МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

“До захисту допущена”
Зав. кафедри комп'ютерних наук
та прикладної математики
Юрій Васильович Турбал
“__” _____20_р.

КВАЛІФІКАЦІЙНА РОБОТА

Інформаційна система розрахунку родинного бюджету

Виконав:
Морозюк Дмитро Юрійович,
група ІПЗ-41
Керівник:
Прищепа О.В., к.ф.-м.н., доцент

Рівне-2024

ЗМІСТ

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
РЕФЕРАТ 3
ВСТУП 4
РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРОБЛЕМИ 6
1.1. Чому важливо аналізувати родинний бюджет 6
1.2. Аналіз готових рішень на ринку 8
РОЗДІЛ 2
РОЗРОБКА ЗАСТОСУНКУ 10
2.1. ASP.NET 10
2.1.1. Моделі додатків ASP.NET 11
2.1.2. Переваги ASP.NET Web API 13
2.1.3. Entity Framework 14
2.1.4. Identity 18
2.1.5. Шар бізнес логіки 21
2.2. React 24
2.2.1. SPA 24
2.2.2. React 26
2.2.3. Функціональні компоненти 28
2.3. Принципи, які реалізовані при виконанні проєкту 31
2.3.1. Нормалізація баз даних 31
2.3.2. ACID принципи 33
2.3.3. SOLID принципи 37
2.3.4. Ін’єкція залежностей (DI) 41
РОЗДІЛ 3.
ТЕСТУВАННЯ РЕАЛІЗОВАНОЇ СИСТЕМИ 43
3.1. Авторизація та автентифікація 43
3.2. Основна функціональність сайту 45
3.2.1. Додавання покупок та доходів 45
3.2.2. Додавання категорій та доступів 50
3.2.3. Статистика та звіти 54
ВИСНОВКИ 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 58

РЕФЕРАТ

Кваліфікаційна робота: 60 с., 41 рисунок, 11 джерел, 1 додаток.

Мета роботи: розробка програмного забезпечення для можливості аналізу родинного бюджету.

Об'єкт дослідження: sнформаційна система управління фінансами для сімейного використання.

Предмет дослідження: управління особистими фінансами за допомогою інформаційної системи.

Методи вивчення: технологія ASP.NET, бібліотека React, MUI, Epplus, фреймворки Identity, Entity Framework.

Реалізована інформаційна система дає можливість користувачам аналізувати їх родинний бюджет, що є досить актуальним, адже більшість наявних і відомих додатків дозволяють аналізувати лише власний бюджет.

Ключові слова: бюджет, інформаційна система, ASP.NET, Entity Framework, Web-API, React, MUI.

ВСТУП

Для досягнення власних фінансових цілей потрібно постійно досліджувати власні покупки та доходи. Багато сімей мають на меті фінансову стабільність та незалежність, для того щоб щасливо проживати на нашій планеті і саме тому додатки та вебресурси для аналізу сімейного бюджету є актуальними сьогодні і будуть такими залишатись надалі.

Такі додатки мали б покращувати та полегшувати сімейне життя громадян нашої країни та й всіх країн загалом. Однією з таких рутинних для сімей справ, які можна покращити – планування та аналіз родинного бюджету. Ця справа є важливою через те, що аналіз сімейного бюджету має вирішальне значення для підтримки фінансового здоров'я та досягнення фінансових цілей. Регулярний аналіз сімейного бюджету є активним кроком до досягнення фінансової стабільності та безпеки. Це дає вам змогу приймати обґрунтовані рішення, визначати пріоритети своїх витрат і впевнено працювати над досягненням своїх фінансових цілей. Розуміючи та контролюючи свої фінанси, ви можете створити більш безпечне та процвітаюче майбутнє для своєї родини та країни.

Вебресурс розроблений за допомогою кількох технологій. Серверна частина буде імплементована на основі ASP.NET платформи, використовуючи Entity Framework як ORM (об'єктно-реляційне відображення або спосіб зв'язку між кодом на бекенді та базою даних), Identity для аутентифікації та авторизації користувачів, бібліотека Epplus для роботи з excel файлами. Клієнтська частина розроблена за допомогою бібліотеки React з підходом функціональних компонент та використанням компонентів MUI.

Даний вебресурс є вашим ідеальним компаньйоном для легкого та впевненого управління сімейними фінансами. Незалежно від того, чи збираєтеся ви відкладати кошти на велику сімейну відпустку, плануєте навчання своїх дітей чи просто намагаєтеся зводити кінці з кінцями щомісяця, цей сайт надає

інструменти та ресурси, необхідні для контролю над вашим фінансовим майбутнім та має наступну функціональність:

- користувачі могли безпечно занотовувати свої покупки та доходи, присвоювати їм певні категорії;
- ділитись своїми списками з іншими користувачами (членами сім'ї) у двох доступах (перший рівень доступу дозволяє ділитись своїми доходами та витратами з можливістю редагування, другий рівень доступу має можливість лише перегляду та аналізу цих даних, який чудово підійде для дітей);
- аналізувати річний бюджет родини – у який місяць сім'я витратила найбільше грошей, у якому вийшло заробити та відкласти найбільшу кількість коштів;
- аналізувати місячний бюджет – скільки грошей було витрачено на певні категорії покупок і скільки отримано грошей в різних категоріях;
- можливість отримати звіт у вигляді excel файлу, який містить інформацію про кожну витрату та дохід в кожному місяці вибраного року та додатково діаграму, яка базується на даних у excel файлі.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРОБЛЕМИ

1.1. Чому важливо аналізувати родинний бюджет

Регулярний аналіз сімейного бюджету має важливе значення для забезпечення фінансового благополуччя та досягнення ваших фінансових цілей.

Для того, щоб отримати фінансовий контроль у сім'ї, потрібно розуміти моделі витрат та уникати перевитрат. Аналіз вашого бюджету допоможе вам визначити, куди йдуть ваші гроші. Відстежуючи свої витрати, ви можете виявити закономірності та визначити, чи ваші витрати відповідають вашим пріоритетам і цінностям. Також регулярний перегляд вашого бюджету допоможе вам залишатися в межах ваших можливостей. Це гарантує, що ви не витрачаєте більше, ніж заробляєте, що може запобігти накопиченню боргів і фінансовому стресу.

За допомогою аналізу власних доходів можна визначити можливості заощаджень. Наприклад, шляхом перерозподілу коштів від непотрібних витрат ви можете збільшити внески на ощадні рахунки, пенсійні фонди або інші фінансові цілі. Ретельний аналіз бюджету може виявити області, де можна скоротити. Незалежно від того, чи йдеться про скорочення відвідування ресторанів, скасування невикористаних підписок або пошук дешевших альтернатив для певних витрат, виявлення цих можливостей може призвести до значної економії.

Аналіз бюджету дає змогу встановлювати реалістичні фінансові цілі та відстежувати прогрес. Незалежно від того, чи це заощадження на відпустку, нову машину чи перший внесок на будинок, наявність чіткого бюджету допоможе вам залишатися зосередженим і мотивованим. Регулярний аналіз бюджету допоможе вам спланувати майбутні витрати, як-от витрати на освіту, надзвичайні ситуації або великі покупки, гарантуючи, що ви будете фінансово підготовлені, коли

прийде час. Можливість створення фонду надзвичайних ситуацій за рахунок знання того, куди йдуть ваші гроші щомісяця, дає змогу розподіляти кошти на рахунок надзвичайних заощаджень, забезпечуючи фінансову подушку для несподіваних витрат. Визначивши та усунувши марнотратні витрати, ви можете виділити більше грошей на погашення заборгованості, скорочення процентних платежів і покращення загального фінансового стану.

Спільний аналіз сімейного бюджету заохочує відкрите спілкування щодо грошових питань, сприяючи розвитку почуття командної роботи та спільної відповідальності між членами сім'ї. Також додаткові дивіденди можна отримати за допомогою залучення дітей до процесу бюджетування навчає їх цінним фінансовим навичкам і важливості відповідального управління грошима з раннього віку.

Такі життєві події, як зміна роботи, витрати на лікування або додавання нового члена сім'ї, можуть вплинути на ваше фінансове становище. Регулярний перегляд бюджету допоможе вам адаптуватися до цих змін і відповідно скорегувати свої фінансові плани. Гнучкий бюджет дозволяє швидко реагувати на зміни в доходах або витратах, забезпечуючи фінансову стабільність навіть у невизначені часи.

Ще однією перевігою аналізу сімейного бюджету є чітке розуміння своїх фінансів, яке може зменшити занепокоєння та стрес, пов'язані з управлінням грошима, що призведе до кращого психічного та емоційного благополуччя. Фінансова стабільність дозволяє вам зосередитися на інших важливих аспектах життя, таких як здоров'я, освіта та особистий розвиток, сприяючи більш збалансованому та повноцінному життю.

Отже, активна участь у процесі регулярного перегляду сімейного бюджету є ключем до забезпечення фінансової стабільності та впевненості. Ця практика дозволяє приймати обґрунтовані рішення, визначати пріоритети витрат і

ефективно досягати своїх фінансових цілей. Отримання чіткого розуміння та контролю над своїми фінансами прокладає шлях до більш безпечного та процвітаючого майбутнього для вашої родини.

1.2. Аналіз готових рішень на ринку

У сфері розподілу родинного бюджету не так і багато існує вже готових рішень, адже це є досить специфічна галузь додатку. Проте існуючі додатки мають певні схожості з даним розробленим програмним забезпеченням: можливість додавати свої категорії, покупки та доходи, порівнювати та аналізувати різні діаграми прибутків та витрат.

Основним плюсом цього програмного забезпечення над іншими – є те, що інші додатки не дозволяють ділитись власними доходами та покупками з іншими користувачами. Це є досить важливою функцією, адже обмін фінансовою інформацією між членами сім'ї може значно покращити управління фінансами та єдність сім'ї. Застосовуючи практику розподілу доходів і витрат, сім'ї можуть створити більш згуртовану та ефективну стратегію управління фінансами, забезпечуючи колективні зусилля для досягнення фінансової стабільності та добробуту.

Окрім того, що можливість ділитись власними доходами та витратами між членами родини дозволяє ефективніше планувати надзвичайні ситуації, сприяє здоровому обговоренню фінансових пріоритетів, викликів і рішень, сприяючи кращим відносинам з грошима, також цей функціонал допоможе залучати дітей до розподілу бюджету і у результаті діти будуть мати кращу фінансову грамотність аналізуючи витрати та доходи батьків.

Додатковим плюсом даного вебресурсу є в тому, що він дозволяє створювати докладні звіти у форматі Excel для комплексного аналізу в автономному режимі. Ідеально підходить для користувачів, які віддають перевагу працювати зі своїми даними в автономному режимі. Також перевага

даного ресурсу над іншими є в тому, що він і весь доступний функціонал є безкоштовними та без реклами, які можуть відволікати або перебивати користувацький досвід.

Отже, даний додаток буде містити досить вагому перевагу у своєму арсеналі у вигляді функціоналу, який дозволить ділитись власними доходами та витратами з іншими користувачами у двох рівнях доступу: перший – користувач, з яким поділились доходами, буде мати можливість редагувати їх; другий – користувач зможе лише переглядати доходи та аналізувати їх, без можливості редагування.

РОЗДІЛ 2
РОЗРОБКА ЗАСТОСУНКУ

2.1. ASP.NET

ASP.NET – це веб-фреймворк із відкритим вихідним кодом, розроблений Microsoft для створення сучасних веб-додатків і сервісів. Він є частиною платформи .NET і дозволяє розробникам створювати динамічні, надійні та масштабовані веб-додатки. ASP.NET підтримує різні моделі програмування, включаючи MVC (Model-View-Controller), Web API, Web Pages і Blazor [2].

Однією з переваг ASP.NET, яка змушує розробників обирати цей фреймворк для створення власних сайтів є те, що ASP.NET пропонує уніфікований досвід розробки для створення веб-інтерфейсів API, динамічних веб-додатків і мікросервісів, спрощуючи процес, зменшуючи потребу у використанні кількох фреймворків. Також цей фреймворк відомий своєю високою продуктивністю, масштабованістю та надійністю, оптимізований для сучасних веб-додатків і ефективно обробляє великий трафік.

ASP.NET фреймворк містить безліч інструментів, як вбудованих так і тих, які можна підключати, для покращення та полегшення роботи команди, яка імплементує сайт. Одним з таких інструментів включає в себе вбудовані функції для автентифікації, авторизації, захисту даних і захисту програм від поширених вразливостей він підтримує галузеві стандартні протоколи, такі як OAuth і OpenID Connect[2]. Даний фреймворк має чудову підтримку механізму ін’єкції залежностей за допомогою вбудованого контейнера для реєстрування залежностей, сприяючи розробці слабозв’язаного та тестованого коду. У свою чергу сторінки Razor надають модель кодування на основі сторінок, яка спрощує створення веб-додатків і підвищує продуктивність, особливо в сценаріях, коли перегляд, орієнтований на сторінку, найбільш прийнятний.

Кросплатформеність за допомогою ASP.NET Core, яка є останньою версією ASP.NET, дозволяє безперебійно працювати у Windows, macOS і Linux. Ця універсальність дозволяє розробникам створювати та легко розгортати програми в різних операційних системах, що нівелює різницю між основними ОС та їхніми дитрибутивами.

Отже, ASP.NET виділяється як надійна структура, яка є ідеальною для розробки сучасних веб-додатків і сервісів. Його виняткова продуктивність, здатність працювати на кількох платформах і великий набір функцій роблять його улюбленим варіантом для розробників, які прагнуть створювати масштабовані веб-програми, які можна підтримувати. ASP.NET надає вам необхідні інструменти та гнучкість для досягнення успіху, незалежно від того, чи працюєте ви над невеликим веб-сайтом чи над масштабним корпоративним рішенням.

2.1.1. Моделі додатків ASP.NET

ASP.NET MVC – це структура, яка використовує шаблон проектування Model-View-Controller для створення веб-додатків. Вона ідеально підходить для додатків, які потребують чіткого розподілу проблем і підтримки складних інтерфейсів користувача. Багато стартапів і організацій різного розміру обирають ASP.NET MVC Core для розробки своїх бізнес-рішень. Ця структура дозволяє їм створювати високоякісне програмне забезпечення, яке є ресурсо ефективним і економічно ефективним для обслуговування. Крім того, бездоганна інтеграція з ASP.NET Core підвищує ефективність основних бізнес-операцій, що робить його кращим вибором для компаній, які прагнуть оптимізувати свої процеси [2].

