорівнює	250	г
Сир твердий	Стандартне обмеження	Менше або дорівнює	250	г
Куряче філе	Стандартне обмеження	Менше або дорівнює	250	г
Свинина грудинка	Стандартне обмеження	Менше або дорівнює	250	г
Свинина лопатка	Стандартне обмеження	Менше або дорівнює	250	г

Рис. 3.33. Стандартні обмеження

Стандартні продукти

Ви повинні бути адміністратором, щоб додавати стандартні продукти харчування до програми.

Після авторизації в якості адміністратора користувач отримає доступ до головного вікна програми, показаного на рис. 3.34.

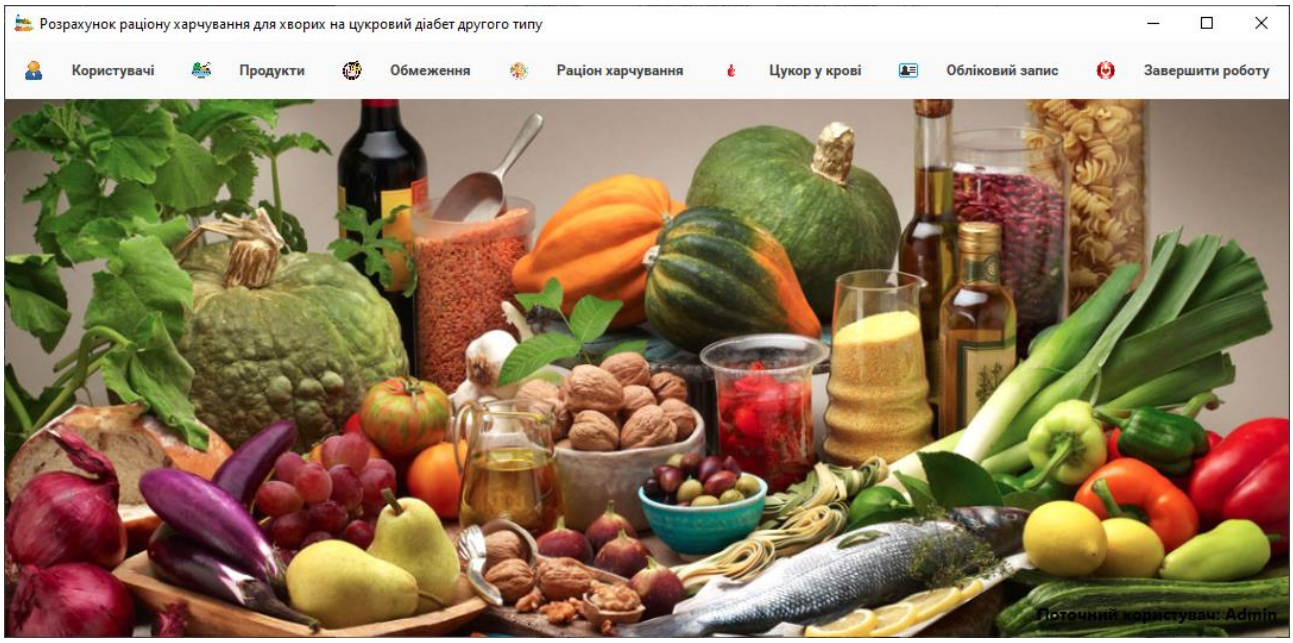


Рис. 3.34. Головне вікно

Щоб ввести інформацію про харчовий продукт, перейдіть до пункту головного меню «Продукти» і натисніть підпункт «Додати продукт». З’явиться сторінка, зображена на рис. 3.35.

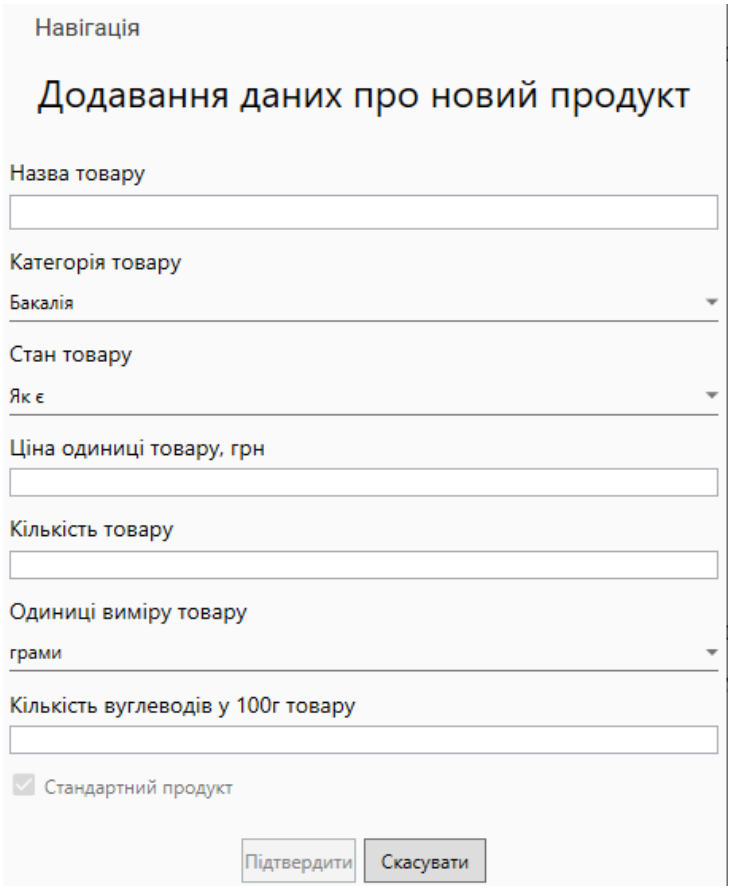


Рис. 3.35. Додавання стандартного продукту

Поля введення і формат даних ідентичні тим, що використовуються у формі для введення інформації про користувальницький товар.

Після того, як ви підтвердили введення інформації про товар, ви можете переглянути його у списку стандартних продовольчих товарів, показаному на рис. 3.36.

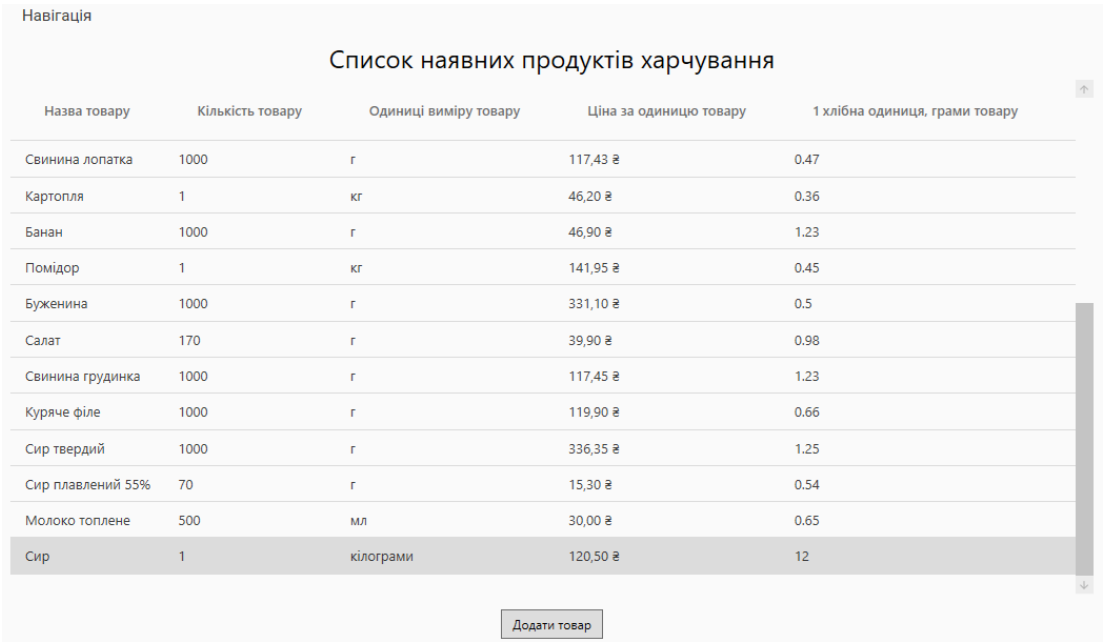


Рис. 3.36. Стандартні продукти

Користувачі

Щоб переглянути список існуючих користувачів системи, натисніть на пункт меню «Користувачі» в головному вікні і перейдіть на сторінку перегляду, показану на рис. 3.37, де відображається ім’я користувача, кількість його/її користувацьких продуктів, обмеження та продуктові кошики тощо.

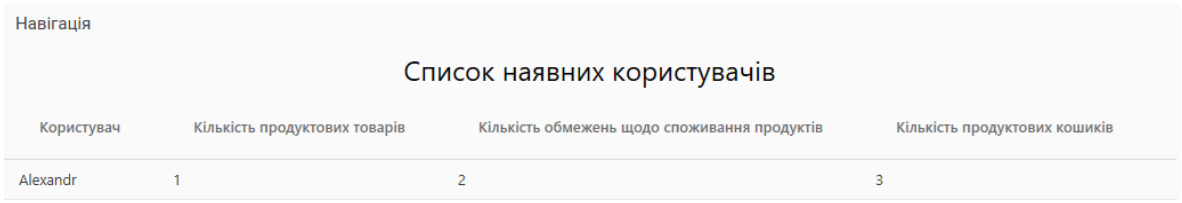


Рис. 3.37. Список користувачів

За необхідності адміністратор може видалити обліковий запис користувача, вибравши команду Видалити з контекстного меню для запису в списку. У повідомленні з підтвердженням наміру видалити обліковий запис користувача оберіть потрібний варіант, див. рис. 3.38.

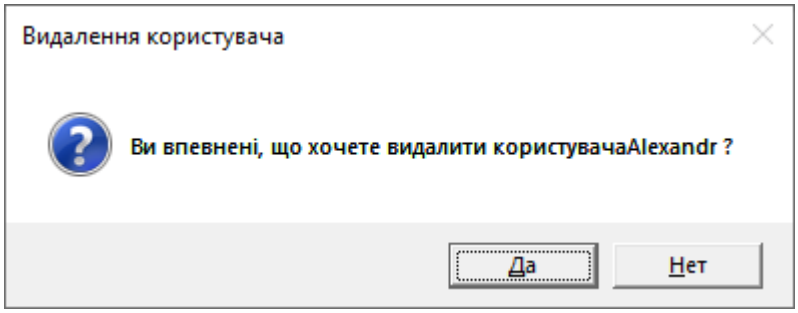


Рис. 3.38. Видалення користувача

Після успішного видалення облікового запису буде показано повідомлення, подібне до показаного на рис. 3.39.

