Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема Розробка сервісу для математичних розрахунків

Спеціальність 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДП.ІПЗ-02.ПЗ
(позначення)

Рецензент
проф. Кузь. М. В.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Богославець Н.В.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
проф. Кузь. М. В.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
доц. Ткачук В.М.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

Козленко М. І.
(посада) (підпис) (дата) (розшифровка підпису)

2020
(рік)

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М. І.

_____
„_____” _____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Богославець Надії Василівні
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка сервісу для математичних розрахунків
затверджена розпорядженням по факультету математики та інформатики від „25” жовтня 2019 р.№7
2. Термін здачі студентом закінченого проекту 22 травня 2020 р.
3. Вихідні дані до дипломного проекту типи математичних обчислень, зразки автоматизації обчислень, технології розробки серверної частини – .NET Core, Entity Framework Core, технології розробки клієнтської частини – HTML, TypeScript, Angular

4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
1. Аналіз предметної області та постановка задачі
2. Проектування програми
3. Реалізація поставленого завдання
4. Бізнес-план

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)

6. Дата видачі завдання 11.09.2019

Керівник _____ Ткачук В. М.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Богославець Н. В.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Аналіз предметної області та постановка задачі	02.12.2019	
2. Проектування програми	15.02.2020	
3. Реалізація поставленого завдання	17.04.2020	
4. Бізнес-план	11.05.2020	
5. Оформлення пояснювальної записки	18.05.2020	

Студент _____ Богославець Н.В.
(підпис) (розшифровка підпису)

Керівник проекту _____ Ткачук В.М.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 77 сторінок (без додатків), 60 рисунків, 29 джерел, 3 додатків на 38 сторінках.

Ключові слова: ОБЧИСЛЕННЯ, ВЕБСЕРВІС, C#, ANGULAR, ENTITY FRAMEWORK, MICROSOFT SQL SERVER.

Об’єктом дослідження є вебсервіс для виконання математичних обчислень.

Мета роботи: скпроектувати та розробити вебсервіс, що надаватиме користувачу можливість виконувати обчислення формул.

Стислий опис тексту пояснювальної записки:

Під час виконання дипломного проекту проведено аналіз предметної області та існуючих програмних продуків. Розроблено вимоги до програмного забезпечення. Проведено проектування роботи програми та бази даних. Реалізовано вебсервіс із багаторівневою клієнт-серверною архітектурою. Здійснено економічне обгрунвання розробки.

ABSTRACT

Explanatory note: 77 pages (without appendix), 60 figures, 29 references, 3 appendix on 38 pages.

Key words: CALCULATION, WEB SERVICE, C#, ANGULAR, ENTITY FRAMEWORK, MICROSOFT SQL SERVER.

Object of study: a web service for performing mathematical calculations.

Brief description of the text of the explanatory note:

During the implementation of the diploma project the analysis of the subject area and existing software products was done. Software requirements are made. The program and database were designed. A web service with a multilayered client-server architecture has been implemented. The economic substantiation of project development is carried out.

ЗМІСТ

ВСТУП

Важко уявити сферу життя людини, де не потрібно використовувати жодних математичних обчислень. Певною мірою це потрібно кожній людині: професійним працівникам в найрізноманітніших галузях, студентам, а також просто у повсякденному житті. Саме тому важливо мати доступний сервіс, який буде здатен виконувати функцію як джерела методів, методик та формул, а також в змозі виконувати розрахунок потрібних величин різноманітних категорій, що дозволить зекономить час та зусилля, а також зменшить вплив людського фактору на точність обчислень.

З вище описаного і випливає актуальність теми. Люди створювали механізми, що можуть допомогти їм із математичними обчисленнями, з давніх-давен, та постійно вдосконалювали їх. Та навіть в умовах сучасності, з безліччю приладів та програмного забезпечення, завжди існує потреба ще більш зручнішого та простішого у використанні механізму — що і є метою та головним завданням дипломної роботи. Оскільки більшість програмного забезпечення для обчислень є надто “важким” та із надлишковим, для більшості людей, функціоналом. На противагу таким програмам є чимало вебсайтів, та їх функціонал є навпаки обмеженим.

Для досягнення мети та виконання основного завдання дипломної роботи, поставлено такі завдання:

- провести аналіз предметної області;
- визначити потрібний функціонал та скласти специфікацію вимог до програмного забезпечення;
- розробити базу даних для забезпечення швидкого доступу до даних;
- створити простий інтерфейс сайту, що не потребуватиме спеціальних навичок для його освоєння;
- ретельно протестувати роботу програми.

Об’єкт дослідження: математичні обчислення.

Предмет дослідження: вебсервіс для математичних обчислень.

Роботу виконано за допомогою методів мови програмування C# та фреймворку Angular. Базу даних розроблено завдяки функціоналу наданим Entity Framework.

Практичним результатом є повноцінний вебзастосунок, який можна використовувати для обчислення різноманітних формул.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття математики та обчислень

Таке визначення, як математика, а в слід за ним і обчислення виникли ще в давні часи до нашої ери внаслідок практичних потреб людини. Їх зміст і характер з плином часу змінювались.

Поняття математики абстраговані від якісних особливостей специфічних для кожного даного кола явищ і предметів [1]. Таким чином будь-яке число не має прив'язки до певного предметного змісту. Воно може відноситися до абсолютно різних предметів і навіть абстрактних явищ. Таким же самим чином геометричні властивості фігур залишаються незмінними не залежно від матеріалу, з якого ця фігура зроблена.

Абстрагування надає математичним поняттям узагальненості, даючи можливість застосовувати математику до найрізноманітніших за природою явищ. Це означає, що одні й ті ж закономірності математики, один і той же математичний апарат можуть бути достатньо успішно застосовані до біологічних, технічних, економічних та інших процесів [1, 2].

Обчислення — процес отримання якогось певного результату чи набору результатів на основі вхідних даних, який включає як арифметичні, так і не арифметичні кроки та слідує за чітко визначеною моделлю, наприклад, алгоритмом. Виконання обчислень вивчає арифметика. Для автоматизації обчислень може використовуватись калькулятор, та інші інструменти [3].

Завдяки абстрактності математики обчислення є інструментом, що успішно застосовується в найрізноманітніших сферах життя, наук та процесів. Тому кожна людина в житті неодноразово стикається з різноманітними обчисленнями. Особливо часто зустрічаються з цим студенти та працівники технічних спеціальностей, не залежно від конкретного напрямку спеціальності.

1.2 Важливість автоматизації обчислень

Різноманітні цифри та числа оточують кожну людину із самого народження, а одним із найважливіших умінь, яких навчають кожного з народження є вміння рахувати. Для найбільш базових розрахунків людина не потребує жодних інструментів, окрім своїх знань, а саме вміння рахувати. Та ця базова навичка не обмежується використання тільки для найпростіших розрахунків.

Оскільки так склалось, що важко назвати сферу життя людини, де не потрібно використовувати жодних математичних обчислень, і обчислення оточують нас як і в повсякденному, так і професійному житті, важливо автоматизувати цей процес. Перш за все автоматизація будь-якого процесу спрощує життя людини, економить ресурси, зокрема час, та зменшує вплив людського фактору на процес і точність результату.

Автоматизація — є одним з напрямів науково-технічного прогресу, який націлений на застосування саморегульованих технічних засобів, економіко-математичних методів і систем керування, що звільняють людину від участі у процесах отримання, перетворення, передавання і використання енергії, матеріалів чи інформації, істотно зменшують міру цієї участі чи трудомісткість виконуваних операцій [4].

Основними перевагами автоматизації є:

- збільшення пропускної здатності або продуктивності;
- підвищення якості та передбачуваності якості;
- підвищена надійність, процесів або продуктів;
- підвищення узгодженості продукції;
- скорочення прямих людських витрат на робочу силу та видатків [4].

Очевидні переваги автоматизації загалом і зокрема обчислень, були зрозуміли ще здавна і люди неодноразово намагались реалізувати це та вдосконалити. Першим, найбільш раннім, із пристосувань для обчислень, найімовірніше були лічильні палички. З часом, природньо, що які і будь-які ранні

пристосування, механізми для обчислень покращувались та ускладнювались і з часом були винайдені абак, логарифмічна лінійка, механічний арифмометр, електронний комп'ютер.

В теперішній час обчислення проводяться із використанням комп’ютерної техніки та різноманітного програмного забезпечення.

Незважаючи на наявність безлічі інструментів, що допомагають у обчисленнях, пошук і створення нових рішень ніколи не зупинявся.

1.3 Аналіз наявних рішень

Математичні обчислення це не нещодавно винайдена річ і є чимало інструментів, що певним чином вирішують ті чи інші труднощі та незручності, що виникають у процесі.

Найпростішим прикладом наявного рішення для обчислень є звичайний калькулятор, який по замовчуванню встановлений на телефоні чи комп’ютері кожного користувача. Та звичайний калькулятор має певний ряд недоліків. Він зручний у використанні для одноразових обчислень, які не містять багато змінних. Тому для постійних та більш складних обчислень варто використовувати більш зручні сервіси.

Загалом сервіси для математичних обчислень можна розділити на два типи: звичайні, що підходять будь-якому користувачу та не потребують специфічних навиків для їх ефективного використання, і професійні інструменти, що надають більше можливостей і в свою чергу потребують певних умінь та ресурсів від користувача. Також їх можна розділити на настільні програми, що потребують інсталяції та онлайн сервіси — вебсайти для обчислень. В більшості настільні програми та вебсайти це сервіси різного рівня, що передбачають застосування при різних цілях та ситуаціях. Та все ж і перше, і друге — сервіси для математичних обчислень, тому варто розглянути їх всіх.

Доволі часто математичні обчислення є знаряддям для дуже складних, комплексних речей. Вони часто використовуються науковцями та певними інженерами для досить специфічних завдань. В такому випадку людям потрібні справді професійні інструменти для роботи. Такими є настільні програми як от MATLAB [5] (Рис. 1.1), Mathcad [6] (Рис. 1.2) тощо.

Такі програми пропонують великий набір функціоналу, необхідного, як вище згадано, для спеціалізованих завдань. Звичайно, в свою чергу, вони потребують значних ресурсів комп'ютера користувача. Що вже не кожний користувач може дозволити і це є значним мінусом, та все ж для простих обчислень, а не складних алгоритмічних завдань, звичайному користувачу такі програми не потрібні. Адже це не просте програмне забезпечення, яке нелегко розробити, тому ліцензія, яка необхідна для роботи програми, досить дорога.

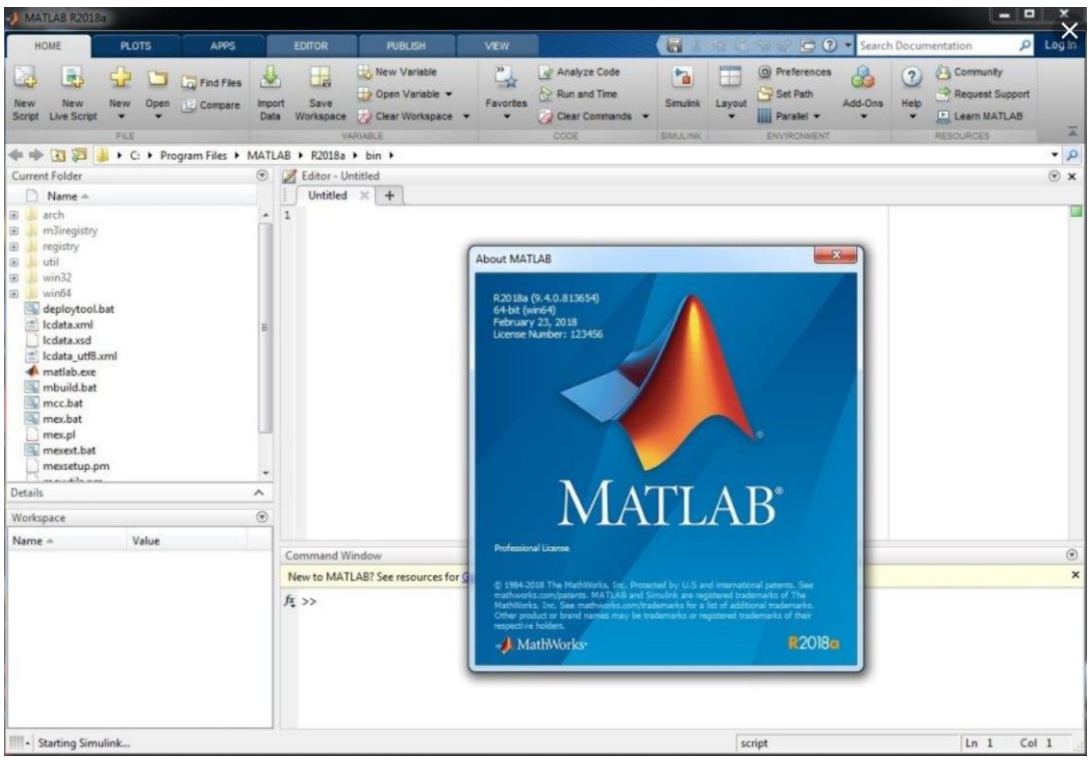


Рисунок 1.1 — MATLAB [7]

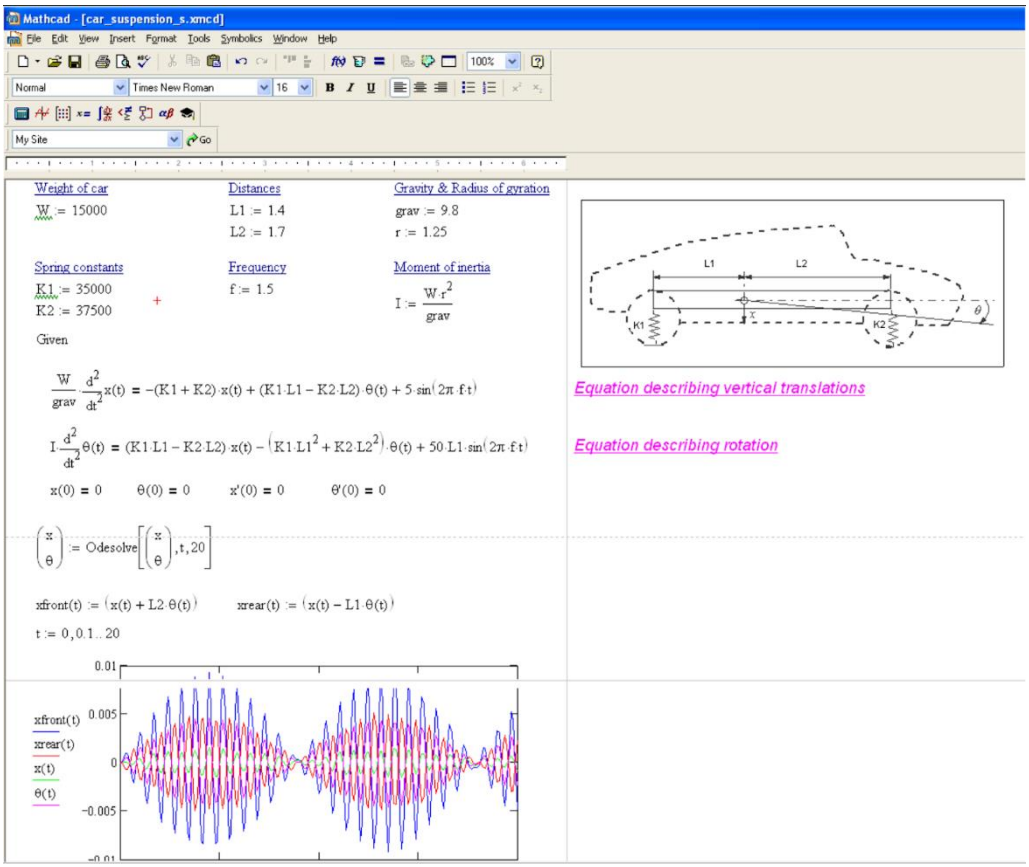


Рисунок 1.2 — Mathcad [6]

Онлайн сервіси (Рис. 1.3) все більше користуються популярністю. Вони не потребують певних мінімальних параметрів техніки, необхідних для встановлення. Користувач може отримати доступ до них просто через будь-який браузер, маючи інтернет з’єднання. Вони надають досить простий функціонал, що може знадобитись користувачу як в повсякденному житті, так і для роботи. Вони значно зручніші ніж звичайні калькулятори, та не такі складні у користуванні та без надлишкового функціоналу, як професійні програми. Вони містять перелік формул, які можна обчислити. Користувач обирає необхідну формулу, вводить параметри і отримує результат. Не потрібно знати спеціального синтаксису, щоб скористатись сервісом.

Проте більшість наявних зараз сайтів пропонують вузький список формул. І найголовнішим недоліком таких сайтів є неможливість самостійно додати шаблон необхідної формули, якої немає в списку, чи навіть створеної самостійно, якої апріорі не може бути на сайті.

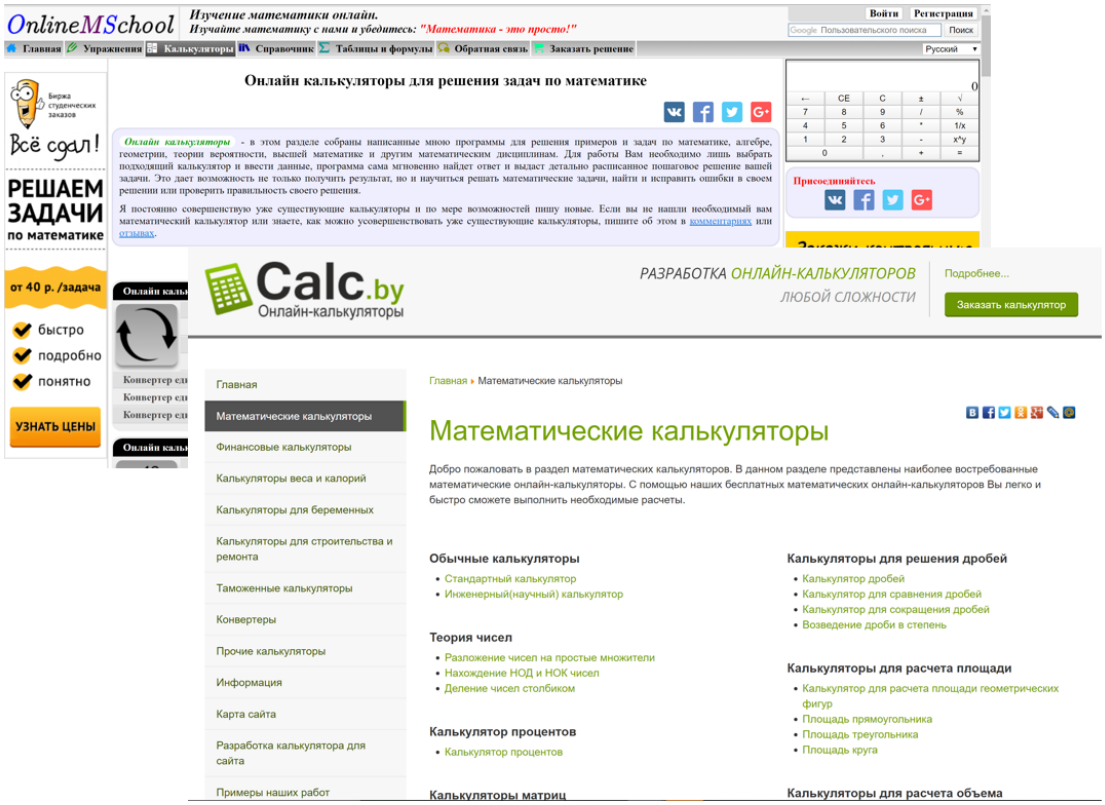


Рисунок 1.3 — Онлайн сервіси для обчислень

Отже, необхідно розробити програмне забезпечення, що перейняло би переваги на разі наявних вебсайтів та в свою чергу надавало користувачу більший функціонал та свободу дій.

1.4 Постановка задачі

Перед початком розробки необхідно чітко поставити завдання та вимоги до програмного забезпечення. Адже залежно від професійності постановки задачі залежить хід робота та кінцевий результат. Вимоги до програмного забезпечення, що розроблятиметься в межах даної роботи, є наступними.

1.4.1 Призначення

Вебзастосунок “Formula” є програмним забезпеченням для математичних обчислень, версії 1.0.

1.4.2 Угоди, прийняті в документах

Дотримання coding conventions притаманні використовуваним мовам програмування, а саме: C# та TypeScript. Використовувати не більше 100 символів в одному рядку. Обов’язкове застосування коментарів XML документації та, за необхідності, звичайні коментарі. Структуроване розбиття файлів по відповідних директоріях. Розділення front-end, back-end частин та тестування. Застосування принципів SOLID.

У дизайні використовувати кольори теми Indigo & Pink, наданої Angular Material Theme, зокрема: #3F51B5, #000000, #FFFFFF, #F44336, #9E9E9E. Використовувати шрифт тексту по замовчуванню - Times New Roman, також дозволяється використовувати сімейство шрифтів monospace.

1.4.3 Передбачувана аудиторія

Дана специфікація вимог розрахована для інженерів програмного забезпечення, front-end та back-end розробників. Документ складається з таких розділів як: введення, загальний опис, функції системи, вимоги до зовнішніх інтерфейсів та нефункціональні вимоги. Кожен розділ містить відповідні до змісту пункти для кращої структуризації інформації.

1.4.4 Границі проекту

Даний проект — сервіс для математичних розрахунків, призначений для виконання математичних обчислень, що застосовуються в різних професійних

сферах та повсякденному житті. Сервіс не має обмежень щодо віку людей, що можуть використовувати його, та жодних інших обмежень. Основна мова інтерфейсу — англійська.

Розробка програмного забезпечення призначення для демонстрації отриманих знань протягом здобуття освітнього рівня Бакалавр та задоволення власних та ринкових потреб.

1.4.5 Загальний погляд на продукт

Даний вебсервіс є продуктом програмного забезпечення, що дозволяє користувачам виконувати обчислення, самостійно додавати власні формули та поширювати її серед інших користувачів. Надає можливість виконувати формули, що містять в собі арифметичні операції, вбудовані функції та формули з бази даних.

1.4.6 Особливості продукту

Особливістю продукту є зручний доступ через будь-який браузер чи то на телефоні, чи на комп'ютері. Розширення бази даних користувачами, що забезпечує широкий вибір загальних та унікальних формул.

Основними функціями вебсервісу є:

- реєстрація та авторизація;
- перегляд публічних списків категорій з формулами;
- додавання власної формули та операції з нею;
- додавання користувацьких формул до публічних колекцій;
- додавання до закладок;
- перегляд історії розрахунків та збереження важливих результатів;
- забезпечення функцій адміністрування сайту.

1.4.7 Класи і характеристики користувачів

Передбачено такі класи користувачів, кожен з яких успадковує можливості попереднього:

- гість — перегляд публічних категорій та формул, перегляд інформації про обрану формулу, виконання обчислення та отримання результату, реєстрація та вхід на сайті;
- користувач — додавання власної формули, оперування власними формулами, запит до адміністратора на поширення формули, додавання до закладок, перегляд історії розрахунків та збереження важливих результатів;
- модератор — підтвердження запитів на додавання формули до публічних колекцій;
- адміністратор — створення модератора та адміністратора.

1.4.8 Операційне середовище

Апаратними засобами для використання сервісу є будь-який пристрій з встановленим браузером та доступом до інтернету. Дані зберігаються у базі даних MS Sql Server, авторизаційному JSON Web Token та cookie.

1.4.9 Обмеження дизайну та реалізації

При розробці програмного забезпечення використовуються компоненти бібліотеки Angular Material, що відповідають вимогам дизайну Material. Усі іконки та значки надаються Material Icon в форматі SVG, що забезпечує відмінну якість зображення при різних розширеннях екранів, а також не навантажує систему. Необхідно уникати використання іконок з інших джерел, щоб підтримувати однорідність дизайну. Кольори, використані в дизайні — в межах Angular Material Theme Indigo & Pink.

Розробка виконується за допомогою методів мови програмування C# та фреймворку Angular. Базу даних розроблено завдяки функціоналу наданим Entity Framework. Для керування версіями, збору даних, відстеження станів та тестування використано Azure DevOps Server.

Апаратними засобами для використання продукту є пристрій з будь-яким браузером та інтернет з'єднанням.

1.4.10 Документація для користувачів

Користувачі повинні бути забезпечені такою документацією:

- інструкція з використання;
- політика конфіденційності.

1.4.11 Припущення і залежності

Можливе некоректне відображення даних на застарілих браузерах, а також незначні відмінності у вигляді в залежності від розміру та розширення екрану пристрою.

1.4.12 Опис і пріоритети

Продукт програмного забезпечення розробляється з використанням основних принципів об'єктно-орієнтованого програмування та SOLID принципів. Дані користувача зберігаються в базі даних, захищений доступ до яких надається через JWT, стандартизованому в RFC 7519.

Першим етапом в реалізації програми є розробка алгоритму валідації даних, обчислення формули, можливість обчислення вкладених формул. Розробка

реєстрації та авторизації користувача. Наступним етапом - реалізація основний функцій користувача та адміністратора.

1.4.13 Послідовність «вплив-реакції»

Функціонал програми передбачає чітку послідовність “вплив-реакція” при натисканні кнопок та переході з однієї сторінки на іншу. Інтерфейс користувача дозволяє інтуїтивно зрозуміти дії, необхідні для досягнення цілі. Програма реагує на натиски мишкою та клавіш на клавіатурі.

1.4.14 Функціональні вимоги

Програма повинна чітко виконувати зазначені дії при взаємодії з відповідною областю. Валідація введених даних та захист особистих даних користувача. Наявність можливості зручно робити обчислювати та самостійно додавати або створювати шаблон формули. Можливість використовувати вкладені формули та базові математичні функції. Забезпечення унікальності імені користувача та електронної пошти.

1.4.15 Інтерфейси користувача

Дизайн інтерфейсу користувача розроблено відповідно до міжнародних вимог та стандартів, а також кольорових схем, що забезпечують адекватне сприйняття усіх кольорів, відтінків та їх поєднання. Дотримано правила зручного і легкого використання зі зрозумілим для користувача інтерфейсом та UX-сценаріями. Виведення підказок при вводі даних та сповіщень при необхідності.