ASP.NET Web API – фреймворк призначений для створення служб HTTP, до яких може отримати доступ широкий спектр клієнтів, наприклад браузери та мобільні пристрої. Він ідеально підходить для розробки RESTful-сервісів, які

підтримують механізми кешування для покращення швидкості відповіді сервера, покращують масштабування за рахунок того, що запити містять усю інформацію серверу для його обробки [2].

ASP.NET Web Pages – легкий фреймворк, призначений для створення простих веб-додатків із односторінковою моделлю [2]. Він використовує синтаксис Razor для вбудовування серверного коду безпосередньо в HTML-сторінки. Razor містить у собі вбудовані функції для пов'язання моделей, легкі способи для відправки форм та відносно легкий синтаксис, який допомагає новачкам легко освоїти даний фреймворк.

Для розробки цієї програми була використана ASP.NET Web API модель, схема:

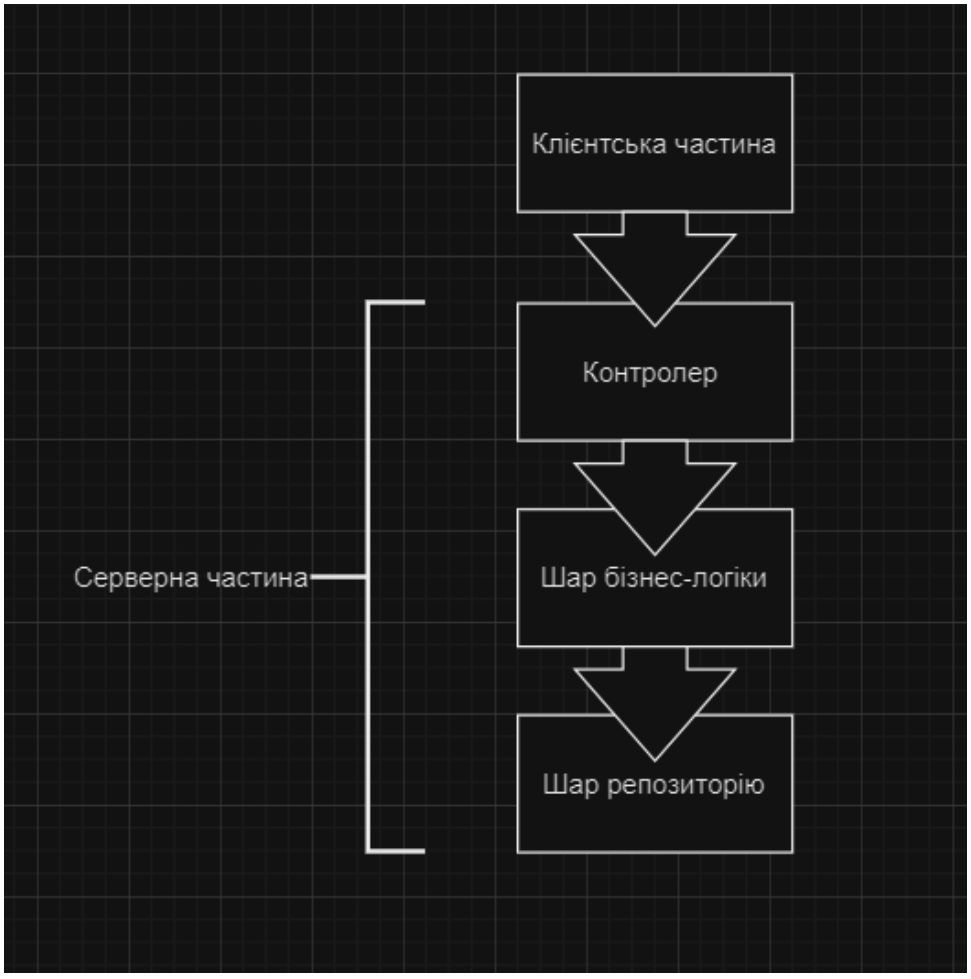


Рис. 2.1. Діаграма програмного забезпечення

2.1.2. Переваги ASP.NET Web API

Для розробки цієї програми була використана ASP.NET Web API модель через переваги її використання та її характеристики. ASP.NET Web API полегшує створення HTTP-сервісів, доступних для широкого кола клієнтів, включаючи браузери, мобільні пристрої та інші веб-програми та пропонує простий підхід до створення сервісів RESTful, що забезпечує безперебійне спілкування та обмін даними між сервером і клієнтами [10].

Маршрутизація ASP.NET Web API є дуже універсальною, тому це дозволяє визначати маршрути, які відображають запити HTTP на різні контролери та дії. Цей аспект може досить легко бути сконфігурованим задля створення унікальної системи маршрутизації. Один зі способів налаштування за допомогою маршрутизації атрибутів ви можете вказувати маршрути безпосередньо в діях контролера, забезпечуючи більший контроль і підвищуючи читабельність маршруту та зручність обслуговування. Також платформа має способи для налаштування формату відповіді за допомогою метаданих та атрибутів (JSON, XML тощо), а також вона автоматично може визначати дані формати для кращої типізації даних для повернення.

Іншими перевагами ASP.NET Web API є відносно легкий спосіб автоматизованого тестування. Розробка як Unit, так і Integration тестів є досить легкою через вбудований контейнер для механізму впровадження залежностей, тому підвищується забезпечення правильності роботи API. Також ASP.NET Web API підтримує асинхронне програмування за допомогою async і await, підвищуючи масштабованість і продуктивність вашого API шляхом ефективного керування операціями вводу-виводу.

Однією з основних ознак Web API є контролери, до яких доступаються фронт-енд запити за інформацією. Один з прикладів таких контролерів:

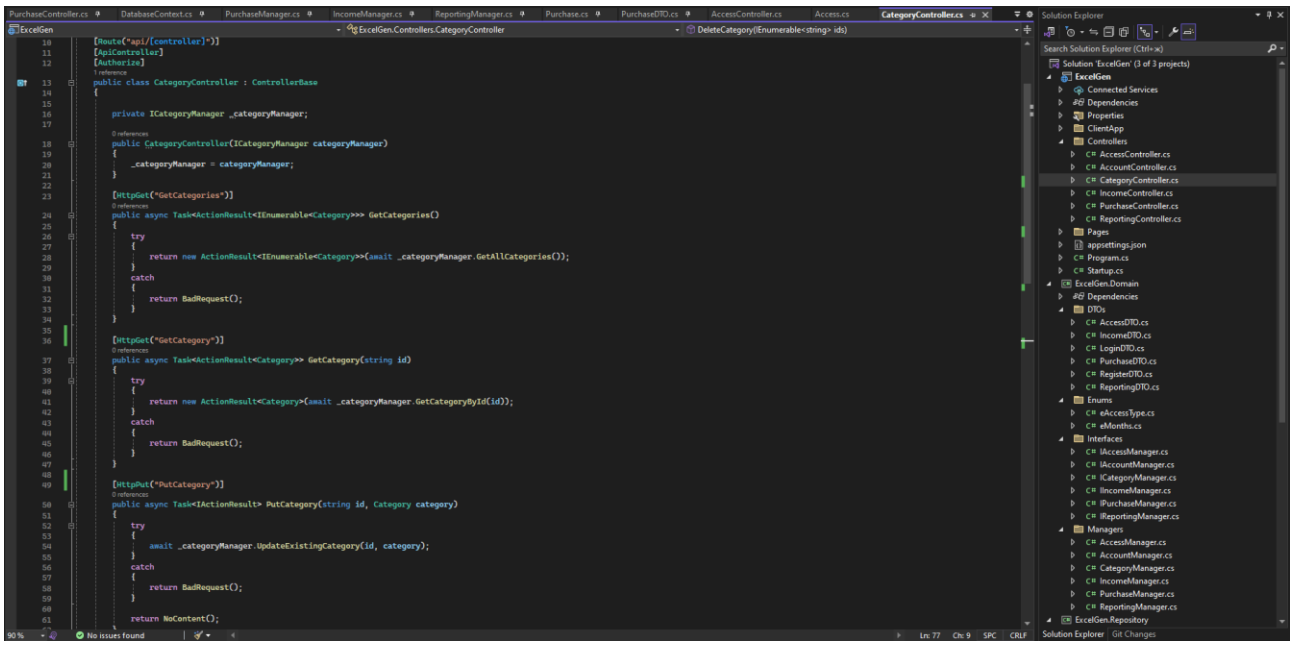


Рис. 2.2. Один з контролерів у додатку

2.1.3. Entity Framework

Entity Framework (EF), фреймворк із відкритим вихідним кодом, розроблений Microsoft, створив справжню революцію у способах взаємодії розробників із реляційними базами даних (SQL). Використовуючи строго типізовані об’єкти .NET, EF плавно усуває проблему зв’язку між кодом програми та таблицями бази даних. Цей підхід суттєво скорочує стандартний код, який традиційно потрібен для доступу до даних, дозволяючи розробникам більше зосереджуватися на логіці додатків і менше на тонкощах бази даних [6].

EF зводить до мінімуму потребу в повторюваному коді доступу до даних, дозволяючи розробникам зосередитися на основних функціях програми тим самим оптимізуючи розробку сайту. Крім того, даний фреймворк нівелює складність різних об’єктів та структур у базі даних за допомогою своїх ORM можливостей та сприяє чистоті, спрощує підтримку коду. Одним з аспектів, за допомогою якого відбувається ця нівеляція є оптимізація взаємодії з базою даних, пропонуючи такі інструменти, як відкладене завантаження (lazy loading),

активне завантаження (eager loading) та оптимізація запитів (IQueryable) для точного налаштування продуктивності програми [6].

Важливою деталлю для вибору даного фреймворку як для новачків, так і для розробників з більшим досвідом, є надійна спільнота Entity Framework та її підтримка від Microsoft. Спільнота в даному випадку слугує основою загалом для Microsoft та дозволяє розміщувати обширну документацію, постійні оновлення та інтеграцію з іншими фреймворками .NET, які дозволяють розробникам підтримувати власні знання та вдосконалювати їх.

Під час розробки даного проекту, було використано Code-first підхід до застосування entity framework, який представляє собою сучасний підхід до розробки додатків, наголошуючи на бездоганній інтеграції дизайну моделі домену зі створенням схеми бази даних.

Перевагами даного підходу є дизайн, орієнтований на домен, полегшення взаємодії з даними, спрощене відображення бази даних та адаптивна зміна бази даних. Розробникам потрібно лише визначити звичайні C# або VB.NET, що будуть представляти сутності домену та інкапсулюватимуть бізнес-логіку та структуровані так, щоб точно відповідати вимогам програми. За допомогою центрального для EF класу DbContext, який є основним компонентом, через який встановлюється з’єднання з базою даних, відбувається контроль запитів, керування зв’язками сутностей та змін у моделі даних. EF Migrations служить динамічним механізмом для зміни схем бази даних з можливістю повернення стану бази даних до певного моменту. Ця автоматизація спрощує процес внесення змін до класів сутностей, забезпечуючи безперебійну синхронізацію бази даних.

Також ще однією перевагою такого підходу є те, що EF використовує інтуїтивно зрозумілі практики для отримання конфігурацій бази даних із класів сутностей. Наприклад, умовна практика за замовчуванням автоматично визначає

такі властивості, як Id або <classname>Id, як первинні ключі, якщо не вказано інше. За допомогою Fluent API розробники можуть задовольнити деталізовані вимоги, що виходять за межі конвенцій. Цей надійний набір інструментів дозволяє точно налаштовувати відображення баз даних, охоплюючи зв'язки, обмеження та інші складні конфігурації, необхідні для узгодженості додатків.

У EF потрібно створити клас, який буде слугувати контекстом та буде містити у собі всі таблиці, для того, щоб в майбутньому через цей клас доступатись до даних у базі даних.

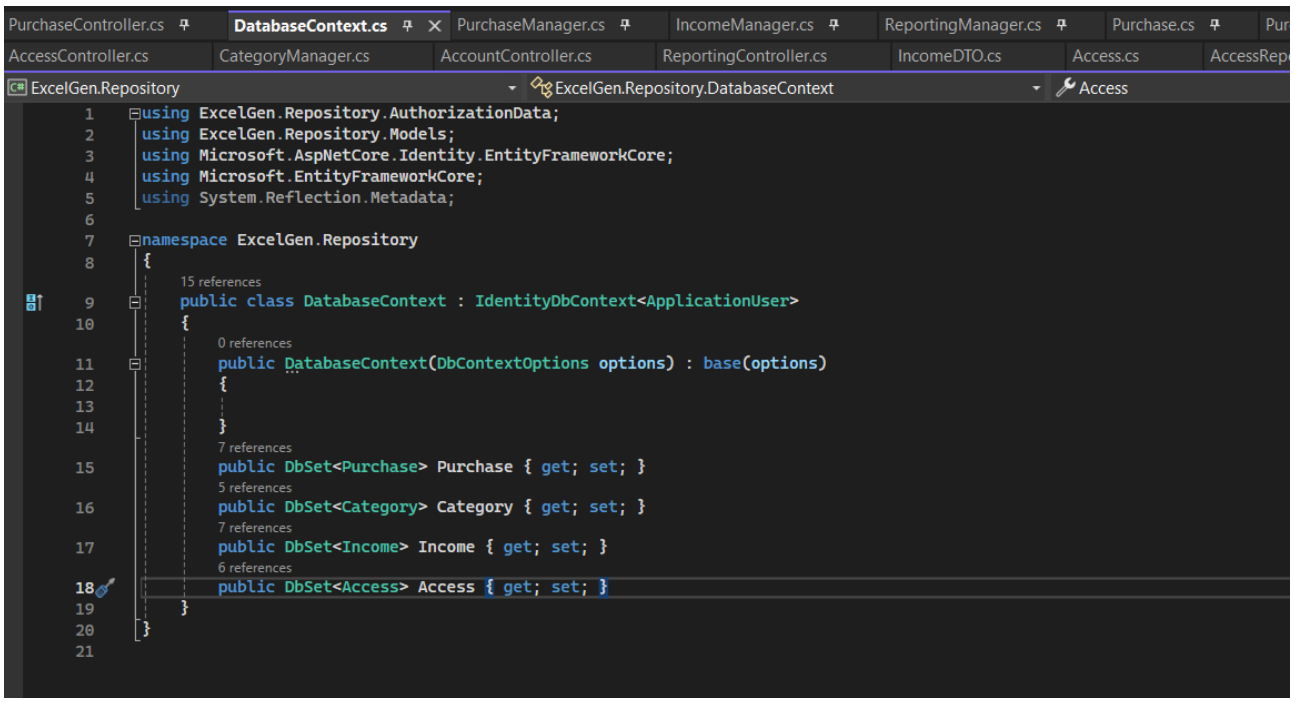


Рис. 2.3. Context клас

Кожна властивість відповідає за таблицю у базі даних. За допомогою атрибутів та метаданих цих атрибутів Entity Framework розуміє певні більш складні структури даних у таблицях (такі, як Primary Key, Foreign Key і т.д.).