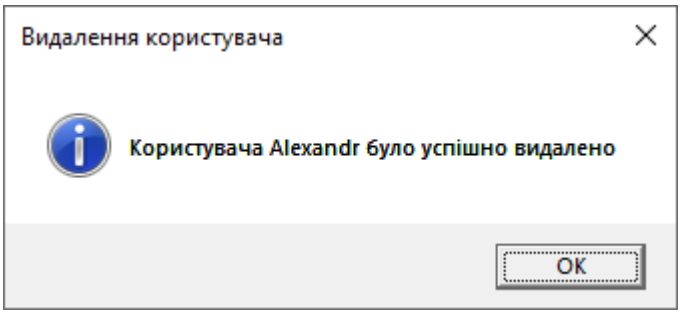


Рис. 3.39. Результат про видалення користувача

Контроль рівню цукру в крові

Щоб скористатися функцією моніторингу рівня цукру в крові, користувач повинен спочатку увійти в систему.

У головному вікні програми слід обрати пункт меню «Цукор в крові» та підпункт «Ввести показники рівня цукру в крові». Після цього користувач буде перенаправлений на сторінку, зображену на рис. 3.40.

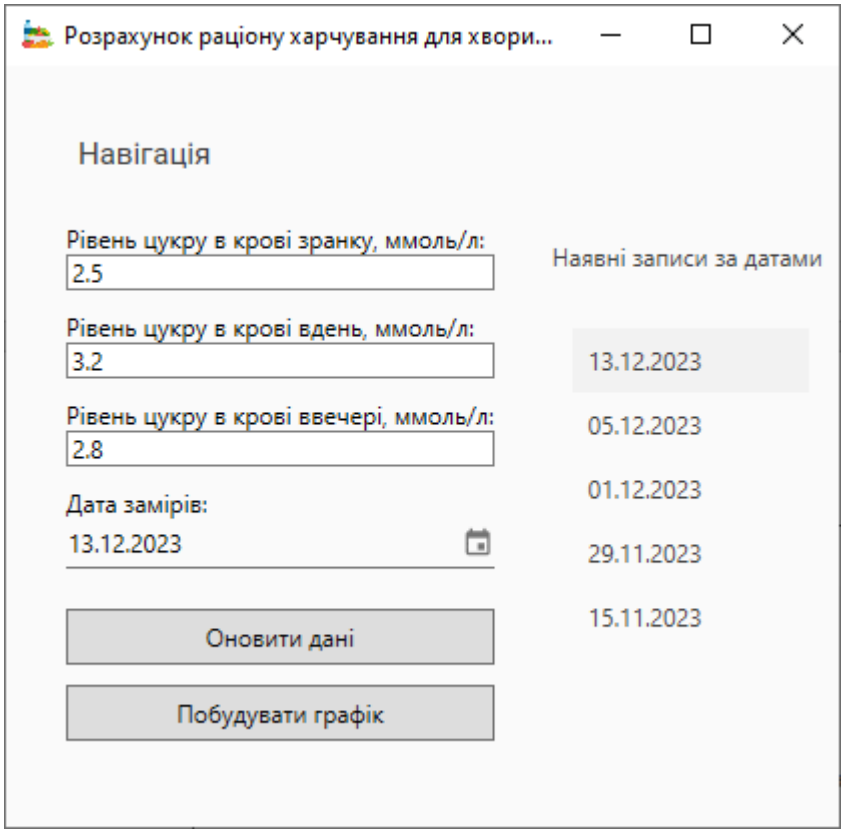


Рис. 3.40. Введення значень замірів рівня цукру

Інформація, що вводиться на цій сторінці, повинна відповідати певним критеріям:

– рівень цукру в крові повинен бути цілим або дійсним числом, більшим за нуль;

– дата вимірювання не може бути пізнішою за поточну дату запуску програми.

Якщо користувач введе невірні дані, відповідні частини графічного інтерфейсу будуть виділені червоним кольором, а кнопка "Додати дані" буде неактивною, доки користувач не введе коректні дані, як показано на рис. 3.41.

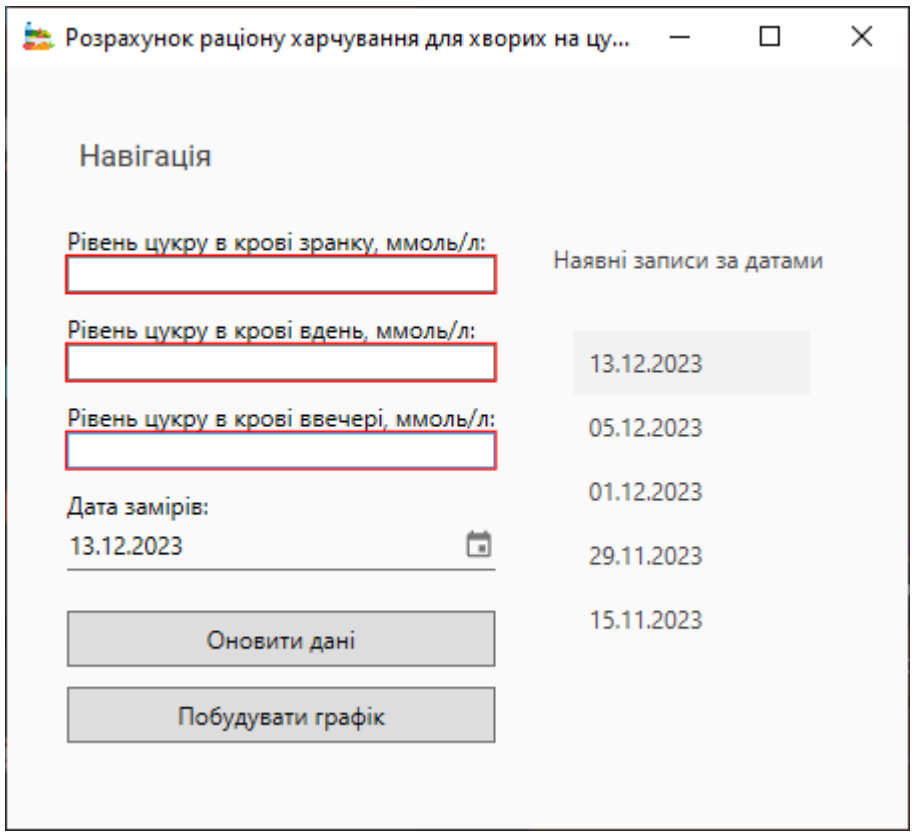


Рис. 3.41. Некоректні значення

Після додавання хоча б одного запису з вимірами, цей запис буде представлений у списку записів, і користувач зможе створювати графіки на основі наявних даних, як показано на рис. 3.42.

Рис. 3.42. Вікно візуалізації

Користувач повинен вибрати тип графіка у спливаючому вікні, що з’явиться:

Точкова діаграма використовується для створення графіка, що показує зміни наявних даних про рівень цукру в кюветі в часі, тоді як гістограма

використовується для порівняння введених чисел з існуючими нормами рівня цукру в крові.

Користувач повинен додатково вибрати дані для побудови графіка - всі проведені вимірювання або середнє значення - і період часу, за який буде показано графік.

Якщо користувач обрав для всіх даних тип графіка точковий, а рівень цукру в крові вимірювався протягом встановленого періоду часу, він отримає графік цих змін у часі з можливістю персоналізувати відображення значень, як показано на рис. 3.43 – 3.44.

Рис. 3.43. Вікно візуалізації з даними за день

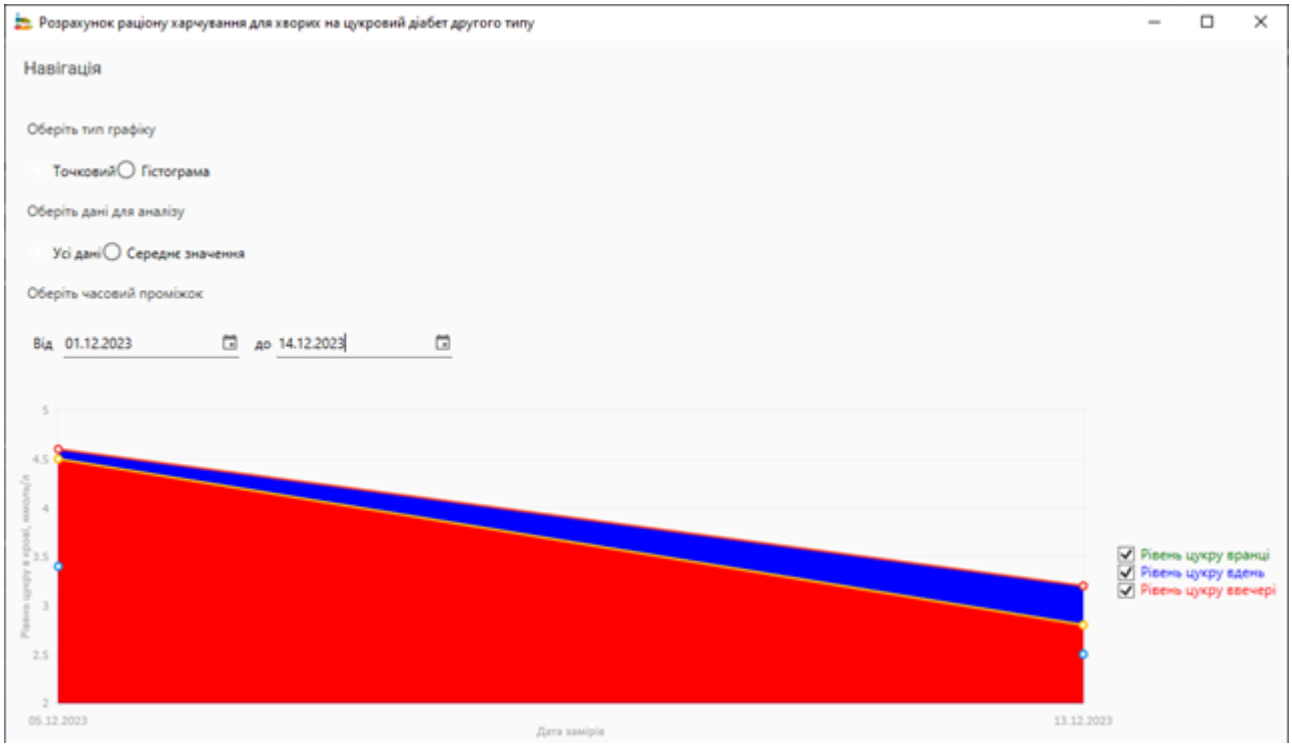


Рис. 3.44. Вікно візуалізації з даними за кілька днів

Користувач повинен вибрати тип графіка «гістограма», щоб побачити порівняння вхідних даних з поточними нормами цукру в крові. Якщо користувач вніс вимірювання за обраний період часу, буде відображено графік, подібний до рис. 3.45.