1.4.16 Програмні інтерфейси

Програмні інтерфейси для реалізації додатку:

- JSON;
- XML;
- .Net Core 3.1;
- Angular 9;
- EF Core 6.

1.4.17 Інтерфейси обладнання

Інтерфейси обладнання відповідають інтерфейсам мобільних та комп'ютерних пристроїв.

1.4.18 Інтерфейси зв'язку і комунікацій

Комунікація із користувачем здійснюється шляхом сповіщень, написів на кожній із сторінок та підказок біля полів введення даних. А також шляхом інтуїтивно зрозумілих зображень та іконок. Також на головній сторінці є короткий опис-інструкція, щоб зорієнтувати користувача на початку використання даного програмного забезпечення.

1.4.19 Вимоги до продуктивності

Продуктивність вебзастосунку визначатиметься через швидкість відповіді на запит користувача, що повинна здійснюватися у межах 600 мс при хорошому інтернет-з’єднанні та у межах 2 с при поганому інтернет-з’єднанні.

1.4.20 Вимоги збереженості даних

Дані користувача зберігаються у базі даних MS Sql Server, доступ до який здійснюється через авторизаційний JSON Web Token з використанням RS256 Signatures. Перевірка коректності введених даних перед занесенням до бази. Аналіз

унікальності даних про користувача. Наявність захисту від виникнення помилок при отриманні даних з БД та запису нових у БД. Унеможливлення видалення одного елементу, якщо він пов’язаний з іншим. Коректність відображення даних відповідного користувача при багато користувальницькому режимі.

1.4.21 Критерії якості програмного забезпечення

Програмне забезпечення повинне відповідати вимогам, зазначеним у даній специфікації, розробленому дизайну, а також таким критеріям якості [8, 9], відповідно до стандарту ДСТУ ISO/IEC TR 9126-2:2008, як:

- надійність;
- ефективність;
- зручність;
- функційність;
- супроводжуваність;
- переносність.

1.4.22 Вимоги до безпеки системи

Програмний код, а в особливості, код серверної частини не повинен бути доступним користувачеві. Усі важливі обчислення та обробка даних повинна відбуватись на серверній частині програми.

Будь-які дані облікового запису користувача не повинні бути у відкритому доступі та не можуть бути переглянуті сторонніми системами.

При отриманні дані користувача хешуються для уникнення витоку приватної інформації власників облікових записів. Усі змінні та методи повинні бути захищені від стороннього доступу та модифікації [10].

Звичайний користувач або гість не має можливості видалити записи, що занесені адміністратором. Користувач може видалити тільки власні записи у персональних колекціях.

Адміністратор чи модератор не мають можливості видаляти персональні дані користувача.

Реалізація поставленого завдання відбуватиметься відповідно до життєвого циклу програмного забезпечення: визначення завдання, проектування, реалізація, тестування та підтримка [11].

2 ПРОЕКТУВАННЯ ПРОГРАМИ

2.1 Стани програми та переходи між ними

Будь-яка програма зводиться до станів та переходів між ними. При написанні програми потрібно мати чітке розуміння того, які стани повинні бути в програми та як повинен відбуватися перехід між ними.

Отже, для початку необхідно змоделювати роботу програми у вигляді діаграми (Рис. 2.1).

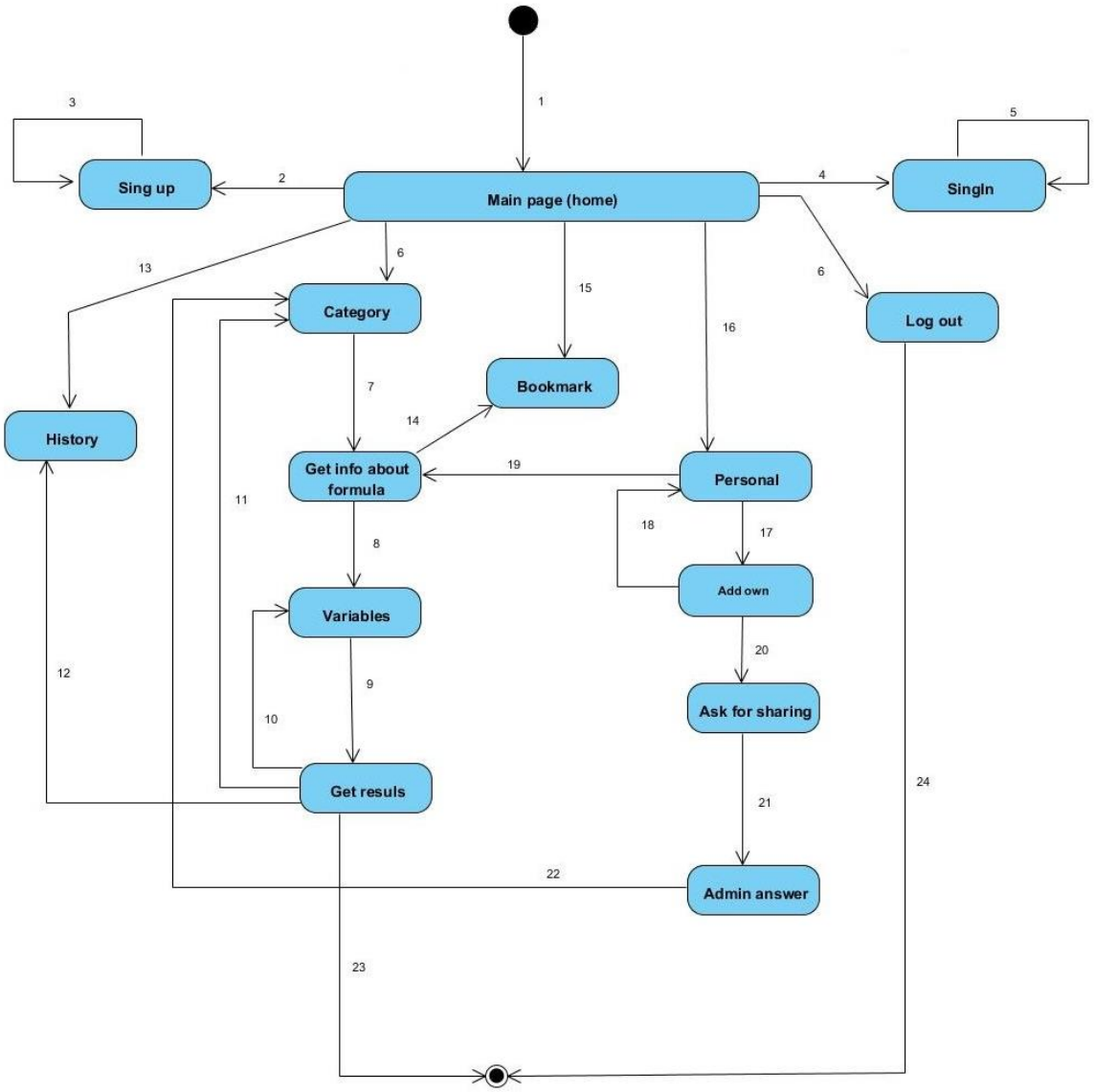


Рисунок 2.1 — Діаграма станів програми

Діаграма станів містить в собі:

- коло — початковий стан програми;
- прямокутник, кути якого заокруглені, — символізує стан програми;
- стрілка — вказує на перехід між станами;
- коло із вписаними колом — вказує на кінцевий стан програми [12].

Як тільки користувач зайшов на вебсайт він опиняється на головній або так званій домашній сторінці програми (1). Саме з цієї сторінки розпочинається робота та переходи до інших станів програми, тобто інших сторінок вебсайту. З будь-якого стану програми користувач може перейти назад на головну сторінку.

З головної сторінки користувачу доступний перехід до реєстрації (2). Для реєстрації користувач повинен ввести певні дані, а саме:

- ім’я користувача;
- електронна адреса;
- пароль;
- підтвердження паролю.

Важливо забезпечити унікальність користувачів, тому якщо у базі даних користувач з введеним іменем чи електронною адресою вже існує — користувачу потрібно спробувати зареєструватись знову (3). Також важливо проводити перевірку введених даних одразу при їх введенні, щоб пришвидшити процес обробки даних і в разі помилки повідомити користувача про це одразу. Ім’я користувача повинно міститись в межах мінімальної та максимальної кількості символів. Електронна адреса повинна відповідати шаблону електронної адреси. Пароль повинен бути «міцним», тобто відповідати високому рівню надійності і містити в собі як мінімум:

- одну велику літеру латинського алфавіту;
- малу літеру латинського алфавіту;
- цифру;
- спеціальний символ.

В сумі кількість символів паролю повинна бути рівна восьми або більше. Звісно пароль необхідно підтвердити, зробити це користувач може просто ввівши його знову.

Якщо користувач уже зареєстрований він може увійти у свій обліковий запис (4). Для входу потрібно ввести дані, які були вказані при реєстрації, а саме: ім’я користувача та пароль. При невдалій спробі увійти на сайт, що може трапитись при неправильно введених даних, користувач повинен спробувати ввести дані і увійти знову (5). Також можлива ситуація, що користувач намагається увійти на сайт, хоча насправді не зареєструвався на ньому. Тоді йому необхідно повернутись на головну сторінку і спочатку все ж таки зареєструватись (2).

Бути зареєстрованим на сайті не обов’язково, користувач може скористатись однією з основних функцій без реєстрації чи входу на вебсайт. Звісно ж, в такому випадку весь спектр можливостей не буде розкрито, але обов’язкове створення облікового запису може відлякувати користувачів. Людина повинна мати можливість оцінити певні функції застосунку і тоді вирішити чи потрібно їй більше.

Одна з основних функцій сервісу — обчислення формул, є доступною незалежно від того чи користувач зареєстрований і чи увійшов він на сайт. Розпочинається все з перегляду списку категорій та формул, що в них містяться (6). При перегляді користувач знаходить необхідну для обчислення формулу та може перейти на сторінку, що містить інформацію про конкретно обрану формулу (7). Програма повинна містити такі дані про формулу, як:

- заголовок;
- ім’я;
- вигляд;
- змінні;
- опис;
- категорія.

Заголовок — назва формули, що може складатись з декількох слів та відображає суть формули.

Ім’я формули — назва формули, що може містити в собі літери та цифри, при цьому розпочинатись вона повинна із літери. Така назва формули необхідна для того, щоб забезпечити можливість виклику однієї формули в іншій. Тобто реалізувати виконання обчислення формули, що в містить собі іншу формулу. Також при створенні власної формули користувач може вказати ім’я будь-якої формули, що вже є на сайті, та використати її як змінну у власній.

Вигляд формули це і є власне сама формула те, як вона виглядає і з яких змінних, операцій та інших формул вона складається.

Змінні — перелік невідомих, не постійних змінних у формулі. Тобто ті змінні, що потрібно ввести перед обчисленням.

Опис — додаткова інформація про формулу. Може містити, наприклад, деталі про елементи формули, що є результатом формули тощо.

Категорія — назва категорії до якої відноситься формула.

Отримавши інформацію про обрану користувачем формулу та переконавшись, що це саме те, що йому потрібно обчислити, він може перейти до введення змінних у формулі (8). В більшості значення змінних це певні числа. Але також користувач може ввести такі сутності замість невідомих змінних:

- число;
- константу, що є в переліку постійних значень на сайті;
- виклик іншої формули, результат якої буде значенням змінної.

Після введення необхідних даних відбувається обчислення результату та Отримання результату користувачем (9). За необхідності користувач може знову обчислити ту ж саму формулу (10) або обрати іншу зі списку (11). Після проведення необхідних обчислень та отримання потрібних результатів користувач може завершити роботу програми (23).

Для користувача, що є зареєстрованим та здійснив вхід в обліковий запис на вебсайті, доступний більший функціонал.

При перегляді інформації про обрану формулу, користувач може додати її в закладки (14). Звісно ж, користувач може перейти до списку зі його закладками одразу із головної сторінки (15). Користувач може управляти записами у своїх закладках, тобто за умови непотрібності формули, користувач може видалити її із закладок.

Після отримання результату має можливість зберегти необхідний результат до історії (12). Повністю вся історія обчислень не зберігається, адже це призведе до великого накопичення записів у базі даних. І загалом зберігати абсолютно всі розрахунки не потрібно і користувачу, важливо надати можливість зберегти найбільш необхідне. Відповідно до роботи зі закладками, користувач може перейти до перегляду історії обчислень одразу із домашньої сторінки (13) і так же само користувач може управляти записами в історії і видаляти надалі непотрібні результати обчислень.

Ще однією базовою функцією сервісу є можливість додавати власні шаблони формул і обчислювати їх в майбутньому. Ця функція доступна тільки для зареєстрованих користувачів, після авторизації на сайті. З домашньої сторінки сайту користувач може перейти до сторінки «personal» (16). На ній відображається перелік формул, які користувач додав особисто або і зовсім придумав їх самостійно.

Для користувача доступна можливість додати необмежену кількість персональних формул (17). Для цього йому необхідно ввести всю інформацію про формулу.

Для спрощення процесу та зменшення ймовірності виникнення помилок, для користувач буде бачити підказки відносно того, що він вводить. При неправильному введенні даних, до прикладу, додаванні пробілу в імені формули, користувачу буде виведено повідомлення про допущену помилку у введені даних. Також для недопущення конфлікту при виклику формули, необхідно переконатись, що ім’я формули унікальне і при спробі створити ще одну із таким же іменем — повідомити користувача.

Після додавання власних формул користувач може отримати всю інформацію про будь-яку із них (19), ввести дані та обчислити її за тим же алгоритмом дій, що й із загальними формулами (8 - 10).

Значним плюсом цієї функції є те, що такі особисті формули користувач може поширювати поміж всіх інших користувачів, поповнюючи загальну базу формул. Звісно, процес поширення власних записів повинен контролюватись модератором чи адміністратором сайту для уникнення поширення невідповідного контенту.

Щоб поширити власноруч додану формулу потрібно відправити запит на її поширення та очікувати відповіді (20). Можливі два варіанти відповіді:

- прийняття — тоді формула додається до загального списку, відповідно до категорії (22). Після цього будь-хто може використовувати цей шаблон та виконувати обчислення;
- відхилення — якщо адміністратор чи модератор сайту не підтвердив запит на поширення формули, вона не додається в загальний список, але надалі залишається в списку особистих формул користувача.

Користувач, який виконав вхід на сайті може вийти із облікового запису і так завершити роботу із програмою (24).

Також коли користувач здійснив вхід на сайті він має доступ до свого облікового запису, де може побачити основну інформацію про свій обліковий запис та налаштування, де він може змінити свій пароль.

Щоб змінити пароль необхідно ввести старий пароль та двічі ввести новий пароль. Як і при реєстрації користувальницький пароль повинен бути високого рівня надійності.

Для кращого розуміння будь-якого класу користувачів як користуватись сервісом, як виконувати обчислення, що він може вводити на місцях змінних та як додавати власну функцію, він матиме доступ до інформативної сторінки, на якій описана інструкція з використання.

На ній же буде зазначено список операторів, built-in функцій та список константних значень.

2.2 Базові функції програми

Програмний продукт, що розробляється є новим, тобто це перша його версія. При запуску програми необхідно максимально реалізувати всі поставлені задачі, та в процесі підтримки програмного забезпечення завжди можна додати щось нове, вдосконалювати вже наявний функціонал, оновити дизайн і загалом програму. Важливо не намагатись зробити ідеальний продукт одразу, а поступово вдосконалювати його.

Та все ж у кожній програмі є якась основа, якісь базові функції, без яких втрачається концепт. Саме їм варто приділити найбільше уваги при проектуванні, розробці та тестуванні програми.

Діаграма послідовностей для базових функцій допомагає детально зобразити взаємозв’язок та послідовність взаємодії між різними об’єктами [13]. Моделювання виконання дії за допомогою таких діаграм допомагає краще продумати алгоритм для їх реалізації.

Базовими функціями даного програмного продукту є власне обчислення та додавання власних формул користувачами, що забезпечить всеохоплюючу загальну базу даних. Діаграма послідовностей для обчислення зображена на рисунку 2.2.

На діаграмі можна побачити наступні об’єкти:

- користувач;
- програма;
- база даних.

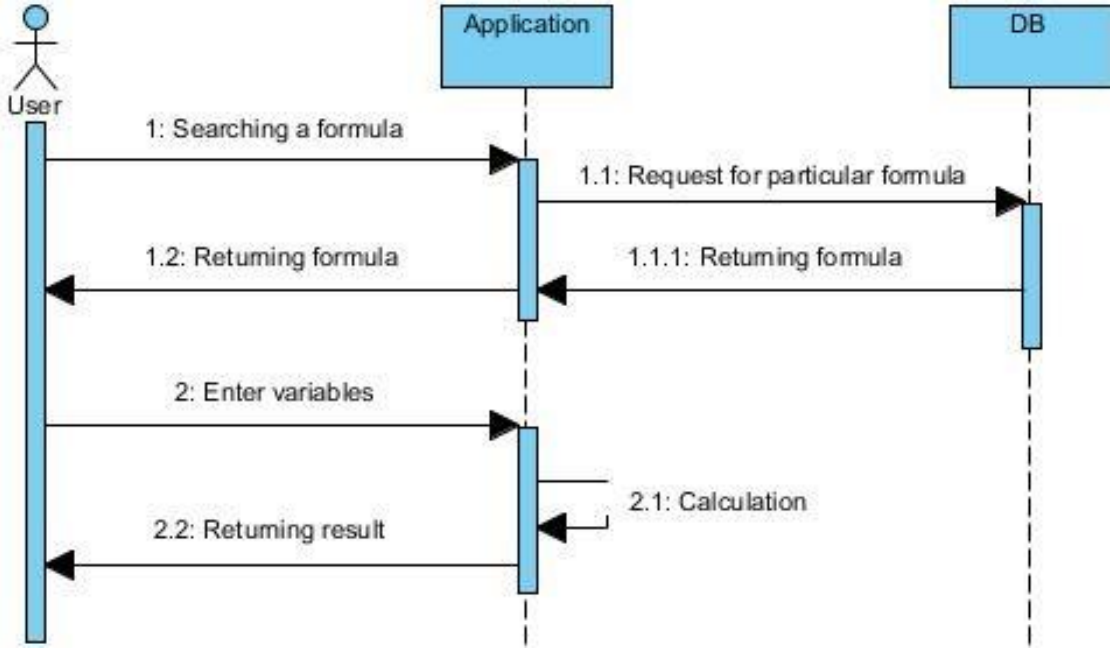


Рисунок 2.2 — Обчислення

Спочатку користувач шукає що саме йому необхідно обчислити, надсилаючи запит програмі. Програма в свою чергу надсилає запит до бази даних. База даних відповідь на запит, повертаючи необхідну формулу.

Наступний крок користувача — введення змінних. Після чого програма виконує обчислення, що включає декілька етапів (Рис. 2.3).

Кожен з етапів виконує певну функцію:

- валідація — перевірка правильності формули;
- заміна — заміна невідомих змінних у формулі значеннями, введеними користувачем;
- додавання нуля перед унарною операцією — щоб не було конфлікту бінарної та унарної операцій, що позначаються одинаковими символами, необхідно замінити додати нуль перед унарною операцією, що перетворить її на бінарну і не змінити результат обчислення;
- формування масиву — оскільки формувала приходить у вигляді рядка, потрібно перетворити її в масив, кожен елемент якого буде окремим елементом формули;

– парсинг — на даному етапі проводиться перетворення звичного запису формули в зворотній польський запис [14];
– обчислення — проводяться операції з числами та обчислення результату.

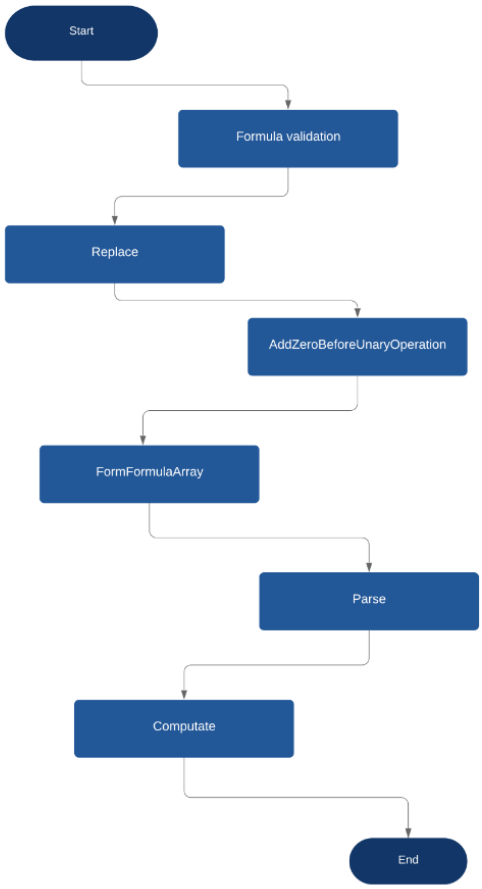


Рисунок 2.3 — Етапи обчислення

Завершальним етапом на діаграмі є відповідь програми користувачу — виведення результату обчислення.

На рисунку 2.4 можна побачити діаграму послідовностей, що відображає послідовність подій при створенні та поширенні формули.

У діаграмі беруть участь наступні об’єкти:

– користувач;
– програма;
– база даних;

– адміністратор або модератор сайту.

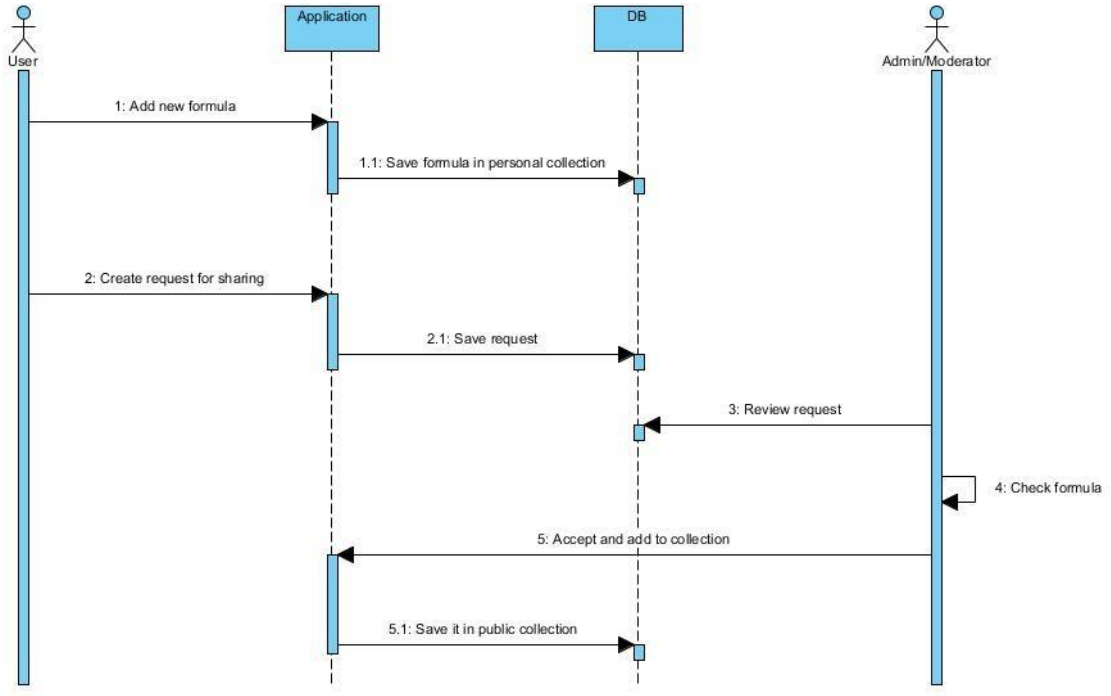


Рисунок 2.4 — Створення та поширення формули користувачем

Першим кроком є додавання користувачем формули. Додавання відбувається через заповнення користувачем форми, де він вказує всі необхідні дані. Програма обробляє та направляє формулу на зберігання до бази даних до списку персональних формул користувача.

Другий крок — користувач створює запит на поширення формули. Цей запит, як і у випадку з формулою, база даних обробляє та зберігає у базі даних.

Час від часу модератори та адміністратори сайту повинні переглядати список запитів користувачів та перевіряти їх — крок три та чотири відповідно.

При успішно пройденій перевірці, адміністратор додає цю формулу до загальної колекції формул і програма зберігає їх і базі даних.

Після цього будь-який користувач може використати додану формулу для обчислення за алгоритмом попередньої діаграми послідовностей.

2.3 Проектування бази даних

При потребі зберігати дані важливо зберігати їх правильно, щоб програма могла максимально ефективно працювати із ними. ER-модель, за якою пізніше буде реалізована база даних, зображена на рисунку 2.5.