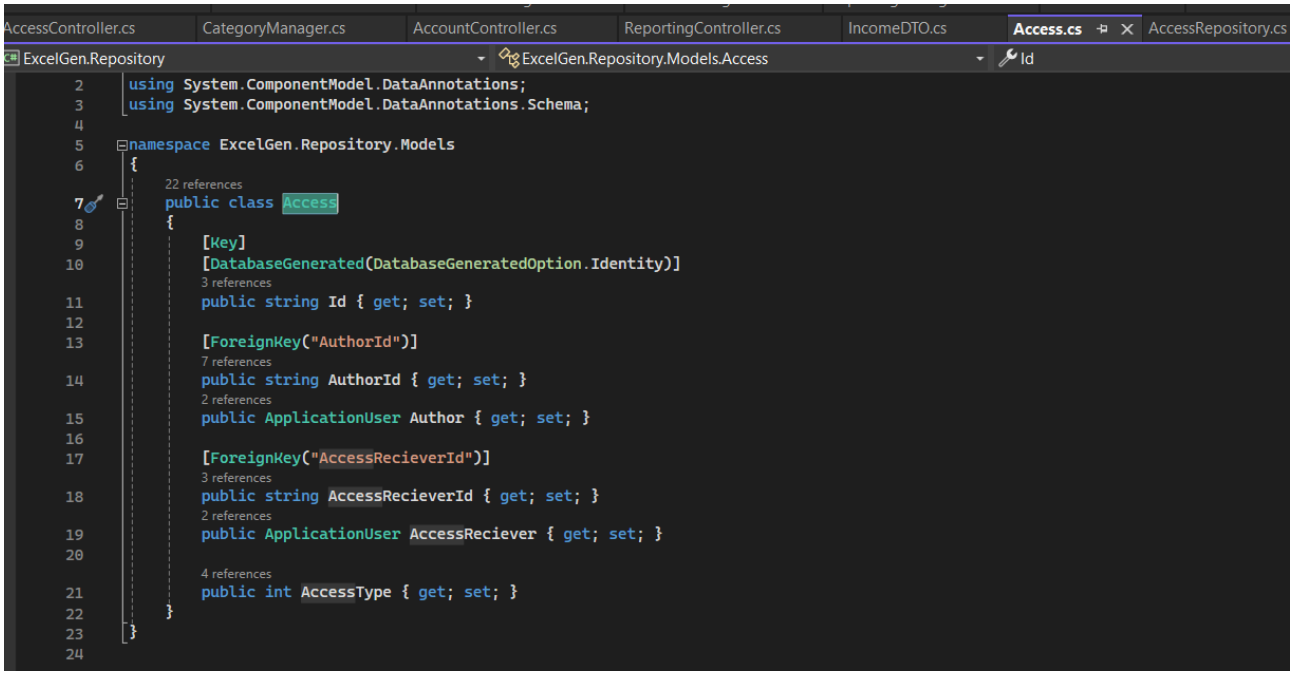


Рис. 2.4. Клас Access, який є таблицею у базі даних.

Зміни до бази даних здійснювались за допомогою підходу code-first, тобто спочатку потрібно змінити клас, який є таблицею у базі даних, або ж додати новий клас до DatabaseContext класу, після того в терміналі Nuget Package потрібно ввести команду “Add-migration <назва міграції>” і “Update-database”. Перша команда створить нову міграцію у папці Migrations, а друга команда додасть зміни у базу даних, яка є підключена за допомогою ConnectionString.

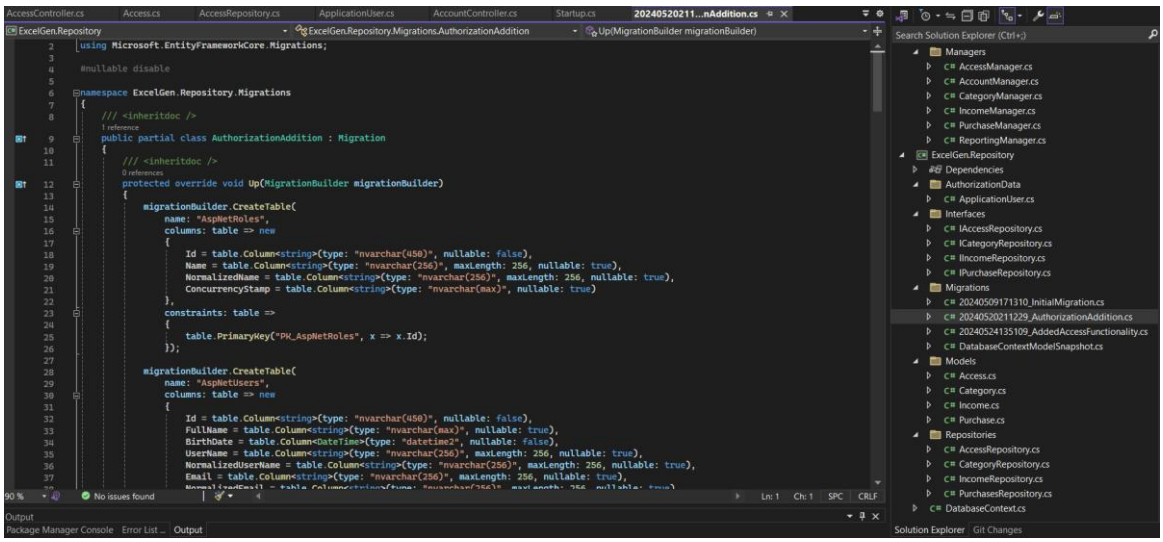


Рис. 2.5. Міграція для додавання авторизації

Для доступу до бази даних кожної таблиці було створено клас репозиторіїв, це було зроблено для того, щоб розподілити обов'язки між класами та в майбутньому мати можливість перевикористати код з одного репозиторію у кількох класах бізнес логіки, тим самим додаток буде легше підтримувати та доповнювати. Саме в класах-репозиторіях ми маємо включення DatabaseContext класу для доступу до бази даних, важливо мати лише включення контексту лише в класах репозиторію, щоб мати менш пов'язану архітектуру.

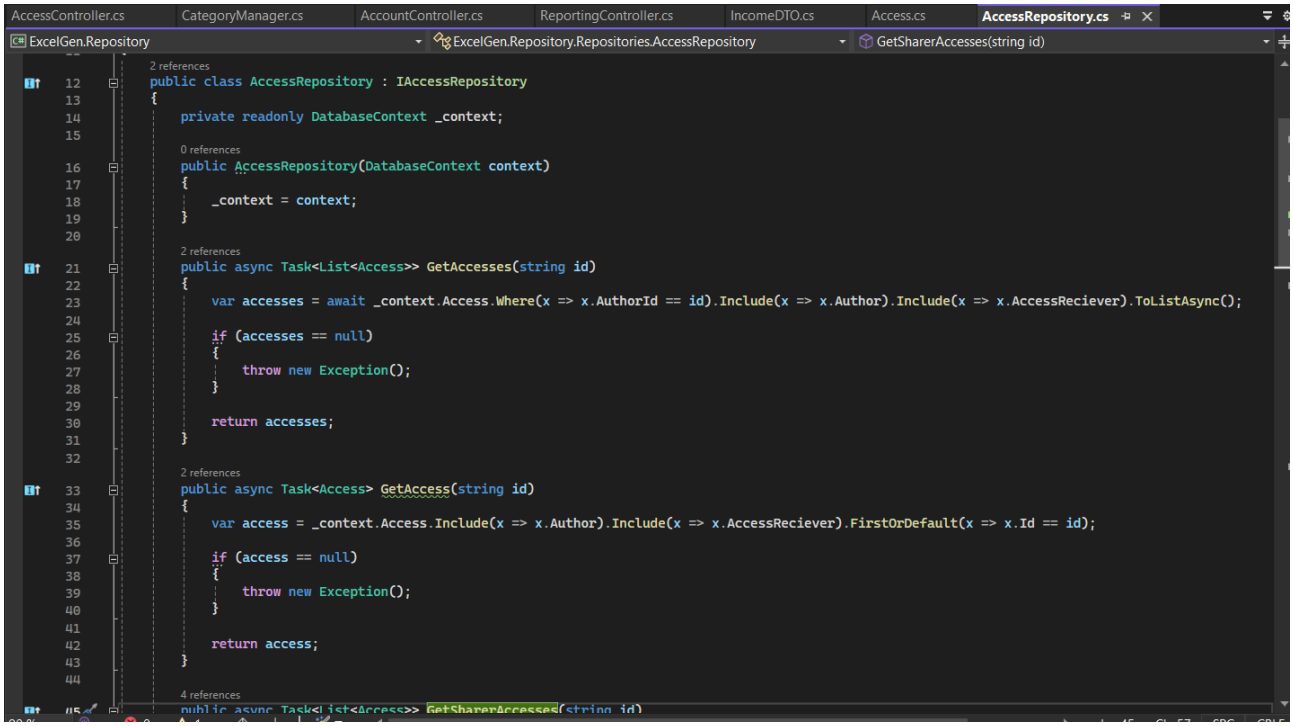


Рис. 2.6. Клас AccessRepository

Отже, Entity Framework спрощує доступ до даних і адміністрування, що робить його найкращим варіантом серед розробників .NET, які створюють сучасні масштабовані програми, які покладаються на реляційні бази даних.

2.1.4. Identity

ASP.NET Identity є важливим фреймворком для додатків ASP.NET, революціонізуючи спосіб обробки розробниками аутентифікації та авторизації користувачів. Identity набагато спрощує реалізацію механізму аутентифікації та

авторизації та має можливість додаткових конфігурацій даних, ролей користувача [7].

ASP.NET Identity надає розробникам різноманітні параметри автентифікації, підтримуючи традиційні налаштування імені користувача та пароля, безпроблемний вхід із соціальних мереж через OAuth та інтеграцію з такими популярними постачальниками ідентифікаційної інформації, як Google і Facebook [7]. Сам спосіб конфігурації визначається в Startup.cs класі і для даного програмного забезпечення використано кукі авторизацію. Розробники мають можливість розширити функціональність ASP.NET Identity, наслідуючись від класу IdentityUser та додавши властивості, які додатково потрібні, або впроваджуючи індивідуальні рішення для зберігання. Ця гнучкість дає чудову можливість розширювати існуючу функціональність аутентифікації та авторизації задля імплементації додаткової логіки.

Identity має чудовий механізм для розширення та реалізації ролей для користувачів, оптимізуючи керування доступом у програмах. Авторизація на основі ролей спрощує реалізацію складної логіки контролю доступу. Також величезну роль у даному фреймворку дано для безпеки даних, адже центральним елементом ASP.NET Identity є надійне сховище користувачів, яке керує важливою інформацією користувачів, такою як хешовані паролі за допомогою криптографічних алгоритмів, таких як PBKDF для покращеної безпеки, адреси електронної пошти та настроювані дані користувачів. Є вбудований механізм підтримки двофакторної автентифікації (2FA), яка посилює безпеку додатків, вимагаючи додаткових кроків перевірки (наприклад, коди SMS, маркери електронної пошти) для доступу користувачів. Розробники налаштовують сувору політику паролів і механізми блокування облікових записів, щоб посилити захист програм [7].

Для підключення Identity потрібно правильно зареєструвати цей сервіс у Startup.cs файлі, а також наслідуватись від типізованого IdentityDbContext класу, де тип, який вказується в дужках – клас користувача.

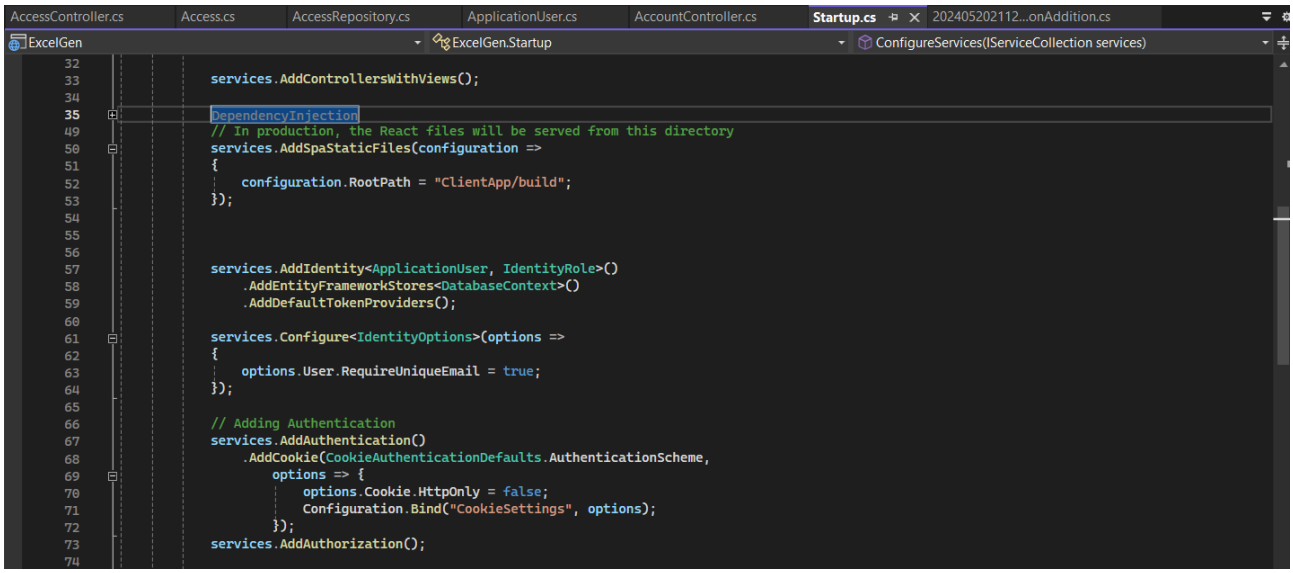


Рис. 2.7. Реєстрація Identity в Startup.cs

Клас користувача повинен наслідуватись від IdentityUser класу і може мати додаткові властивості, які будуть зберігатись у колонках таблиці [dbo].[AspNetUsers].