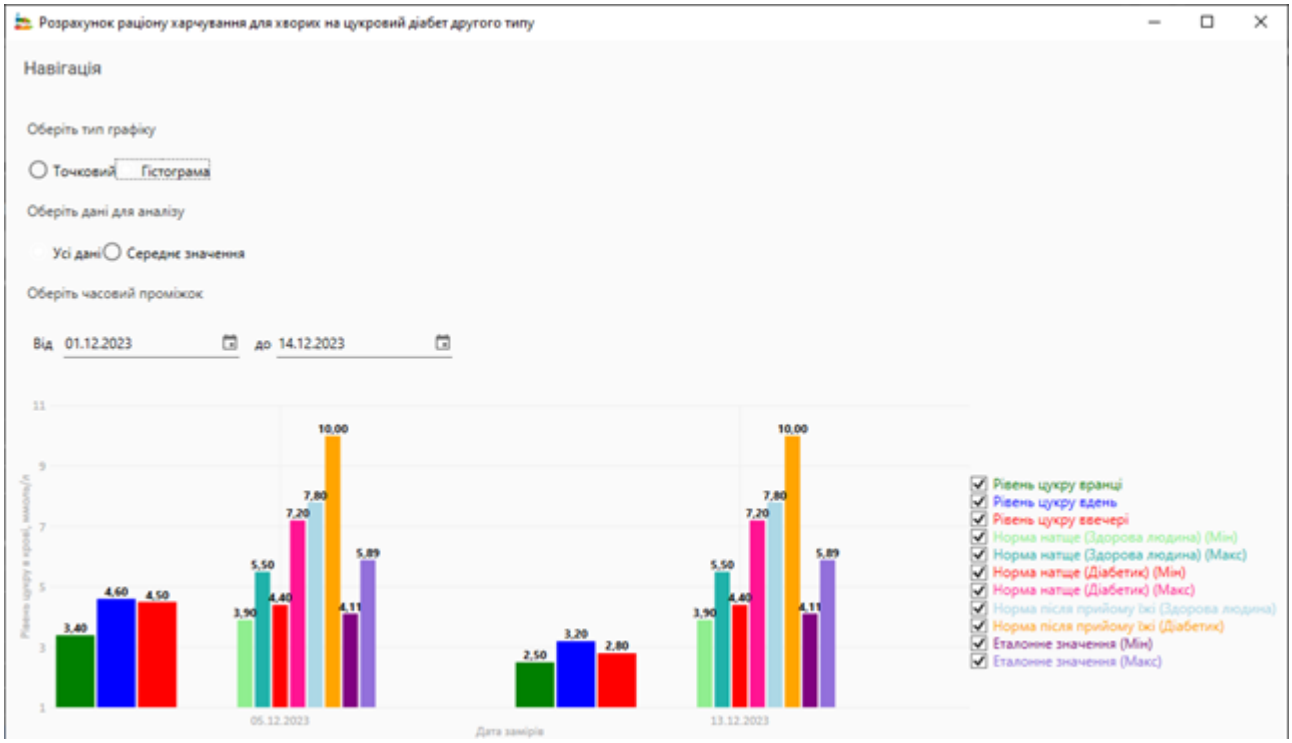
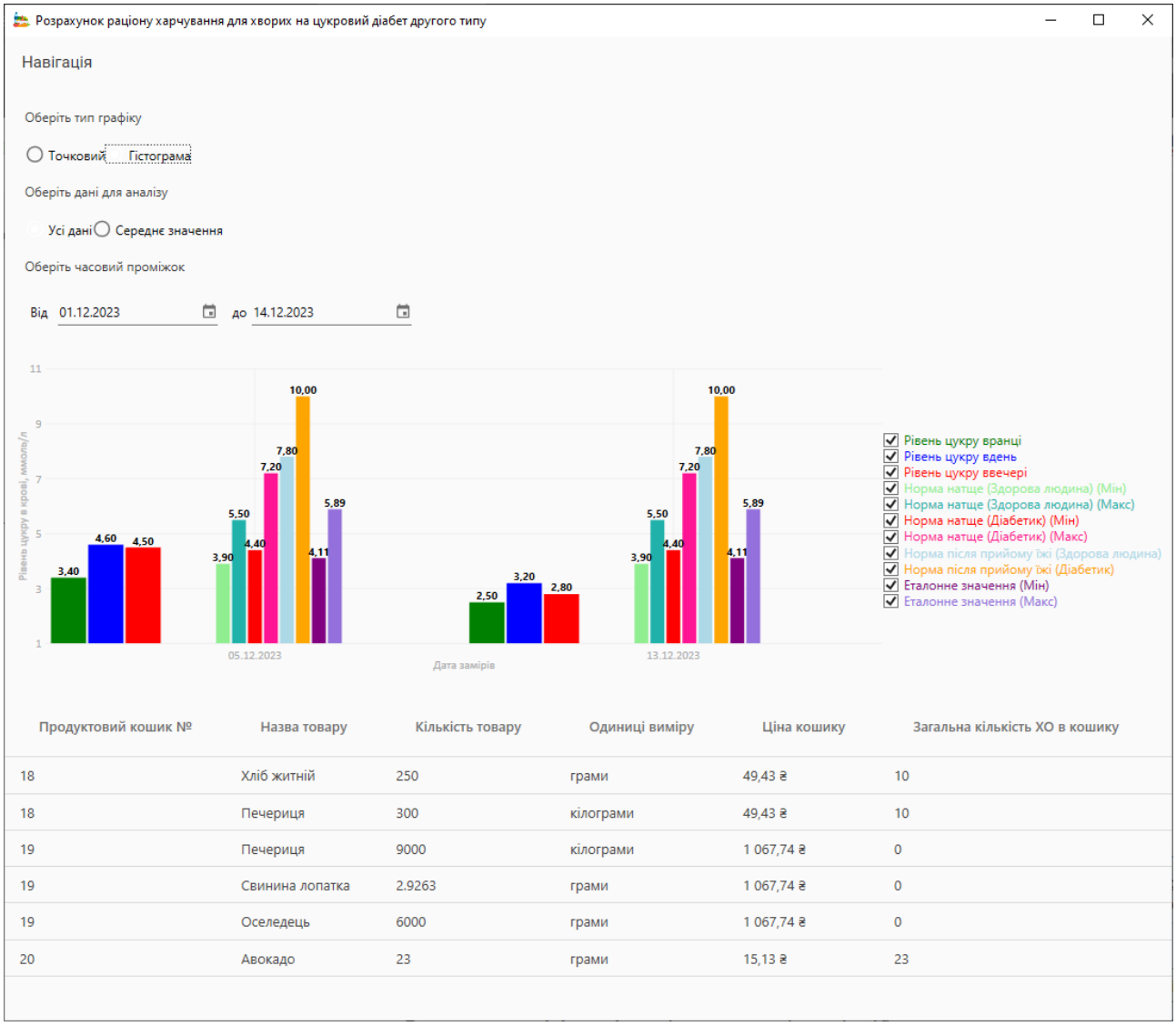


Рис. 3.45. Відображена гістограма

Якщо користувач натисне на графік своїх показників, і у нього є розрахунки харчування на певну дату, вони будуть подані під графіком, як показано на рис. 3.46.



Продуктовий кошик №	Назва товару	Кількість товару	Одиниці виміру	Ціна кошику	Загальна кількість ХО в кошику
18	Хліб житній	250	грами	49,43 ₴	10
18	Печериця	300	кілограми	49,43 ₴	10
19	Печериця	9000	кілограми	1 067,74 ₴	0
19	Свинина лопатка	2.9263	грами	1 067,74 ₴	0
19	Оселедець	6000	грами	1 067,74 ₴	0
20	Авокадо	23	грами	15,13 ₴	23

Рис. 3.45. Графік із результатами обчислень раціону

Вихід з облікового запису

За необхідності користувач може вийти з облікового запису, вибравши «Обліковий запис» – «Вийти» або «Обліковий запис» – «Профіль» – «Вийти». При спробі виходу з облікового запису користувач повинен підтвердити свої наміри у спосіб, зображений на рис. 3.46.

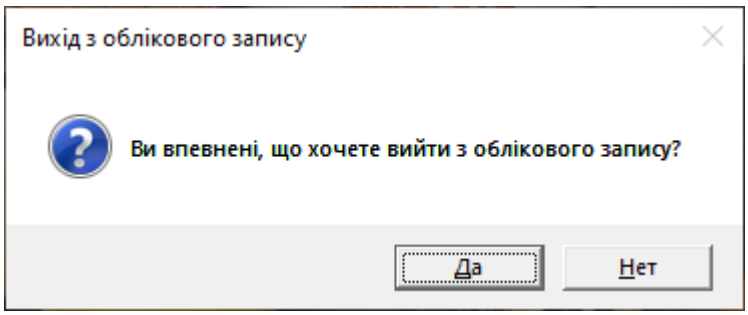


Рис. 3.46. Запит на підтвердження виходу з облікового запису

Після підтвердження користувачеві буде запропоновано форму запиту до авторизації.

Завершення роботи програми

Щоб вийти з програми, виберіть пункт меню "Завершити роботу" на головній формі, а потім підтвердіть своє бажання, як показано на рис. 3.47.

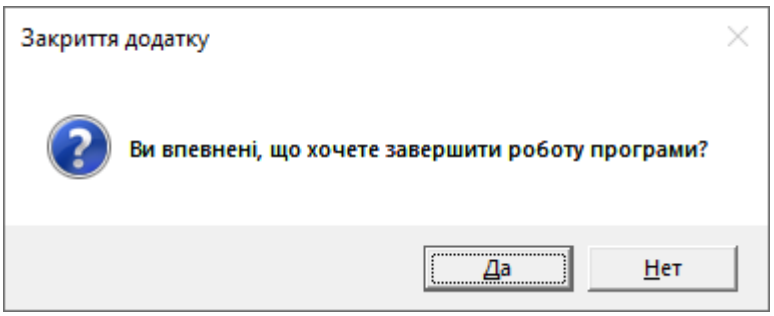


Рис. 2.41. Запит на завершення роботи програми

3.6. Висновки

У цьому розділі було розглянуто організацію побудованої бази даних проекту, яка складається з 13 таблиць і призначена для зберігання даних відповідно до предметної області.

Також було розглянуто шаблон проектування MVVM для створення архітектури програмного забезпечення, фреймворк Windows Presentation Foundation, механізм зв’язування даних, ядро Entity Framework Core для забезпечення зв’язку з базою даних та інструментарій LINQ для отримання значень з бази даних під час роботи з додатком.

Було описано програмне забезпечення для розробки, зокрема середовище розробки Microsoft Visual Studio 2022 Community, мову програмування C# та систему управління базами даних PostgreSQL.

Було розроблено програмне забезпечення для розрахунку раціону харчування осіб, хворих на цукровий діабет 2 типу.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено програмне забезпечення для розрахунку раціону харчування хворих на цукровий діабет другого типу.