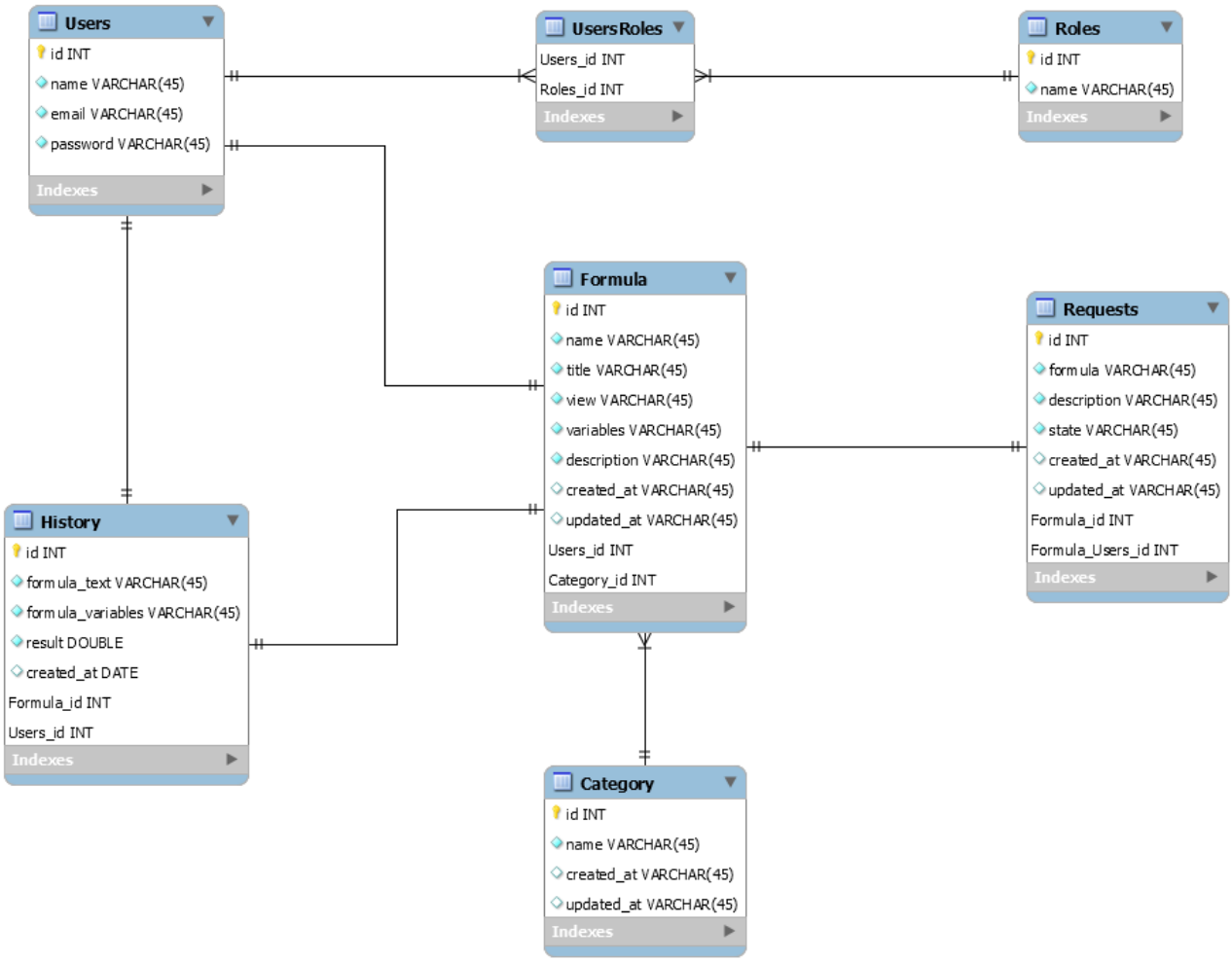


Рисунок 2.5 — ER-модель

Модель відображає основні сутності програми, інформацію про які необхідно зберігати в базі даних, і те, як вони пов’язані між собою [15]:

– формули;
– категорії;

– історія;
– запити;
– користувачі;
– ролі;
– зв’язок ролей та користувачів.

Кожна таблиця неодмінно містить поле, що являється первинним ключем, який обов’язково є унікальним в межах однієї таблиці. Він необхідний для однозначного зв’язку та ідентифікації записів. Зазвичай тип цього ключа є цілим числом, адже операції із числами відбуваються швидше, ніж операції з рядками. Нижче можна побачити детальніше вміст кожної із таблиць, окрім первинного ключа.

Таблиця «Formula», як і планувалось на попередніх етапах моделювання програми, містить такі дані:

– ім’я;
– заголовок;
– вигляд;
– змінні;
– опис;
– дату створення;
– дату оновлення;
– ідентифікатор автора;
– ідентифікатор категорії.

Ім’я повинно унікальним у всіх формул. Не допускаються значення null у полях. Таблиця пов’язана із таблицями користувачів та категорій.

Таблиця «Category»:

– назву;
– дату створення;
– дату оновлення.

Щоб отримати список формул певної категорії необхідно звернутись до таблиці із ними за первинним ключем таблиці категорій та вторинним ключем category_id у таблиці зі формулами.

Таблиця «History» пов’язана із таблицею користувачів та формулами і містить наступні дані:

- текст формули;
- змінні;
- результат;
- дата створення;
- ідентифікатор формули;
- ідентифікатор користувача.

Таблиця «Requests» також пов’язана із формулою та користувачем та містить наступне:

- формула;
- опис;
- стан;
- дату створення;
- дату оновлення;

Таблиця «Users»:

- ім’я;
- електронну адресу;
- пароль.

Таблиця «Roles» містить назву ролі. Між користувачем і роллю зв’язок типу багато-до-багатьох, для відображення якого створюється проміжна таблиця «UsersRoles».

2.4 Аналіз поставленої задачі та user story

Маючи чітко поставлену задачу та вимоги до програмного забезпечення, потрібно розбити її на більш дрібні та конкретні завдання. Для ефективної роботи і роботи із досить великий проектом необхідно використовувати такі сервіси для розробників, як наприклад, Azure DevOps Server [16, 17]. Він дозволяє не тільки керувати версіями програми та виконувати збір даних, але й дозволяє планувати хід розробки, носити вимоги до програмного продукту та розбивати більші завдання на дрібніші і відслідковувати прогрес реалізації проекту [18].

На основі поставлених вимог до програмного забезпечення було складено такі основні елементи product backlog, що зображені на рисунку 2.6.

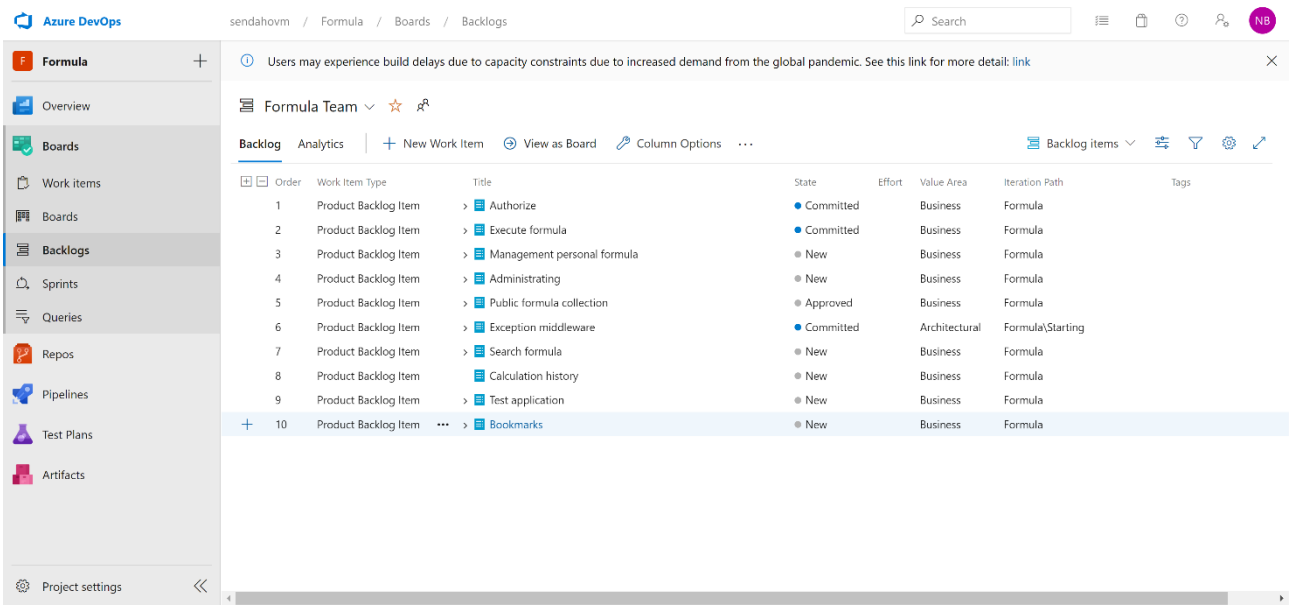


Рисунок 2.6 — Product backlog

Після зазначення в беклогах головних функцій необхідно додати до них конкретні, неабстрактні завдання та поступово імплементувати їх. Також до завдань варто додавати user story, що відображають бажання користувача з потенційних класів користувачів. Це дозволяє краще зрозуміти як саме потрібно реалізувати те чи інше завдання. Приклад user story можна побачити на рисунку 2.7.

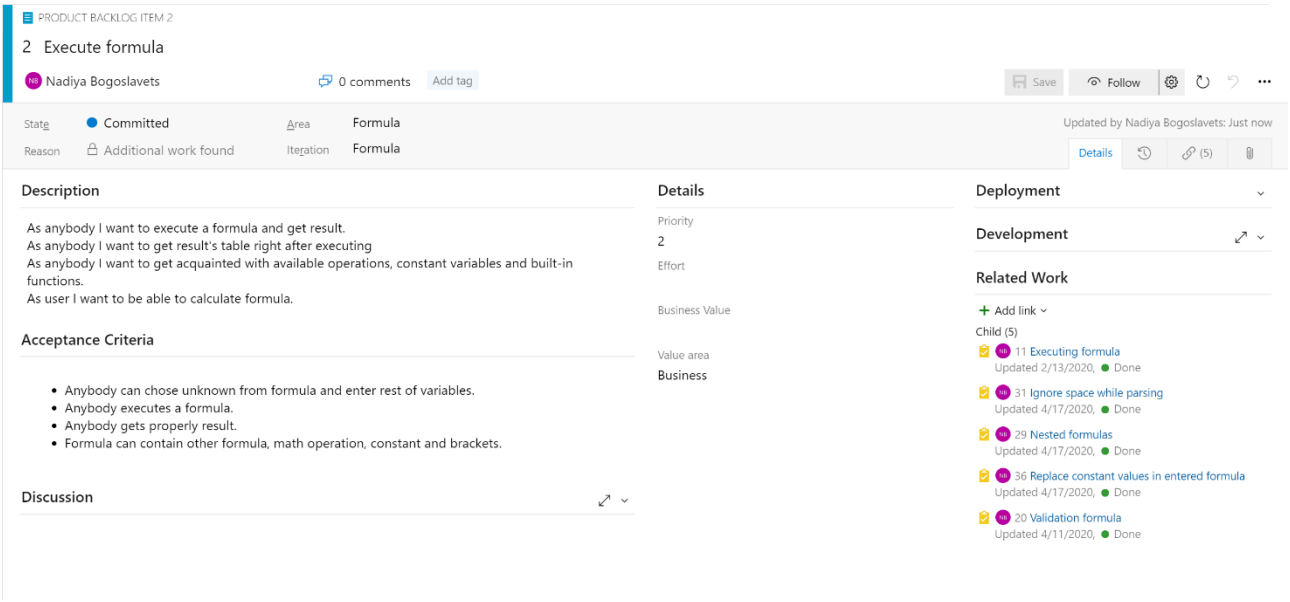


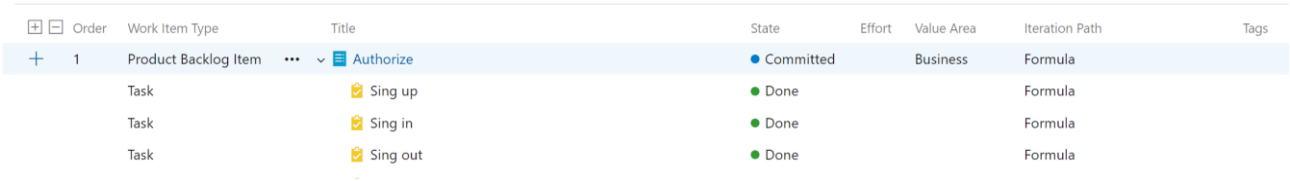
Рисунок 2.7 — Приклад user story

В описі додано тези, що відображають що саме, які функції необхідні для користувачів, що можуть бути в будь-якому класі користувачі: незареєстровані користувачі, так звані гості, зареєстровані користувачі чи адміністратори. Також додано тези про необхідні функції, що доступні при певних умовах.

В межах розробки даного програмного продукту створено такі основні завдання, приклади яких можна побачити на рисунках 2.8 — 2.12.

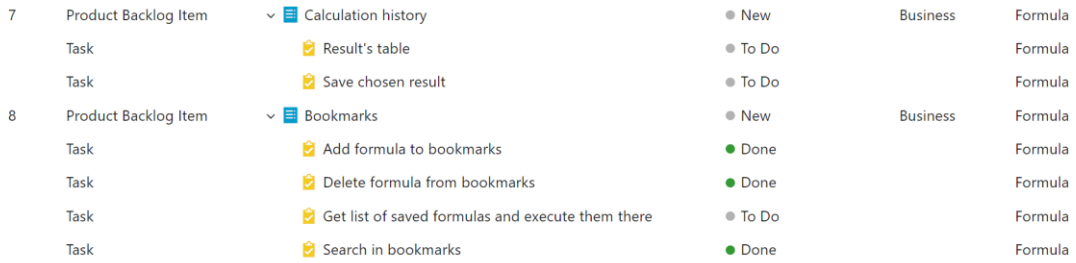
2	Product Backlog Item	Execute formula	Committed	Business	Formula
	Task	Executing formula	Done		Formula
	Task	Validation formula	Done		Formula
	Task	Validation exception	Done		Formula
	Task	Nested formulas	Done		Formula
	Task	Ignore space while parsing	Done		Formula
	Task	Replace constant values in entered formula	Done		Formula
3	Product Backlog Item	Management personal formula	New	Business	Formula
	Task	Create/view/edit/delete formula	Done		Formula
	Task	Share formula	Done		Formula
	Task	Request to admin	To Do		Formula
4	Product Backlog Item	Public formula collection	Approved	Business	Formula
	Task	Admin create/view/edit/delete public formula	Done		Formula
5	Product Backlog Item	Search formula	New	Business	Formula
	Task	Search	To Do		Formula

Рисунок 2.8 — Завдання для забезпечення функції обчислення



Order	Work Item Type	Title	State	Effort	Value Area	Iteration Path	Tags
1	Product Backlog Item	Authorize	Committed		Business	Formula	
	Task	Sing up	Done			Formula	
	Task	Sing in	Done			Formula	
	Task	Sing out	Done			Formula	

Рисунок 2.9 — Завдання авторизації



7	Product Backlog Item	Calculation history	New	Business	Formula
	Task	Result's table	To Do		Formula
	Task	Save chosen result	To Do		Formula
8	Product Backlog Item	Bookmarks	New	Business	Formula
	Task	Add formula to bookmarks	Done		Formula
	Task	Delete formula from bookmarks	Done		Formula
	Task	Get list of saved formulas and execute them there	To Do		Formula
	Task	Search in bookmarks	Done		Formula

Рисунок 2.10 — Закладки та історія

10	Product Backlog Item	Test application	New	Business	Formula
	Task	Test validation service	Done		Formula
	Task	Test computation service	Done		Formula
	Task	Test built-in functions	Done		Formula

Рисунок 2.11 — Завдання для тестування процесу обчислення

Тестування аплікації є необхідною частиною при розробці програмного забезпечення. Рівень покриття коду тестами повинен бути не нижчим ніж 80%. Найважливішою частиною програми, що потребує ретельного тестування, є ті класи, сервіси та функції, що напряму стосуються обчислення. Такими є обчислення вбудованих функцій, валідація введеної формули і власне алгоритм обчислення.

9	Product Backlog Item	Exception middleware	Committed	Architectural	Formula\Starting
	Task	Add implementation for exception middleware	Done		Formula\Starting

Рисунок 2.12 — Архітектурні завдання

Додавати завдання можна не тільки на основі бізнес логіки та основних функцій, але і завдання з точки зору архітектури та features, додаткові особливі функції. В ході роботи, безумовно, додаються нові завдання, тим паче такі, що пов’язані з технічними питаннями, покращеннями ефективності програми тощо. Також до завдання можна додавати пріоритеті, тип завдання та зв’язність завдання між собою.

Структурованість та чіткість завдань допомагає розробити кращі рішення для поставлених завдань, не пропустити жодного важливого завдання та завершити розробку в необхідні терміни.

3 РЕАЛІЗАЦІЯ ПОСТАВЛЕНОГО ЗАВДАННЯ

3.1 Структура програми

Програму реалізовано за допомогою методів мови програмування C# та TypeScript. Серверну частину програми створено на базі фрейморку Asp .Net Core 3.1 [19, 20], а користувальницьку — на базі Angular 8 [21]. Модульне тестування програми на базі xUnit додані окремим проектом. Структура проекту зображена на рисунку 3.1.

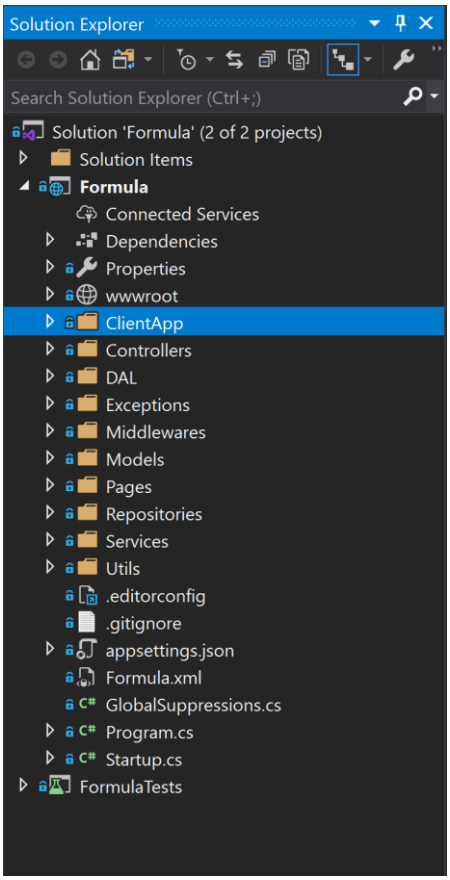


Рисунок 3.1 — Структура проекту

Директорія «FormulaTests» містить модульні тести програми. Директорія «Formula» це і є власне проект, що містить в собі серверну, клієнтську частини, а також конфігурації і залежності.

Залежності проекту це сторонні модулі, пакети, що необхідні для розробки чи спрощують її (Рис. 3.2).

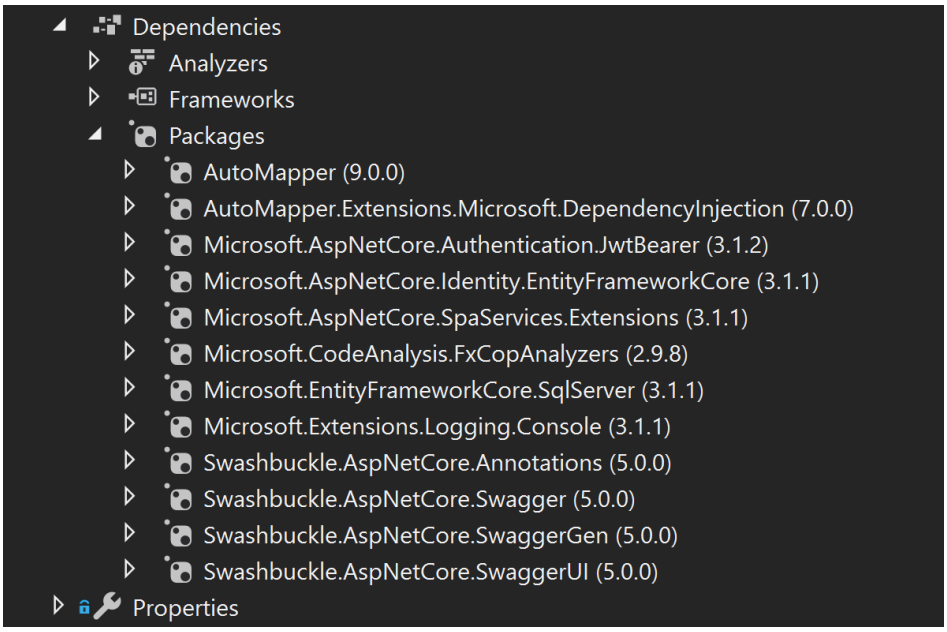


Рисунок 3.2 — Модулі, використані при розробці

Основними складовими серверної частини програми є:

- ClientApp — клієнтська частина аплікації;
- Controllers — містить контролери програми, що відповідають за маршрутизацію запитів, що приходять із клієнтської частини;
- DAL — містить моделі, контекст та файл ініціалізації бази даних;
- Exeptions — містить класи специфічних для програми виключень;
- Middlewares — містить проміжний обробник виключень при опрацюванні запиту;
- Models — проміжні моделі між клієнтською та серверною частинами;
- Repositories — класи, через які здійснюється доступ до бази даних;
- Services — класи, які містять основну логіку серверної частини аплікації;

– Utils — містить допоміжні класи, що відповідають за макетні перетворення моделей, константні значення, містять список ролей та загальні розширення методів класів;
– Startup.cs — міститься логіку запуску програми. В ньому відбувається ініціалізація важливих складових програми, реєстрація сервісів та задаються необхідні конфігурації.

Клієнтська частина аплікації також чітко структурована, побачити це можна на рисунку 3.3.

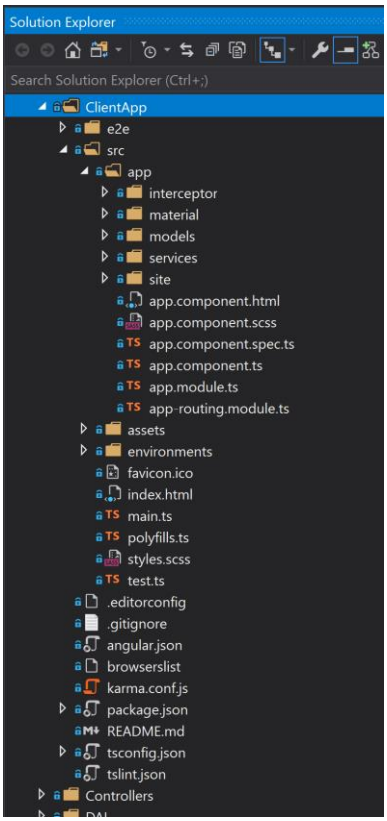


Рисунок 3.3 — Структура клієнтської частини програми

Основними складовими клієнтської частини програми є:

– e2e — містить тести інтерфейсу вебзастосунку;
– interceptor — включає json web token до кожного, надісланого до серверу, запиту;

- material — описує компоненти Material, що використовуються у програмі;
- models — моделі, що заповнюються введеними користувачем даними, та передаються на серверну частину аплікації;
- services — містить класи з основною логікою клієнтської частини, а також з'єднання з логікою серверної частини;
- site — містить компоненти вигляду сторінки, їх html, scss та tf файли;
- app.component.* — файли корінного компонента аплікації;
- app.module.ts — файл, в якому необхідно вказати всі сервіси та компоненти аплікації;
- asserts — статичні ресурси;
- environments — можливі середовища запуску програми.

Окрім вище перелічених файлів, в проекті містять ще деякі системні файли, що створюються автоматично.

Програма поділена на такі шари, як репозиторії, сервіси та контролери на стороні серверної частини. Також шаровість присутня і на стороні клієнтської частини програми. Кожен шар програми відповідає за певні конкретні функції і використовує нижній шар аплікації. Це дозволяє писати гнучку програму і при необхідності легко додавати зміни.

3.2 Створення і доступ до бази даних

Базу даних створено за розробленою ER-моделлю, використовуючи Microsoft SQL Server. Для з'єднання зі сервером використовується connection string, прописаний у файлі appsettings.json.

Робота зі базою даних виконується допомогою Entity Framework — об'єктно-орієнтованої технології доступу до даних [22]. Таблиці у базі даних створюються на основі звичайних класів та їх полів. Модель, що відображає сутність у базі даних можна побачити на прикладі сутності «Category» на рисунку 3.4.

Клас імплементує вміст відповідно до інтерфейсу IModel, що додає службові поля, які вказують ким так коли запис було створено чи оновлено.

```
using ...

namespace Formula.DAL.Models
{
    /// <summary>
    /// Category model for db
    /// </summary>
    public class Category : IModel
    {
        /// <summary>
        /// Category Id
        /// </summary>
        public int Id { get; set; }

        /// <summary>
        /// Category Name
        /// </summary>
        public string Name { get; set; } = "";

        /// <summary>
        /// Formula Id
        /// </summary>
        public List<FormulaRecord> Formulas { get; set; } = new List<FormulaRecord>();

        #region History properties
        [NotMapped]
        public string? CreatedBy { get; set; }

        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public DateTime CreatedAt { get; set; }

        [NotMapped]
        public string? UpdatedBy { get; set; }

        [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
        public DateTime UpdatedAt { get; set; }
        #endregion
    }
}
```

Рисунок 3.4 — Модель для збереження даних до бази

Поле int Id у моделі відповідає первинному ключу у таблиці бази даних. Поле string Name відображає назву категорії, а List<FormulaRecords> Formula — позначає зв’язок категорій та формул, як один-до-багатьох, тобто одна категорія містить список формул, а одна формула може міститись в одній з категорій.

У випадку, якщо між сутностями зв’язок один-до-одного, то тип поля, що буде вторинним ключем, у таблиці це тип об’єкту з яким необхідно зв’язатись. Якщо полю одразу присвоїти значення, то це буде значення по замовчуванню. Якщо значення у базі даних може бути null, то біля типу поля потрібно поставити знак запитання, що вказуватиме на це.

Таким чином описано всі необхідні моделі для збереження даних у таблицях бази:

– Category;
– FormulaRecord;
– History;
– Request;
– User;

Безпосередньо створення таблиць у базі даних (Рис. 3.5) та автоматичне встановлення дати у службових полях (Рис. 3.6) описано у файлі ApplicationDBContext.cs.

```
1   using ...
9
10  namespace Formula.DAL
11  {
12      /// <summary> DB context
15      public class ApplicationDbContext : IdentityDbContext<User>, IApplicationDbContext
16      {
17          /// <summary> DB logger writes db query to console
20          public static readonly ILoggerFactory loggerFactory
21              = LoggerFactory.Create(builder => { builder.AddConsole().AddDebug(); });
22
23          /// <summary> DB context constructor
27          public ApplicationDbContext(DbContextOptions dbContextOptions)
28          : base(dbContextOptions)
29          {
30
31              Database.EnsureCreated();
32          }
33
34          /// <summary> Table of users in DB
37          public override DbSet<User> Users { get; set; }
38
39          /// <summary> Table of formulas in DB
42          public virtual DbSet<FormulaRecord> FormulaRecords { get; set; }
43
44          /// <summary> Table of request for admins
47          public virtual DbSet<Request> Requests {get;set;}
48
49          /// <summary> Table of categories
52          public virtual DbSet<Category> Categories { get; set; }
53
54          /// <summary> Saved user calculations
57          public virtual DbSet<History> History { get; set; }
```

Рисунок 3.5 — Створення бази даних та таблиць

```
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder?.Entity<Request>()
                .Property(entity => entity.State)
                .HasConversion<int>();

            modelBuilder?.Entity<FormulaRecord>()
                .Property(b => b.CreatedAt)
                .HasDefaultValueSql("getdate()");

            modelBuilder?.Entity<FormulaRecord>()
                .Property(b => b.UpdatedAt)
                .HasDefaultValueSql("getdate()");

            modelBuilder?.Entity<Request>()
                .Property(b => b.CreatedAt)
                .HasDefaultValueSql("getdate()");

            modelBuilder?.Entity<Request>()
                .Property(b => b.UpdatedAt)
                .HasDefaultValueSql("getdate()");

            modelBuilder?.Entity<History>()
                .Property(b => b.CreatedAt)
                .HasDefaultValueSql("getdate()");

            base.OnModelCreating(modelBuilder);

        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseLoggerFactory(loggerFactory);
        }
    }
}
```

Рисунок 3.6 — Автоматичне заповнення дат у службових полях

Таблицю «Roles» та проміжну зв’язну таблицю «UserRoles» створювати вручну не потрібно, так як для цього використано функціонал, наданий Entity Framework і вони створюються автоматично при ініціалізації бази даних.