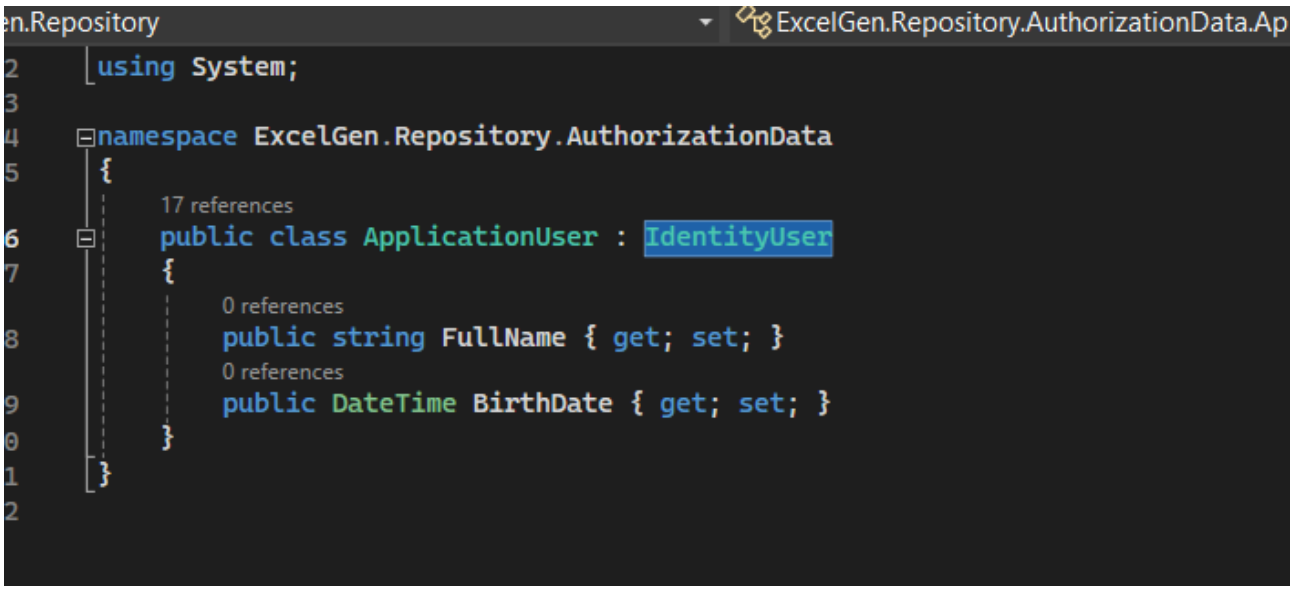


Рис. 2.8. Клас користувача ApplicationUser

По суті, ASP.NET Identity служить наріжним каменем для модернізації безпеки користувачів у програмах ASP.NET. Віддаючи пріоритет гнучкості, безпеці та бездоганній інтеграції, він дає змогу розробникам впроваджувати складні механізми аутентифікації та авторизації, захищаючи дані користувачів і підвищуючи загальну надійність додатків.

2.1.5. Шар бізнес логіки

У веб-розробці шар бізнес-логіки (BLL) відіграє ключову роль, керуючи основними операціями та правилами програми. Діючи як посередник між інтерфейсом користувача (рівень контролера) і взаємодією з базою даних (рівень репозиторіїв), він забезпечує належну обробку даних і правильного виконання функціоналу [8].

Основною характеристикою цього шару якраз є забезпечення виконання бізнес-логіки та правил сайту, BLL забезпечує перевірку, обробку даних відповідно до вимог, бізнес-операцій: керує складними операціями, такими як обробка замовлень, управління запасами, фінансові операції та завдання, пов’язані з доменом.

Також додатковою причиною для розподілу шару функціональності у окремий клас є те, що BLL підтримує чисту, придатну для обслуговування та масштабовану архітектуру додатків. Тим самим ми розподіляємо обов’язки і це сприяє до більш модульної кодової бази, що спрощує оновлення та обслуговування. Такі класи можна повторно використовувати логіку у різних контролерах та ізоляція бізнес-логіки полегшує написання як юніт, так і автоматизованих тестів і перевіряє правильність реалізації бізнес-правил [8].

Тому для правильного розподілу обов’язків, потрібно створити шар бізнес-логіки (я назвав ці класи Manager). Це необхідно для легкої підтримки та розробки програмного забезпечення у майбутньому, адже це забезпечує слабку

пов’язаність між різними класами і тим самим спрощує перевикористання існуючого коду і розширення функціоналу.

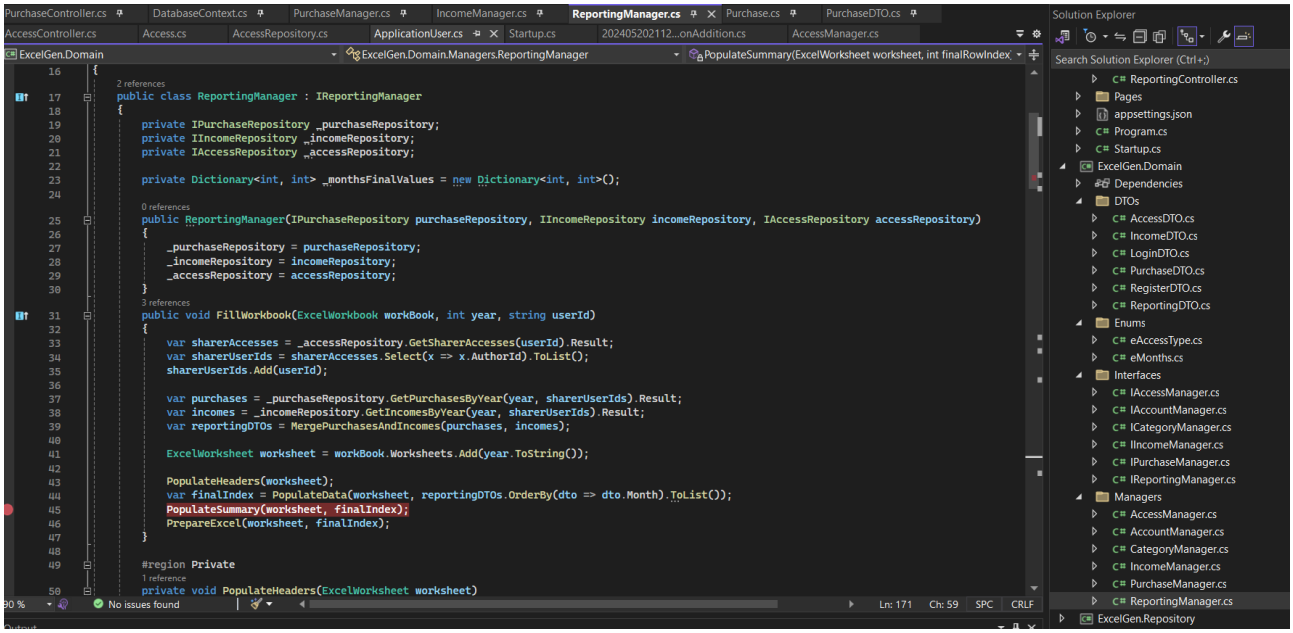


Рис. 2.9. Приклад класу для бізнес-логіки, ReportingManager

Контролер повинен викликати лише певний метод з Manager, тобто шару бізнес-логіки, усі інші дії повинні вже відбуватись у самому класі Manager, тому що саме так розподіляються обов’язки кожного шару. Саме тому в проекті, який є шаром бізнес-логіки ExcelGen.Domain є DTO класи, енами та різні калькуляції.

DTO (data transfer object) – це класи, які мають на меті розширення або зменшення переліку даних, які ми хочемо передати від шару до шару.

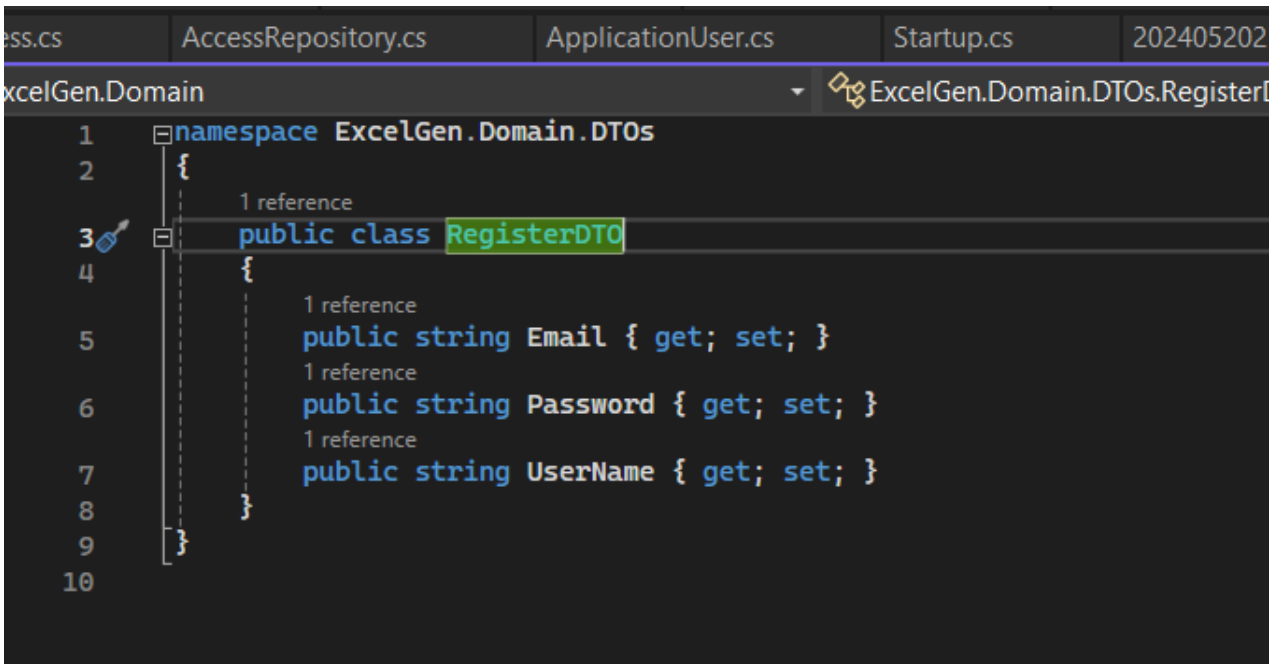


Рис. 2.10. RegisterDTO

Енами – спеціальні конструкції в мові c#, яка надає можливість пов’язати число в базі даних з певним логічним значенням.

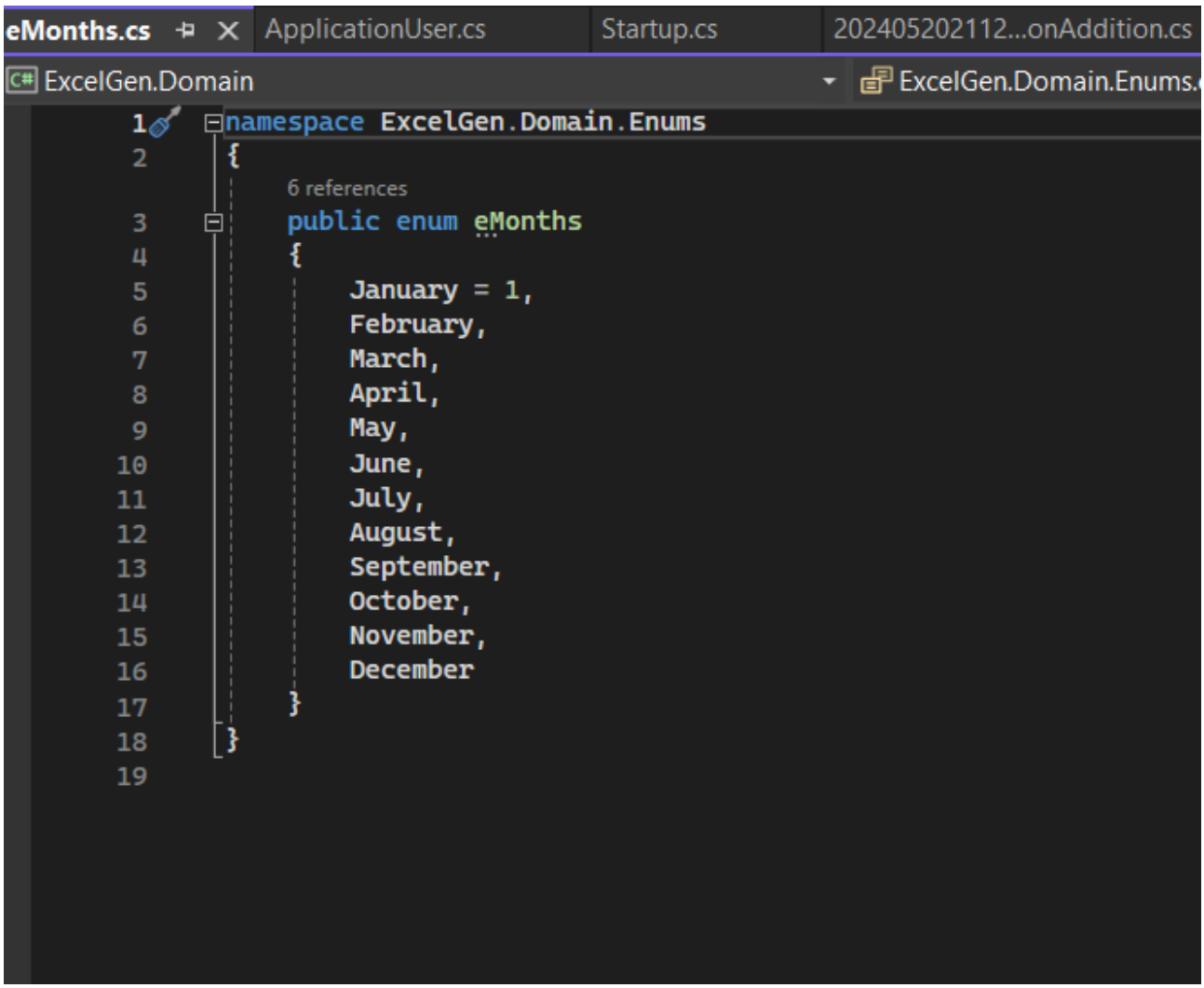


Рис. 2.11. eMonths енам

Таким чином, рівень бізнес-логіки є незамінним у веб-розробці, забезпечує дотримання бізнес-правил і обробляє дані для задоволення потреб додатків. Підтримуючи чіткий розподіл обов'язків, ми покращуємо придатність до обслуговування, масштабованість, безпеку та тестування, суттєво сприяючи надійності програми.