Було проведено дослідження предметної області та рекомендацій щодо щоденної кількості споживання вуглеводів. За результатами досліджень було побудовано та реалізовано обчислення математичної моделі з використанням засобів мови C#.

Інтерфейс користувача програмного додатку створено з використанням технології Windows Presentation Foundation. Для підключення до бази даних PostgreSQL використано технології Entity Framework Core.

Створений програмний продукт має зручний традиційний інтерфейс користувача, який дозволяє користувачеві налаштовувати розміри програмних вікон.

Програма містить механізм реєстрації та авторизації, створений з дотриманням сучасних заходів безпеки для збереження результатів обчислень та персоналізації облікового запису користувача.

За результатами аналізу ефективності розробленого програмного забезпечення були зроблені наступні висновки:

– програмне забезпечення є ефективним інструментом для розрахунку раціону харчування з урахуванням харчової цінності та кількості вуглеводів, необхідних для задоволення щоденних потреб користувача, виходячи з його індивідуальних особливостей та рівня фізичної активності відповідно до медичних рекомендацій;

– правильна конфігурація обмежень дозволяє змоделювати продуктовий кошик, який відповідає потребам добового споживання вуглеводів і при цьому має мінімально можливу вартість для заданого набору продуктів;

– створене програмне забезпечення дозволяє користувачеві контролювати рівень глюкози в крові та порівнювати його з медичними нормами, а також з переліком розрахованих продуктових кошиків;

– оскільки створена програма призначена для оптимізації раціону харчування на основі добового споживання вуглеводів, надані результати можуть містити деякі неточності. Для підвищення точності розрахунків рекомендується додати до переліку даних для розрахунків вітаміни, харчові волокна та інші поживні речовини.

– оскільки метою програмного продукту є мінімізація загальної вартості набору товарів, обраних користувачем, в результаті розрахунків не всі продукти потрапляють до отриманого кошика, що зменшує його різноманітність.

Напрямком подальшого розвитку програми оптимізації набору продуктів є додавання у розрахунки більше факторів, що визначають здорове різноманіття продуктів харчування.

За результатами виконаної роботи було розроблено доповідь для міжнародної конференції [32].

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. МЕТОДИЧНІ РЕКОМЕНДАЦІЇ до виконання кваліфікаційних робіт здобувачами другого (магістерського) рівня вищої освіти спеціальностей 121 «Інженерія програмного забезпечення» / / Б.І. Мороз, О.В. Іванченко, О.В. Реута, О.С. Шевцова; М-во освіти і науки України, Нац. техн. ун-т “Дніпровська політехніка”. – Дніпро : НТУ «ДП», 2021. – 56 с.

2. Diabetes [Електронний ресурс] // World Health Organization (WHO). – Режим доступу: https://www.who.int/news-room/fact-sheets/detail/diabetes (дата звернення: 16.09.2023). – Назва з екрана..

3. Diabetes [Електронний ресурс] // World Health Organization (WHO). – Режим доступу: https://www.who.int/news-room/fact-sheets/detail/diabetes (дата звернення: 16.09.2023). – Назва з екрана.

4. Facts & figures [Електронний ресурс] // International Diabetes Federation. – Режим доступу: https://idf.org/about-diabetes/diabetes-facts-figures/ (дата звернення: 17.09.2023). – Назва з екрана.

5. Type 2 Diabetes [Електронний ресурс] // Centers for Disease Control and Prevention. – Режим доступу: https://www.cdc.gov/diabetes/basics/type2.html (дата звернення: 17.09.2023). – Назва з екрана.

6. Williams Textbook of Endocrinology E-Book / Shlomo MD Melmed [та ін.]. – 12-те вид. – [Б. м. : б. в.], 2011. – 1904 с.

7. Про зв’язок вуглеводів і діабету [Електронний ресурс] // Все про діабет. – Режим доступу: https://diabet.org.ua/pro-zvyazok-vuhlevodiv-i-diabetu/ (дата звернення: 18.09.2023). – Назва з екрана.

8. Dietary carbohydrate restriction as the first approach in diabetes management: Critical review and evidence base. ScienceDirect. [Електронний ресурс] // Science Direct – Режим доступу: https://www.sciencedirect.com/science/article/pii/S0899900714003323 (дата звернення: 18.09.2023).– Назва з екрана.

9. Low-carbohydrate diet in type 2 diabetes: stable improvement of bodyweight and glycemic control during 44 months follow-up [Електронний ресурс] // PubMed Central (PMC). – Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2424054/ (дата звернення: 20.09.2023). – Назва з екрана.

10. Fletcher J. Carbs and diabetes: Relationship, benefits, risks, and more [Електронний ресурс] / Jenna Fletcher // Medical and health information. – Режим доступу: https://www.medicalnewstoday.com/articles/carbs-and-diabetes?utm_source=Sailthru%20Email&utm_medium=Email&utm_campaign=dedicated&utm_content=2021-04-18&apid=35785309&rvid=ba09cd4713629821ea5c8e1040a2de2d9000b5f0814c7433bf42491e692bb526&fbclid=IwAR0_qK5esdlJ84XgUuoGdGcFYYF94TMoWkOvpwA-uGuOrNFTO_5YUO12HqM (дата звернення: 20.09.2023). – Назва з екрана.

11. Види цукру, їхнє походження та метаболічний шлях [Електронний ресурс] // Національний університет біоресурсів і природокористування України. – Режим доступу: https://nubip.edu.ua/node/114069 (дата звернення: 20.09.2023). – Назва з екрана.

12. Норма цукру в крові та як слідкувати за його рівнем. – Державна установа "Одеський обласний центр контролю та профілактики хвороб Міністерства охорони здоров’я України" [Електронний ресурс] // Державна установа "Одеський обласний центр контролю та профілактики хвороб Міністерства охорони здоров’я України" – Державна Установа "ОДЕСЬКИЙ ОБЛАСТНИЙ ЦЕНТР КОНТРОЛЮ ТА ПРОФІЛАКТИКИ ХВОРОБ МОЗ УКРАЇНИ". – Режим доступу: https://oolc.od.ua/норма-цукру-в-крові-та-як-слідкувати-за/ (дата звернення: 05.10.2023). – Назва з екрана.

13. СахарОК - інтернет-журнал про цукровий діабет. – Режим доступу: https://mysugar.media/pdf/view/100 (дата звернення: 05.10.2023). – Назва з екрана.

14. Blood sugar testing: Why, when and how [Електронний ресурс] // Mayo Clinic. – Режим доступу: https://www.mayoclinic.org/diseases-

conditions/diabetes/in-depth/blood-sugar/art-20046628 (дата звернення: 08.10.2023). – Назва з екрана.

15. Вимірювання рівня цукру в крові: чому, коли і як - Поради - DIAWIN - Товари для людей з цукровим діабетом [Електронний ресурс] // DIAWIN - Товари для людей з цукровим діабетом. – Режим доступу: http://www.diawin.com.ua/pages/vymiryuvania_rivnya_cukru_v_krovi (дата звернення: 08.10.2023). – Назва з екрана.

16. Аліна У. Норма цукру в крові та як слідкувати за його рівнем [Електронний ресурс] / Урбанович Аліна // Medialt. – Режим доступу: https://medialt.clinic/blog/diagnostyka/norma-tsukru-v-krovi (дата звернення: 09.10.2023). – Назва з екрана.

17. MVVM (Model View ViewModel) Architecture Pattern in Android - GeeksforGeeks [Електронний ресурс] // GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/ (дата звернення: 11.10.2023). – Назва з екрана.

18. What Is MVVM Architecture? [Електронний ресурс] // Built In. – Режим доступу: https://builtin.com/software-engineering-perspectives/mvvm-architecture (дата звернення: 11.10.2023). – Назва з екрана.

19. MVVM: Model-View-ViewModel Architecture | Ramotion Branding Agency [Електронний ресурс] // Web Design, UI/UX, Branding, and App Development Blog. – Режим доступу: https://www.ramotion.com/blog/what-is-mvvm/#section-advantages-of-mvvm (дата звернення: 13.10.2023). – Назва з екрана.

20. Bondar S. Working with Windows Presentation Foundation | Reintech media [Електронний ресурс] / Sasha Bondar // Software Developers as a Service | Reintech. – Режим доступу: https://reintech.io/blog/tutorial-introduction-working-windows-presentation-foundation (дата звернення: 13.10.2023). – Назва з екрана.

21. WPF vs. WinForms - The complete WPF tutorial [Електронний ресурс] // Welcome - The complete WPF tutorial. – Режим доступу: https://wpf-

tutorial.com/about-wpf/wpf-vs-winforms/ (дата звернення: 15.10.2023). – Назва з екрана.

22. Karia R. Entity Framework Core [Електронний ресурс] / Ravi Karia // Entity Framework Tutorial. – Режим доступу: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx (дата звернення: 15.10.2023). – Назва з екрана.

23. Overview of Entity Framework Core - EF Core [Електронний ресурс] // Microsoft Learn: Build skills that open doors in your career. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 15.10.2023). – Назва з екрана.

24. Language Integrated Query (LINQ) in C# - C# [Електронний ресурс] // Microsoft Learn: Build skills that open doors in your career. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/linq/ (дата звернення: 16.10.2023). – Назва з екрана.

25. PostgreSQL: About [Електронний ресурс] // PostgreSQL: The world’s most advanced open source database. – Режим доступу: https://www.postgresql.org/about/ (дата звернення: 17.10.2023). – Назва з екрана.

26. Why use PostgreSQL as a Database for my Next Project in 2022 - Fulcrum [Електронний ресурс] // Fulcrum. – Режим доступу: https://fulcrum.rocks/blog/why-use-postgresql-database#mongodb-vs-postgresql-1 (дата звернення: 17.10.2023). – Назва з екрана.

27. PostgreSQL Advantages: Benefits of Using PostgreSQL [Електронний ресурс] // Prisma’s Data Guide. – Режим доступу: https://www.prisma.io/dataguide/postgresql/benefits-of-postgresql (дата звернення: 18.10.2023). – Назва з екрана.

28. A tour of C# - Overview - C# [Електронний ресурс] // Microsoft Learn: Build skills that open doors in your career. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (дата звернення: 20.10.2023). – Назва з екрана.

29. What is C# used for? [Електронний ресурс] // Stackify. – Режим доступу: https://stackify.com/what-is-c-used-for/ (дата звернення: 20.10.2023). – Назва з екрана.

30. Angella A. 15 reasons why you should learn C# in 2023 [Електронний ресурс] / Andrea Angella // LinkedIn: Log In or Sign Up. – Режим доступу: https://www.linkedin.com/pulse/15-reasons-why-you-should-learn-c-2023-andrea-angella (дата звернення: 25.10.2023). – Назва з екрана.