Клас DbInitializer.cs відповідає за заповнення бази даних деякими значення при першому доступі до неї. Саме тут відбувається реєстрація всіх ролей: адміністратора, модератора та звичайного користувача, за допомогою вище згаданого RoleManager.

Також необхідно, щоб при створенні бази даних, було створено адміністратора за замовчуванням. І в майбутньому він міг створити модератора чи іншого адміністратора. Створення першого адміністратора, що зображено на рисунку 3.7. Спочатку йде перевірка чи є хоч якийсь користувач із роллю адміністратора в базі даних. Якщо його нема, то створюється користувач з стандартним іменем, електронною адресою та паролем і зберігається до бази даних.

```
        /// <summary> Seed user. Create super admin
        private async Task SeedUsers()
        {
            if (!_userManager.Users.Where(u => u.UserName == "Administrator").Any())
            {
                User user = new User
                {
                    UserName = "Administrator",
                    Email = "admin@localhost"
                };

                IdentityResult result = await _userManager.CreateAsync(user, "Pass-1111");

                if (result.Succeeded)
                {
                    await _userManager.AddToRoleAsync(user, RoleType.Admin);
                }
            }
        }
    }
}
```

Рисунок 3.7 — Створення першого адміністратора

Таким чином реалізовано необхідну базу даних за спроектованою раніше ER-моделлю. Безпосередній доступ до даних виконується на рівні репозиторіїв. Відповідно до кожної моделі даних створено клас-репозиторій, де реалізовано необхідні операції для управління даними, а також базові операції керування даними CRUD — «створення, зчитування, зміна і видалення» [23]. Приклад такого класу можна побачити на рисунку 3.8 а, б.

```
using Formula.DAL;
using Formula.DAL.Models;
using Formula.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Formula.Repositories
{
    /// <summary> Request repository
    public class RequestRepository : IRequestRepository
    {
        private readonly ApplicationDbContext dbContext;

        /// <summary> Constructor with initialization of needed services
        public RequestRepository(ApplicationDbContext applicationDBContext)
        {
            dbContext = applicationDBContext;
        }


        /// <summary> Create request
        public async Task Create(Request request)
        {
            await dbContext.Requests.AddAsync(request);
            await dbContext.SaveChangesAsync();
        }

        /// <summary> Read all requests
        public IEnumerable<Request> GetAll()
        {
            return dbContext.Requests;
        }

        /// <summary> Read particular request
        public Request GetParticular(int id)
        {
            return dbContext.Requests.Where(r => r.Id == id).First();
        }

        /// <summary> Update request
        public async Task Update(Request request)
        {
            dbContext.Entry(request).State = EntityState.Modified;
            await dbContext.SaveChangesAsync();
        }

        /// <summary> Delete request
        public async Task Delete(int id)
        {
            var model = dbContext.Requests.Find(id);
            dbContext.Requests.Remove(model);
            await dbContext.SaveChangesAsync();
        }
```

а)

```
        /// <summary> Count of requests
        public int GetCountOfAll()
        {
            return dbContext.Requests.Count();
        }

        /// <summary> Count of requests created by user
        public int GetCountOfAllRequestByUser(string userId)
        {
            return dbContext.Requests.Count(req => req.CreatedBy == userId);
        }

        /// <summary> Return all user requests
        public IEnumerable<Request> GetAllRequest(int skip, int take)
        {
            return dbContext.Requests
                .OrderBy(req => req.CreatedAt)
                .Skip(skip)
                .Take(take)
                .AsEnumerable();
        }

        /// <summary> Return all user requests for specific user
        public IEnumerable<Request> GetRequestForUser(int skip, int take, string userId)
        {
            return dbContext.Requests
                .Where(req => req.CreatedBy == userId)
                .OrderBy(req => req.CreatedAt)
                .Skip(skip)
                .Take(take)
                .AsEnumerable();
        }
    }
}
```

б)

Рисунок 3.8 — Реалізація request repository,

а — операції CRUD, б — додаткові операції

Кожен репозиторій імплементує інтерфейс, де описано які саме методи повинні бути реалізовані, що додає ще один шар абстракції та гнучкості програмі.

3.3 Основна логіка програми

Основна логіка програми написана на серверній частині аплікації на рівні сервісів (Рис. 3.9).

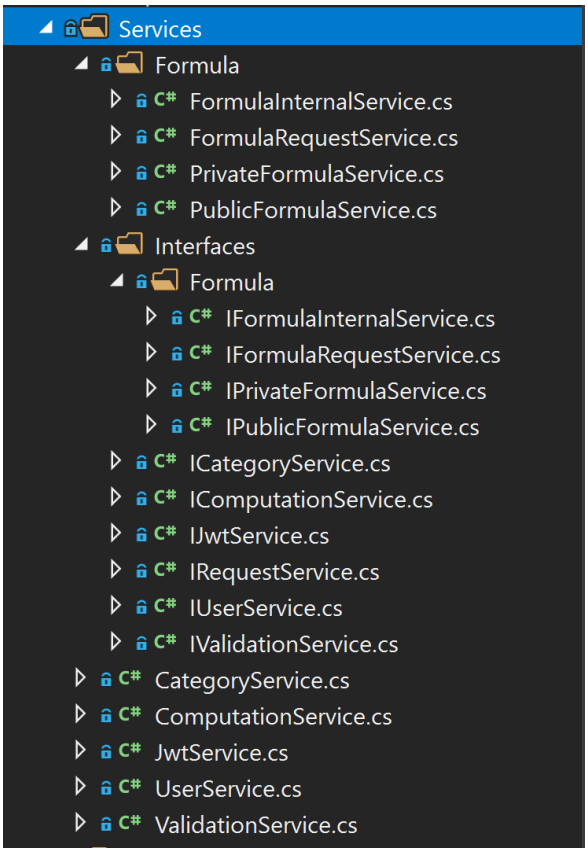


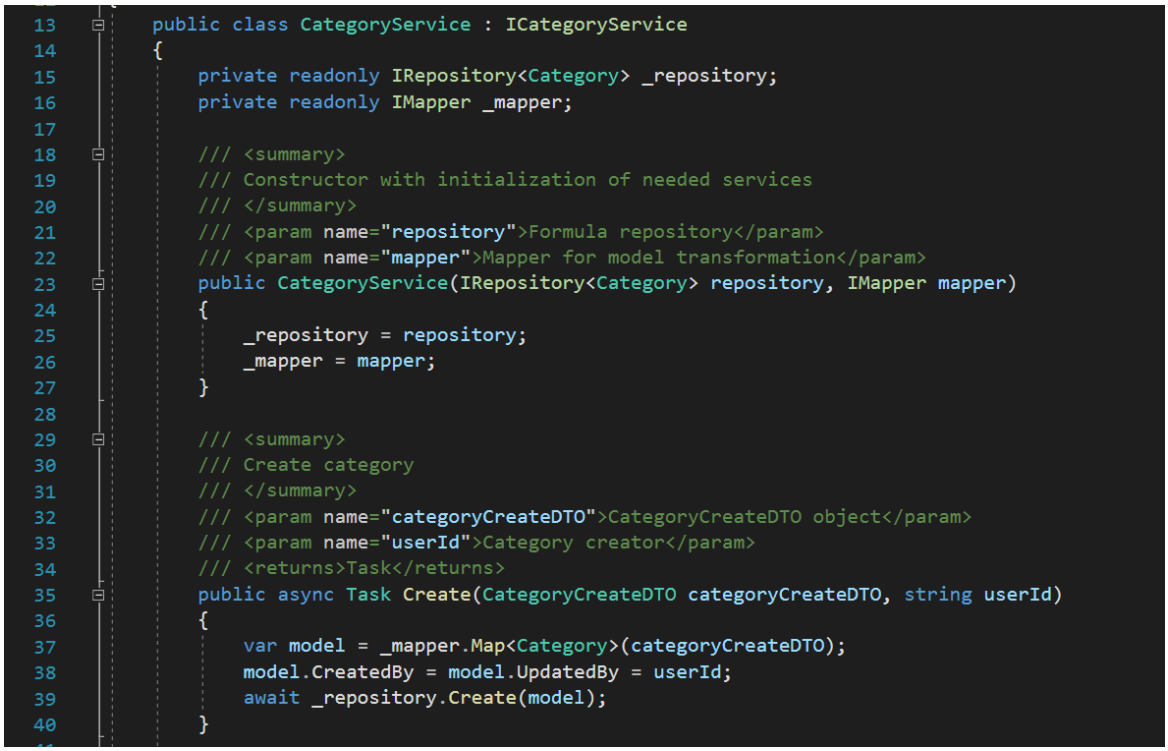
Рисунок 3.9 — Сервіси серверної частини програми

Сервіси можуть використовувати шар репозиторів та реалізовувати обробку даних перед занесенням до бази даних чи після отримання з бази для передачі на клієнтську частину. І виконують обробку даних, не тільки пов’язану із базою даних. Логіка, що написана у сервісах структурована наступним чином:

- FormulaInternalService — функціонал для внутрішнього використання у програмі, як от перевірка чи існує така формула в базі даних чи знайти формулу за її іменем;
- FormulaRequestService — функціонал управління запитами користувачів на поширення формул:
    - AddRequest;
    - ApproveRequest;
    - DeclineRequest;
    - GetCountOfAll;
    - GetCountOfAllRequestByUser;
    - GetAllRequest;
    - GetRequestForUser.
- PrivateFormulaService — базовий функціонал для управління персональними формулами;
- PublicFormulaService — базовий функціонал для управління формулами із загального списку;
- CategoryService — базовий функціонал для керування категоріями;
- ComputationService — логіка обчислення формули;
- JwtService — генерація json web token для авторизації користувача;
- UserService — функціонал, що стосується користувача, як от управління закладками та історією;
- ValidationService — логіка перевірки формули на її правильність і можливість існування.

Приклад методу сервісу, що обробляє дані з бази, та використовуєш шар репозиторіїв можна побачити на рисунку 3.10.

Сервіс використовує відповідний йому репозиторій і так здійснює доступ до даних бази. Також сервіс використовує мапер, що здійснює перетворення моделей за шаблоном.



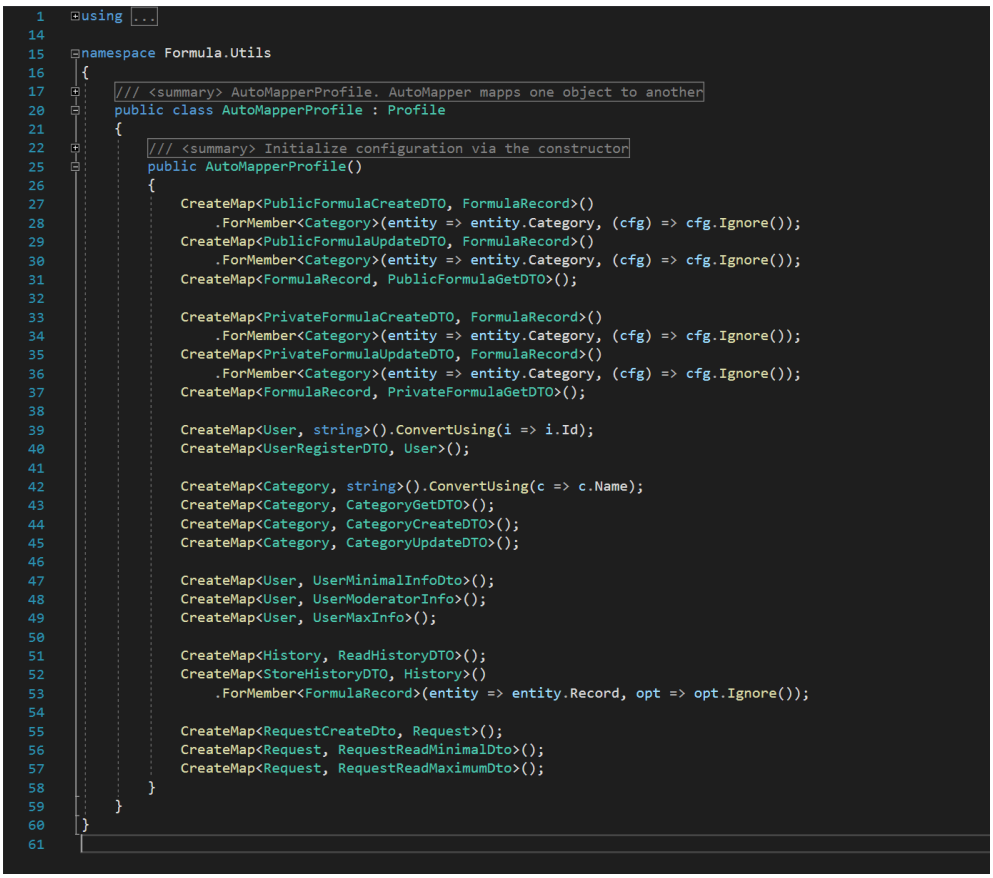
```
public class CategoryService : ICategoryService
{
    private readonly IRepository<Category> _repository;
    private readonly IMapper _mapper;

    /// <summary>
    /// Constructor with initialization of needed services
    /// </summary>
    /// <param name="repository">Formula repository</param>
    /// <param name="mapper">Mapper for model transformation</param>
    public CategoryService(IRepository<Category> repository, IMapper mapper)
    {
        _repository = repository;
        _mapper = mapper;
    }

    /// <summary>
    /// Create category
    /// </summary>
    /// <param name="categoryCreateDTO">CategoryCreateDTO object</param>
    /// <param name="userId">Category creator</param>
    /// <returns>Task</returns>
    public async Task Create(CategoryCreateDTO categoryCreateDTO, string userId)
    {
        var model = _mapper.Map<Category>(categoryCreateDTO);
        model.CreatedBy = model.UpdatedBy = userId;
        await _repository.Create(model);
    }
```

Рисунок 3.10 — Category service

Для створення категорій сервіс повинен отримати CategotyCreateDTO та ідентифікатор користувача, що створює запис. CategotyCreateDTO — модель даних, схожа до моделі Categoty, що зберігається до бази. Проте вона містить тільки ті поля, що задіяні при створенні категорії. Наприклад, ідентифікатор категорій користувач не вводить, це первинний клюс, що інкрементується самостійно. Все, що потрібно ввести користувачу при створенні категорії це ім’я категорій, тому CategotyCreateDTO містить тільки одне поле — Name. В самому ж методі відбувається перетворення моделі в ту, що необхідна репозиторію та виклик метода репозиторія, що створить і збереже запис.

DTO (Data Transfer Object) являється шаблоном проектування, що застосовується для передачі даних між підсистемами програми. Такі об’єкти не містять в собі жодної логіки, а тільки ті поля, які потрібно використати для тих чи інших задач, щоб не навантажувати функції великою кількістю непотрібних на даний момент аргументів [24].

У файлі AutoMapperProfile.cs, що є утилітою, вказано як саме перетворювати моделі між собою (Рис. 3.11). Якщо типи полів у моделей не співпадають, то необхідно вказати правило перетворення.



```
using ...

namespace Formula.Utils
{
    /// <summary> AutoMapperProfile. AutoMapper mapps one object to another
    public class AutoMapperProfile : Profile
    {
        /// <summary> Initialize configuration via the constructor
        public AutoMapperProfile()
        {
            CreateMap<PublicFormulaCreateDTO, FormulaRecord>()
                .ForMember<Category>(entity => entity.Category, (cfg) => cfg.Ignore());
            CreateMap<PublicFormulaUpdateDTO, FormulaRecord>()
                .ForMember<Category>(entity => entity.Category, (cfg) => cfg.Ignore());
            CreateMap<FormulaRecord, PublicFormulaGetDTO>();

            CreateMap<PrivateFormulaCreateDTO, FormulaRecord>()
                .ForMember<Category>(entity => entity.Category, (cfg) => cfg.Ignore());
            CreateMap<PrivateFormulaUpdateDTO, FormulaRecord>()
                .ForMember<Category>(entity => entity.Category, (cfg) => cfg.Ignore());
            CreateMap<FormulaRecord, PrivateFormulaGetDTO>();

            CreateMap<User, string>().ConvertUsing(i => i.Id);
            CreateMap<UserRegisterDTO, User>();

            CreateMap<Category, string>().ConvertUsing(c => c.Name);
            CreateMap<Category, CategoryGetDTO>();
            CreateMap<Category, CategoryCreateDTO>();
            CreateMap<Category, CategoryUpdateDTO>();

            CreateMap<User, UserMinimalInfoDto>();
            CreateMap<User, UserModeratorInfo>();
            CreateMap<User, UserMaxInfo>();

            CreateMap<History, ReadHistoryDTO>();
            CreateMap<StoreHistoryDTO, History>()
                .ForMember<FormulaRecord>(entity => entity.Record, opt => opt.Ignore());

            CreateMap<RequestCreateDto, Request>();
            CreateMap<Request, RequestReadMinimalDto>();
            CreateMap<Request, RequestReadMaximumDto>();
        }
    }
}
```

Рисунок 3.11 — Шаблони перетворення моделей

За таким же принципом здійснюється читання записів у базі даних, оновлення та видалення.

Сервіси, в яких міститься найголовніша логіка це ValidationService та ComputationService. Кожна формула, що створюється, повинна бути ретельно перевірена, щоб не виникали помилки при обчислені. Формула являє собою рядок символів, послідовність яких має певні правила. Звісно ж, є обмеження і що до символів, формула може складатись із:

– відкриваючої дужки;

– закриваючої дужки;
– бінарної операції;
– унарної орепації;
– постфіксної операції;
– числа, що може містити крапку;
– змінної, що починається з літери та може містити цифри;
– виклик вкладеної формули.

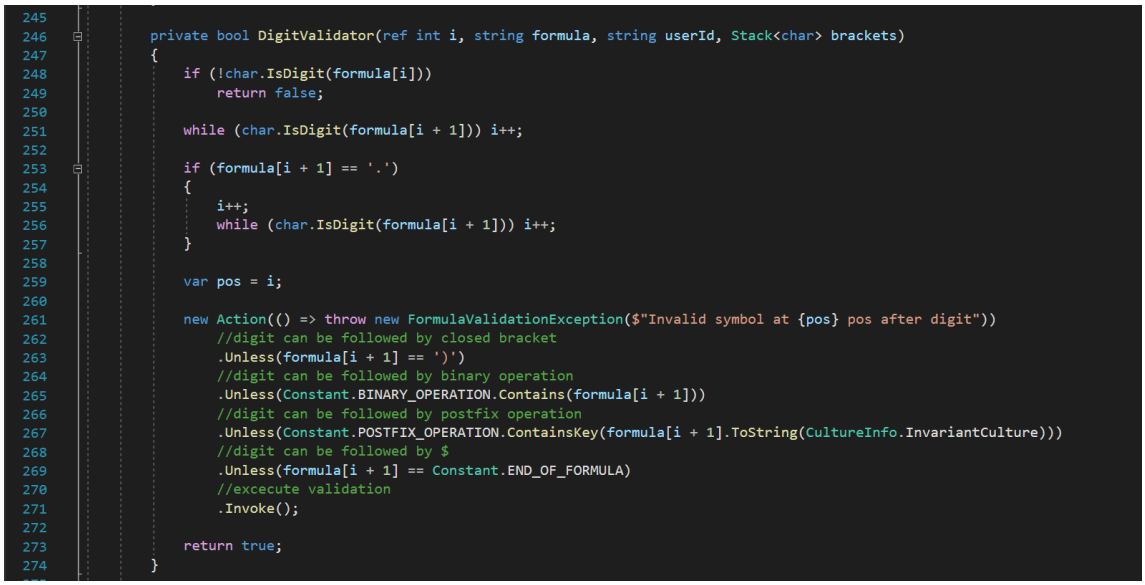
Якщо кількість дужок не співпадає, або послідовність елементів не прийнятна, то така формула не може бути створена чи допущена до обчислення.

ValidationService містить список валідаторів, кожен з яких перевіряє чи наступний символ формули є можливим. Для покращення швидкодії, спочатку перевіряється чи кількість відкриваючих та закриваючих дужок співпадає, якщо ні, то подальша перевірка не має сенсу і не виконується. Якщо це етап успішно пройдено, то з формули забирається всі відступи та додаються спеціальні символи на початок та кінець, а далі формула проходить через наступні валідотори:

– StartValidator;
– OpenBracketValidator;
– CloseBracketValidator;
– BinaryOperationValidator;
– UnaryOperationValidator;
– PostfixOperationValidator;
– DigitValidator;
– IdentificatorValidator.

Кожен із символів формули, в залежності від своєї суті, перевіряється валідатором. Перевірка є успішною, якщо наступний символ формули може йти за поточним. Якщо такі символи не можуть іти один за одним — буде повідомлено про помилку і вказано на якій позиції знайдено це виявлено. На рисунку 3.12

зображено валідатор для чисел. За аналогічним принципом працюють і інші валідатори.



```
245
246     private bool DigitValidator(ref int i, string formula, string userId, Stack<char> brackets)
247     {
248         if (!char.IsDigit(formula[i]))
249             return false;
250
251         while (char.IsDigit(formula[i + 1])) i++;
252
253         if (formula[i + 1] == '.')
254         {
255             i++;
256             while (char.IsDigit(formula[i + 1])) i++;
257         }
258
259         var pos = i;
260
261         new Action(() => throw new FormulaValidationException($"Invalid symbol at {pos} pos after digit"))
262             //digit can be followed by closed bracket
263             .Unless(formula[i + 1] == ')')
264             //digit can be followed by binary operation
265             .Unless(Constant.BINARY_OPERATION.Contains(formula[i + 1]))
266             //digit can be followed by postfix operation
267             .Unless(Constant.POSTFIX_OPERATION.ContainsKey(formula[i + 1].ToString(CultureInfo.InvariantCulture)))
268             //digit can be followed by $
269             .Unless(formula[i + 1] == Constant.END_OF_FORMULA)
270             //excecute validation
271             .Invoke();
272
273         return true;
274     }
```

Рисунок 3.12 — Digit validator

Якщо поточний символ цифра, потрібно знайти кінець числа, адже число може містити багато цифр і одну крапку. Тоді перевіряється чи наступний символ може іти після числа. Після числа не може бути нічого іншого, окрім вказаних в переліку символів: закриваюча дужка, бінарна операція, постфіксна операція чи кінець формули.

Якщо поточний символ літера, то необхідно перевірити чи це змінна чи це вкладена формула. Оскільки тіло вкладеної формули обов’язково повинно бути в дужках, то визначити це легко. І якщо це формула, то вона може містити кілька аргументів, що розділяються комою, та містити в собі ще інші вкладені формули. Тому якщо поточний символ це початок формули, її вміст перевіряється в окремому методі (Рис. 3.13). Вміст дужок формули розділяється за комами на компоненти. Потім всі компоненти перевіряються на наявність у списку вбудованих чи у базі даних, якщо вони вже існують, то спочатку перевіряється

правильність кількості аргументів. Якщо все пройдено успішно, то ці компоненти по черзі проходять перевірку валідаторами формули.

```
private bool FunctionAnalyzer(int functionNamePos, string userId, ref int i, string formula)
{
    string functionName = formula.Substring(functionNamePos, (i - functionNamePos) + 1);

    var startPos = i + 2;

    int bracketCount = 0;

    List<int> commas = new List<int>();

    int endPos;

    for (endPos = i; endPos < formula.Length - 1; endPos++)
    {
        if (formula[endPos + 1] == ',')
        {
            if (bracketCount == 1)
                commas.Add(endPos + 1);
        }
        else if (formula[endPos + 1] == '(')
            bracketCount++;
        else if (formula[endPos + 1] == ')')
        {
            bracketCount--;
            if (bracketCount == 0)
                break;
        }
    }
    endPos++;

    List<string> components = new List<string>();

    if (commas.Any())
    {
        int componentStartPos = startPos;

        foreach (var index in commas)
        {
            components.Add(formula[componentStartPos..index]);
            componentStartPos = index + 1;
        }
        components.Add(formula[componentStartPos..endPos]);
    }
    else
    {
        components.Add(formula[startPos..endPos]);
    }

    i = endPos;

    ValidateFunctionArrity(functionName, functionNamePos, components.Count, userId);

    foreach (var component in components)
    {
        Validate(component, userId);
    }

    return true;
}
```

Рисунок 3.13 — Аналіз вмісту вкладеної формули

Формула, що успішно пройшла перевірку правильності можна обчислити за допомогою методів класу ComputationService. Як зазначалось при проектуванні програми, обчислення проходить у декілька етапів.

До обчислення приходить об’єкт класу ComputationRequestDTO, що містить два поля типу string: формулу та її аргументи. Спочатку відбувається підстановка змінних у формулі. При підстановці можлива ситуація, коли змінна у формулі є від’ємною і її аргумент також від’ємний. Після підстановки значення у формулі отримаємо «--», що є неприйнятним, тому після підстановки перевіряється наявність двох мінусів під ряд і їх замінюється на «+».

Наступним кроком є додавання нуля перед унарною операцією в змінній. Оскільки символи унарних та бінарних операцій одинакові, то це призведе до помилки при обчисленні. Додавання нуля перед унарною операцією не вплине на результат виконання і вирішить дану проблему. Після даного кроку залишається повноцінна формула у вигляді рядка.