2.2. React

2.2.1. SPA

Односторінкові програми (SPA) революціонізують веб-розробку, створюючи всі умови для динамічної взаємодії з користувачем та оновлюючи

вміст поточної сторінки, не вимагаючи повного перезавантаження сторінки з сервера. Цей метод створює більш бездоганну роботу, яка швидко реагує, подібну до десктопних програм [11].

SPA використовують AJAX (асинхронний JavaScript і XML) для асинхронного отримання даних із сервера, що дозволяє оновлювати частини веб-сторінки за допомогою отриманих даних без перезавантаження всієї сторінки [11]. Вибіркові оновлення: оновлюються лише необхідні частини сторінки, мінімізуючи передачу даних і збільшуючи час завантаження. Також цей метод розробки клієнтської частини забезпечує більш чутливий інтерфейс, уникаючи повних перезавантажень сторінок і зменшуючи кількість звернень до сервера, що сприяє швидкій взаємодії інтерфейсу з користувачем.

Також SPA підтримує такі додаткові функції, як маршрутизація на клієнтській стороні фреймворків React, Angular і Vue.js, тобто керують обробленням навігації та показом сторінок [11]. Під час зміни URL-адреси скрипт сторінки не змінюються і це забезпечує плавний перегляд сторінки та імплементацію різного роду анімацій. Серед додаткових функцій можна підкреслити також оновлення сторінок в режимі реального часу, сповіщення та миттєві зміни вмісту, є звичайними для SPA, що сприяє розширенню даної технології для користувачів.

Основною перевагою SPA є покращена продуктивність, адже ця технологія обробляють більшість взаємодій на стороні клієнта, зменшуючи навантаження на сервер, витягуються лише важливі дані, а не повні сторінки HTML. Також покращений час завантаження сторінки через те, що скрипт сторінки завантажується лише раз і пізніше різні компоненти, скрипт яких уже є в браузері, відображаються за допомогою механізму маршрутизації.

Популярні фреймворки та бібліотеки для SPA:

React. Бібліотека, розроблена Facebook для створення інтерфейсів користувача. React використовує віртуальну DOM для ефективного оновлення та пропонує компонентну архітектуру.

Angular. Комплексна структура, розроблена Google, яка надає повний набір інструментів для створення SPA, включаючи двостороннє зв'язування даних, ін'єкцію залежностей і потужні шаблони.

Vue.js. Прогресивний фреймворк, який легко інтегрувати в проекти. Vue.js пропонує реактивне зв'язування даних і компонентну архітектуру, що робить його придатним як для простих, так і для складних програм.

Підводячи підсумок, односторінкові програми представляють сучасний підхід до веб-розробки, пропонуючи багатий та широкий досвід користувача шляхом динамічного оновлення вмісту без повного перезавантаження сторінки.

2.2.2. React

React, бібліотека JavaScript з відкритим вихідним кодом, розроблена компанією Facebook, та є дуже популярною для створення динамічних користувацьких інтерфейсів, особливо в односторінкових програмах. Його головна перевага полягає в створенні багаторазових компонентів інтерфейсу користувача, які можна повторно використовувати та вони ефективно керують і оновлюють дані в міру їх завантаження [3].

Основними характеристиками для даної бібліотеки є розробка компонент як самодостатніх, так і вкладених, що забезпечує повторному використанню коду та підвищує легкість підтримки таких програмних рішень. Самодостатні компоненти керують власним станом ха допомогою ключових хуків таких як useState для керування станом, useEffect для обробки подій життєвого циклу і useContext для впровадження контекстного API. Вкладені компоненти забезпечують використання компонентів у інших компонентах, важливим для

вкладених компонент механізмом є props, він використовується для передачі даних від батьківських компонентів до дочірніх компонентів, забезпечуючи прямолінійний потік даних і їх узгоджену обробку.

Також варто відмітити реалізацію віртуального DOM у React, адже цим самим дана бібліотека спрощує процес оновлення реальної DOM завдяки тому, що зміни у компоненті впливають на оновлення віртуального DOM, яка в свою чергу визначає оновлення, які потрібні застосувати до реальної DOM. За допомогою цього механізму було досягнуто покращення продуктивності та покращення користувацького досвіду через мінімізування взаємодії з реальним DOM [3].

Головною перевагою React у порівнянні з іншими фреймворками для роботи з клієнтською частиною програми є підвищена продуктивність за рахунок зміни в першу чергу віртуального DOM. Кросплатформеність є іншою перевагою цієї бібліотеки, адже з її допомогою розробники можуть створювати веб додатки, мобільні (використовуючи React Native) та десктопні (через Electron) [3].

Важливими аспектами для роботи з React є хороша спільнота та доступні ресурси для вивчення або розширення знань про React. Спільнота розробників забезпечує попит використання даної бібліотеки та допомогу з аналізом існуючих проблем інших розробників створюючи чудову екосистему. Екосистема React є досить розвиненою, адже існують додаткові інструменти для розробки такі як Redux для керування станом, React Router для маршрутизації та Next.js для відтворення на стороні сервера. Велика кількість документації навчальних посібників та прикладів використовується для допомоги розробникам різних рівнів.

React має основний компонент App, у якому за допомогою компоненти Route встановлюється, яка компонента буде відображена при певній url адресі.

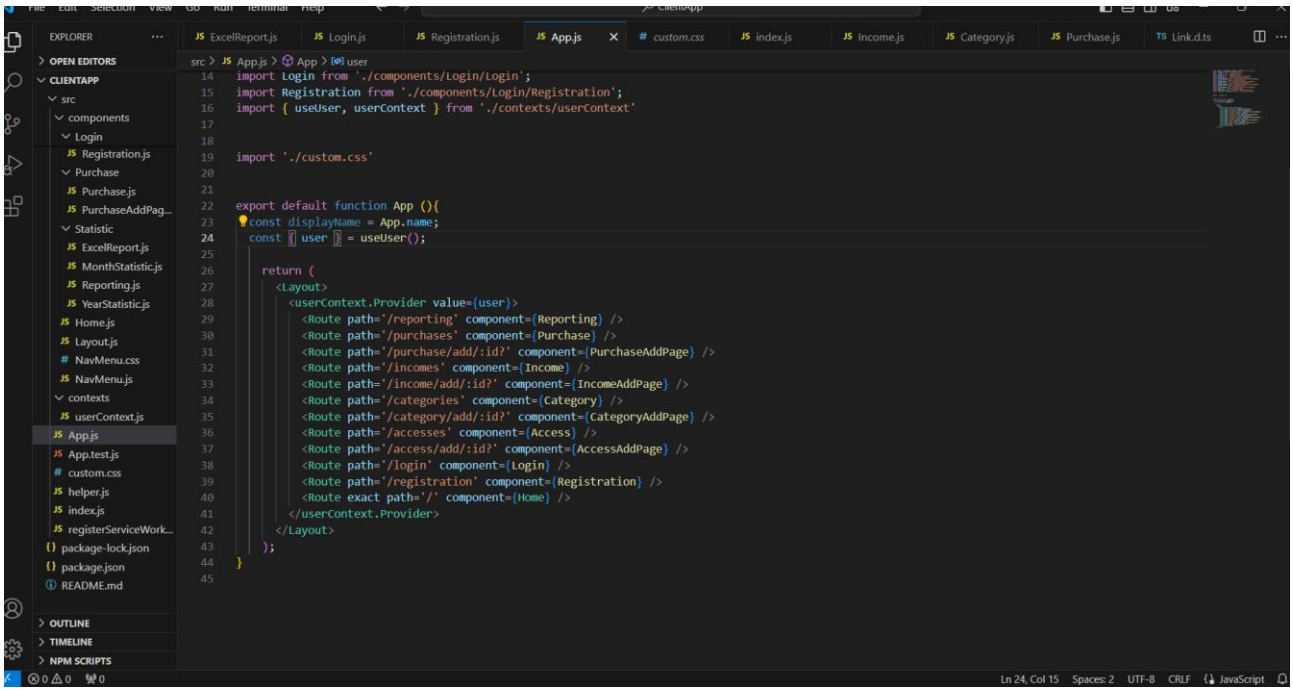


Рис. 2.12. Компонент App

По суті, React — це надійна та адаптована бібліотека для розробки сучасних інтерфейсів користувача. Його акцент на багаторазових компонентах, ефективності віртуальної DOM і підтримуючій екосистемі робить його кращим вибором для розробників, які прагнуть створювати динамічні та високопродуктивні веб-додатки.

2.2.3. Функціональні компоненти

І Redux, і функціональні компоненти з хуками є ключовими в сучасній React-розробці, кожен з них слугує різним цілям і має свої переваги [2].

Redux полягає в тому, щоб зберігати всі дані в загальному контейнері, за допомогою якого відбувається керування глобальним станом програми. Таким чином буде підтримуватись єдине джерело правдивої інформації, адже скрізь, де потрібно використати дані, інформація буде братись з одного й того ж самого місця. Зміна стану програми відбувається за допомогою спеціальних методів – дій та обробляються чистими функціями, редукторами, що робить зміни в станах відстежуваними і зрозумілими.

Перевагами Redux є передбачувані зміни станів через використання дій та редукторів, Redux DevTools дозволяє відстежувати стан у певний момент роботи програми, що робить процес дебагу більш легким та зрозумілим. Проте цей підхід також має складнощі у використанні такі, як Redux може вводити значну кількість шаблонного коду, що може ускладнити початкове налаштування, а також складність розуміння таких понять, як дії, редуктори та схема Redux може бути складною для початківців.

Підхід функціональних компонент полягає у тому, щоб створювати окремі модульні частинки, які будуть використовувати хуки для керування станом та методами життєвого циклу. Особливості полягають саме у використанні хуків таких як useState та useEffect, useContext також розробники можуть створювати власні хуки для інкапсуляції та повторного використання логіки загальних хуків.

Перевагами функціональних компонентів є зменшення кількості шаблонного коду, необхідного для створення функціональних компонент, повторне використання за допомогою створених компонент або створених розробником користувацьких хуків. Проте і цей метод має певні складнощі при роботі, яка в загальному випадку є при малому досвіді роботи з функціональними компонентами. Такими складнощами є нагромадження управління складною логікою, якщо вона погано структурована, а також надмірне використання хуків може призвести до проблем проблем з продуктивністю, якщо неправильно використовувати дані хуки.

Для розробки цього програмного забезпечення був вибраний підхід функціональних компонент з використанням MUI компонент. За допомогою useState хука зберігаються дані для використання в компоненті, useEffect використовується для отримання даних або ініціалізації стейтів, у даній програмній реалізації було використано функцію fetch для запитів на серверну частину проекту.

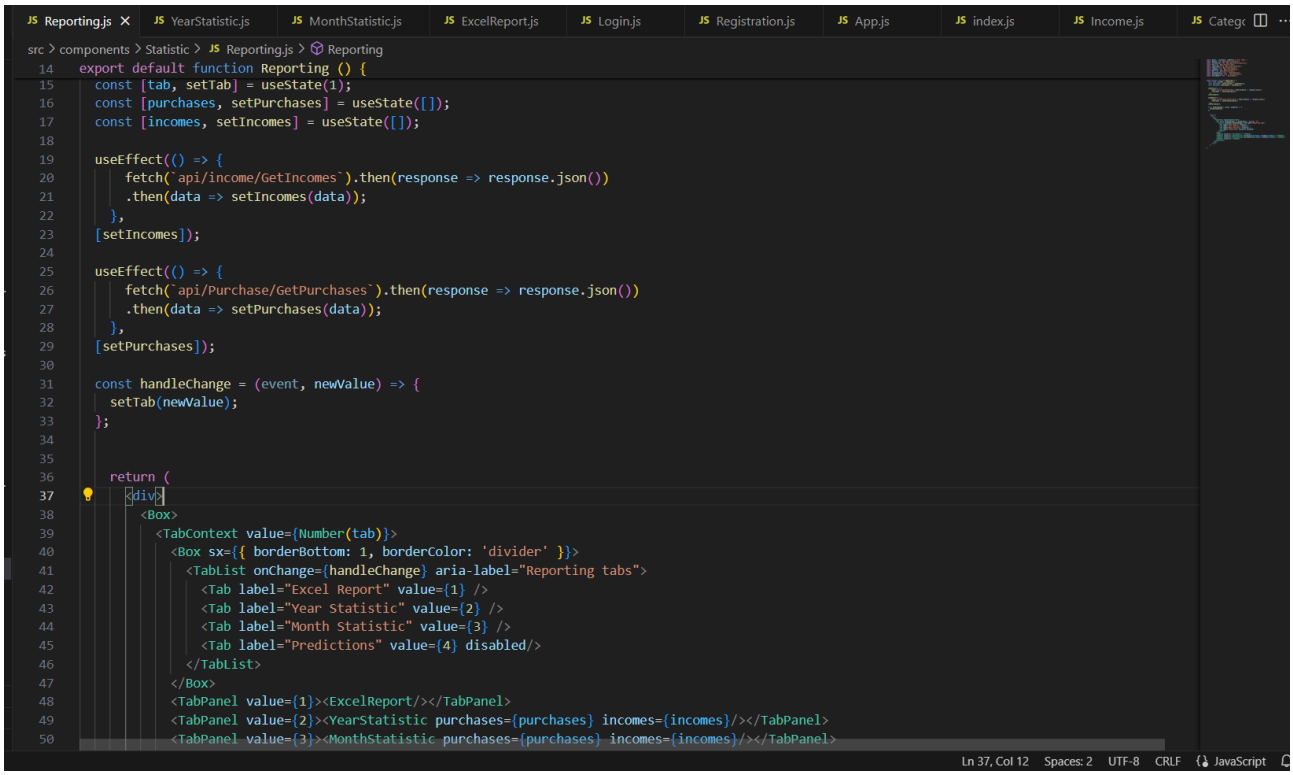


Рис. 2.13. Функціональна компонента Reporting

Використовуйте функціональні компоненти з хуками задля спрощення завдань управління станами, а також для роботи зі станами та побічними ефектами в межах окремих компонентів або невеликого набору компонентів.

Таким чином, Redux та функціональні компоненти з хуками є взаємодоповнюючими інструментами в екосистемі React. Redux відмінно підходить для управління глобальним станом у великих додатках, в той час як хуки спрощують управління станами та побічними ефектами у функціональних компонентах, що робить їх обох цінними в залежності від конкретних потреб вашого додатку.