31. Visual Studio: IDE and Code Editor for Software Developers and Teams [Електронний ресурс] // Visual Studio. – Режим доступу: https://visualstudio.microsoft.com/ (дата звернення: 25.10.2023). – Назва з екрана.

32. Проблеми використання інформаційних технологій в освіті, науці тапромисловості: XVII міжнар. конф. (24 листопада 2022 р., м. Дніпро): зб. наук. пр. [Електронний ресурс] / ред. кол.: О.О. Азюковський та ін.; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Електрон. текст. дані – Дніпро: НТУ «ДП», 2023. – № 7. – 217 с. – Режим доступу: https://ir.nmu.org.ua/handle/123456789/163499. – Назва з екрана.

ДОДАТОК А

КОД ПРОГРАМИ

Модель-представлення для забезпечення введення користувачем значень рівню цукру в крові

```
using DevExpress.Mvvm;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Globalization;
using System.Linq;
using System.Text.RegularExpressions;
using System.Windows;
using System.Windows.Input;
using WpfApp2TypeDiabet.Commands;
using WpfApp2TypeDiabet.DBServices;
using WpfApp2TypeDiabet.Models;
using WpfApp2TypeDiabet.Pages;
using WpfApp2TypeDiabet.Services;

namespace WpfApp2TypeDiabet.ViewModels
{
    //Модель-представлення для забезпечення введення користувачем
значень рівню цукру в крові
    public class BloodSugarLevelDataEntryViewModel : BindableBase,
INotifyPropertyChanged
    {
        private DateTime _date;
        private double _morningLevel;
        private double _afternoonLevel;
        private double _eveningLevel;
        //сервіс навігації
        private readonly NavigationService _navigation;
        //користувальницький сервіс
        private readonly UserService _userService;
        //конструктор з параметрами
        public BloodSugarLevelDataEntryViewModel(NavigationService
navigation, UserService userService)
        {
            _navigation = navigation;
            _userService = userService;

            LoadUserDataFromDatabase(_userService.CurrentUser.id);
            InitializeUIFromUserDataList();
        }
        //команда переходу на головну сторінку
        public ICommand GoToMainCommand => new DelegateCommand(() =>
        {
            MessageBoxResult result = MessageBox.Show("Ви впевнені, що
хочете скасувати обчислення?",
```

```
                "Скасування обчислень", MessageBoxButton.YesNo,
MessageBoxImage.Question);
            if (result == MessageBoxResult.Yes)
            {
                if (_userService.CurrentUser.IsSuperUser)
                {
                    _navigation.Navigate(new AdminMainPage());
                }
                else
                {
                    if
(_userService.CurrentUser.UserName.Equals("Guest"))
                    {
                        _navigation.Navigate(new GuestMainPage());
                    }
                    else
                    {
                        _navigation.Navigate(new UserMainPage());
                    }
                }
            }
        });
        //команда переходу на попередню сторінку
        public ICommand GoBackCommand => new DelegateCommand(() =>
        {
            MessageBoxResult result = MessageBox.Show("Ви впевнені, що
хочете скасувати обчислення?",
                "Скасування обчислень", MessageBoxButton.YesNo,
MessageBoxImage.Question);
            if (result == MessageBoxResult.Yes)
            {
                _navigation.GoBack();
            }
        });
        //команда переходу на сторінку побудови графіків
        public ICommand OpenPlotWindowCommand => new DelegateCommand(()
=>
        {
            _navigation.Navigate(new BloodSugarLevelPlot());
        });

        private string _addOrUpdateButtonText = "Додати дані";
        //властивість для встановлення тексту кнопки на
користувальницькому інтерфейсі
        public string AddOrUpdateButtonText
        {
            get { return _addOrUpdateButtonText; }
            set
            {
                _addOrUpdateButtonText = value;
                OnPropertyChanged(nameof(AddOrUpdateButtonText));
            }
        }
        //метод для ініціалізації даних поточного користувача
        public void CurrentUserInitialize()
        {
            CurrentUserData.MorningLevel = MorningLevel;
```

```
            CurrentUserData.AfternoonLevel = AfternoonLevel;
            CurrentUserData.EveningLevel = EveningLevel;
            CurrentUserData.Age = _userService.CurrentUser.Age;
            CurrentUserData.Date = SelectedDate;

            OnPropertyChanged(nameof(CurrentUserData));
        }
        //список даних про користувача
        public ObservableCollection<UserDataModel> UserDataList { get;
set; } = new ObservableCollection<UserDataModel>();

        private UserDataModel _currentUserData = new();
        //властивість для взаємодії з даними користувача
        public UserDataModel CurrentUserData
        {
            get { return _currentUserData; }
            set
            {
                _currentUserData = value;
                OnPropertyChanged(nameof(CurrentUserData));
            }
        }

        private UserDataModel _selectedUserData;
        //властивість для взаємодії користувача з обраними даними
        public UserDataModel SelectedUserData
        {
            get { return _selectedUserData; }
            set
            {
                if (_selectedUserData != value)
                {
                    _selectedUserData = value;
                    OnPropertyChanged(nameof(SelectedUserData));

                    UpdateUIFromSelectedRecord();
                    ValidateData();
                }
            }
        }
        //метод для оновлення значень елементів графічного інтерфейсу на
основі обраного запису
        private void UpdateUIFromSelectedRecord()
        {
            if (SelectedUserData != null)
            {
                MorningLevel = SelectedUserData.MorningLevel;
                AfternoonLevel = SelectedUserData.AfternoonLevel;
                EveningLevel = SelectedUserData.EveningLevel;
                SelectedDate = SelectedUserData.Date;

                AddOrUpdateButtonText = "Оновити дані";
            }
        }
        //метод для ініціалізації елементів графічного інтерфейсу зі
списку даних користувача
        private void InitializeUIFromUserDataList()
```

```
        {
            if (UserDataList.Count > 0)
            {
                SelectedUserData = UserDataList[0];
            }
            else
            {
                ClearUI();
            }
        }

        private DateTime _selectedDate = DateTime.Now;
        //властивість для збереження та встановлення обраної дати
        public DateTime SelectedDate
        {
            get { return _selectedDate; }
            set
            {
                _selectedDate = value;
                OnPropertyChanged(nameof(SelectedDate));
                ValidateData();
            }
        }

        private bool isPlotButtonEnabled;
        //властивість для встановлення стану доступності кнопки
відкриття сторінки побудови графіків
        public bool IsPlotButtonEnabled
        {
            get { return isPlotButtonEnabled; }
            set
            {
                isPlotButtonEnabled = value && UserDataList.Count > 0;
                OnPropertyChanged(nameof(IsPlotButtonEnabled));
            }
        }

        public DateTime Date
        {
            get { return _date; }
            set { _date = value.Date; OnPropertyChanged(nameof(Date)); }
        }
        //властивість для збереження даних про введений ранковий рівень
цукру в крові
        public double MorningLevel
        {
            get { return _morningLevel; }
            set
            {
                if (_morningLevel != value)
                {
                    _morningLevel = value;
                    OnPropertyChanged(nameof(MorningLevel));
                    ValidateData();
                }
            }
        }
```

```
        //властивість для збереження даних про введений денний рівень
цукру в крові
        public double AfternoonLevel
        {
            get { return _afternoonLevel; }
            set
            {
                if (_afternoonLevel != value)
                {
                    _afternoonLevel = value;
                    OnPropertyChanged(nameof(AfternoonLevel));
                    ValidateData();
                }
            }
        }
        //властивість для збереження даних про введений вечірній рівень
цукру в крові
        public double EveningLevel
        {
            get { return _eveningLevel; }
            set
            {
                if (_eveningLevel != value)
                {
                    _eveningLevel = value;
                    OnPropertyChanged(nameof(EveningLevel));
                    ValidateData();
                }
            }
        }

        private bool _isDataInvalid;
        //властивіть-показник, чи введені дані хибними
        public bool IsDataInvalid
        {
            get { return _isDataInvalid; }
            set
            {
                if (_isDataInvalid != value)
                {
                    _isDataInvalid = value;
                    OnPropertyChanged(nameof(IsDataInvalid));
                    (AddDataCommand as
RelayCommand)?.RaiseCanExecuteChanged();
                }
            }
        }
        //метод для перевірки введених даних
        private void ValidateData()
        {
            string doublePattern = @"^\d+(\.\d+)?$";
            Regex regex = new Regex(doublePattern);

            bool isValid =
regex.IsMatch(MorningLevel.ToString(CultureInfo.InvariantCulture))
                && MorningLevel > 0.0
```

```
                &&
regex.IsMatch(AfternoonLevel.ToString(CultureInfo.InvariantCulture))
                && AfternoonLevel > 0.0
                &&
regex.IsMatch(EveningLevel.ToString(CultureInfo.InvariantCulture))
                && EveningLevel > 0.0
                && SelectedDate <= DateTime.Today.Date;

            IsDataInvalid = !isValid;
        }

        private ICommand _addDataCommand;
        //команда для прив’язки елементу графічного інтерфейсу при
додаванні даних
        public ICommand AddDataCommand
        {
            get
            {
                return _addDataCommand ?? (_addDataCommand = new
RelayCommand(
                    ExecuteAddOrUpdateDataCommand,
                    () => !IsDataInvalid));
            }
        }
        //метод для додавання і оновлення даних
        private void ExecuteAddOrUpdateDataCommand()
        {
            if (!IsDataInvalid)
            {
                if (UserDataList.Any(u => u.Date == SelectedDate))
                {
                    var existingUserData = UserDataList.First(u =>
u.Date == SelectedDate);
                    existingUserData.Age = _userService.CurrentUser.Age;
                    existingUserData.MorningLevel = MorningLevel;
                    existingUserData.AfternoonLevel = AfternoonLevel;
                    existingUserData.EveningLevel = EveningLevel;
                    existingUserData.Date = SelectedDate;
                }
                else
                {
                    CurrentUserInitialize();
                    SaveUserDataToDatabase(CurrentUserData);

LoadUserDataFromDatabase(_userService.CurrentUser.id);
                }

                ClearUI();
                (AddDataCommand as
RelayCommand)?.RaiseCanExecuteChanged();

                AddOrUpdateButtonText = "Додати дані";
            }
        }
        //метод для встановлення даних за замовчуванням елементам
графічного інтерфейсу
        private void ClearUI()
```

```
        {
            SelectedDate = DateTime.Today.Date;

            MorningLevel = 0.0;
            AfternoonLevel = 0.0;
            EveningLevel = 0.0;

            CurrentUserInitialize();
        }
        //метод для збереження даних у базу даних
        public void SaveUserDataToDatabase(UserDataModel userData)
        {
            using (var context = new ApplicationContext())
            {
                var existingUser = context.Users.FirstOrDefault(u =>
u.id == _userService.CurrentUser.id);
                var newIndicators = new Indicators
                {
                    MorningLevel = userData.MorningLevel,
                    AfternoonLevel = userData.AfternoonLevel,
                    EveningLevel = userData.EveningLevel,
                    AverageLevel = userData.AverageLevel,

                    IndicatorDate = userData.Date
                };

                context.Indicators.Add(newIndicators);

                var userIndic = new UserIndic
                {
                    User = existingUser,
                    Indicators = newIndicators
                };

                context.UserIndic.Add(userIndic);

                context.SaveChanges();
            }
        }

        private string _formattedDate;
        //властивість для збереження відформатованої дати
        public string FormattedDate
        {
            get { return _formattedDate; }
            set
            {
                _formattedDate = value;
                if (DateTime.TryParseExact(value, "dd.MM.yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out var date))
                {
                    CurrentUserData.Date = date;
                    OnPropertyChanged(nameof(FormattedDate));
                }
            }
        }
        //метод для завантаження даних із бази даних
```

```
        public void LoadUserDataFromDatabase(int userId)
        {
            using (var context = new ApplicationContext())
            {
                var userIndics = context.UserIndic
                    .Include(ui => ui.User)
                    .Include(ui => ui.Indicators)
                    .Where(ui => ui.User.id == userId)
                    .ToList();

                UserDataList.Clear();

                foreach (var userIndic in userIndics)
                {
                    var userDataModel = new UserDataModel
                    {
                        Date = userIndic.Indicators.IndicatorDate,
                        MorningLevel =
userIndic.Indicators.MorningLevel,
                        AfternoonLevel =
userIndic.Indicators.AfternoonLevel,
                        EveningLevel =
userIndic.Indicators.EveningLevel,
                        Age = userIndic.User.Age
                    };

                    UserDataList.Add(userDataModel);
                }
                if (UserDataList.Count > 0)
                {
                    FormattedDate =
UserDataList[0].Date.ToString("dd.MM.yyyy");
                }
            }
            IsPlotButtonEnabled = UserDataList.Count > 0;
            OnPropertyChanged(nameof(IsPlotButtonEnabled));
        }

        public event PropertyChangedEventHandler PropertyChanged;
        //метод для нотифікації, коли змінюється значення елементу
графічного інтерфейсу
        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));

            if (propertyName != nameof(IsPlotButtonEnabled))
            {
                OnPropertyChanged(nameof(IsPlotButtonEnabled));
            }
            if (propertyName != nameof(IsDataInvalid))
            {
                ValidateData();
            }

            if (propertyName == nameof(SelectedUserData))
            {
```

```
                UpdateUIFromSelectedRecord();
            }
        }
    }
}
```