Із вигляду рядка формулу потрібно перетворити в масив її елементів. Тому на третьому кроці кожен символ формули проходить ідентифікацію та додається до вихідного масиву. Якщо це число чи літера, то, як у випадку валідації, перевіряється із скількох символів вони складаються і додається в масив одним елементом.

Саме обчислення виконується за принципом зворотного польського запису — форма запису математичних виразів, в якій знаки операцій розташовано після операндів [25]. Обчислення базується на операціях з стеком, що забезпечує швидкість обчислення, а також така форму запису звільняє від необхідності працювати із дужками. Тому із масиву елементів формула перетворюється в стек елементів (Рис. 3.14).

```
foreach (var f in formula)
{
    if (f[0] == '(')
    {
        tempStack.Push(f);
    }
    else if (f[0] == ')')
    {
        while (!(tempStack.Peek()[0] == '('))
        {
            preparedFormula.Push(tempStack.Pop());
        }
        if (tempStack.Peek()[0] == '(')
        {
            tempStack.Pop();
        }
    }
    else if(char.IsDigit(f[0]) || Constant.POSTFIX_OPERATION.ContainsKey(f) || Constant.CONSTANT_VALUE.ContainsKey(f))
    {
        preparedFormula.Push(f);
    }
    else if (Constant.OPERATORS.ContainsKey(f))
    {
        while (!(tempStack.IsEmpty() || tempStack.Peek()[0] == '('
            && (Constant.BUILT_IN_FUNCTION.ContainsKey(tempStack.Peek())
                || ( Constant.PRIORITY.ContainsKey(tempStack.Peek()) && Constant.PRIORITY[tempStack.Peek()] <= Constant.PRIORITY[f])
                || ( !Constant.PRIORITY.ContainsKey(tempStack.Peek()) && char.IsLetter(tempStack.Peek()[0])))))
        {
            preparedFormula.Push(tempStack.Pop());

        }

        tempStack.Push(f);
    }
    else if (Constant.BUILT_IN_FUNCTION.ContainsKey(f))
    {
        tempStack.Push(f);
    }
    else if(_formulaService.IsExistFormula(f, userId))
    {
        tempStack.Push(f);
    }
}

while (!tempStack.IsEmpty())
{
    preparedFormula.Push(tempStack.Pop());
}
while (!preparedFormula.IsEmpty())
{
    finishedFormula.Push(preparedFormula.Pop());
}

return finishedFormula;
}
```

Рисунок 3.14 — Перетворення формули в стек

Коли вже готовий стек з елементами формули, відбувається саме обчислення, алгоритм якого можна побачити на рисунку 3.15. Алгоритм обробки вкладених формул можна побачити на рисунку 3.16.

```
/// <summary>
/// Computate formula
/// </summary>
/// <param name="formula">Formula in polish notation</param>
/// <param name="userId">Null or user Id if user authenticated</param>
/// <returns>Formula result</returns>
private double Computate(Stack<string> formula, string userId)
{
    Stack<double> data = new Stack<double>();

    while (!formula.IsEmpty())
    {
        var f = formula.Pop();

        if (char.IsDigit(f[0]))
        {
            data.Push(double.Parse(f, CultureInfo.InvariantCulture));
        }
        else if (Constant.OPERATORS.ContainsKey(f))
        {
            Constant.OPERATORS[f](data);
        }
        else if (Constant.BUILT_IN_FUNCTION.ContainsKey(f))
        {
            Constant.BUILT_IN_FUNCTION[f](data);
        }
        else if (Constant.POSTFIX_OPERATION.ContainsKey(f))
        {
            Constant.POSTFIX_OPERATION[f](data);
        }
        else if (Constant.CONSTANT_VALUE.ContainsKey(f))
        {
            Constant.CONSTANT_VALUE[f](data);
        }
        else
        {
            ProcessInnerFunction(_formulaService.GetByName(f, userId), data, userId);
        }
    }

    return data.Pop();
}
```

Рисунок 3.15 — Обчислення формули

```
/// <summary>
/// Processing sub formula from db.
/// For it we recreate FormulaRequest object and
/// run full Computate process at end of it we push to stack result of computation.
/// It allow use subformula in other subformula
/// </summary>
/// <param name="formulaRecord"></param>
/// <param name="data"></param>
/// <param name="userId">Null or user Id if user authenticated</param>
private void ProcessInnerFunction(FormulaRecord formulaRecord, Stack<double> data, string userId)
{
    //Build arguments for request
    var args = formulaRecord.Variables
        .Split(';')
        .Select(arg => arg + "=" + data.Pop().ToString(CultureInfo.InvariantCulture))
        .Aggregate((item1, item2) => item1 + ";" + item2);

    //Build request object
    var formulaRequest = new ComputationRequestDTO
    {
        Formula = formulaRecord.View,
        Args = args
    };

    //Computate formula
    var result =
        Computate(
            Parse(
                FormFormulaArray(
                    AddZeroBeforeUnaryOperation(
                        Replace(formulaRequest)
                        )
                    )
                ,
                userId
                )
            ,
            userId
            );
    //Store result
    data.Push(result);
}
```

Рисунок 3.16 — Обробка вкладних формул

Весь код класів Constant, ValidationService та ComputationService можна побачити у додатку А.

3.4 Рівень контролерів та клієнтська частина аплікації

Код на рівні контролерів це верхній шар серверної частини аплікації, що відповідає за перетворення внутрішніх результатів у модель представлення та транспорт даних. Контролери викликаються за певним маршрутом, використовують відповідні їм сервіси та повертають результат їх роботи на клієнтську частину. Код у контролерах досить простий та схожий, приклад можна побачити на рисунку 3.17.

```
34          [HttpPost]
35          [Route("computate")]
36          public ComputationRequestResultDTO Computation([FromBody]ComputationRequestDTO formulaRequest)
37          {
38              if(IsAnonymous)
39              {
40                  return _computationService.CalculateFormula(formulaRequest, null);
41              }
42              else
43              {
44                  return _computationService.CalculateFormula(formulaRequest, UserId);
45              }
46          }
47      }
48  }
```

Рисунок 3.17 — Computation controller

В межах проекту створено контролери з такими призначеннями:

- BaseController — відповідає за визначення ролі користувача;
- PrivateFormulaRepositoryController, "api/formula/private" — запити щодо приватних формул;
- PublicFormulaRepositoryController, "api/formula/public" — запити щодо загальних формул;
- CategoryController, "api/category" — запити щодо категорій;
- ComputationController, "api/formula" — запит на обчислення формули;

– RequestController, "api/requests" — запити щодо поширення приватних формул;
– UserController, "api/users" — поділяється на три групи запитів:
  - запити щодо авторизації, реєстрації користувачів;
  - запити щодо закладок;
  - запити щодо історії обчислень.

Сервіси клієнтської частини аплікації звертаються до серверної частини за допомогою маршрутів, вказаних біля методів контролерів (Рис. 3.18).

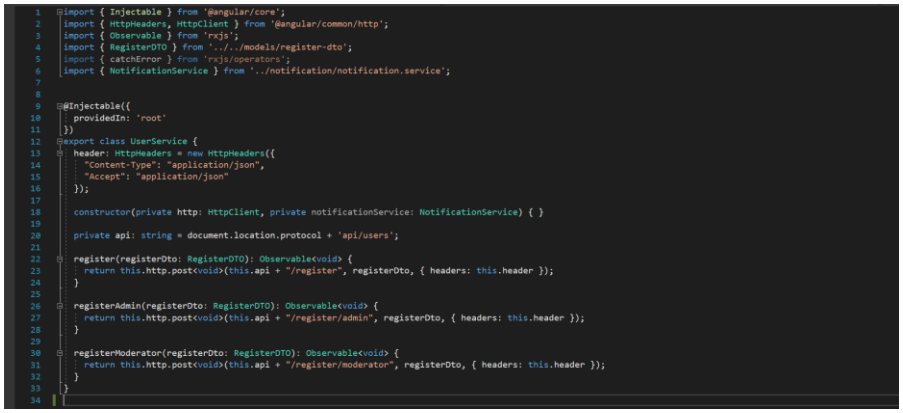


Рисунок 3.18 — User service клієнтської частини

Усі сервіси клієнтської частини програми працюють за таким же принципом, список сервісів можна побачити на рисунку 3.19.

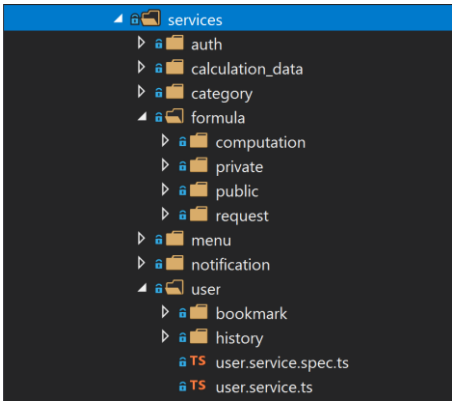


Рисунок 3.19 — Сервіси клієнтської частини

Інколи деяка логіка програми залишається на стороні клієнта, та в даному проекті вся логіка по максимуму знаходиться на серверній частині, що надає певну безпеку. Тож на клієнтській частині залишається тільки логіка, пов’язана із виглядом програми, наприклад, вивід сповіщень.

Вигляд програми складається з окремих компонентів (Рис. 3.20), що містять в собі html, scss та ts файли.

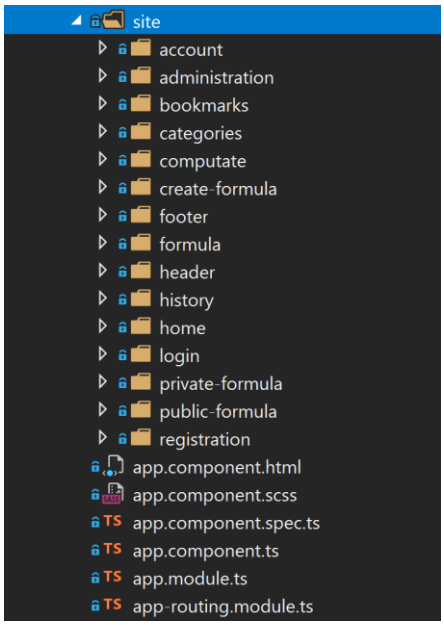
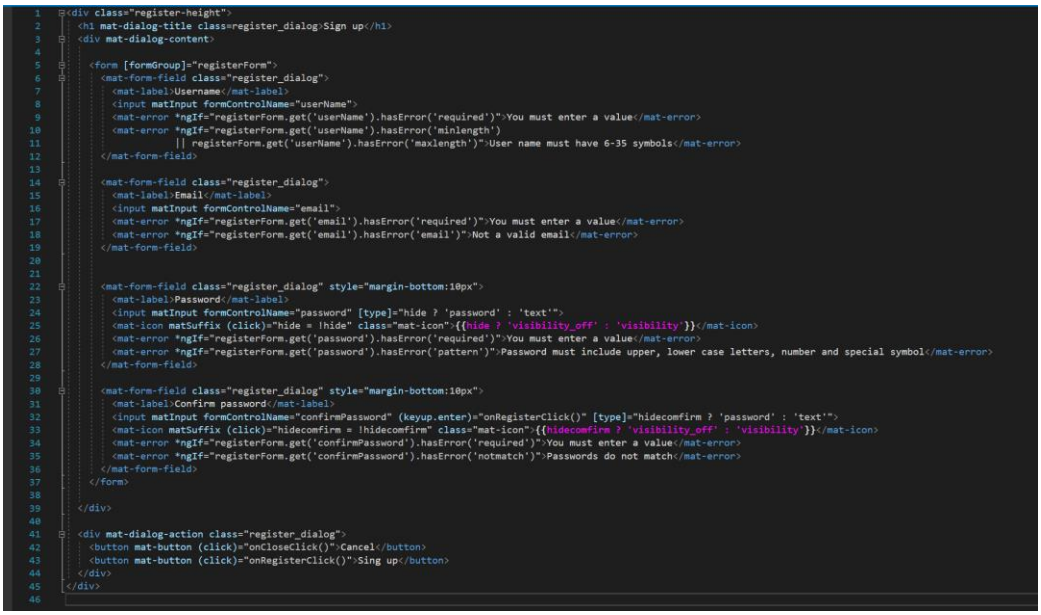


Рисунок 3.20 — Компоненти вигляду програми

Алгоритм роботи із компонентами наведено на прикладі registration component на рисунках 3.21 — 3.23.

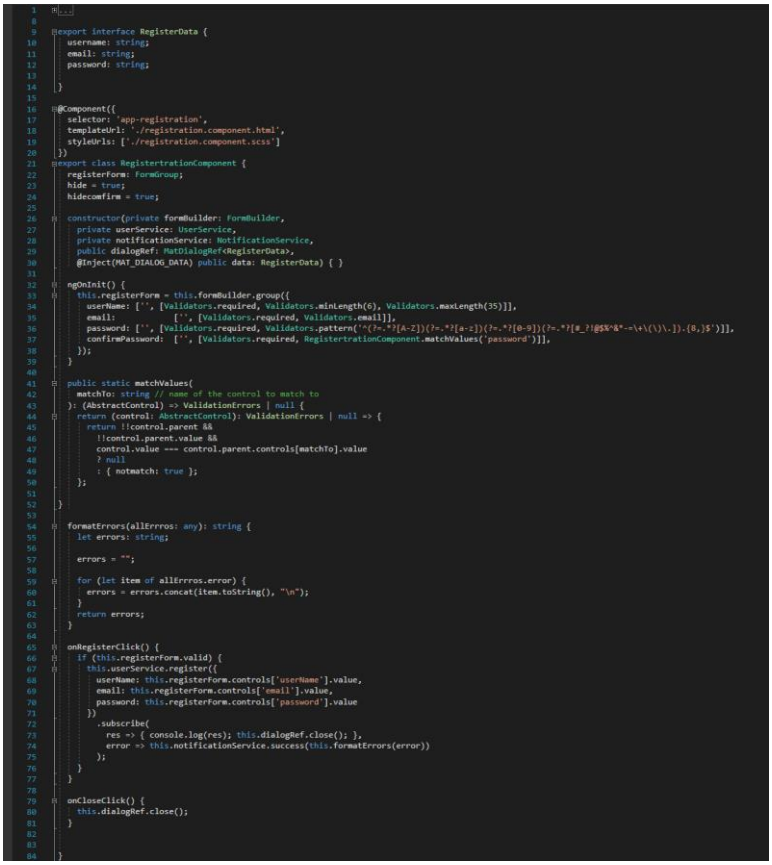
```scss
.register_dialog {
  width: 280px;
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  flex-flow: row;
}

.register-height {
  height: 400px;
  margin-top: 55px;
}

.mat-icon {
  font-size: 15px;
}

::ng-deep .register_dialog .mat-form-field-flex {
  width: 200px !important;
}
```

Рисунок 3.21 — Файл registration.component.scss



```
<div class="register-height">
  <h1 mat-dialog-title class=register_dialog>Sign up</h1>
  <div mat-dialog-content>

    <form [formGroup]="registerForm">
      <mat-form-field class="register_dialog">
        <mat-label>Username</mat-label>
        <input matInput formControlName="userName">
        <mat-error *ngIf="registerForm.get('userName').hasError('required')">You must enter a value</mat-error>
        <mat-error *ngIf="registerForm.get('userName').hasError('minlength')
                  || registerForm.get('userName').hasError('maxlength')">User name must have 6-35 symbols</mat-error>
      </mat-form-field>

      <mat-form-field class="register_dialog">
        <mat-label>Email</mat-label>
        <input matInput formControlName="email">
        <mat-error *ngIf="registerForm.get('email').hasError('required')">You must enter a value</mat-error>
        <mat-error *ngIf="registerForm.get('email').hasError('email')">Not a valid email</mat-error>
      </mat-form-field>


      <mat-form-field class="register_dialog" style="margin-bottom:10px">
        <mat-label>Password</mat-label>
        <input matInput formControlName="password" [type]="hide ? 'password' : 'text'">
        <mat-icon matSuffix (click)="hide = !hide" class="mat-icon">{{hide ? 'visibility_off' : 'visibility'}}</mat-icon>
        <mat-error *ngIf="registerForm.get('password').hasError('required')">You must enter a value</mat-error>
        <mat-error *ngIf="registerForm.get('password').hasError('pattern')">Password must include upper, lower case letters, number and special symbol</mat-error>
      </mat-form-field>

      <mat-form-field class="register_dialog" style="margin-bottom:10px">
        <mat-label>Confirm password</mat-label>
        <input matInput formControlName="confirmPassword" (keyup.enter)="onRegisterClick()" [type]="hidecomfirm ? 'password' : 'text'">
        <mat-icon matSuffix (click)="hidecomfirm = !hidecomfirm" class="mat-icon">{{hidecomfirm ? 'visibility_off' : 'visibility'}}</mat-icon>
        <mat-error *ngIf="registerForm.get('confirmPassword').hasError('required')">You must enter a value</mat-error>
        <mat-error *ngIf="registerForm.get('confirmPassword').hasError('notmatch')">Passwords do not match</mat-error>
      </mat-form-field>
    </form>

  </div>

  <div mat-dialog-action class="register_dialog">
    <button mat-button (click)="onCloseClick()">Cancel</button>
    <button mat-button (click)="onRegisterClick()">Sing up</button>
  </div>
</div>
```

Рисунок 3.22 — Файл registration.component.html



```
...

export interface RegisterData {
  username: string;
  email: string;
  password: string;

}

@Component({
  selector: 'app-registration',
  templateUrl: './registration.component.html',
  styleUrls: ['./registration.component.scss']
})
export class RegistertrationComponent {
  registerform: FormGroup;
  hide = true;
  hidecomfirm = true;

  constructor(private formBuilder: FormBuilder,
    private userService: UserService,
    private notificationService: NotificationService,
    public dialogRef: MatDialogRef<RegisterData>,
    @Inject(MAT_DIALOG_DATA) public data: RegisterData) { }

  ngOnInit() {
    this.registerform = this.formBuilder.group({
      userName: ['', [Validators.required, Validators.minLength(6), Validators.maxLength(35)]],
      email:          ['', [Validators.required, Validators.email]],
      password: ['', [Validators.required, Validators.pattern('^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#_?!@$%^&*-]).{8,}$')]],
      confirmPassword:  ['', [Validators.required, RegistertrationComponent.matchValues('password')]],
    });
  }

  public static matchValues(
    matchTo: string // name of the control to match to
  ): (AbstractControl) => ValidationErrors | null {
    return (control: AbstractControl): ValidationErrors | null => {
      return !!control.parent &&
        !!control.parent.value &&
        control.value === control.parent.controls[matchTo].value
        ? null
        : { notmatch: true };
    };
  }

  formatErrors(allErrros: any): string {
    let errors: string;

    errors = "";

    for (let item of allErrros.error) {
      errors = errors.concat(item.toString(), "\n");
    }
    return errors;
  }

  onRegisterClick() {
    if (this.registerForm.valid) {
      this.userService.register({
        userName: this.registerForm.controls['userName'].value,
        email: this.registerForm.controls['email'].value,
        password: this.registerForm.controls['password'].value
      })
        .subscribe(
          res => { console.log(res); this.dialogRef.close(); },
          error => this.notificationService.success(this.formatErrors(error))
        );
    }
  }

  onCloseClick() {
    this.dialogRef.close();
  }

}
```

Рисунок 3.23 — Файл registration.component.ts

Вигляд кожного компоненту описаний у html файлі, за необхідності стилі додаються у scss файлі, а у файлі з розширенням ts описаний функціонал, що

використовуються у шаблоні. Кінцевий вигляд вікна реєстрації зображений на рисунку 3.24.

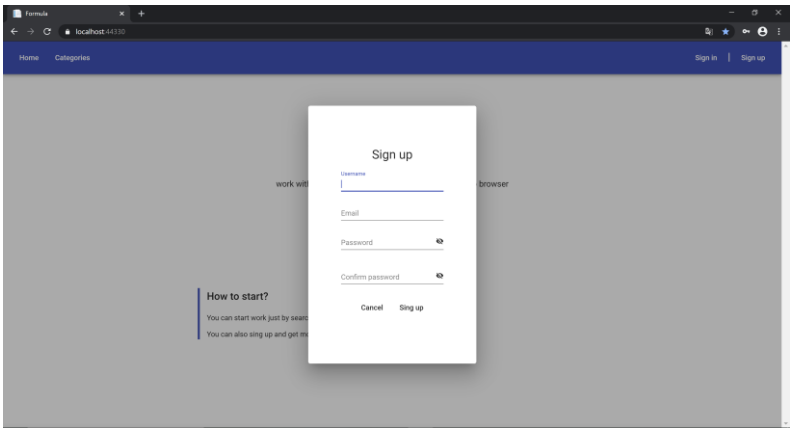


Рисунок 3.24 — Кінцевий вигляд вікна реєстрації користувача

Важливою сторінкою це сторінка обчислення формули, обраної користувачем (Рис. 3.25). Вона є одинаковою для будь-якої формули, адже поля для введення змінних генеруються автоматично відносно кожної формули.

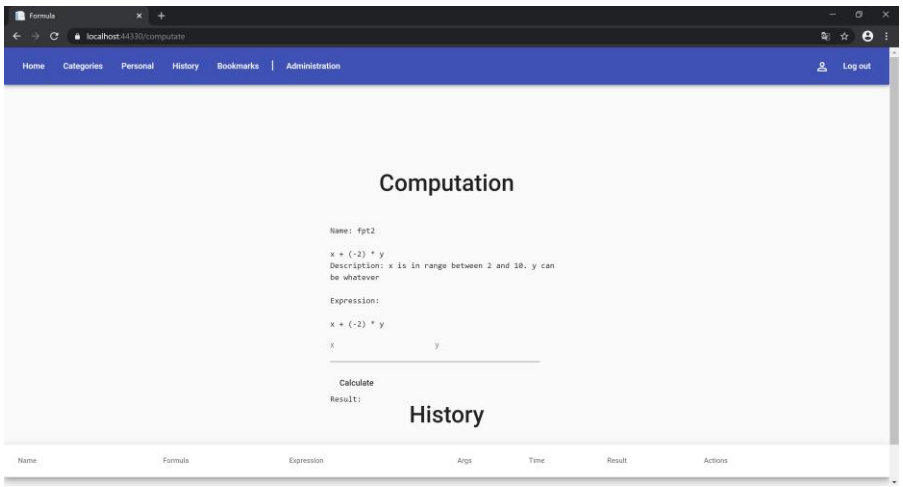


Рисунок 3.25 — Обчислення формули

На даній сторінці відображена основна інформація про формулу, зокрема її опис, де можуть знаходитись підказки щодо змінних формул. Під час вводу

значень змінних, вони одрау відображаються у полі «Expression» і користувач бачить повністю формулу в звичному для його вигляді. Після натиску на кнопку «Calculate», виконується обчислення і виводиться результат. Також результат виводиться до таблиці з історією.

Для виконання обчислення система використовує відповідні складові серверної частини програми. В коді компоненту даної сторінки виконується підготовка даних і вилик сервісу клієнтської частини, що в свою чергу викликає сервіс серверної частини (Рис. 3.26).

```
calculate() {
  let request: ComputationRequestDTO;
  let argsValue = "";
  let isValid = true;
  this.fields.forEach(item => {
    if (isValid) {
      if (item.value == "") {
        isValid = false;
      } else {
        argsValue = argsValue + item.name + "=" + item.value + ";";
      }
    }
  })
  if (!isValid)
    return;
  argsValue = argsValue.substr(0, argsValue.length - 1);
  request = {
    formula: this.computationData.formula,
    args: argsValue
  };
  this.computationService.calculate(request).subscribe(result => {
    this.result = result.result.toString();
    let record: HistoryRecord = new HistoryRecord();
    record.name = this.computationData.name;
    record.formula = this.computationData.formula;
    record.args = argsValue;
    record.expression = this.formula;
    record.result = this.result;
    record.time = Date.now();
    this.calculationDataService.history.push(record);
    this.refreshHistory();
  },
    error => this.notificationService.warning(error.message));
  }
```

Рисунок 3.26 —Метод обробки запиту обчислення

Кінцевий вигляд решти компонентів та програми в цілому можна побачити у додатку Б.

3.5 Тестування програми

Тестування програми є важливим етапом у розробці програмного забезпечення. В даному проекті найважливішим частиною програми, що потребує ретельного тестування, є методи, що відповідають за валідацію формул та їх обчислення. Рівень покриття даних методів тестами можна побачити на рисунках 3.27 та 3.28. Результат запуску всіх тестів можна побачити на рисунку 3.29.