2.3. Принципи, які реалізовані при виконанні проєкту

2.3.1. Нормалізація баз даних

Нормалізація бази даних є важливим методом у проектуванні реляційних баз даних, спрямованим на структурування даних для зменшення надмірності та підвищення цілісності даних. Це передбачає розбиття бази даних на кілька таблиць і встановлення чітких зв’язків між ними. Основними цілями є усунення надлишкових даних, забезпечення логічних залежностей даних і спрощення обслуговування бази даних [5].

Основними принципами нормалізації є мінімізація надмірності даних і через це покращується узгодженість та цілісність даних. Мінімізація надлишковості даних відбувається за рахунок видалення дублікатів також це допомагає зберегти простір для зберігання та мінімізувати ризик невідповідностей. Мінімізування невідповідностей забезпечує зменшення аномалій (проблем з додаванням, оновленням і видаленням) даних. Також нормалізація допомагає краще побудувати логічну структуру бази даних, яка в результаті гарантуватиме побудову більш ефективних та простих запитів до бази даних.

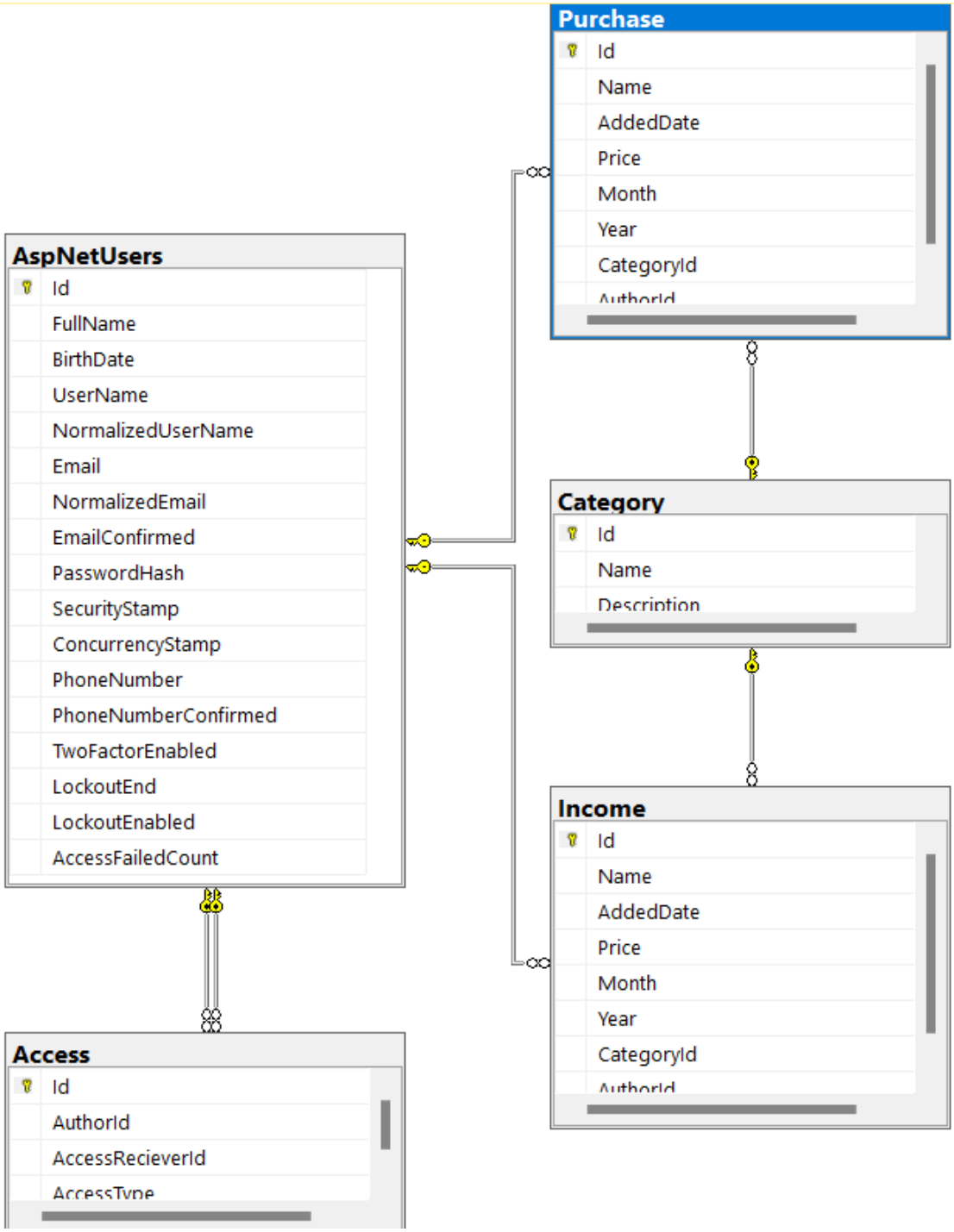


Рис. 2.14. Структура таблиць у базі даних

Отже, нормалізація бази даних є критично важливою практикою в розробці ефективних, надійних і придатних для обслуговування реляційних баз даних. Упорядковуючи дані в логічні ненадлишкові таблиці та забезпечуючи логічні залежності даних, нормалізація значно підвищує цілісність даних і зменшує надмірність. Незважаючи на складність і потенційні компроміси щодо

продуктивності, переваги точності даних, узгодженості та простого обслуговування роблять нормалізацію незамінним аспектом проектування бази даних.

2.3.2. ACID принципи

ACID — це абревіатура, яка означає атомарність, консистенцію (узгодженість), ізоляцію та довговічність. Ці принципи є фундаментальними для забезпечення надійності та цілісності транзакцій бази даних. Кожен принцип стосується конкретних аспектів того, як транзакції обробляються та керуються в системі бази даних [1].

Ключові принципи ACID:

Атомарність – гарантує, що кожна транзакція розглядається як єдина неподільна одиниця роботи. Або всі операції в рамках транзакції завершені успішно, або жодна з них. Реалізація: якщо будь-яка частина транзакції виходить з ладу, уся транзакція відкочується, залишаючи базу даних у вихідному стані.

Узгодженість – гарантує, що транзакція переводить базу даних з одного дійсного стану в інший, зберігаючи попередньо визначені правила та обмеження бази даних. Реалізація: база даних повинна бути в узгодженому стані до і після транзакції. Будь-яка транзакція повинна гарантувати, що всі обмеження цілісності (такі як первинні ключі, зовнішні ключі та унікальні обмеження) не порушуються.

Ізоляція – гарантує, що виконання транзакцій ізольовано одне від одного. Транзакції не повинні заважати одна одній, а проміжні результати транзакції мають бути невидимі для інших транзакцій. Реалізація: ізоляція зазвичай досягається за допомогою механізмів блокування та рівнів ізоляції (таких як незафіксоване читання, зафіксоване читання, повторюване читання та можливість серіалізації) для керування одночасними транзакціями.

Довговічність – гарантує, що після здійснення транзакції її наслідки постійно записуються в базу даних, навіть у разі збою системи.Реалізація: довговічність зазвичай досягається за допомогою таких методів, як журнали транзакцій, які записують зміни, внесені транзакціями. Ці журнали можна використовувати для відновлення узгодженого стану бази даних у разі збою.

ACID принципи є важливими, адже вони гарантують надійність бази даних, цілісність даних, допомагають обробити одночасний доступ до бази та відновлення системи у разі непередбачуваних несправностей. Цілісність даних, реалізована за допомогою транзакцій, запобігає неправдивість даних і гарантує, що всі операції обмежені визначеними для бази даних правилами [1].

ACID принципи імплементовані у Entity Framework за допомогою таких механізмів:

Атомарність може бути реалізована через DbContext.SaveChanges(): коли ви викликаєте SaveChanges() у DbContext, EF обгортає всі зміни в транзакції. Якщо під час процесу збереження виникає помилка, EF автоматично відкочує транзакцію. Для ще більшого кастомізованого керування механізмом транзакцій можна використати DbContextTransaction.

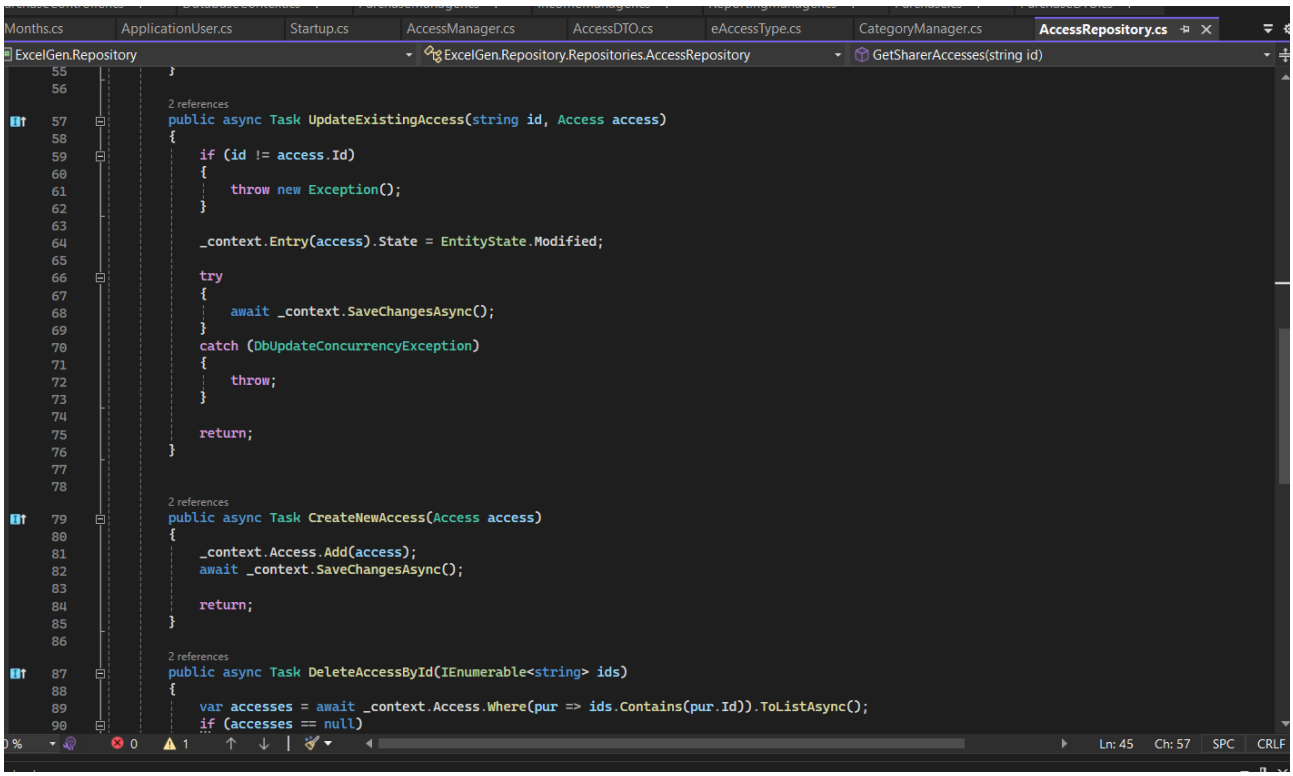


Рис. 2.15. Приклад транзакціїу методі SaveChangesAsync()

Узгодженість у EF підтримується шляхом забезпечення відповідності всіх операцій бази даних визначеним правилам цілісності даних, таким як первинні ключі, зовнішні ключі, унікальні обмеження та інші обмеження відносин. Анотації даних і Fluent API: EF дозволяє визначати обмеження за допомогою анотацій даних або Fluent API, щоб переконатися, що схема бази даних відповідає вашій моделі домену. Також підтримується через валідацію: EF забезпечує перехоплення для перевірки перед збереженням змін у базі даних, гарантуючи, що дані відповідають визначеним правилам.

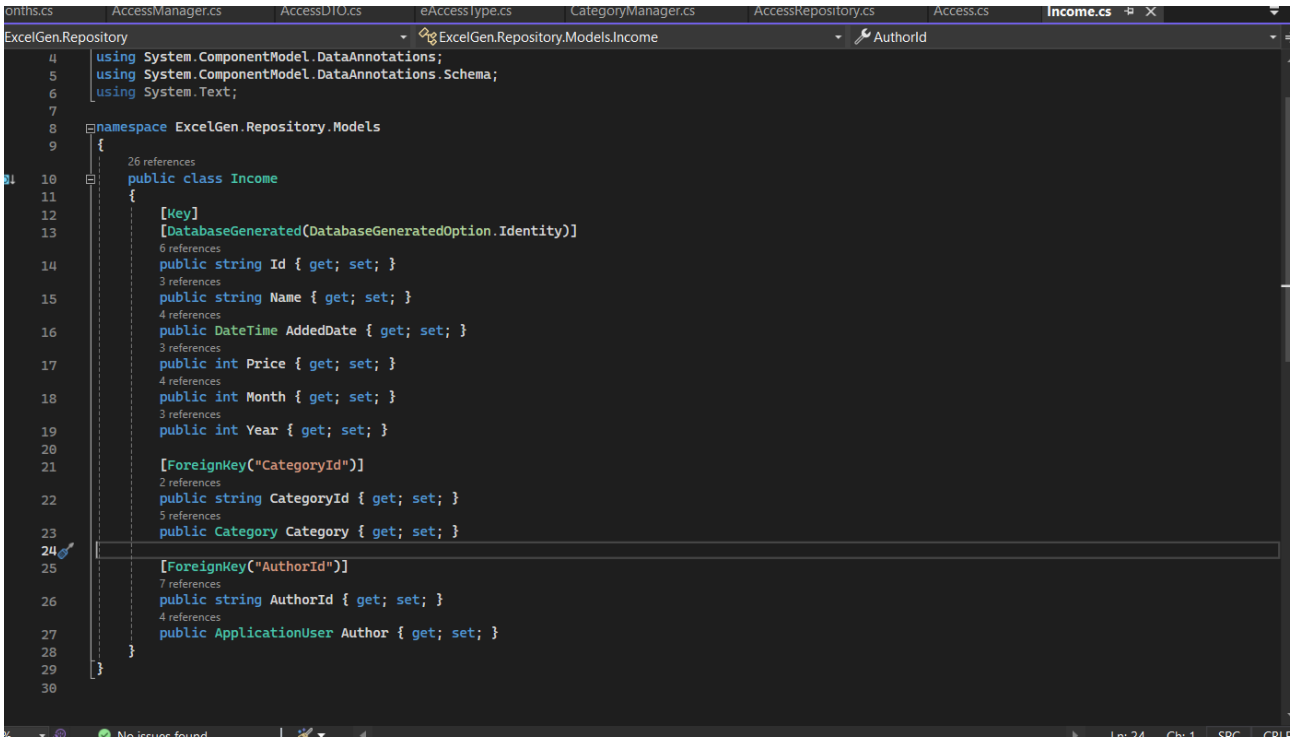


Рис. 2.16. Приклад анотації даних у класі Income

Ізоляція гарантує, що транзакції забезпечені ізольовано одна від одної. EF розміщується на базовій базі даних для керування рівнями ізоляції транзакцій.