Модель-представлення для забезпечення введення користувачем значень рівню цукру в крові

```
using DevExpress.Mvvm;
using LiveCharts;
using LiveCharts.Wpf;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using WpfApp2TypeDiabet.DBServices;
using WpfApp2TypeDiabet.Models;
using WpfApp2TypeDiabet.Pages;
using WpfApp2TypeDiabet.Services;
using static WpfApp2TypeDiabet.ViewModels.BasketListPageViewModel;

namespace WpfApp2TypeDiabet.ViewModels
{
    //Модель-представлення для забезпечення введення користувачем
значень рівню цукру в крові
    public class BloodSugarLevelPlotViewModel : BindableBase,
INotifyPropertyChanged
    {
        //перелік товарів
        public ObservableCollection<GoodBasket> GoodList { get; set; } =
new ObservableCollection<GoodBasket>();
        //перелік продуктових кошиків
        public ObservableCollection<BasketToView> GoodBaskets { get;
set; } = new ObservableCollection<BasketToView>();
        //допоміжні сервіси
        private readonly NavigationService _navigation;
        private readonly UserService _userService;
        private ObservableCollection<UserDataModel> _userDataList;
        private readonly GoodBasketService _goodBasketService;
        private readonly GoodInBasketService _goodInBasketService;
        private readonly GoodInShopService _goodInShopService;
        private readonly OptimizeService _optimizeService;
        //конструктор з параметрами
        public BloodSugarLevelPlotViewModel(NavigationService
navigation, UserService userService,
            OptimizeService optimizeService, GoodBasketService
goodBasketService,
            GoodInBasketService goodInBasketService, GoodInShopService
goodInShopService)
        {
```

```
            _navigation = navigation;
            _userService = userService;
            UserDataList = new ObservableCollection<UserDataModel>();
            _optimizeService = optimizeService;
            _goodBasketService = goodBasketService;
            _goodInBasketService = goodInBasketService;
            _goodInShopService = goodInShopService;

            using var context = new ApplicationContext();
            var userIndics = context.UserIndic
                .Include(ui => ui.User)
                .Include(ui => ui.Indicators)
                .Where(ui => ui.User.id == _userService.CurrentUser.id)
                .ToList();

            UserDataList.Clear();

            foreach (var userIndic in userIndics)
            {
                var userDataModel = new UserDataModel
                {
                    Date = userIndic.Indicators.IndicatorDate,
                    MorningLevel = userIndic.Indicators.MorningLevel,
                    AfternoonLevel =
userIndic.Indicators.AfternoonLevel,
                    EveningLevel = userIndic.Indicators.EveningLevel,
                    Age = userIndic.User.Age
                };

                UserDataList.Add(userDataModel);
            }

            SeriesCollection = new SeriesCollection();
            LegendItems = new ObservableCollection<LegendItem>();

            foreach (var legendItem in LegendItems)
            {
                legendItem.IsCheckedChanged +=
LegendItem_IsCheckedChanged;
            }

        }
        //команда переходу на головну сторінку
        public ICommand GoToMainCommand => new DelegateCommand(() =>
        {
            MessageBoxResult result = MessageBox.Show("Ви впевнені, що
хочете припинити перегляд результатів?",
                "Припинення перегляду", MessageBoxButton.YesNo,
MessageBoxImage.Question);
            if (result == MessageBoxResult.Yes)
            {
                if (_userService.CurrentUser.IsSuperUser)
                {
                    _navigation.Navigate(new AdminMainPage());
                }
                else
                {
```

```
                    if
(_userService.CurrentUser.UserName.Equals("Guest"))
                    {
                        _navigation.Navigate(new GuestMainPage());
                    }
                    else
                    {
                        _navigation.Navigate(new UserMainPage());
                    }
                }
            }
        });
        //команда переходу на попереднью сторінку
        public ICommand GoBackCommand => new DelegateCommand(() =>
        {
            MessageBoxResult result = MessageBox.Show("Ви впевнені, що
хочете припинити перегляд результатів?",
                "Припинення перегляду", MessageBoxButton.YesNo,
MessageBoxImage.Question);
            if (result == MessageBoxResult.Yes)
            {
                _navigation.GoBack();
            }
        });
        //властивість-нотифікатор відбувшихся змін
        public event PropertyChangedEventHandler PropertyChanged;
        //метод для нотифікації, коли змінюється значення елементу
графічного інтерфейсу
        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
        //властивість для збереження списку даних про користувача
        public ObservableCollection<UserDataModel> UserDataList
        {
            get { return _userDataList; }
            set
            {
                _userDataList = value;
                OnPropertyChanged(nameof(UserDataList));
                UpdateChartData(_userDataList);
            }
        }

        private SeriesCollection _seriesCollection;
        //властивість для взаємодії із серіями на графіку
        public SeriesCollection SeriesCollection
        {
            get { return _seriesCollection; }
            set { _seriesCollection = value;
OnPropertyChanged(nameof(SeriesCollection)); }
        }

        private List<string> _labels;
        //властивість для взаємодії із підписами вісей на графіку
        public List<string> Labels
```

```
{
    get { return _labels; }
    set { _labels = value; OnPropertyChanged(nameof(Labels)); }
}

private bool _isPointed = true;
//Властивість-показник типу графіку - точковий
public bool IsPointed
{
    get { return _isPointed; }
    set
    {
        _isPointed = value;
        OnPropertyChanged(nameof(IsPointed));
        UpdateChartData(_userDataList);
    }
}

private bool _isHistogram = false;
//Властивість-показник типу графіку - гістограма
public bool IsHistogram
{
    get { return _isHistogram; }
    set
    {
        _isHistogram = value;
        OnPropertyChanged(nameof(IsHistogram));
        UpdateChartData(_userDataList);
    }
}

private bool _isAllData = true;
//Властивість-показник даних для аналізу - усі дані
public bool IsAllData
{
    get { return _isAllData; }
    set
    {
        _isAllData = value;
        OnPropertyChanged(nameof(IsAllData));
        UpdateChartData(_userDataList);
    }
}

private bool _isAverageData = false;
//Властивість-показник даних для аналізу - середнє значення
public bool IsAverageData
{
    get { return _isAverageData; }
    set
    {
        _isAverageData = value;
        OnPropertyChanged(nameof(IsAverageData));
        UpdateChartData(_userDataList);
    }
}
```

```
        private DateTime? _startDate;
        //властивість для взаємодії із початковою датою для відображення
даних
        public DateTime? StartDate
        {
            get { return _startDate; }
            set
            {
                _startDate = value;
                OnPropertyChanged(nameof(StartDate));
                UpdateChartData(UserDataList);
            }
        }

        private DateTime? _endDate;
        //властивість для взаємодії із кінцевою датою для відображення
даних
        public DateTime? EndDate
        {
            get { return _endDate; }
            set
            {
                _endDate = value;
                OnPropertyChanged(nameof(EndDate));
                UpdateChartData(UserDataList);
            }
        }

        private ObservableCollection<LegendItem> _legendItems;
        //колекція елемент ів легенди графіку
        public ObservableCollection<LegendItem> LegendItems
        {
            get { return _legendItems; }
            set { _legendItems = value;
OnPropertyChanged(nameof(LegendItems)); }
        }

        private Visibility _errorMessageVisibility =
Visibility.Collapsed;
        //властивість для встановлення видимості повідомлення про
помилку
        public Visibility ErrorMessageVisibility
        {
            get { return _errorMessageVisibility; }
            set
            {
                _errorMessageVisibility = value;
                OnPropertyChanged(nameof(ErrorMessageVisibility));
            }
        }
        private Visibility _chartVisibility = Visibility.Visible;
        //властивість для встановлення видимості графіку
        public Visibility ChartVisibility
        {
            get { return _chartVisibility; }
            set
            {
```

```
                _chartVisibility = value;
                OnPropertyChanged(nameof(ChartVisibility));
            }
        }
        private Visibility _legendVisibility = Visibility.Visible;
        //властивість для встановлення видимості легенди
        public Visibility LegendVisibility
        {
            get { return _legendVisibility; }
            set
            {
                _chartVisibility = value;
                OnPropertyChanged(nameof(LegendVisibility));
            }
        }
        //метод для оновлення даних на графіку
        private void UpdateChartData(ObservableCollection<UserDataModel>
userDataList)
        {
            var filteredData = FilterData(userDataList);

            if (SeriesCollection != null)
            {
                SeriesCollection.Clear();
            }
            if (filteredData.Count == 0)
            {
                ShowErrorMessage();
                HideChart();
            }
            else
            {
                HideErrorMessage();
                ShowChart();

                if (IsHistogram)
                {
                    CreateHistogramSeries(filteredData);
                }
                else
                {
                    CreateLineSeries(filteredData);
                }
                ShowBaskets();
            }
        }
        //метод для отримання списку даних із застосуванням фільтрації
        private List<UserDataModel>
FilterData(ObservableCollection<UserDataModel> userDataList)
        {
            var startDate = StartDate;
            var endDate = EndDate;

            if (!startDate.HasValue || !endDate.HasValue)
            {
                return new List<UserDataModel>();
            }
```

```
            var filteredData = userDataList.Where(u => u.Date >=
startDate.Value && u.Date <= endDate.Value).OrderBy(u =>
u.Date).ToList();

            if (IsAllData)
            {
                return filteredData;
            }
            else if (IsAverageData)
            {
                return filteredData.Select(u =>
                {
                    u.MorningLevel = u.AverageLevel;
                    u.AfternoonLevel = u.AverageLevel;
                    u.EveningLevel = u.AverageLevel;
                    return u;
                }).ToList();
            }

            return filteredData;
        }
        //метод для створення гістограми на основі вказаних даних
        private void CreateHistogramSeries(List<UserDataModel>
filteredData)
        {
            LegendItems.Clear();

            SeriesCollection.Clear();

            Labels = filteredData.Select(u =>
u.Date.ToString("dd/MM/yyyy")).ToList();

            if (IsAllData)
            {
                CreateAndAddColumnSeries(filteredData, "Рівень цукру
вранці", Brushes.Green);
                CreateAndAddColumnSeries(filteredData, "Рівень цукру
вдень", Brushes.Blue);
                CreateAndAddColumnSeries(filteredData, "Рівень цукру
ввечері", Brushes.Red);
            }
            else
            {
                CreateAndAddColumnSeries(filteredData, "Середній рівень
цукру", Brushes.Green);
            }

            var currentUser = filteredData.FirstOrDefault();
            if (currentUser != null)
            {
                AddNormSeries(currentUser.Age);
            }
        }
        //метод для відображення графіків медичних норм рівня цукру в
крові
        private void AddNormSeries(int age)
```

```
        {
            var norms = new List<(string Title, Color Color, double
Value)>
            {
                ("Норма натще (Здорова людина) (Мін)",
Colors.LightGreen, CalculateFastingNormNonDiabetic().Min),
                ("Норма натще (Здорова людина) (Макс)",