	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
ComputationService	0	0,00%	294	100,00%
AddZeroBeforeUnaryOperation...	0	0,00%	20	100,00%
CalculateFormula(Formula.Mod...	0	0,00%	12	100,00%
Computate(System.Collections....	0	0,00%	39	100,00%
ComputationService(Formula.S...	0	0,00%	2	100,00%
FormFormulaArray(string)	0	0,00%	68	100,00%
Parse(System.Collections.Gener...	0	0,00%	96	100,00%
ProcessInnerFunction(Formula....	0	0,00%	19	100,00%
Replace(Formula.Models.Form...	0	0,00%	38	100,00%

Рисунок 3.27 — Покриття тестами ComputationService

	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
ValidationService	25	6,96%	334	93,04%
BinaryOperationValidator(ref in...	0	0,00%	16	100,00%
CloseBracketValidator(ref int, st...	2	8,33%	22	91,67%
DigitValidator(ref int, string, stri...	0	0,00%	32	100,00%
FunctionAnalyzer(int, string, ref...	0	0,00%	50	100,00%
IdentificatorValidator(ref int, stri...	1	2,94%	33	97,06%
IsBracketExsist(string)	0	0,00%	15	100,00%
OpenBracketValidator(ref int, st...	0	0,00%	20	100,00%
PostfixOperationValidator(ref in...	0	0,00%	17	100,00%
StartValidator(ref int, string, stri...	0	0,00%	20	100,00%
UnaryOperationValidator(ref int...	11	68,75%	5	31,25%
Validate(string, string)	2	4,76%	40	95,24%
ValidateFormula(string, string)	0	0,00%	5	100,00%
ValidateFunctionArrity(string, in...	9	25,00%	27	75,00%
ValidationService(Formula.Servi...	0	0,00%	32	100,00%

Рисунок 3.28 — Покриття тестами ValidationService

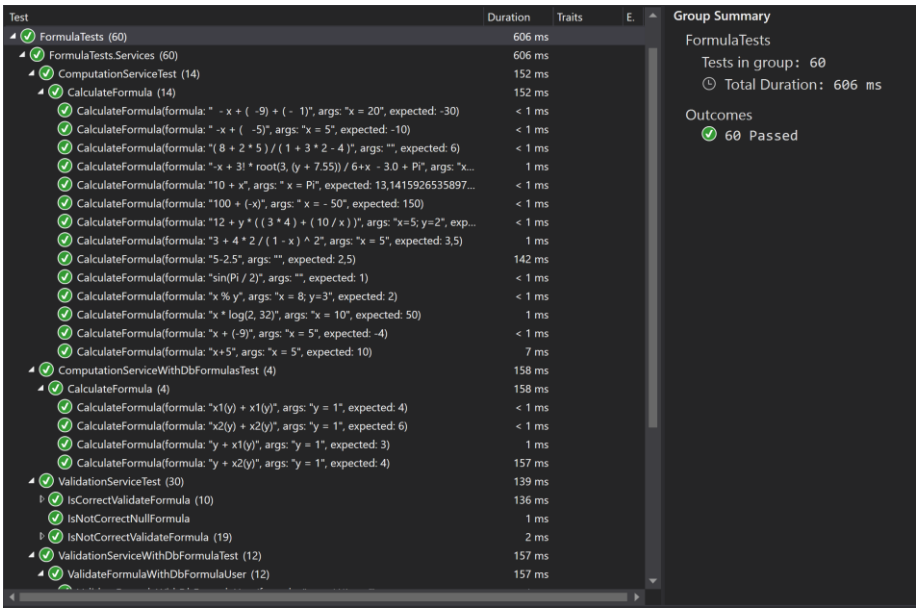


Рисунок 3.29 — Результат запуску всіх тестів

Як можна переконатись за допомогою даних рисунків, рівень покриття тестами високий. Кожен із методів, що містить найважливішу логіку роботи програми був протестовано. Код тестів можна побачити у додатку В.

4 БІЗНЕС-ПЛАН

4.1 Резюме

Розроблене програмне забезпечення надаватиме послуги для людей, яким необхідно виконати певні математичні обчислення та виконувати функцію джерела та опису різноманітних формул.

Цільовою аудиторією, тобто майбутніми клієнтами, є учні, студенти, люди різних професій, яким потрібно здійснювати обчислення та люди, яким знадобиться здійснити обчислення в повсякденному житті.

Для реалізації потрібно залучати 40 000 гривень.

На розробку програмного забезпечення залучено особисті кошти та кошти, отримані від фонду інституту східноєвропейських досліджень.

Планова виручка становить 156 000 гривень за перший рік роботи рік та 13 000 гривень в середньому за місяць.

Організаційно-правова форма майбутнього бізнесу — діяльність особи-підприємця.

Проект реалізується однією людиною.

Рентабельність проекту становить 1.34.

4.2 Маркетинг

Потенційним клієнтам буде запропоновано вебсервіс що спростить виконання рутинних і не тільки обчислень з можливістю додавати шаблони власних формул.

Програмний продукт задовільнятиме такі потреби клієнта як:

- економія часу;
- точність обчислень;
- джерело формул із описом;
- можливість поділитись своїми записами;

- зручність у використанні
- простота використання;
- непотрібність встановлювати програму;
- мобільність.

Розроблений сервіс можна використовувати в найрізноманітніших сферах діяльності, оскільки математичні обчислення є достатньо абстрактними і застосувати їх можна до чого завгодно. Він підійде для людей, яких обчислення це рутинні робочі справи, для специфічних обчислень, шаблон для яких людина може просто створити та просто в буденному житті.

Унікальність відмінність даного вебсайту в тому, що людина може додати шаблон для обчислення формули самостійно. Також передбачена можливість використати будь-яку наявну формулу як змінні в іншій. Результати обчислень можна зберігати, а формулу додати до закладок, що доступного не на всіх вебсайтах для обчислення формул.

Сторінка та форма для введення змінних і обчислення формул одна і змінюється динамічно в залежності від кількості необхідних значень. Введені значення та формули ретельно перевіряються. Обчислення виконуються на основі зворотного польського запису, що забезпечує швидкодію. Сервіс простий у використанні та з інтуїтивно зрозумілим інтерфейсом користувача.

Скористатися даним сайтом користувач зможе знайшовши його за запитом у пошукових системах чи перейшовши за посилання, розміщеним на інших платформах та інформативних групах соціальних мереж.

Програма буде постійно підтримуватись та оновлюватись. При виявленні будь-якого роду недоліків користувач матиме можливість звернутись до цієї підтримки. Потенційними недоліками можуть бути:

- некоректне відображення інтерфейсу користувача в залежності від пристрою;
- відсутність можливості проводити алгоритмічні розрахунки.

В майбутньому до функціоналу програми буде додано можливість виконувати розв’язки різного роду задач із покроковим виведенням розв’язування

та результату. Інтерфейс користувача також буде оновлюватись та покращуватись впродовж підтримки програми.

Ринок збуту для даного програмного забезпечення немає обмежень за географічним фактором. Користувач може проживати в будь-якому населеному пунктів, все що необхідно для користування сервісом це доступ до браузера та інтернет з'єднання. Також немає обмежень щодо віку, статі, соціального стану, рівня знань чи роду діяльності користувачів.

Потенційними клієнтами також є люди та організації, які хочуть прорекламувати свої послуги на вебсайті. Попит на використання даних послуг є стабільним.

Конкурентами даного програмного забезпечення є схожі вебсайти, які також дозволяють виконувати математичні обчислення.

Відмінною рисою таких сайтів є неможливість додати шаблон формули самостійно взагалі, а на деяких їх необхідно замовляти. Також не на всіх вебзастосунках передбачено можливості зберегти розрахунки чи додати в закладки формулу. Використання функціоналу сайтів конкурентів є безкоштовним.

Додавання власної формули користувачем та можливість її поширення поміж решти буде володіти конкурентними перевагами перед іншими вебзастосунками. Завдяки цій функції база даних сайту буде всеохоплюючою, що створює ще одну перевагу над конкурентами.

Відвідування та використання функціоналу сайту буде безкоштовним, прибуток забезпечуватиметься за рахунок рекламних оголошень на ньому та добровільної підтримки від користувачів.

За допомогою оглядів Google Trends можна зрозуміти як часто люди здійснюють пошук певного терміну [26]. Основним пошуковим словом, що напряму пов’язане із даним програмним продуктом є слово calculate (обчислити). На рисунку 4.1 можна побачити графік запитів “calculate ” у світі, а на рисунку 4.2 — графік запитів “обчислити ” в Україні і зрозуміти рівень зацікавленості людей.

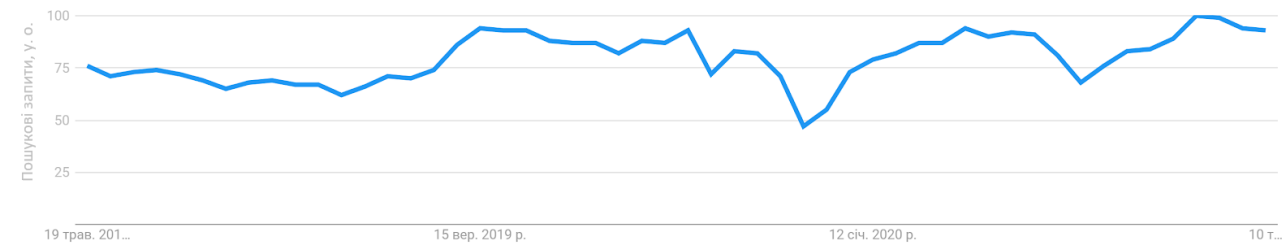


Рисунок 4.1 — графік запитів “calculate ” у світі

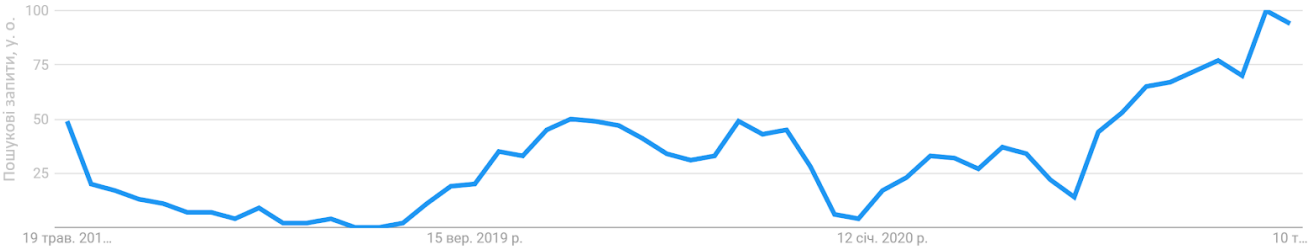


Рисунок 4.2 — графік запитів “обчислити ” в Україні

Найвища точка на графіку — 100 символізує пік популярності, інші числа – популярність запиту відносно піку популярності. Точка нуль — замало інформації про запит.

Зважаючи на дані графіків можна переконатись, що люди дійсно звертаються до інтернет ресурсів за допомогою в обчисленнях. Популярність запиту у світі тримається на високому рівні, а в Україні поступово зростає.

Оскільки прибуток буде залежати на пряму від кількості рівня зацікавленості темою та рівня відвідування сайту, можна зробити наступні прогнози щодо відвідуваності сайту:

- песимістичний (Рис. 4.3);
- реалістичний (Рис. 4.4);
- оптимістичний (Рис. 4.5).

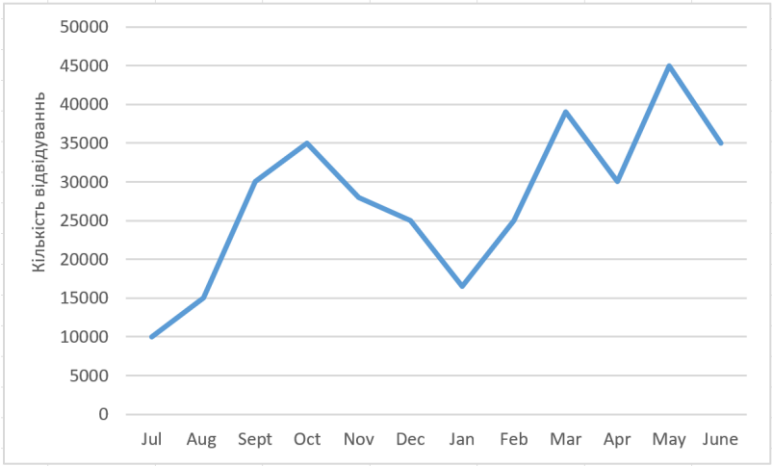


Рисунок 4.3 — Песимістичний прогноз

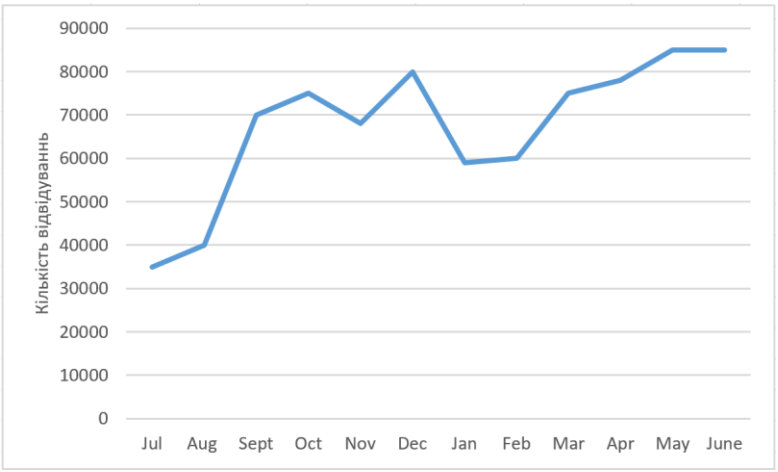


Рисунок 4.4 — Реалістичний прогноз

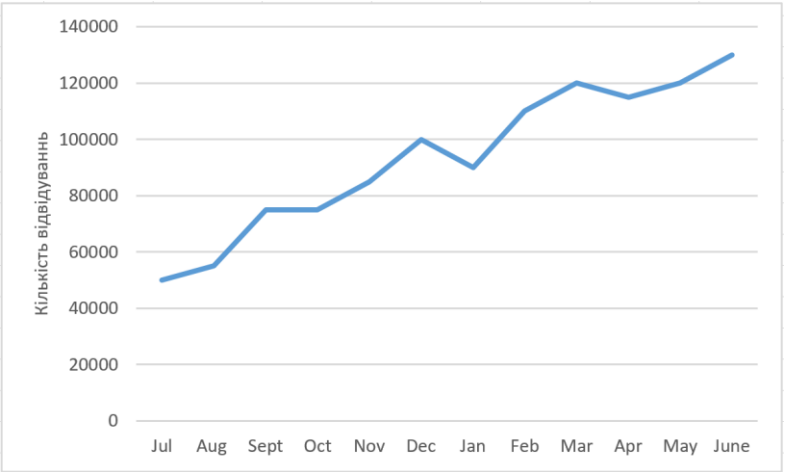


Рисунок 4.5 — Оптимістичний прогноз

Про запуск і старт роботи програми потенційні користувачі зможуть на презентації та з оголошень в соціальних мережах.

Функціонал сайт доступних безкоштовно. Ціна рекламних оголошень буде виставлена з огляду на рівень цін на ринку, відвідуваність сайту користувачами та попитом клієнтів.

На початкове просування вебсайту та рекламу програми буде інвестовано приблизно 15% від загального бюджету.

Рекламуючи товар необхідно буде вказати загальну інформацію про вебсайт з акцентом на його перевагах.

Розміщення реклами можливе в соціальних мережах та різноманітних сайтах і буде відбуватись не тільки на початку запуску програми, але і впродовж підтримки, на це буде відведено 5-10% від щомісячного прибутку.

4.3 Нормативно-правові нюанси

Статус, який найбільше підійде бізнесу — діяльність фізичної особи підприємця.

Незважаючи на те, що фізична особа-підприємець насправді не видом юридичної особи, і через це не має організаційно-правової форми, для спрощення класифікації, відповідний код КОПФГ включений до Класифікатору організаційно-правових форм господарювання [27].

Такий вибір найкраще підходить для початкового етапу діяльності, передбачає нескладну реєстрацію та отримання статусу та надає можливість спрощеної системи оподаткування [28].

Форма бізнесу може бути змінена у майбутньому з огляду на його розвиток.

При потребі професійної підтримки здійснюватимуться консультації з юристом, представниками банку чи бухгалтером.

4.4 Необхідні фінансові вкладення та складання бюджету

До виробничих потужностей, необхідних для розробки програмного забезпечення належить одне приміщення у вигляді кімнати з доступом до електроенергії та один ноутбук достатньої потужності для розробки програмного забезпечення. Також необхідна ліцензія на середовище розробки Visual Studio.

Календарний план реалізації проекту відповідає календарному плану дипломної роботи.

Визначення трудомісткості розробки програмного забезпечення. Трудомісткість програмного забезпечення визначається через суму наступних значень:

- витрати праці на підготовку й опис поставленої задачі — 10 люд/год;
- витрати праці на дослідження алгоритму рішення задачі — 9 люд/год;
- витрати праці на розробку блок-схеми алгоритму — 16 люд/год;
- витрати праці на програмування по готовій блок-схемі — 55 люд/год;
- витрати праці на налагодження програми на ЕОМ — 70 люд/год;
- витрати праці на підготовку документації — 30 люд/год [29].

Трудомісткість даного програмного забезпечення становить 191,5 людино-годин.

На оплату години роботи програміста початківця в середньому необхідно 62,5 грн/год, отже, в межах реалізації даного програмного забезпечення на заплату потрібно виділити 11 968,75 гривень.

Витрати до отримання перших прибутків. До впровадження програмного забезпечення необхідні такі витрати:

- оплата ліцензії ПЗ, необхідного для розробки — 1215 грн/міс;
- витрати на електроенергію — 250 грн/міс;
- витрати на купівлю доменного імені — 324 грн/міс;
- оплата хостингу — 1620 грн/міс.

Орієнтовна сума даних витрати становить 3409 грн/міс. При реалізації даного проекту не потрібно витрачати гроші на закупівлю устаткування чи оренду приміщення.

Амортизаційні відрахування становлять 15% від витрат на розробку програмного забезпечення. На випадок непередбачуваної ситуації відкладено 5% від вкладень, та в подальшому від прибутку.

Орієнтовна тривалість початкового організаційного періоду бізнесу - 1 місяць. Термін отримання дозвільних документів - 1 тиждень.

Фінансування реалізації проекту відбувається до початку реалізації в повному обсязі.

Система оподаткування та розміри податків. На початковому етапі впровадження програмного забезпечення розмір податків, за спрощеною системою оподаткування, становить не більше ніж 10% від мінімальної заробітної плати і становлять 472 гривні на місяць [28].

План очікуваних прибутків. Очікуваний прибуток від реалізації проекту за перший рік становить в середньому 13 000 грн/місяць. В залежності від справдження прогнозів, буде кореговано кількість реклами на сайті, щоб дотримуватись цілі.

Постійними витратами є:

- на хостинг сайту;
- підтримку доменного імені;
- оплату ліцензії програмного забезпечення, необхідного для супровіду та підтримки програми, а також її подальшого вдосконалення;
- на електроенергію;
- на сплату податків.

Сума цих витрат становить приблизно 3 800 грн. На початковому етапі запуску програми витрати на просування та рекламу також постійно присутні і становлять 10% від загального прибутку.

В підсумку протягом першого року від початку реалізації програмного продукту постійні витрати становлять 40% від загального прибутку.

Зекономлені під час розробки гроші буде витрачено на додаткову оплату праці та просування продукту.

Постійний прибуток становить 60% від загального прибутку і дорівнює 7 800 гривень на місяць в середньому за перший рік реалізації програмного продукту та поступово зростатиме.

Розрахунок показників проекту. Загальна сума чистого прибутку за перший на перший рік становить 93 600 гривень.

Рентабельність продукту, як частка чистого доходу на фінансові вкладення в проекту мінус одиниця, становить 1.34. Рентабельність продукту повинна становити більше ніж 0, що справджується.

Таким чином впровадження даного програмного продукту є економічно вигідним.

ВИСНОВКИ

Завданням цього дипломного проекту на тему «Розробка сервісу для математичних обчислень» було створення вебзастосунку для математичних обчислень. Для реалізації даного завдання було проведено аналіз предметної області. Здійснено аналіз переваг та недоліків розглянутих наявних рішень на ринку. Завдяки цьому було визначено критерії, яким повинен відповідати даний продукт, та поставлені чіткі вимоги, що охоплюють питання дизайну, функціоналу та продуктивності програми. Було розроблено моделі поведінки програми та спроектовано модель бази даних.

Для реалізації поставленого завдання було використано методи мови програмування C# та TypeScript, що підходять для ефективної розробки багаторівневої клієнт-серверної архітектури. Базу даних розроблено завдяки функціоналу наданим Entity Framework.

Дотримуючись заданих вимог, було розроблено вебзастосунок для виконання математичних обчислень та виконано поставлені завдання.

Функціонал розробленого програмного продукту буде розширюватись та вдосконалюватись в майбутньому. А саме заплановано додати такі можливості для користувачів, як:

- виконання алгоритмічних задач;
- поступове виведенням розв’язку та проміжних результатів;
- можливість коментувати та обговорювати формули чи задачі;
- можливість конвертації величин;
- покращення інтерфейсу користувача, зокрема можливість обрати темну тему інтерфейсу;
- побудова графіків функцій;
- можливість реєстрації через соціальні мережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Математика. URL:
   https://uk.wikipedia.org/wiki/Математика
   (дата звернення: 13.01.2020)
2. Michiel Hazewinkel. Encyclopaedia of Mathematics. Springer Science & Business Media. ISBN 978-94-015-1239-8, 6 December 2012.
3. Обчислення. URL:
   https://uk.wikipedia.org/wiki/Обчислення
   (дата звернення: 14.01.2020).
4. Автоматизація. URL:
   https://uk.wikipedia.org/wiki/Автоматизація
   (дата звернення: 16.01.2020).
5. MathWorks. MATLAB. URL:
   https://www.mathworks.com/products/matlab.html
   (дата звернення: 28.01.2020).
6. Mathcad. URL:
   https://uk.wikipedia.org/wiki/Mathcad
   (дата звернення: 05.02.2020).
7. Matlab. URL:
   https://clickdown.org/tag/download-mathworks-matlab-r2018a-windows-linux-full-license/
   (дата звернення: 05.02.2020).
8. Кузь М.В., Соловко Я.Т., Андрейко В.М. Методологія формування узагальненого критерію якості програмного забезпечення в умовах невизначеності. Вісник Вінницького політехнічного інституту. Вінниця, 2015. №5. С. 104-107.

9. Віткін Л.М., Кузь М.В., Андрейко В.М. Методика визначення якісних показників програмного забезпечення засобів вимірювальної техніки. Вимірювальна та обчислювальна техніка в технологічних процесах. Хмельницький, 2017. №1 (57). С. 109-113.
10. M. Kozlenko, V. Tkachuk, and M. Dutchak, "Software implementation of microcomputer based intrusion detection and prevention system with binary neural network," in Proc. 2nd International Scientific-Practical Conference "Problems of Cyber Security of Information and Telecommunication Systems" (PCSITS), O. Oksiiuk et al, Eds. Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, Apr. 11-12, 2019, pp. 371-373.
11. Життєвий цикл програмного забезпечення. URL: https://uk.wikipedia.org/wiki/Життєвий_цикл_програмного_забезпечення (дата звернення: 09.02.2020).
12. Діаграма станів (UML). URL: https://uk.wikipedia.org/wiki/Діаграма_станів_(UML) (дата звернення: 14.02.2020).
13. Діаграма послідовності. URL: https://uk.wikipedia.org/wiki/Діаграма_послідовності (дата звернення: 17.02.2020).
14. Lambert M. Surhone; Mariam T. Tennoe; Susan F. Henssonow. Reverse Polish Notation. VDM Publishing. ISBN 978-613-1-25976-0, 15 August 2010.
15. David C. Hay. UML and Data Modeling: A Reconciliation. Technics Publications. ISBN 978-1-935504-19-1, 2011.
16. Azure DevOps Server. URL: https://azure.microsoft.com/en-gb/services/devops/server/ (дата звернення: 19.02.2020).
17. Joachim Rossberg. Agile Project Management with Azure DevOps: Concepts, Templates, and Metrics. Apress. pp. 14–. ISBN 978-1-4842-4483-8, 27 April 2019.
18. Team Foundation Server. URL:

https://uk.wikipedia.org/wiki/Team_Foundation_Server
(дата звернення: 19.02.2020).
19. Gabriel Baptista; Francesco Abbruzzese. Hands-On Software Architecture with C# 8 and .NET Core 3: Architecting software solutions using microservices, DevOps, and design patterns for Azure Cloud. Packt Publishing Ltd. ISBN 978-1-78980-328-0, 29 November 2019.
20. Kenneth Yamikani Fukizi; Jason De Oliveira; Michel Bruchet. Learn ASP.NET Core 3: Develop modern web applications with ASP.NET Core 3, Visual Studio 2019, and Azure, 2nd Edition. Packt Publishing. ISBN 978-1-78961-887-7, 27 December 2019.
21. Valerio De Sanctis. ASP.NET Core 3 and Angular 9: Full stack web development with .NET Core 3.1 and Angular 9, 3rd Edition. Packt Publishing. ISBN 978-1-78961-148-9, 14 February 2020.
22. ADO.NET Entity Framework . URL:
https://ru.wikipedia.org/wiki/ADO.NET_Entity_Framework
(дата звернення: 24.02.2020).
23. CRUD. URL:
https://uk.wikipedia.org/wiki/CRUD
(дата звернення: 25.02.2020).
24. DTO. URL:
https://uk.wikipedia.org/wiki/DTO
(дата звернення: 01.03.2020).
25. Польський інверсний запис. URL:
https://uk.wikipedia.org/wiki/Польський_інверсний_запис
(дата звернення: 03.03.2020).
26. Google Trends. URL:
https://uk.wikipedia.org/wiki/Google_Trends
(дата звернення: 12.03.2020).
27. Фізична особа-підприємець. URL:
https://uk.wikipedia.org/wiki/Фізична_особа-підприємець

(дата звернення: 13.05.2020).

28. Спрощена система оподаткування, обліку та звітності суб'єктів малого підприємництва в Україні. URL: https://uk.wikipedia.org/wiki/Спрощена_система_оподаткування,_обліку_та_звітності_суб%27єктів_малого_підприємництва_в_Україні (дата звернення: 15.05.2020).
29. Економіка, визначення трудомісткості розробки програмного забезпечення - розробка системи обліку. URL: http://jak.bono.odessa.ua/articles/ekonomika-viznachennja-trudomistkosti-rozrobki.php (дата звернення: 16.05.2020).