Один із способів реалізації ізоляції в EF – рівні ізоляції транзакцій: під час використання явних транзакцій можна вказати рівень ізоляції. Також ізоляція отримується шляхом обробки паралелізму: EF підтримує оптимістичний паралелізм за допомогою маркерів паралелізму. Ви можете змінити властивість як маркер паралелізму, і EF мати наявність змін перед фіксацією.

Довговічність гарантує, що після здійснення транзакції зміни залишаються постійними навіть у разі збою системи. EF використовує механізми довговічності, які надає базова база даних. Реалізована через операцію фіксації: коли викликається SaveChanges(), EF гарантує, що транзакція фіксується в базі даних, яка використовує свої механізми журналювання та відновлення, щоб гарантувати довговічність. Інший спосіб реалізації довговічності конфігурація бази даних: налаштування бази даних для використання таких функцій, як

ведення журналу з попереднім записом (WAL), гарантує довговічність транзакцій.

У підсумку можемо констатувати, що принципи ACID є основоположними для забезпечення надійності, цілісності та стійкості транзакцій бази даних. Дотримуючись принципів атомарності, послідовності, ізоляції та довговічності, системи баз даних можуть ефективно керувати транзакціями, підтримувати цілісність даних і забезпечувати надійне відновлення системи. Незважаючи на проблеми з продуктивністю та складністю, особливо в розподілених середовищах, властивості ACID залишаються критичними для багатьох програм, які вимагають суворої цілісності та узгодженості даних.

2.3.3. SOLID принципи

Принципи SOLID – це набір із п’яти принципів проектування в об’єктно-орієнтованому програмуванні, спрямованих на створення більш зрозумілого, гнучкого та зручного для обслуговування програмного забезпечення. Ці принципи були введені Робертом К. Мартіном і основою для сучасної розробки програмного забезпечення [9].

Принцип єдиної відповідальності (SRP) – у класу має бути лише одна причина для змін, тобто у нього має бути лише одна робота чи відповідальність. Він забезпечує цілеспрямовану мету класу, тобто кожен клас повинен зосередитись на своєму завданні або відповідальності, це сприяє кращому розумінні логіки класу та покращує тестування і підтримку. Іншим важливим аспектом цього принципу є розподіл проблем, завдяки цьому зміни в одній частині коду будуть мати меншим вплив на інший код, тобто код буде менш пов’язаним [9].

Одним з таких прикладів може бути контролер – методи контролера мають єдиний обов’язок – отримати запит та сформувати відповідь. У даному випадку,

заповнити ексель, яка буде у відповіді повинен інший клас, а саме ReportingManager.

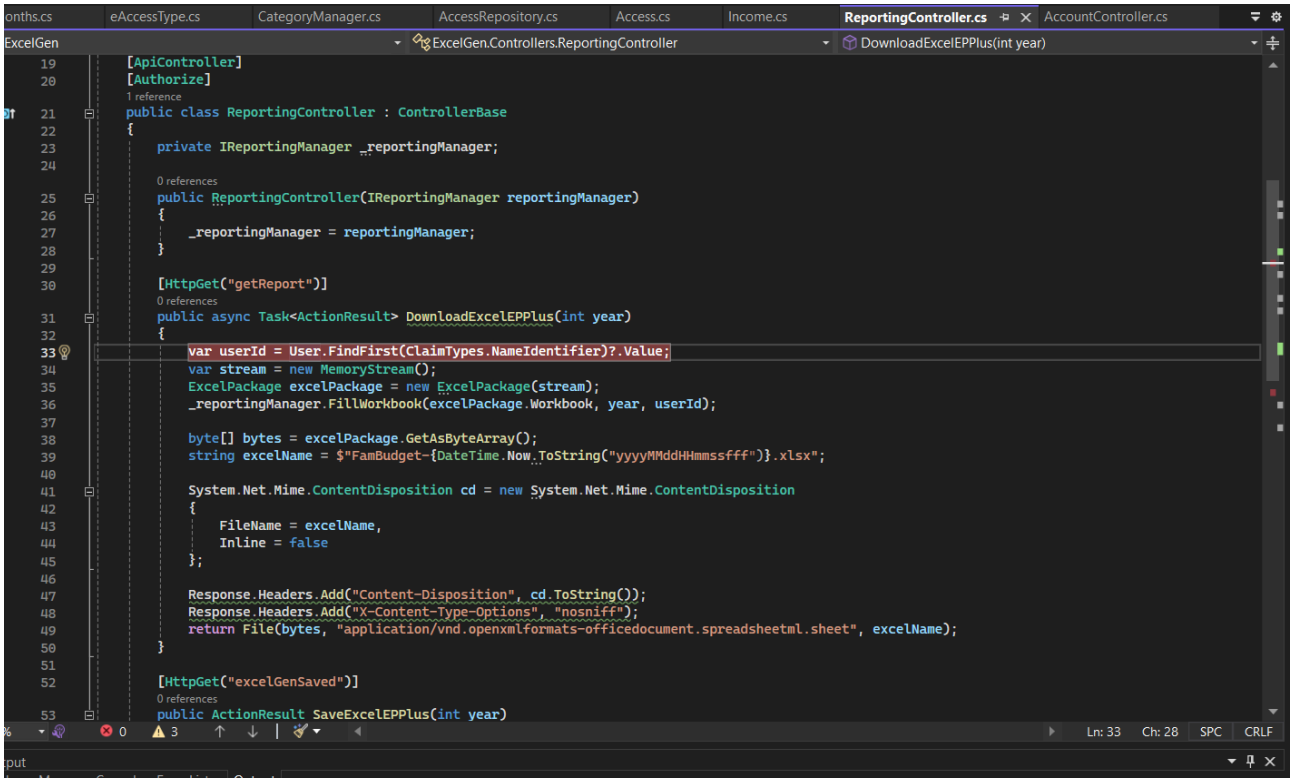


Рис. 2.17. Клас ReportingController

2. Принцип відкритості/закритості (OCP) – програмні сутності (класи, модулі, функції тощо) мають бути відкритими для розширення, але закритими для модифікації [9]. За допомогою розширюваної поведінки, яку надає цей принцип, новий код можна додати шляхом розширення існуючих класів, а не їх зміною. Це зменшує ризик появи помилок у вже протестованому коді.

Репозиторії є прикладом OCP принципу, адже розподіл обов'язків дозволяє створювати в репозиторії нові методи без модифікації існуючих і завдяки цьому класи є слабо пов'язаними.

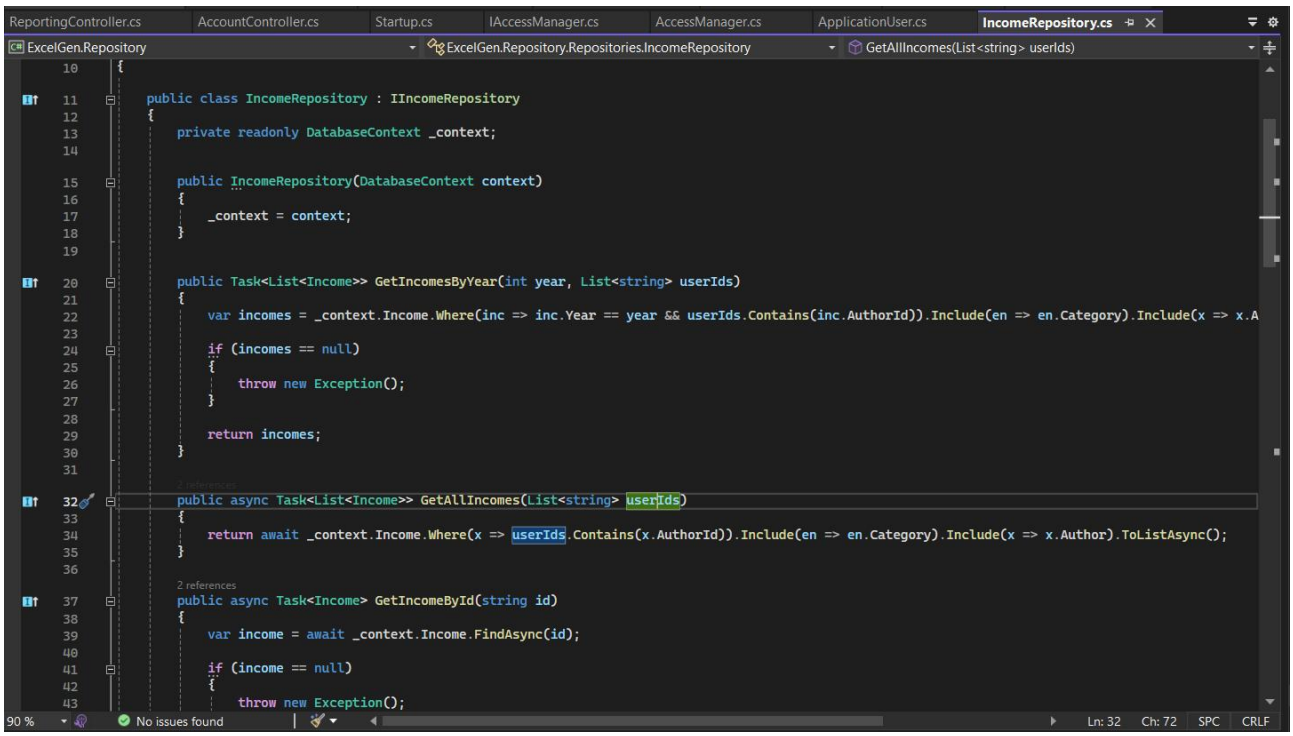


Рис. 2.18. Клас репозиторію

3. Принцип заміни Ліскова (LSP) – об’єкти суперкласу повинні бути замінними на об’єкти підкласу без впливу на коректність програми [9]. Основним у цьому принципі є узгодженість поведінки та забезпечення сумісності класів. Похідні класи можуть лише розширювати поведінку базового класу, але не змінювати основну поведінку батьківського, тобто суперклас можна замінити дочірніми не змінюючи поведінку програми.

Один з прикладів є використання IdentityUser замість ApplicationUser, адже вони мають однакове логічне значення, ApplicationUser лише певним чином розширює функціонал IdentityUser, але вони обоє означають клас користувача.

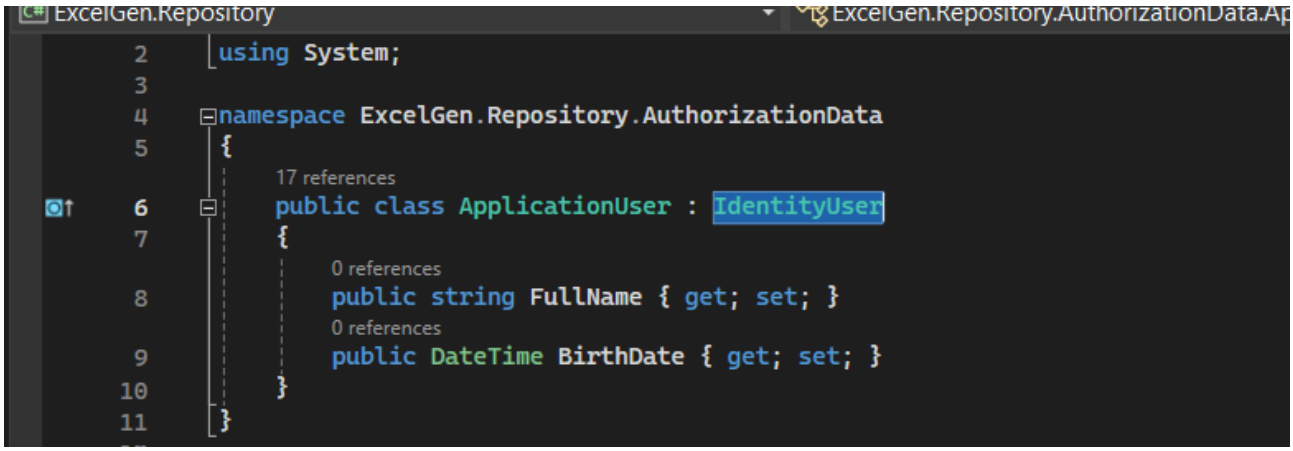


Рис. 2.19. Приклад взаємозамінних класів

4. Принцип сегрегації інтерфейсу (ISP) – клієнт не повинен бути змушений залежати від інтерфейсів, які він не використовує [9]. Це дозволяє класам, які імплементують методи інтерфейсу, реалізовувати лише потрібні їм функції та робить систему більш модульною, що у результаті зменшує вплив змін на систему.

Рис. 2.20. Принцип розподілу інтерфейсів

5. Принцип інверсії залежностей (DIP) – модулі високого рівня не повинні залежати від модулів низького рівня. Обидва мають залежати від абстракцій. Крім того, абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій [9]. Цей принцип дозволяє зробити систему менш пов’язаною, гнучкішою та легшою для підтримки за рахунок залежності від абстракцій, а не конкретних реалізацій. Також гнучкість покращується за допомогою того, що зміни у модулях низького рівня не впливають на модулі високого рівня, аж

допоки абстракція не зміниться. Одним з механізмів реалізації цього принципу є ін’єкція залежностей, яка реалізована в коді.

Підсумовуючи, принципи SOLID є основними вказівками для створення надійного об’єктно-орієнтованого програмного забезпечення, яке зручно підтримувати та масштабується. Вони допомагають розробникам проектувати системи, які легше розуміти, розширювати та підтримувати, сприяючи належним методам проектування та зменшуючи залежності та складності.

2.3.4. Ін’єкція залежностей (DI)

Впровадження залежностей (DI) — це важливий шаблон проектування в інженерії програмного забезпечення, який визначає, як об’єкти отримують свої залежності. Замість того, щоб об’єкт створював власні залежності, вони надаються зовнішнім джерелом, таким як фреймворк або контейнер. Такий підхід сприяє слабкому зв’язку, роблячи додатки більш зручними для тестування та обслуговування [4].

Впровадження залежностей реалізується за допомогою інверсії керування (IoC). Інверсія керування (IoC) – це принцип, за допомогою якого контроль над створенням об’єкта та керування ним передається від самого об’єкта до зовнішньої сутності (контейнера). DI — це конкретна реалізація IoC, де залежності впроваджуються в об’єкти зовнішньою структурою або контейнером [4].