Colors.LightSeaGreen, CalculateFastingNormNonDiabetic().Max),
                ("Норма натще (Діабетик) (Мін)", Colors.Red,
CalculateFastingNormDiabetic().Min),
                ("Норма натще (Діабетик) (Макс)", Colors.DeepPink,
CalculateFastingNormDiabetic().Max),
                ("Норма після прийому їжі (Здорова людина)",
Colors.LightBlue, CalculatePostMealNormNonDiabetic()),
                ("Норма після прийому їжі (Діабетик)", Colors.Orange,
CalculatePostMealNormDiabetic()),
                ("Еталонне значення (Мін)", Colors.Purple,
CalculateReferenceValues(age).Min),
                ("Еталонне значення (Макс)", Colors.MediumPurple,
CalculateReferenceValues(age).Max),
            };

            foreach (var norm in norms)
            {
                CreateAndAddColumnNormSeries(norm.Title, norm.Color,
norm.Value);
            }
        }
        //метод для створення та додавання стовбчатої серії на графік
для норм
        private void CreateAndAddColumnNormSeries(string title, Color
color, double norm)
        {
            var series = CreateColumnSeries(title, color, norm);

            var legendItem = LegendItems.FirstOrDefault(item =>
item.SeriesTitle == title);
            if (legendItem == null)
            {
                legendItem = CreateLegendItem(series);
            }
            else
            {
                series.Visibility = legendItem.IsChecked ?
Visibility.Visible : Visibility.Hidden;
            }

            legendItem.AssociatedSeries.Add(series);

            SeriesCollection.Add(series);
        }
        //метод для додавання лінійної серії до графіку
        private void CreateLineSeries(List<UserDataModel> filteredData)
        {
            LegendItems.Clear();
            if (IsAllData)
            {
```

```
                CreateAndAddLineSeries(filteredData, "Рівень цукру
вранці", Brushes.Green);
                CreateAndAddLineSeries(filteredData, "Рівень цукру
вдень", Brushes.Blue);
                CreateAndAddLineSeries(filteredData, "Рівень цукру
ввечері", Brushes.Red);

                Labels = filteredData.Select(u =>
u.Date.ToString("dd/MM/yyyy")).ToList();
            }
            else
            {
                CreateAndAddLineSeries(filteredData, "Середній рівень
цукру", Brushes.Green);

                Labels = filteredData.Select(u =>
u.Date.ToString("dd/MM/yyyy")).ToList();
            }
        }
        //метод для створення та додавання стовбчатої серії на графік
для даних користувача
        private void CreateAndAddColumnSeries(List<UserDataModel>
filteredData, string title, SolidColorBrush fillBrush)
        {
            var series = new ColumnSeries
            {
                Title = title,
                Values = new ChartValues<double>(filteredData.Select(u
=> GetSeriesValue(u, title))),
                Fill = fillBrush,
                DataLabels = true,
                LabelPoint = point => $"{point.Y:N2}",
                Foreground = Brushes.Black,
                StrokeThickness = 0
            };
            var legendItem = CreateLegendItem(series);

            series.Visibility = legendItem.IsChecked ?
Visibility.Visible : Visibility.Hidden;

            SeriesCollection.Add(series);
        }
        //метод для створення та додавання лінійної серії на графік для
даних користувача
        private void CreateAndAddLineSeries(List<UserDataModel>
filteredData, string title, SolidColorBrush fillBrush)
        {
            var series = new LineSeries
            {
                Title = title,
                Values = new ChartValues<double>(filteredData.Select(u
=> GetSeriesValue(u, title))),
                Fill = fillBrush
            };
            var legendItem = CreateLegendItem(series);
```

```
            series.Visibility = legendItem.IsChecked ?
Visibility.Visible : Visibility.Hidden;

            SeriesCollection.Add(series);
        }
        //метод для створення та додавання елемента-легенди на
користувальницький інтерфейс
        private LegendItem CreateLegendItem(Series series)
        {
            SolidColorBrush fillBrush = null;

            if (series is ColumnSeries columnSeries)
            {
                fillBrush = columnSeries.Fill as SolidColorBrush;
            }
            else if (series is LineSeries lineSeries)
            {
                fillBrush = lineSeries.Fill as SolidColorBrush;
            }

            var legendItem = LegendItems.FirstOrDefault(item =>
item.SeriesTitle == series.Title);
            if (legendItem == null)
            {
                legendItem = new LegendItem
                {
                    SeriesTitle = series.Title,
                    Color = fillBrush,
                    IsChecked = true,
                    Visibility = series?.Visibility ??
Visibility.Visible
                };
                legendItem.IsCheckedChanged +=
LegendItem_IsCheckedChanged;
                LegendItems.Add(legendItem);
            }

            legendItem.AssociatedSeries.Add(series);

            return legendItem;
        }
        //метод для отримання значень серії
        private double GetSeriesValue(UserDataModel userData, string
title)
        {
            switch (title)
            {
                case "Рівень цукру вранці":
                    return userData.MorningLevel;
                case "Рівень цукру вдень":
                    return userData.AfternoonLevel;
                case "Рівень цукру ввечері":
                    return userData.EveningLevel;
                case "Середній рівень цукру":
                    return userData.AverageLevel;
                default:
                    return 0;
```

```
            }
        }
        //метод-нотифікатор зміни стану елемента-легенди
        private void LegendItem_IsCheckedChanged(object sender,
EventArgs e)
        {
            var legendItem = sender as LegendItem;

            if (legendItem != null && legendItem.AssociatedSeries !=
null)
            {
                foreach (var associatedseries in
legendItem.AssociatedSeries)
                {
                    associatedseries.Visibility = legendItem.IsChecked ?
Visibility.Visible : Visibility.Hidden;
                }
            }
        }
        //метод для показу повідомлення про помилку
        private void ShowErrorMessage()
        {
            ErrorMessageVisibility = Visibility.Visible;
        }
        //метод для приховання повідомлення про помилку
        private void HideErrorMessage()
        {
            ErrorMessageVisibility = Visibility.Collapsed;
        }
        //метод для показу графіку
        private void ShowChart()
        {
            ChartVisibility = Visibility.Visible;
            LegendVisibility = Visibility.Visible;
        }
        //метод для приховання графіку
        private void HideChart()
        {
            ChartVisibility = Visibility.Collapsed;
            LegendVisibility = Visibility.Collapsed;
        }
        //метод для створення стовбчастої серії
        private ColumnSeries CreateColumnSeries(string title, Color
color, double norm)
        {
            var series = new ColumnSeries
            {
                Title = title,
                Values = new ChartValues<double> { norm, norm },
                Fill = new SolidColorBrush(color),
                MaxColumnWidth = 15,
                DataLabels = true,
                LabelPoint = point => $"{point.Y:N2}",
                Foreground = Brushes.Black,
                StrokeThickness = 0
            };
```

```
            return series;
        }
        //методи для отримання значень про норми рівня цукру у крові
        public (double Min, double Max)
CalculateFastingNormNonDiabetic()
        {
            double minNorm, maxNorm;

            minNorm = 3.9;
            maxNorm = 5.5;

            return (minNorm, maxNorm);
        }

        public (double Min, double Max) CalculateFastingNormDiabetic()
        {
            double minNorm, maxNorm;

            minNorm = 4.4;
            maxNorm = 7.2;

            return (minNorm, maxNorm);
        }

        public double CalculatePostMealNormDiabetic()
        {
            double maxNorm = 10.0;
            return maxNorm;
        }

        public double CalculatePostMealNormNonDiabetic()
        {
            double maxNorm = 7.8;
            return maxNorm;
        }

        public (double Min, double Max) CalculateReferenceValues(int
age)
        {
            double minNorm = 0, maxNorm = 0;

            if (age <= 14)
            {
                minNorm = 3.33;
                maxNorm = 5.55;
            }
            else if (age >= 18 && age <= 59)
            {
                minNorm = 4.11;
                maxNorm = 5.89;
            }
            else if (age >= 60 && age <= 90)
            {
                minNorm = 4.56;
                maxNorm = 6.38;
            }
            else if (age > 90)
```

```
            {
                minNorm = 4.16;
                maxNorm = 6.72;
            }

            return (minNorm, maxNorm);
        }
        //метод для відображення поля наявних продуктових кошиків
        public void ShowBaskets()
        {
            if (GoodBaskets.Any())
            {
                GoodBaskets.Clear();
            }
            if (GoodList.Any())
            {
                GoodList.Clear();
            }
            GoodList =
_goodBasketService.GetAllBaskets(_userService.CurrentUser);
            foreach (var b in GoodList)
            {
                if((b.OptimizationDate >= StartDate &&
b.OptimizationDate <= EndDate) ||
                    ((b.PeriodEnd >= StartDate && b.PeriodEnd <=
EndDate)))
                {
                    b.GoodInBasket =
_goodInBasketService.GetGoodInBaskets(b);
                    foreach (var g in b.GoodInBasket)
                    {
                        BasketToView basket = new BasketToView();
                        basket.GoodName =
_goodInShopService.GetGoodByShopID(g.GoodInShopID).GoodName;
                        basket.GoodAmount = g.Amount;
                        basket.Price = b.Price;
                        basket.BU = b.BU;
                        basket.Units =
_goodInShopService.GetGoodInShop(g.GoodInShopID).GoodUnits;
                        basket.BasketNO = b.id;
                        GoodBaskets.Add(basket);
                    }
                }
            }
        }
    }
    //програмний клас для реалізації елемента-легенди
    public class LegendItem : INotifyPropertyChanged
    {
        private bool isChecked;
        private string seriesTitle;
        private SolidColorBrush color;
        //властивість-стан
        public bool IsChecked
        {
            get { return isChecked; }
            set
```

```
            {
                if (isChecked != value)
                {
                    isChecked = value;
                    OnPropertyChanged(nameof(IsChecked));

                    IsCheckedChanged?.Invoke(this, EventArgs.Empty);
                }
            }
        }
        //властивість-назва серії
        public string SeriesTitle
        {
            get { return seriesTitle; }
            set
            {
                seriesTitle = value;
                OnPropertyChanged(nameof(SeriesTitle));
            }
        }
        //властивість-колір
        public SolidColorBrush Color
        {
            get { return color; }
            set
            {
                color = value;
                OnPropertyChanged(nameof(Color));
            }
        }

        private Visibility _visibility;
        //властивість-показник видимості
        public Visibility Visibility
        {
            get { return _visibility; }
            set { _visibility = value;
OnPropertyChanged(nameof(Visibility)); }
        }

        private ObservableCollection<Series> _associatedSeries;
        //список асоційованих серій з поточним елементом легенди
        public ObservableCollection<Series> AssociatedSeries
        {
            get { return _associatedSeries ?? (_associatedSeries = new
ObservableCollection<Series>()); }
            set
            {
                _associatedSeries = value;
                OnPropertyChanged(nameof(AssociatedSeries));
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(string propertyName)
        {
```

```
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }

        public event EventHandler IsCheckedChanged;
    }
}
```