ДОДАТОК А

Лістинг програми, клас Constant.cs

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

namespace Formula.Utils
{
    /// <summary>
    /// Constant variables
    /// </summary>
    public static class Constant
    {
        /// <summary>
        /// Binary operators and their results
        /// </summary>
        public readonly static Dictionary<string, Action<Stack<double>>> OPERATORS
            = new Dictionary<string, Action<Stack<double>>>
        {
            {"+", stack => stack.Push(stack.Pop() + stack.Pop()) },

            {"-", stack =>
                {
                double temp = stack.Pop();
                stack.Push(stack.Pop() - temp);
                }
            },

            {"*", stack => stack.Push(stack.Pop() * stack.Pop()) },

            {"/", stack =>
                {
                    double temp = stack.Pop();
                    stack.Push(stack.Pop() / temp);
                }
            },

            {"%", stack =>
                {
                    double temp = stack.Pop();
                    stack.Push(stack.Pop() % temp);
                }
            },

            {"^", stack =>
                {
                    double temp = stack.Pop();
                    stack.Push(Math.Pow(stack.Pop(), temp));
                }
            }
        };

        /// <summary>
        /// Priority of operators
        /// </summary>
```

```
        public readonly static Dictionary<string, int> PRIORITY = new Dictionary<string, in
        {
            {"^", 1},
            {"*", 2},
            {"/", 2},
            {"%", 2},
            {"+", 3},
            {"-", 3}
        };

        /// <summary>
        /// List of binary operators
        /// </summary>
        public readonly static List<char> BINARY_OPERATION = new List<char>
        {
            '+', '-', '*', '/', '^', '%'
        };

        /// <summary>
        /// List of unary operators
        /// </summary>
        public readonly static List<char> UNARY_OPERATION = new List<char>
        {
            '+', '-'
        };

        /// <summary>
        /// List of postfix operations
        /// </summary>
        public readonly static Dictionary<string, Action<Stack<double>>> POSTFIX_OPERATION
            = new Dictionary<string, Action<Stack<double>>>
        {
                {"!",
                    stack =>
                    {

                        Func<int, int> fact = null;
                        fact = value => value == 0 ? 1 : fact(value - 1) * value;
                        stack.Push(fact((int)stack.Pop()));
                    }
                 }

        };

        /// <summary>
        /// Symbol show that it is start of formula
        /// </summary>
        public readonly static char START_OF_FORMULA = '#';

        /// <summary>
        /// Symbol show that it is end of formula
        /// </summary>
        public readonly static char END_OF_FORMULA = '$';

        /// <summary>
        /// List of Constant values
        /// </summary>
        public readonly static Dictionary<string, Action<Stack<double>>> CONSTANT_VALUE =
new Dictionary<string, Action<Stack<double>>>
        {
            {"Pi", stack => stack.Push(Math.PI) }
```

```
};

/// <summary>

/// List of basic built-in functions
/// </summary>
public readonly static Dictionary<string, Action<Stack<double>>> BUILT_IN_FUNCTION
    = new Dictionary<string, Action<Stack<double>>>
    {
        {"sin", stack => stack.Push(Math.Sin(stack.Pop())) },

        {"cos", stack => stack.Push(Math.Cos(stack.Pop())) },

        {"tg", stack => stack.Push(Math.Tan(stack.Pop())) },

        {"ctg", stack => stack.Push(1 / Math.Tan(stack.Pop())) },

        {"sec", stack => stack.Push(1 / Math.Cos(stack.Pop())) },

        {"cosec", stack => stack.Push(1 / Math.Sin(stack.Pop())) },

        {"arcsin", stack => stack.Push(Math.Asin(stack.Pop())) },

        {"arccos", stack => stack.Push(Math.Acos(stack.Pop())) },

        {"arctg", stack => stack.Push(Math.Atan(stack.Pop())) },

        {"arcctg", stack => stack.Push(Math.PI / 2 - Math.Atan(stack.Pop())) },

        {"arcsec", stack => stack.Push(Math.Acos(1 / stack.Pop())) },

        {"arccosec", stack => stack.Push(Math.Asin(1 / stack.Pop())) },

        {"sh", stack => stack.Push(Math.Sinh(stack.Pop())) },

        {"ch", stack => stack.Push(Math.Cosh(stack.Pop())) },

        {"th", stack => stack.Push(Math.Tanh(stack.Pop())) },

        {"cth", stack => stack.Push(1 / Math.Tanh(stack.Pop())) },

        {"sech", stack => stack.Push(1 / Math.Cosh(stack.Pop())) },

        {"csch", stack => stack.Push(1 / Math.Sinh(stack.Pop())) },


        {"sqrt", stack => stack.Push(Math.Sqrt(stack.Pop())) },

        {"abs", stack => stack.Push(Math.Abs(stack.Pop())) },

        {"exp", stack => stack.Push(Math.Exp(stack.Pop())) },

        {"ln", stack => stack.Push(Math.Log(stack.Pop())) },

        {"lg", stack => stack.Push(Math.Log10(stack.Pop())) },

        {"lb", stack => stack.Push(Math.Log(stack.Pop(), 2)) },

        {"log", stack =>
                {
                    double temp = stack.Pop();
```

```
                        stack.Push(Math.Log(temp, stack.Pop()));
                    }
                },

                {"root", stack =>
                    {
                        double temp = stack.Pop();
                        stack.Push(Math.Pow(temp, 1.0 / stack.Pop()));
                    }
                },

                {"pow", stack =>
                    {
                        double temp = stack.Pop();
                        stack.Push(Math.Pow(temp, stack.Pop()));
                    }
                },

            };

        /// <summary>
        /// Max history record per user
        /// </summary>
        static public int UserHistoryRecordMaxCount => 20;
    }

}
```

Лістинг програми, клас ValidationService.cs

```
using Formula.Exceptions;

using Formula.Services.Interfaces;

using Formula.Services.Interfaces.Formula;

using Formula.Utils;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Threading.Tasks;


namespace Formula.Services

{

    /// <summary>
```

```
/// Validation for computation formula
/// </summary>
public class ValidationService : IValidationService
{
    private readonly IFormulaInternalService _formulaService;

    /// <summary>
    /// Constructor for getting depedencies
    /// </summary>
    /// <param name="formulaService">Need for validation function arrity</param>
    public ValidationService(IFormulaInternalService formulaService)
    {
        _formulaService = formulaService;

        Validators = new List<Validator>()
        {
            StartValidator,
            OpenBracketValidator,
            CloseBracketValidator,
            BinaryOperationValidator,
            UnaryOperationValidator,
            PostfixOperationValidator,
            DigitValidator,
            IdentificatorValidator
        };

        SimpleValidators = new List<Validator>()
        {
            StartValidator,
            BinaryOperationValidator,
            UnaryOperationValidator,
            PostfixOperationValidator,
            DigitValidator,
            IdentificatorValidator
```

```
            };
        }

        delegate bool Validator(ref int i, string formula, string userId, Stack<char>
brackets);

        /// <summary>
        /// Full list of validators
        /// </summary>
        private readonly List<Validator> Validators;

        /// <summary>
        /// List of validators without Brackets validators
        /// </summary>
        private readonly List<Validator> SimpleValidators;

        private bool IsBracketExsist(string formula)
        {
            var openedBracket = formula.Count(a => a == '(');
            var closedBracket = formula.Count(a => a == ')');

            if (!(openedBracket == closedBracket))
                throw new FormulaValidationException("Open and close brackets count not
equal");

            if (openedBracket == 0) return false;

            return true;
        }

        private void Validate(string formula, string userId)
        {
            bool isBracketExist = IsBracketExsist(formula);

            if (formula.Contains(Constant.START_OF_FORMULA, StringComparison.Ordinal)
```

```
            || formula.Contains(Constant.END_OF_FORMULA, StringComparison.Ordinal))
            throw new FormulaValidationException("Invalid formula pass");

        formula = Constant.START_OF_FORMULA
            + formula.Where(c => !char.IsWhiteSpace(c)).Aggregate("", (str, ch) => str
+= ch)
            + Constant.END_OF_FORMULA;

        Stack<char> brackets = new Stack<char>();

        bool isValidatorFound = false;

        var selectedValidator = (isBracketExist) ? Validators : SimpleValidators;

        for (int i = 0; i < formula.Length - 1; i++)
        {
            isValidatorFound = false;
            foreach (var validator in selectedValidator)
            {
                if (validator(ref i, formula, userId, brackets))
                {
                    isValidatorFound = true;
                    break;
                }
            }

            if (!isValidatorFound)
                throw new ArgumentException("Invalid formula passed");
        }
    }

    private bool StartValidator(ref int i, string formula, string userId, Stack<char>
brackets)
    {
        if (formula[i] != '#')
```

```
                return false;


            new Action(() => throw new FormulaValidationException("Invalid start formula"))
                //formula can start with digit
                .Unless(char.IsDigit(formula[i + 1]))
                //formula can start with letter
                .Unless(char.IsLetter(formula[i + 1]))
                //formula can start with opened bracket
                .Unless(formula[i + 1] == '(')


                //formula can start with unary operation
                .Unless(Constant.UNARY_OPERATION.Contains(formula[i + 1]))
                //excecute validation
                .Invoke();


            return true;
        }


        private bool OpenBracketValidator(ref int i, string formula, string userId,
Stack<char> brackets)
        {
            if (formula[i] != '(')
                return false;


            var pos = i;


            new Action(() => throw new FormulaValidationException($"Invalid symbol at {pos}
pos after open bracket"))
                //opened bracket can be followed by digit
                .Unless(char.IsDigit(formula[i + 1]))
                //opened bracket can be followed by letter
                .Unless(char.IsLetter(formula[i + 1]))
                //opened bracket can be followed by other opened bracket
                .Unless(formula[i + 1] == '(')
                //opened bracket can be followed by unary operation
```

```
                .Unless(Constant.UNARY_OPERATION.Contains(formula[i + 1]))
                //excecute validation
                .Invoke();

            brackets.Push(formula[i]);

            return true;
        }

        private bool CloseBracketValidator(ref int i, string formula, string userId,
Stack<char> brackets)
        {
            if (formula[i] != ')')
                return false;

            var pos = i;

            new Action(() => throw new FormulaValidationException($"Invalid symbol at {pos}
pos after close bracket"))
                //closed bracket can be followed by other closed bracket
                .Unless(formula[i + 1] == ')')
                //closed bracket can be followed by binary operation
                .Unless(Constant.BINARY_OPERATION.Contains(formula[i + 1]))
                //closed bracket can be followed by postfix operation
                .Unless(Constant.POSTFIX_OPERATION.ContainsKey(formula[i +
1].ToString(CultureInfo.InvariantCulture)))
                //closed bracket can be followed by $
                .Unless(formula[i + 1] == Constant.END_OF_FORMULA)
                //excecute validation
                .Invoke();

            if (brackets.Peek() == '(')
            {
                brackets.Pop();
                return true;
            }
```

```
            else
            {
                throw new ArgumentException("Invalid formula passed");
            }
        }

        private bool BinaryOperationValidator(ref int i, string formula, string userId,
Stack<char> brackets)
        {
            if (!Constant.BINARY_OPERATION.Contains(formula[i]))
                return false;

            var pos = i;

            new Action(() => throw new FormulaValidationException($"Invalid symbol at {pos}
pos after binary operation"))
                //binary operation can be followed by digit
                .Unless(char.IsDigit(formula[i + 1]))
                //binary operation can be followed by letter
                .Unless(char.IsLetter(formula[i + 1]))
                //binary operation can be followed by opened bracket
                .Unless(formula[i + 1] == '(')
                //excecute validation
                .Invoke();

            return true;
        }

        private bool UnaryOperationValidator(ref int i, string formula, string userId,
Stack<char> brackets)
        {
            if (!Constant.UNARY_OPERATION.Contains(formula[i]))
                return false;

            var pos = i;
```

```
            new Action(() => throw new FormulaValidationException($"Invalid symbol at {pos}
pos after unary operation"))

                 //unary operation can be followed by digit

                 .Unless(char.IsDigit(formula[i + 1]))

                 //unary operation can be followed by letter

                 .Unless(char.IsLetter(formula[i + 1]))

                 //unary operation can be followed by opened bracket

                 .Unless(formula[i + 1] == '(')

                 //excecute validation

                 .Invoke();


            return true;

        }


        private bool PostfixOperationValidator(ref int i, string formula, string userId,
Stack<char> brackets)

        {

            if
(!Constant.POSTFIX_OPERATION.ContainsKey(formula[i].ToString(CultureInfo.InvariantCulture)))

                return false;


            var pos = i;


            new Action(() => throw new FormulaValidationException($"Invalid symbol at {pos}
pos after postfix operation"))

                //postfix operation can be followed by binary operation

                .Unless(Constant.BINARY_OPERATION.Contains(formula[i + 1]))

                //postfix operation can be followed by closed bracket

                .Unless(formula[i + 1] == ')')

                //postfix operation can be followed by $

                .Unless(formula[i + 1] == Constant.END_OF_FORMULA)

                //excecute validation

                .Invoke();


            return true;

        }
```

```
        private bool DigitValidator(ref int i, string formula, string userId, Stack<char>
brackets)
        {
            if (!char.IsDigit(formula[i]))
                return false;

            while (char.IsDigit(formula[i + 1])) i++;

            if (formula[i + 1] == '.')
            {
                i++;
                while (char.IsDigit(formula[i + 1])) i++;
            }

            var pos = i;

            new Action(() => throw new FormulaValidationException($"Invalid symbol at {pos}
pos after digit"))
                //digit can be followed by closed bracket
                .Unless(formula[i + 1] == ')')
                //digit can be followed by binary operation
                .Unless(Constant.BINARY_OPERATION.Contains(formula[i + 1]))
                //digit can be followed by postfix operation
                .Unless(Constant.POSTFIX_OPERATION.ContainsKey(formula[i +
1].ToString(CultureInfo.InvariantCulture)))
                //digit can be followed by $
                .Unless(formula[i + 1] == Constant.END_OF_FORMULA)
                //excecute validation
                .Invoke();

            return true;
        }

        private bool IdentificatorValidator(ref int i, string formula, string userId,
Stack<char> brackets)
```

```
        {
            if (!char.IsLetter(formula[i]))
                return false;

            var startPos = i;

            while (char.IsLetter(formula[i + 1]) || char.IsDigit(formula[i + 1]))
            {
                i++;
            }

            if (formula[i + 1] == '(')
            {
                return FunctionAnalyzer(startPos, userId, ref i, formula);
            }

            var pos = i;

            new Action(() => throw new FormulaValidationException($"Invalid symbol at {pos}
pos after identificator"))
                //letter can be followed by binary operation
                .Unless(Constant.BINARY_OPERATION.Contains(formula[i + 1]))
                //letter can be followed by closed bracket
                .Unless(formula[i + 1] == ')')
                //letter can be followed by postfix operation
                .Unless(Constant.POSTFIX_OPERATION.ContainsKey(formula[i +
1].ToString(CultureInfo.InvariantCulture)))
                //letter can be followed by $
                .Unless(formula[i + 1] == Constant.END_OF_FORMULA)
                //excecute validation
                .Invoke();

            return true;
        }
```

```
        private void ValidateFunctionArrity(string functionName, int position, int
argsCount, string userId)
        {
            if (Constant.BUILT_IN_FUNCTION.ContainsKey(functionName))
            {
                switch (functionName)
                {
                    case "log":
                    case "root":
                    case "pow":
                        if (argsCount != 2)
                        {
                            throw new FormulaValidationException
                                ($"Invalid function argument count for {functionName} at
{position} pos");
                        }

                        break;
                    default:
                        if (argsCount != 1)
                        {
                            throw new FormulaValidationException
                                ($"Invalid function argument count for {functionName} at
{position} pos");
                        }

                        break;
                }
            }
            else
            {
                if (_formulaService.IsExistFormula(functionName, userId))
                {
                    var record = _formulaService.GetByName(functionName, userId);
                    var recordArgsCount = 0;
```

```
                    if (record.Variables.Length > 0)
                    {
                        recordArgsCount = record.Variables.Count(ch => ch == ';') + 1;
                    }

                    if (argsCount != recordArgsCount)
                    {
                        throw new FormulaValidationException
                            ($"Invalid function argument count for {functionName} at
{position} pos");
                    }
                }
                else
                {
                    throw new FormulaValidationException
                            ($"Function {functionName} not found at {position} pos");
                }
            }
        }

        private bool FunctionAnalyzer(int functionNamePos, string userId, ref int i, string
formula)
        {
            string functionName = formula.Substring(functionNamePos, (i - functionNamePos) +
1);

            var startPos = i + 2;

            int bracketCount = 0;

            List<int> commas = new List<int>();

            int endPos;

            for (endPos = i; endPos < formula.Length - 1; endPos++)
            {
```

```
    if (formula[endPos + 1] == ',')
    {
        if (bracketCount == 1)
            commas.Add(endPos + 1);
    }
    else if (formula[endPos + 1] == '(')
        bracketCount++;
    else if (formula[endPos + 1] == ')')
    {
        bracketCount--;
        if (bracketCount == 0)
            break;
    }
}
endPos++;

List<string> components = new List<string>();

if (commas.Any())
{
    int componentStartPos = startPos;

    foreach (var index in commas)
    {
        components.Add(formula[componentStartPos..index]);
        componentStartPos = index + 1;
    }
    components.Add(formula[componentStartPos..endPos]);
}
else
{
    components.Add(formula[startPos..endPos]);
}
```

```
            i = endPos;

            ValidateFunctionArrity(functionName, functionNamePos, components.Count, userId);

            foreach (var component in components)
            {
                Validate(component, userId);
            }

            return true;
        }

        /// <summary>
        /// Validate formula
        /// </summary>
        /// <param name="formula">Formula</param>
        /// <param name="userId">User Id or null if user not authenticated</param>
        public void ValidateFormula(string formula, string userId)
        {
            if (formula == null)
                throw new NullReferenceException();

            Validate(formula, userId);
        }
    }
}
```

Лістинг програми, клас ComputationService.cs

```
using Formula.Utils;
using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
```

```
using System.Globalization;
using Formula.Services.Interfaces;
using System.Linq;
using Formula.DAL.Models;
using Formula.Services.Interfaces.Formula;
using Formula.Models.Formula.Computation;

namespace Formula.Services
{
    /// <summary>
    /// Service for computation formula
    /// </summary>
    public class ComputationService : IComputationService
    {
        private readonly IValidationService _validationService;
        private readonly IFormulaInternalService _formulaService;

        /// <summary>
        /// Empty computation service constructor
        /// </summary>
        public ComputationService(IValidationService validationService,
IFormulaInternalService formulaService)
        {
            _validationService = validationService;
            _formulaService = formulaService;
        }

        /// <summary>
        /// Replace argument in formula with entered values
        /// </summary>
        /// <param name="formulaRequest">Formula and formula arguments</param>
        /// <returns>Formula with known arguments</returns>
        private string Replace(ComputationRequestDTO formulaRequest)
        {
            var args = formulaRequest.Args.Trim().Split(";");
```

```
            var formula = formulaRequest.Formula.Trim();


            if (string.IsNullOrEmpty(formulaRequest.Args))
            {
                return formula;
            }


            foreach (var arg in args)
            {
                var keyedArg = arg.Split("=");


                formula = formula.Replace(keyedArg[0].Trim(), keyedArg[1].Trim(),
StringComparison.Ordinal);
            }


            formula = formula.Where(c => !char.IsWhiteSpace(c)).Aggregate("", (str, ch) =>
str += ch);


            for (int i = 1; i < formula.Length - 2; i++)
            {
                if (formula[i] == '-' && formula[i + 1] == '-')
                {
                    formula = formula.Replace("--", "+", StringComparison.Ordinal);
                }
            }



            return formula;
        }


        /// <summary>
        /// Add zero before unary operations
        /// </summary>
        /// <param name="replacedFormula">Formula with known arguments</param>
        /// <returns>Formula with known arguments</returns>
```

```
private string AddZeroBeforeUnaryOperation(string replacedFormula)
{
    var formula = replacedFormula;
    int j = 1;

    if (Constant.UNARY_OPERATION.Contains(replacedFormula[0]))
    {
        formula = formula.Insert(0, "0 ");
        j += 2;
    }

    for (int i = 1; i < replacedFormula.Length - 2; i++, j++)
    {
        if ((replacedFormula[i] == '('
            && (Constant.UNARY_OPERATION.Contains(replacedFormula[i + 1]))))
        {
            formula = formula.Insert(j + 1, " 0");
            j += 2;
        }
    }

    return formula;
}
/// <summary>
/// Form an array of formula elements from formula string
/// </summary>
/// <param name="formula">Formula string with known arguments</param>
/// <returns>List of formula elements</returns>
private List<string> FormFormulaArray(string formula)
{
    List<string> formulaArray = new List<string>();

    formula += Constant.END_OF_FORMULA;
```

```
            for (int i = 0; i < formula.Length; i++)
            {
                if (formula[i] == '(' || formula[i] == ')' || formula[i] == ','
                    || Constant.BINARY_OPERATION.Contains(formula[i])
                    || Constant.UNARY_OPERATION.Contains(formula[i])
                    ||
Constant.POSTFIX_OPERATION.ContainsKey(formula[i].ToString(CultureInfo.InvariantCulture)))
                {
                    formulaArray.Add(formula[i].ToString(CultureInfo.InvariantCulture));
                }
                else if (char.IsDigit(formula[i]))
                {
                    var startPossition = i;

                    while (char.IsDigit(formula[i + 1])) i++;

                    if (formula[i + 1] == '.')
                    {
                        i++;
                        while (char.IsDigit(formula[i + 1])) i++;
                    }

                    var endPossition = i;

                    formulaArray.Add(formula.Substring(startPossition, endPossition -
startPossition + 1));
                }
                else if (char.IsLetter(formula[i]))
                {
                    var startPossition = i;

                    while (char.IsLetter(formula[i + 1]) || char.IsDigit(formula[i + 1]))
                    {
                        i++;
                    }
```

```
                    var endPossition = i;

                    formulaArray.Add(formula.Substring(startPossition, endPossition -
startPossition + 1));
                }

            }

            return formulaArray;
        }

        /// <summary>
        /// Tramsform formula into polish notation
        /// </summary>
        /// <param name="formula">List of formula elements</param>
        /// <param name="userId">Null or user Id if user authenticated</param>
        /// <returns>Formula in polish notation</returns>
        private Stack<string> Parse(List<string> formula, string userId)
        {
            Stack<string> finishedFormula = new Stack<string>();
            Stack<string> preparedFormula = new Stack<string>();
            Stack<string> tempStack = new Stack<string>();

            foreach (var f in formula)
            {
                if (f[0] == '(')
                {
                    tempStack.Push(f);
                }
                else if (f[0] == ')')
                {
                    while (!(tempStack.Peek()[0] == '('))
                    {
                        preparedFormula.Push(tempStack.Pop());
```

```
                    }
                    if (tempStack.Peek()[0] == '(')
                    {
                        tempStack.Pop();
                    }
                }
                else if(char.IsDigit(f[0]) || Constant.POSTFIX_OPERATION.ContainsKey(f) ||
Constant.CONSTANT_VALUE.ContainsKey(f))
                {
                    preparedFormula.Push(f);
                }
                else if (Constant.OPERATORS.ContainsKey(f))
                {
                    while (!(tempStack.IsEmpty() || tempStack.Peek()[0] == '(')
                        && (Constant.BUILT_IN_FUNCTION.ContainsKey(tempStack.Peek())
                            || ( Constant.PRIORITY.ContainsKey(tempStack.Peek()) &&
Constant.PRIORITY[tempStack.Peek()] <= Constant.PRIORITY[f])
                            || ( !Constant.PRIORITY.ContainsKey(tempStack.Peek()) &&
char.IsLetter(tempStack.Peek()[0]))))
                    {
                        preparedFormula.Push(tempStack.Pop());


                    }


                    tempStack.Push(f);
                }
                else if (Constant.BUILT_IN_FUNCTION.ContainsKey(f))
                {
                    tempStack.Push(f);
                }
                else if(_formulaService.IsExistFormula(f, userId))
                {
                    tempStack.Push(f);
                }
            }
```

```
            while (!tempStack.IsEmpty())
            {
                preparedFormula.Push(tempStack.Pop());
            }
            while (!preparedFormula.IsEmpty())
            {
                finishedFormula.Push(preparedFormula.Pop());
            }

            return finishedFormula;
        }

        /// <summary>
        /// Processing sub formula from db.
        /// For it we recreate FormulaRequest object and
        /// run full Computate process at end of it we push to stack result of computation.
        /// It allow use subformula in other subformula
        /// </summary>
        /// <param name="formulaRecord"></param>
        /// <param name="data"></param>
        /// <param name="userId">Null or user Id if user authenticated</param>
        private void ProcessInnerFunction(FormulaRecord formulaRecord, Stack<double> data,
string userId)
        {
            //Build arguments for request
            var args = formulaRecord.Variables
                .Split(';')
                .Select(arg => arg + "=" +
data.Pop().ToString(CultureInfo.InvariantCulture))
                .Aggregate((item1, item2) => item1 + ";" + item2);

            //Build request object
            var formulaRequest = new ComputationRequestDTO
            {
                Formula = formulaRecord.View,
```

```
            Args = args
        };

        //Computate formula
        var result =
            Computate(
                Parse(
                    FormFormulaArray(
                        AddZeroBeforeUnaryOperation(
                            Replace(formulaRequest)
                            )
                        )
                    ,
                    userId
                    )
                ,
                userId
                );
        //Store result
        data.Push(result);
    }

    /// <summary>
    /// Computate formula
    /// </summary>
    /// <param name="formula">Formula in polish notation</param>
    /// <param name="userId">Null or user Id if user authenticated</param>
    /// <returns>Formula result</returns>
    private double Computate(Stack<string> formula, string userId)
    {
        Stack<double> data = new Stack<double>();

        while (!formula.IsEmpty())
        {
```

```
                var f = formula.Pop();

                if (char.IsDigit(f[0]))
                {
                    data.Push(double.Parse(f, CultureInfo.InvariantCulture));
                }
                else if (Constant.OPERATORS.ContainsKey(f))
                {
                    Constant.OPERATORS[f](data);
                }
                else if (Constant.BUILT_IN_FUNCTION.ContainsKey(f))
                {
                    Constant.BUILT_IN_FUNCTION[f](data);
                }
                else if (Constant.POSTFIX_OPERATION.ContainsKey(f))
                {
                    Constant.POSTFIX_OPERATION[f](data);
                }
                else if (Constant.CONSTANT_VALUE.ContainsKey(f))
                {
                    Constant.CONSTANT_VALUE[f](data);
                }
                else
                {
                    ProcessInnerFunction(_formulaService.GetByName(f, userId), data,
userId);
                }
            }

            return data.Pop();
        }

        /// <summary>
        /// Calculate formula
        /// </summary>
```

Продовження Додатку А

```
        /// <param name="formulaRequest">Formula and arguments</param>
        /// <param name="requestBy">Null or user Id if user authenticated</param>
        /// <returns>Formula result</returns>
        public ComputationRequestResultDTO CalculateFormula(ComputationRequestDTO
formulaRequest, string requestBy)
        {
            Contract.Assume(formulaRequest != null);

            _validationService.ValidateFormula(formulaRequest.Formula, requestBy);

            return new ComputationRequestResultDTO{
                Result =
                Computate(
                    Parse(
                        FormFormulaArray(
                            AddZeroBeforeUnaryOperation(
                                Replace(formulaRequest)
                                )
                            )
                        ,
                        requestBy
                        )
                    ,
                    requestBy
                    )
            };
        }
    }
}
```

ДОДАТОК Б

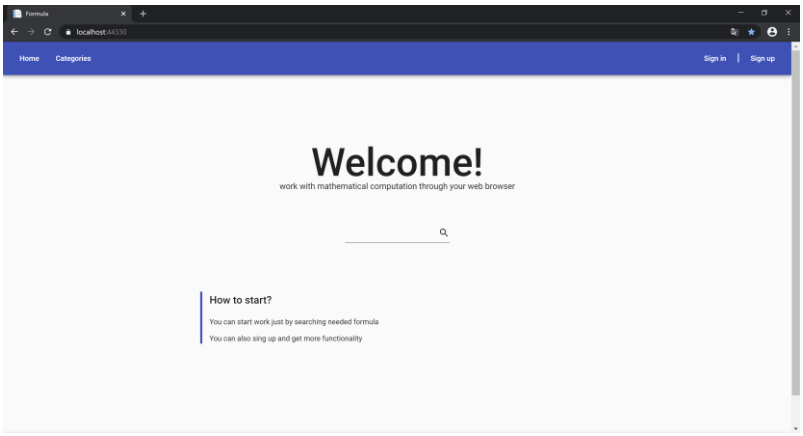


Рисунок Б.1 — Домашня сторінка

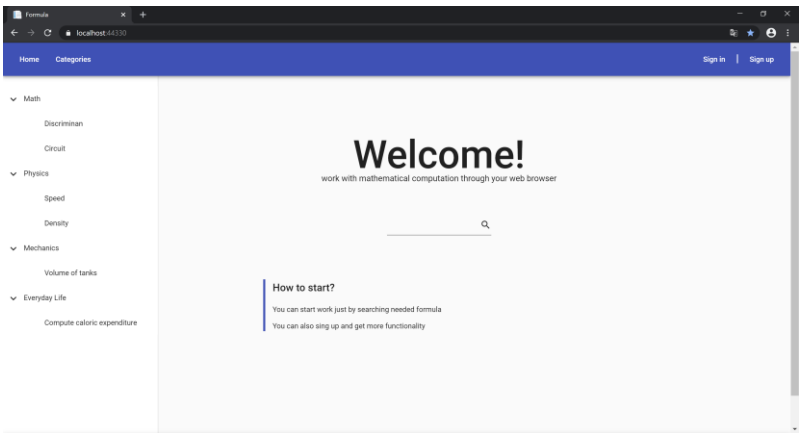


Рисунок Б.2 — Вигляд категорій

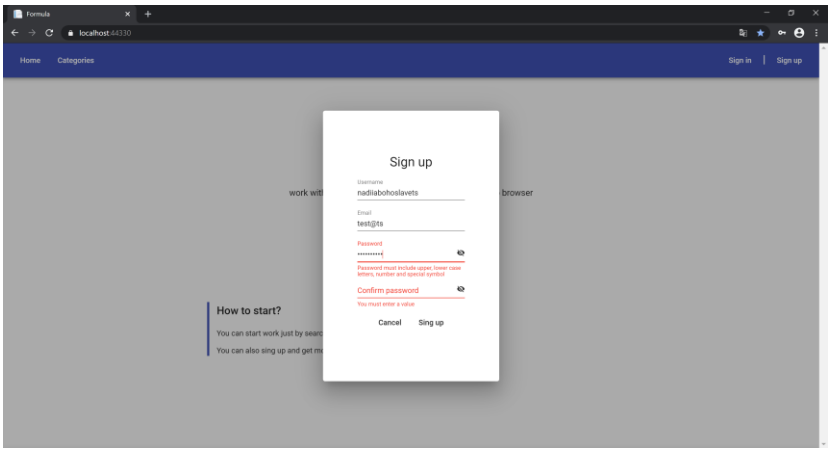


Рисунок Б.3 — Повідомлення про неправильне введення даних

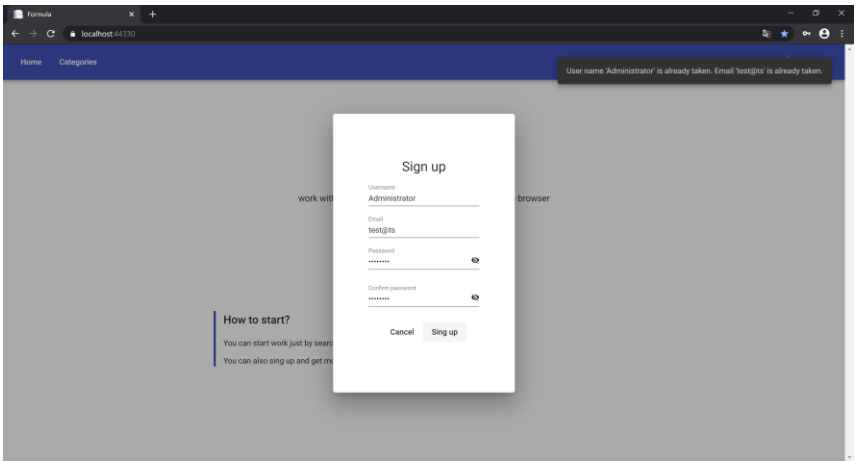


Рисунок Б.4 — Приклад вигляду сповіщення

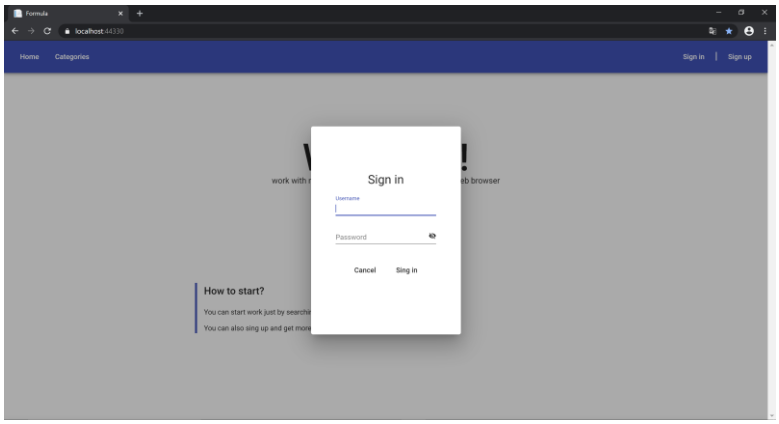


Рисунок Б.5 — Вікно авторизації

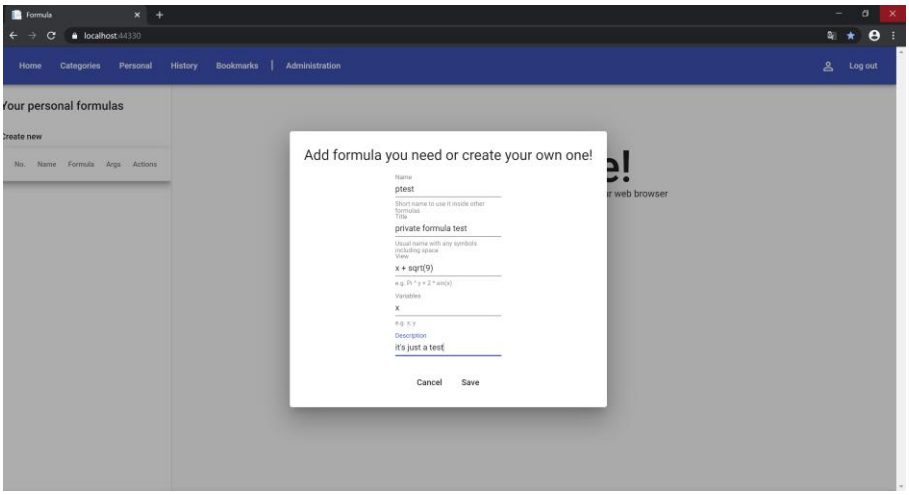


Рисунок Б.6 — Додавання власної формули

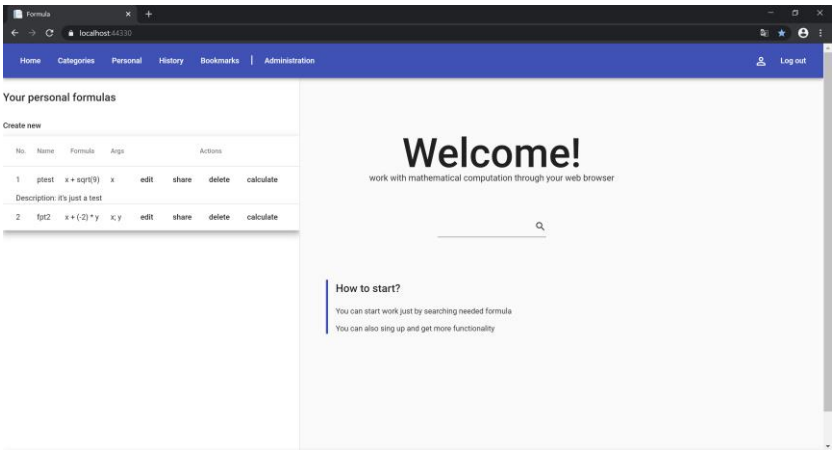


Рисунок Б.7 — Перегляд особистих формул

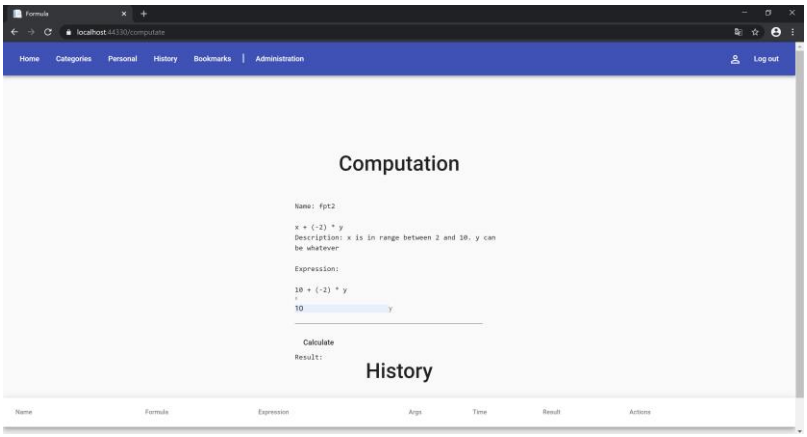


Рисунок Б.8 — Вікно обчислення фомули

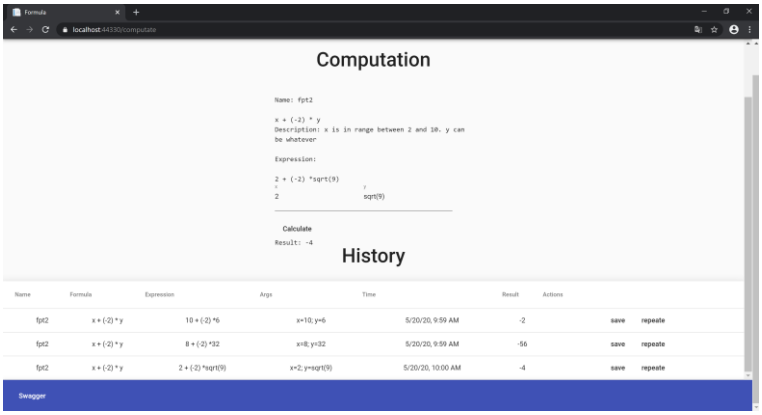


Рисунок Б.9 — Обчислення вкладеної формули та історія

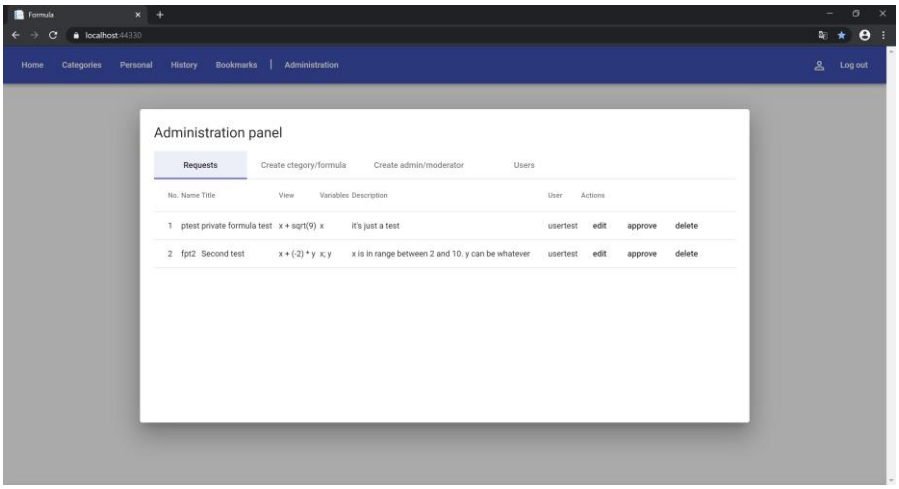


Рисунок Б.10 — Запити на поширення формул

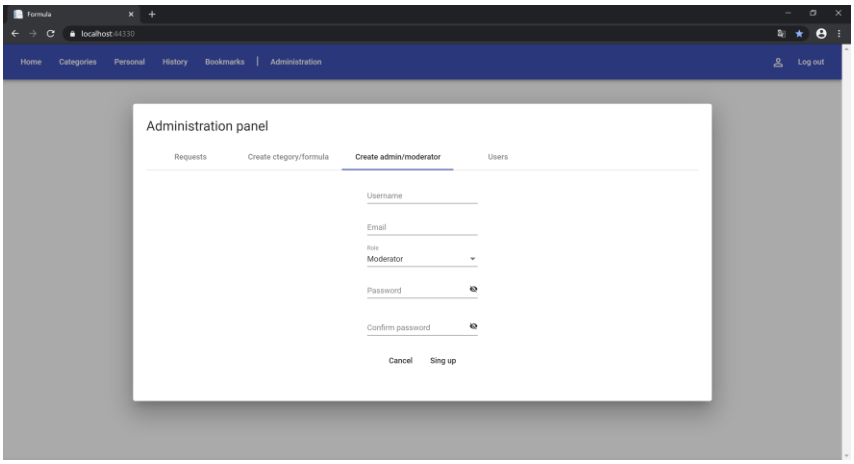


Рисунок Б.11 — Створення модератора, адміністратора

ДОДАТОК В

Лістинг програми, клас ComputationServiceTest.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Xunit;
using Formula.Services;
using Formula.Models;
using Moq;
using Formula.Services.Interfaces.Formula;
using Formula.Models.Formula.Computation;

namespace FormulaTests.Services
{
    public class ComputationServiceTest
    {
        [Theory]
        [InlineData("x+5", "x = 5" , 10.0)] //(simple formula)

        [InlineData(" -x + (   -5)", "x = 5" , -10.0)] //(formula with unary operations)

        [InlineData("12 + y * ( ( 3 * 4 ) + ( 10 / x ) )", "x=5; y=2" , 40)] //(usual
formula)

        [InlineData("( 8 + 2 * 5 ) / ( 1 + 3 * 2 - 4 )", "" , 6)] //(usual formula)

        [InlineData("3 + 4 * 2 / ( 1 - x ) ^ 2", "x = 5" , 3.5)] //(formula with ^)

        [InlineData("5-2.5", "" , 2.5)] //(formula with double)

        [InlineData("x + (-9)", "x = 5" , -4)] //(formula with minus digit)

        [InlineData("x * log(2, 32)", "x = 10" , 50)] //(formula with log)

        [InlineData("-x + 3! * root(3, (y + 7.55)) / 6+x  - 3.0 + Pi", "x = 6; y = 19.45",
Math.PI)] //(complex formula)

        [InlineData("x % y", "x = 8; y=3" , 2)] //(formula with %)

        [InlineData("sin(Pi / 2)", "" , 1.0)] //(formula with sin)

        [InlineData("  - x + (  -9) + ( -  1)", "x = 20", -30)] //(formula with different
spaces)

        [InlineData("100 + (-x)", " x = - 50",  150)] //(formula with minus armument and
minus value )

        [InlineData("10 + x", " x = Pi",  10 + Math.PI)] //(formula with minus armument and
minus value )
        public void CalculateFormula(string formula, string args, double expected)
        {
            //Create formula service mock
            var formulaServiceMock = new Mock<IFormulaInternalService>();
```

```
            formulaServiceMock.Setup(mock => mock.IsExistFormula(It.IsAny<string>(),
It.IsAny<string>())).Returns(false);
            //create computation service
            ComputationService cs = new ComputationService(new
ValidationService(formulaServiceMock.Object), formulaServiceMock.Object);

            ComputationRequestDTO fr = new ComputationRequestDTO{ Formula = formula, Args =
args };

            var result = cs.CalculateFormula(fr, null);
            Assert.Equal(expected, result.Result);
        }
    }
}
```

Лістинг програми, клас ComputationServiceWithDbFormulasTest.cs

```
using Formula.DAL.Models;
using Formula.Models;
using Formula.Models.Formula.Computation;
using Formula.Services;
using Formula.Services.Interfaces.Formula;
using Moq;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Xunit;

namespace FormulaTests.Services
{
    public class ComputationServiceWithDbFormulasTest
    {
        [Theory]
        [InlineData("y + x1(y)", "y = 1", 3)]
        [InlineData("y + x2(y)", "y = 1", 4)]
        [InlineData("x1(y) + x1(y)", "y = 1", 4)]
        [InlineData("x2(y) + x2(y)", "y = 1", 6)]
        public void CalculateFormula(string formula, string args, double expected)
        {
            //Create formula service mock
            var formulaServiceMock = BuildFormulaServiceMock();
            //create computation service
            ComputationService cs = new ComputationService(new
ValidationService(formulaServiceMock.Object), formulaServiceMock.Object);

            ComputationRequestDTO fr = new ComputationRequestDTO{ Formula = formula, Args =
args };
            var result = cs.CalculateFormula(fr, null);
            Assert.Equal(expected, result.Result);
        }

        private readonly List<FormulaRecord> formulaRecords = new List<FormulaRecord>()
        {
            new FormulaRecord{
```

```
                Name = "x1",
                View = "x + 1",
                Description = "Test1",
                Variables = "x"
            },
            new FormulaRecord{
                Name = "x2",
                View = "y + x1(y)", // 1 + 2, 1 + 2
                Description = "Test2",
                Variables = "y"
            },
            new FormulaRecord{
                Name = "x3",
                View = "y + x2(z)",
                Description = "Test1",
                Variables = "z;y"
            },
            new FormulaRecord{
                Name = "x4",
                View = "y + x3(1, z) + x2(1)", // 1 + x3(1, 1) + x2(1)
                Description = "Test1",
                Variables = "z;y"
            }
        };

        private Mock<IFormulaInternalService> BuildFormulaServiceMock()
        {
            //Create formula service mock
            var formulaServiceMock = new Mock<IFormulaInternalService>();
            //Assign lambda as method body
            formulaServiceMock
                 .Setup(mock => mock.IsExistFormula(It.IsAny<string>(), It.IsAny<string>()))
                 .Returns<string, bool>((name, publicy) => formulaRecords.Any(formula =>
formula.Name == name));

            formulaServiceMock
                 .Setup(mock => mock.GetByName(It.IsAny<string>(), It.IsAny<string>()))
                 .Returns<string, bool>((name, publicy) =>
formulaRecords.FirstOrDefault(formula => formula.Name == name));

            return formulaServiceMock;
        }
    }
}
```

Лістинг програми, клас ValidationServiceTest.cs

```
using Formula.Exceptions;
using Formula.Services;
using Formula.Services.Interfaces.Formula;
using Moq;
using System;
using System.Collections.Generic;
using System.Text;
using Xunit;

namespace FormulaTests.Services
{
```

```
    public class ValidationServiceTest
    {
        [Theory]

        [InlineData("x + 1 +(6*(-2))")]          //(simple formula)

        [InlineData("-(t * 2 + (+2))")]          //(simple formula)

        [InlineData("3.659 + v - t")]          //(formula with double)

        [InlineData("x! - f % fx * 3.0 ^ 2")]          //(formula with postfix and binary
operators)

        [InlineData("x! - (f + fx)!")]          //(formula with postfix operator)

        [InlineData("100 + (-x)")]          //(formula with unaty minus)

        [InlineData("x + sin(x + cos(3+2)) + 6")]          //(formula with complex
functions)

        [InlineData("pow(x + y, log(x , pi))")]          //(formula with functions)

        [InlineData("sin(6) + sin(x + 5) + pow(x, 2) + pow(x * y, exp(2 * y))")]
//(complex formula)

        [InlineData("-sin(-6.3551) + sin(x! + (+5!))! + pow(x, 2) /6 + pow(x ^ y, exp(2 *
y))")]     //(complex formula)
        public void IsCorrectValidateFormula(string formula)
        {
            //Create formula service mock
            var formulaServiceMock = new Mock<IFormulaInternalService>();
            formulaServiceMock.Setup(mock => mock.IsExistFormula(It.IsAny<string>(),
It.IsAny<string>())).Returns(false);
            //create validation service
            ValidationService vs = new ValidationService(formulaServiceMock.Object);

            vs.ValidateFormula(formula, null);
        }


        [Theory]

        [InlineData("x + 1 + ( 6 * (-2) ) )")]           //(invalid brackets)

        [InlineData("-( *t * 2 + (+2))")]                  //(binary operation after "(" )

        [InlineData("v - t * 6 /")]                           //(binary operation in the
end)

        [InlineData("6.035.0 + x - u")]                    //(invalid digit)

        [InlineData("x + sin(x, yi!(3+2)) + 6")]         //(invalid use of postfix
operation)

        [InlineData(") 5 + s")]                           //(start with ")")

        [InlineData("> 8 * j")]                            //(start with binary operation)

        [InlineData("-x + 3! * root(3, (y, + 7.55)) / 6+x  - 3.0 + Pi")]     //(invalid
comma)
```

```
        [InlineData("")]                //(empty formula)

        [InlineData(" ")]               //(only space)

        [InlineData("!5 + 8")]               //(start with postfix opeartion)

        [InlineData("# k + 8 ")]               //(contains # symbol)

        [InlineData("$ k + 8 ")]               //(contains $ symbol)

        [InlineData("= + 5 +  8")]             //(contains invalid symbol)

        [InlineData("~ + 5 +  8")]             //(contains invalid symbol)

        [InlineData("₴ + 5 +  8")]             //(contains invalid symbol)

        [InlineData("'")]             //(contains invalid symbol)

        [InlineData(", +5 + 8)")]               //(contains comma outside function)

        [InlineData("- / f + b")]               //(binary after unary/binary operation)
        public void IsNotCorrectValidateFormula(string formula)
        {
            //Create formula service mock
            var formulaServiceMock = new Mock<IFormulaInternalService>();
            formulaServiceMock.Setup(mock => mock.IsExistFormula(It.IsAny<string>(),
It.IsAny<string>())).Returns(false);
            //create validation service
            ValidationService vs = new ValidationService(formulaServiceMock.Object);

            Assert.Throws<FormulaValidationException>(() => vs.ValidateFormula(formula,
null));
        }

        [Fact]
        public void IsNotCorrectNullFormula()
        {
            //Create formula service mock
            var formulaServiceMock = new Mock<IFormulaInternalService>();
            formulaServiceMock.Setup(mock => mock.IsExistFormula(It.IsAny<string>(),
It.IsAny<string>())).Returns(false);
            //create validation service
            ValidationService vs = new ValidationService(formulaServiceMock.Object);

            Assert.Throws<NullReferenceException>(() => vs.ValidateFormula(null, null));
        }

    }
}
```

Лістинг програми, клас ValidationServiceWithDbFormulaTest.cs

```
using Formula.DAL.Models;
using Moq;
using System;
```

```
using System.Collections.Generic;
using System.Text;
using System.Linq;
using Xunit;
using Formula.Services;
using Formula.Services.Interfaces.Formula;

namespace FormulaTests.Services
{
    public class ValidationServiceWithDbFormulaTest
    {
        private Mock<IFormulaInternalService> BuildFormulaServiceMock()
        {
           //Create formula service mock
           var formulaServiceMock = new Mock<IFormulaInternalService>();
           //Assign lambda as method body
           formulaServiceMock
                .Setup(mock => mock.IsExistFormula(It.IsAny<string>(), It.IsAny<string>()))
                .Returns<string, bool>((name, publicy) => formulaRecords.Any(formula =>
formula.Name == name));

           formulaServiceMock
                .Setup(mock => mock.GetByName(It.IsAny<string>(), It.IsAny<string>()))
                .Returns<string, bool>((name, publicy) =>
formulaRecords.FirstOrDefault(formula => formula.Name == name));

           return formulaServiceMock;
        }

        [Theory]
        [InlineData("x + x1(1) + y")] //With const arg
        [InlineData("x + x1(x) + y")] //With one arg
        [InlineData("x + x1(y) + y")] //With one arg
        [InlineData("x + x2(2) + y")] //With const arg
        [InlineData("x + x2(x) + y")] //With one arg
        [InlineData("x + x2(y) + y")] //With one arg
        [InlineData("x + x3(y, x) + y")] //With two arg
        [InlineData("x + x3(1, x) + y")] //With const and arg
        [InlineData("x + x3(1, 3) + y")] //With two consts
        [InlineData("x + x4(x, y) + y")] //With two arg
        [InlineData("x + x4(1, y) + y")] //With const arg
        [InlineData("x + x4(1, 2) + y")] //With two consts
        public void ValidateFormulaWithDbFormulaUser(string formula)
        {
            var formulaServiceMock = BuildFormulaServiceMock();
            //create validation service
            var vs = new ValidationService(formulaServiceMock.Object);

            vs.ValidateFormula(formula, null);
        }

        private readonly List<FormulaRecord> formulaRecords = new List<FormulaRecord>()
        {
            new FormulaRecord{
                Name = "x1",
                View = "x + 1",
                Description = "Test1",
                Variables = "x"
            },
            new FormulaRecord{
                Name = "x2",
```

Продовження Додатку В

```
                View = "x + x1(x)",
                Description = "Test2",
                Variables = "x"
            },
            new FormulaRecord{
                Name = "x3",
                View = "y + x2(x)",
                Description = "Test1",
                Variables = "x;y"
            },
            new FormulaRecord{
                Name = "x4",
                View = "y + x3(1, x) + x2(1)",
                Description = "Test1",
                Variables = "x;y"
            },
        };
    }
}
```