ДОДАТОК Б

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота_Коваленко.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота_Коваленко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Програма_Коваленко.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Презентація_Коваленко.ppt	Презентація кваліфікаційної роботи

ДОДАТОК В

ТЕЗИ ДОПОВІДІ НА МІЖНАРОДНІЙ КОНФЕРЕНЦІЇ

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»
Ройтлінгенський університет техніки та економіки (Німеччина)
Еслінгенський університет прикладних наук (Німеччина)
Технічний університет Фрайберзька гірнича академія (Німеччина)
Університет Кобленц-Ландау (Німеччина)
Краківська гірничо-металургійна академія (Польща)
Вроцлавський технічний університет (Польща)
Дніпропетровський національний університет імені Олеся Гончара
ДКХ «Дніпровський машинобудівний завод»
ДАТ «КБ Дніпровське»

Національний технічний університет ДНІПРОВСЬКА ПОЛІТЕХНІКА 1899

ПРОБЛЕМИ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ В ОСВІТІ, НАУЦІ ТА ПРОМИСЛОВОСТІ

XVII МІЖНАРОДНА КОНФЕРЕНЦІЯ

24 листопада 2022 року
м. Дніпро

Збірник наукових праць
№ 7

Дніпро
НТУ «ДП»
2023

УДК 004.415.3:681.6

М.Г. Бердник[1], О.С. Коваленко[1]
[1]Національний технічний університет «Дніпровська політехніка», Дніпро, Україна

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ МОДЕЛЮВАННЯ ОПТИМАЛЬНОГО РАЦІОНУ ХАРЧУВАННЯ

Анотація. Результатом дослідження є створений програмний додаток для оптимізації раціону харчування хворих на цукровий діабет другого типу шляхом аналізу їх персональних даних та на основі обраного продуктового кошику.

Ключові слова: *раціон харчування, хлібна одиниця, оптимізація, програмний додаток, програмні засоби C#.*

Вступ. Пошук та аналіз наявних програмних засобів з оптимізації раціону харчування показав, що такі або не існують на сучасному ринку, або не покривають весь необхідний користувачеві функціонал. Це робить питання розробки програмного забезпечення моделювання оптимального раціону харчування досить актуальним. Розробка такого програмного забезпечення зумовлюється необхідністю пересічної людини, особливо, хворої на цукровий діабет другого типу, постійно слідкувати за своїм раціоном харчування задля недопущення перевищення норм споживання вуглеводів. Метою дослідження є створення програмного додатку мовою C# для оптимізації харчування на основі споживання хлібних одиниць.

Основний зміст роботи. Цукровий діабет – це захворювання, яке зумовлюються підвищенням рівня глюкози в крові через частковий або повний брак гормону інсуліну. Цукровий діабет може спричиняти низку інших захворювань, ураження внутрішніх органів, судин, нервової системи тощо.

Зважаючи на причини виникнення захворювання для хворих на цукровий діабет вкрай важливо мати збалансований раціон харчування, зокрема, не перевищувати допустимі норми щоденного вживання вуглеводів, адже вони найсильніше впливають на підвищення рівня цукру в крові. Розрахувати оптимальний склад продуктового кошику самостійно для багатьох людей, яким необхідно власноруч контролювати рівень цукру в крові, майже неможливо. Це робить актуальною необхідність розробки програмного засобу, який максимально спростить та пришвидшить процес таких обчислень.

Хлібна одиниця (ХО) – це штучний параметр для оцінки вмісту вуглеводів в конкретному продукті харчування. Одна хлібна одиниця рівнозначна 10 г простих вуглеводів в продукті або 12 г вуглеводів включаючи клітковину. Щоденне обмеження щодо споживання вуглеводів при цукровому діабеті будь-якого типу становить 18-24 ХО. У випадку, коли індекс маси тіла (ІМТ) людини не перевищує показник у 18,49 одиниць – тобто спостерігається недостатність ваги – щоденне обмеження споживання вуглеводів становить 18-

30 ХО. Якщо ІМТ перевищує 35 одиниць – кількість вуглеводів на день становить 10 ХО. При ІМТ вище 40 одиниць – 6-8 ХО на день.

Окрім цих рекомендацій існують обмеження споживання вуглеводів в залежності від віку та статі людини, а також від статі людини та рівня її фізичного навантаження.

Розглянемо поведінку споживача, який має намір за добу закупити продукти в кількості: $x_i,\ i = 0,1 \dots n-1$, за ціною p_i. На закупку продуктів споживач виділяє z гривень. Вказані величини повинні бути зв'язані співвідношенням:

$$\sum_{i=0}^{n-1} p_i x_i = z, z > 0$$

На суму z товари можуть бути закуплені неоднозначно. Тому з усіх варіантів закупок споживач повинен обрати найкращий.

Для цього покупець повинен обрати стратегію закупок:

$$z \to min$$

$$P_1 < \sum_{i=0}^{n-1} x_i * XO_i < P_2 \qquad 1.2)$$

де P_1 – мінімальна кількість хлібних одиниць,

P_2 – максимальна кількість хлібних одиниць,

XO_i – кількість хлібних одиниць в і-му продукті.

У якості методу оптимізації математичної моделі було обрано послідовну версію методу найменших квадратів з методології квадратичного програмування.

Послідовне квадратичне програмування – один з найбільш ефективних алгоритмів загального призначення, головною ідеєю якого є послідовне вирішення задач квадратичного програмування, апроксимуючих певну задачу оптимізації.

Програмний додаток реалізовано на основі архітектурного шаблону проектування MVVM (Model – View – ViewModel), який передбачає розмеження візуальної частини додатку від внутрішньої логіки його роботи:

1. Model – описує дані, які використовуються в додатку. Моделі можуть містити логіку валідації даних, проте не повинні визначати логіку відображення цих даних або взаємодії з елементами графічного інтерфейсу.

2. View – представлення графічного інтерфейсу, з яким взаємодіє користувач. Представлення визначає перелік та розміщення графічних елементів керування та логіку взаємодії користувача з ними за допомогою команд.

3. ViewModel – модель представлення – своєрідний посередник між моделлю та її представленням. Модель представлення призначена для переправлення даних, введених через графічний інтерфейс до моделі, та навпаки – при зміні даних в моделі – відобразити ці зміни у графічному інтерфейсі. Також модель представлення реалізує команди взаємодії з графічним інтерфейсом, наприклад, команду зміни представлення (навігацію).

Висновки. Створений програмний додаток призначений для використання в побуті людьми, які мають бажання чи яким необхідно контролювати рівень цукру в крові і надасть можливість виконувати наступні дії:

– проводити оптимізацію свого раціону харчування;

– створювати обліковий запис у системі для збереження результатів оптимізації;

– авторизуватися в системі для отримання можливості додавати користувальницькі продукти харчування та обмеження щодо їхнього споживання;

– додавати та редагувати користувальницькі продукти харчування та обмеження щодо їхнього споживання;

– переглядати список стандартних та користувальницьких продуктів та обмежень;

– додавати нові продукти до списку стандартних продуктів харчування;

– додавати нові стандартні обмеження щодо споживання продуктів харчування.

УДК 621.391.14:519

М.О. Алексєєв[1], О.М. Алексєєв[1], А.Є. Ляшенко[1]
[1]Національний технічний університет «Дніпровська політехніка», Дніпро, Україна

КОНТРОЛЬ ПАРАМЕТРІВ РОТОРНИХ ОБ'ЄКТІВ З ВИКОРИСТАННЯМ ЧАСТОТНО-ХВИЛЕВОГО СПЕКТРАЛЬНОГО АНАЛІЗУ

Анотація. В роботі пропонується метод контролю функціонального стану роторного об'єкту по частотно-хвилевому спектру звукометричного сигналу. Метод дозволяє визначити координати джерела звукометричного сигналу.

Ключові слова: *звукометричний сигнал, частотно-хвильовий спектр, координати джерел обурення.*

Вступ. При функціонуванні роторних об'єктів управління, які є складними динамічними системами, що складаються з безлічі підсистем, що взаємодіють, виникають обурювальні сили і моменти, прикладені в різних місцях об'єкта. Реакцією динамічної системи на сили, що обурюють, є