Даний механізм забезпечує слабкий зв’язок у системі за допомогою відокремлення обов’язку створення залежностей у бізнес-логіки об’єкта, тим самим покращуючи модульність програми. За допомогою модульності програми покращується ізольованість тестів, підтримка проектів та легкість рефакторингу. Також покращується повторне використання компонентів, адже вони є розподіленими по окремих модулях, які можна перевикористати, або з легкістю замінити одну на іншу.

У ASP.NET є вбудований контейнер для Dependency Injection, який знаходиться в Startup.cs класі і реєструє залежності при запуску сервера.

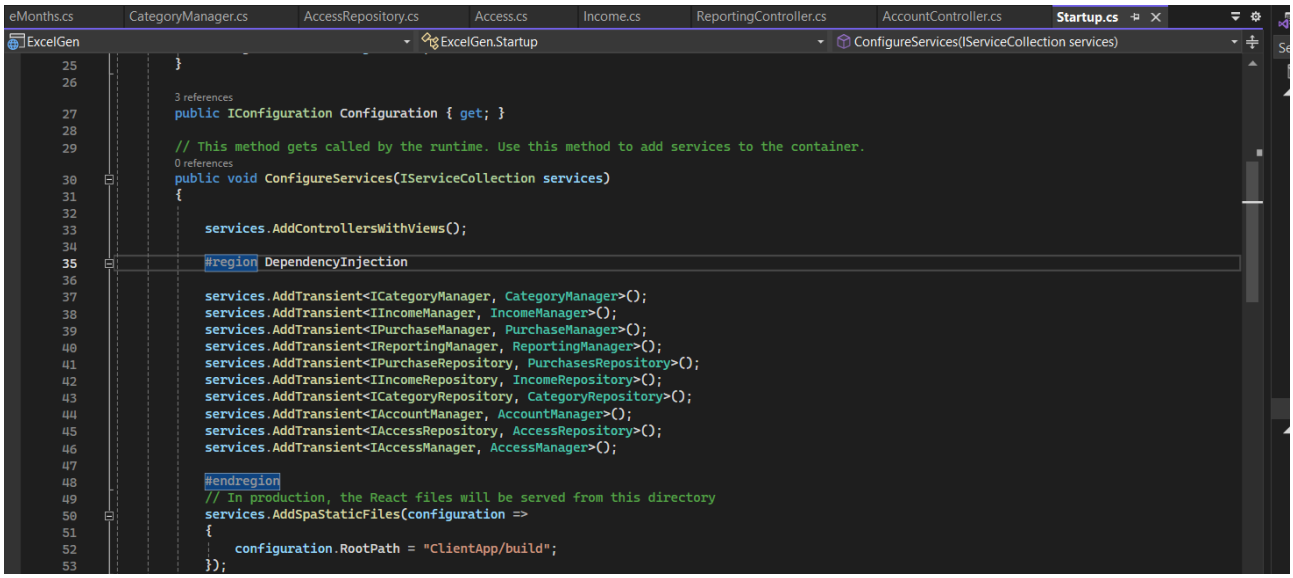


Рис. 2.21. Контейнер для ін’єкції залежностей

Отже, впровадження залежностей — це важливий шаблон проектування, який значно покращує модульність, можливість тестування та обслуговування програмних систем. Призначаючи створення та керування залежностями зовнішній сутності, DI забезпечує слабий зв’язок і дотримується принципів інверсії контролю. Це призводить до того, що програми є більш гнучкими, легшими для тестування та обслуговування, з чітким розподілом завдань і покращеним повторним використанням коду.

РОЗДІЛ 3.
ТЕСТУВАННЯ РЕАЛІЗОВАНОЇ СИСТЕМИ

3.1. Авторизація та автентифікація

При вході на сторінку вебресурсу користувач бачить основну сторінку, на якій вказані мету, причини використання даного вебресурсу та прохання для реєстрації або авторизацію на сторінку, адже весь функціонал доступний лише для авторизованих користувачів:

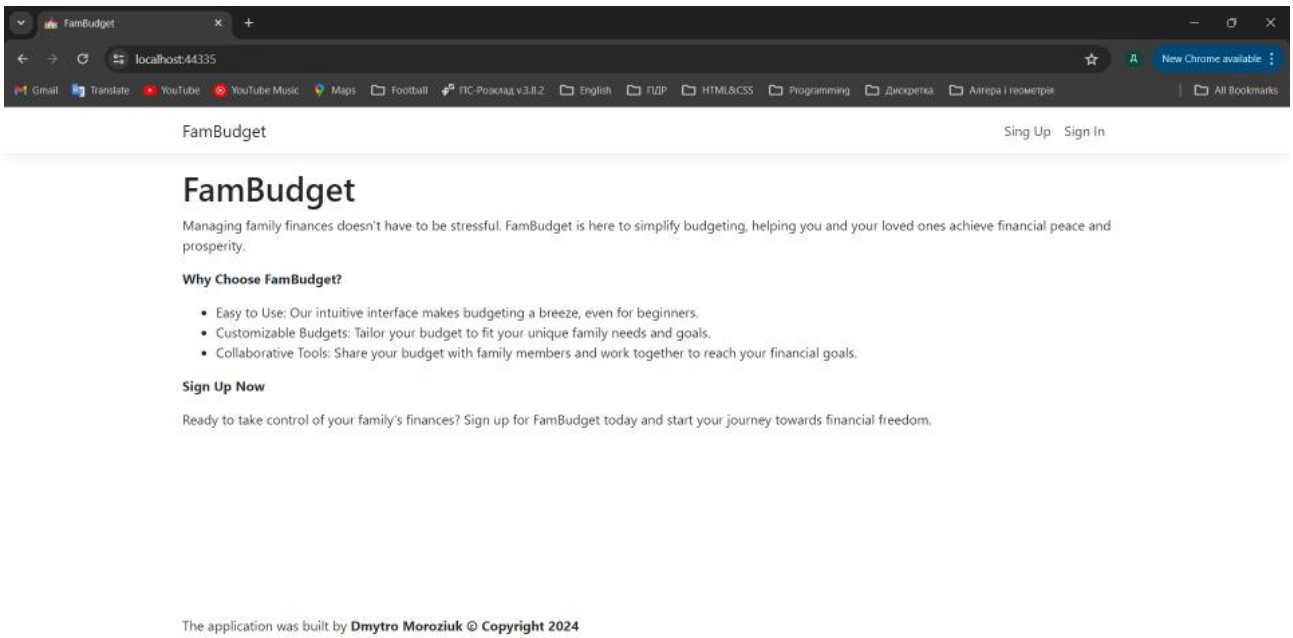


Рис. 3.1. Вітальна сторінка вебресурсу

Отже, для неавторизованих користувачів у меню доступні два пункти – зареєструватись та залогінитись на сайт, при нажатті на ці пункти користувач бачить наступні сторінки відповідно:

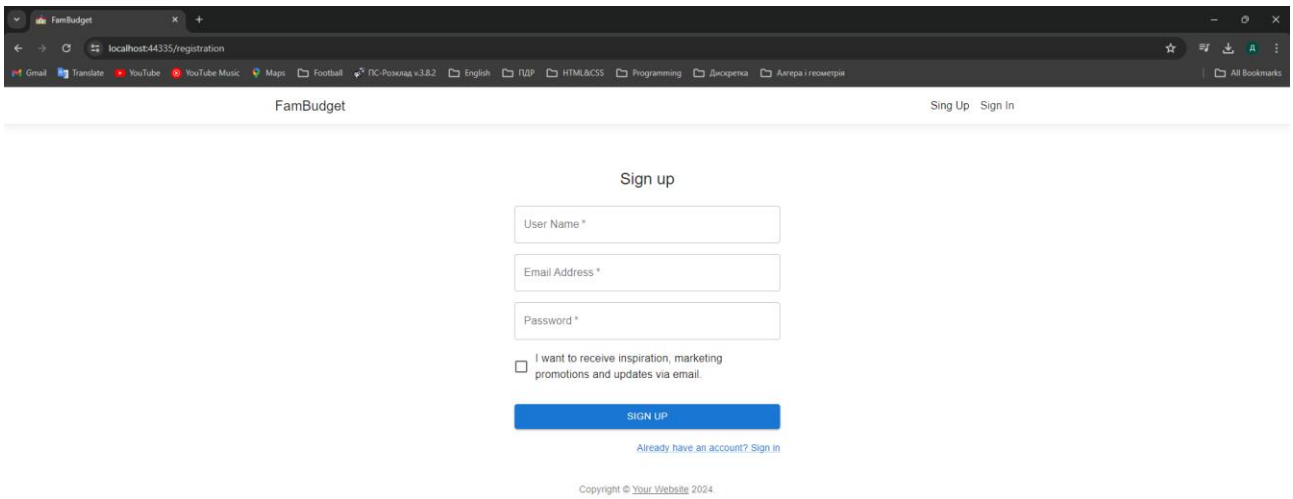


Рис. 3.2. Сторінка реєстрації

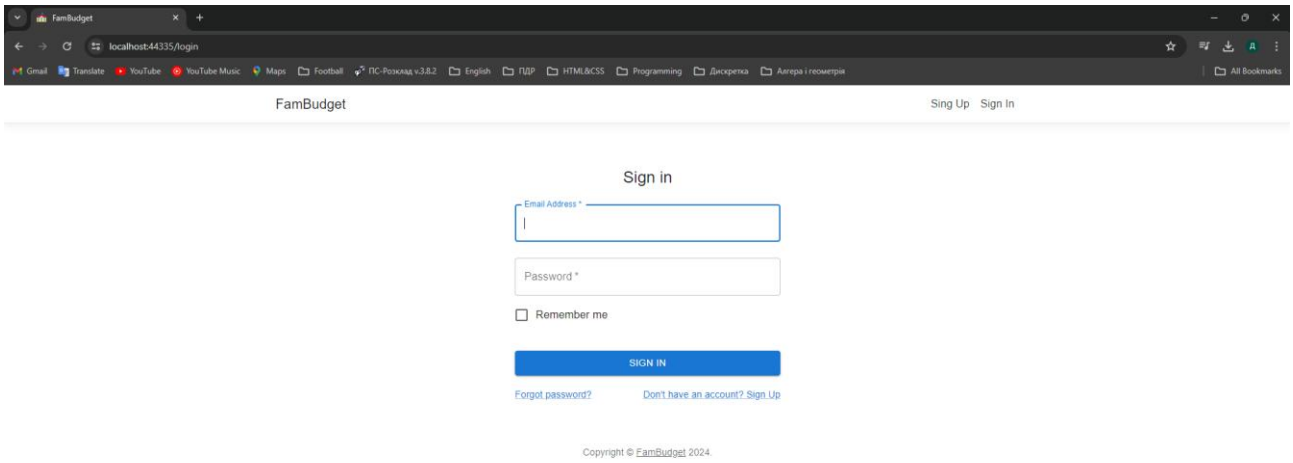


Рис. 3.3. Сторінка логіну

3.2. Основна функціональність сайту

3.2.1. Додавання покупок та доходів

Для авторизованого користувача вже відкривається розширене меню навігації, яке містить в собі наступні опції:

1. Purchases – меню покупок, у якому можна редагувати, видаляти, переглянути і додавати власні покупки та покупки користувачів, які поділились з вами своїми.
2. Incomes – меню доходів, у якому можна редагувати, видаляти, переглянути і додавати власні доходи та доходи користувачів, які поділились з вами своїми.
3. Categories – пункт меню, у якому можна переглядати, редагувати, видаляти та додавати нові категорії.
4. Reports – пункт меню, у якому можна аналізувати власні доходи та покупки у річній та місячній статистиці, скачати ексель файл з річною статистикою.
5. Access Manager – пункт меню, у якому можна переглянути, видалити, поредагувати та додати новий доступ до своїх покупок та доходів для інших користувачів.
6. Log out – пункт меню, за допомогою якого можна вийти з власного облікового запису.

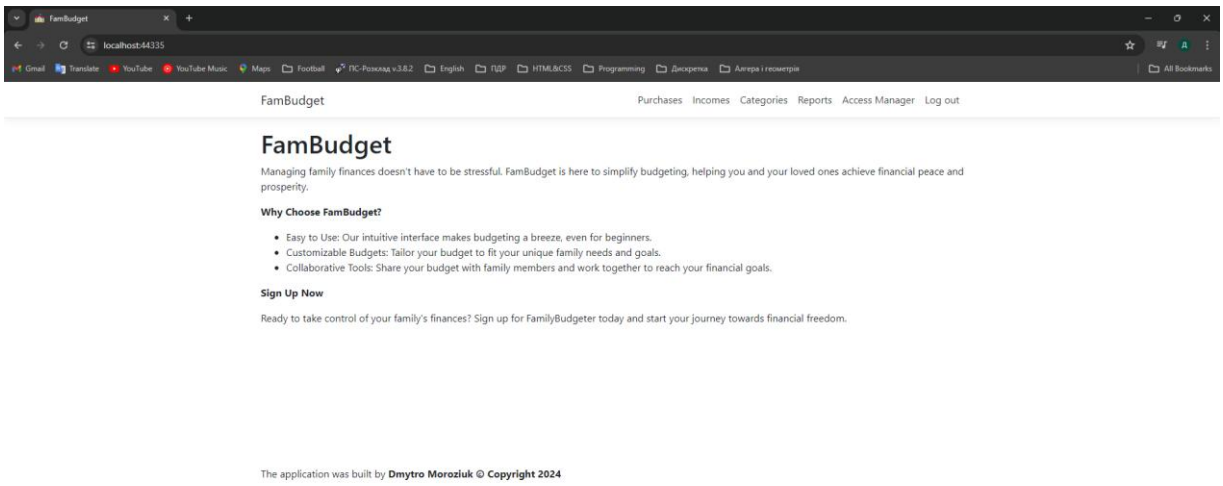


Рис. 3.4. Меню авторизованого користувача

При відкритті пункту меню Purchases, користувач бачить список доступних для нього покупок у вигляді таблиці з даними в кінці таблиці у кожного рядка є кнопки редагування та видалення. Кнопки повинні бути клікабельними, якщо це покупка цього користувача, та можуть бути не клікабельними, якщо це покупки, якими з даним користувачем поділились у режимі “Readonly” (“Лише для читання”). При натиску на кнопку Delete, покупка зникає з таблиці, але вона все ще є у базі даних, для того, щоб видалити цю покупку з бази даних, потрібно натиснути на кнопку збереження “Save”.

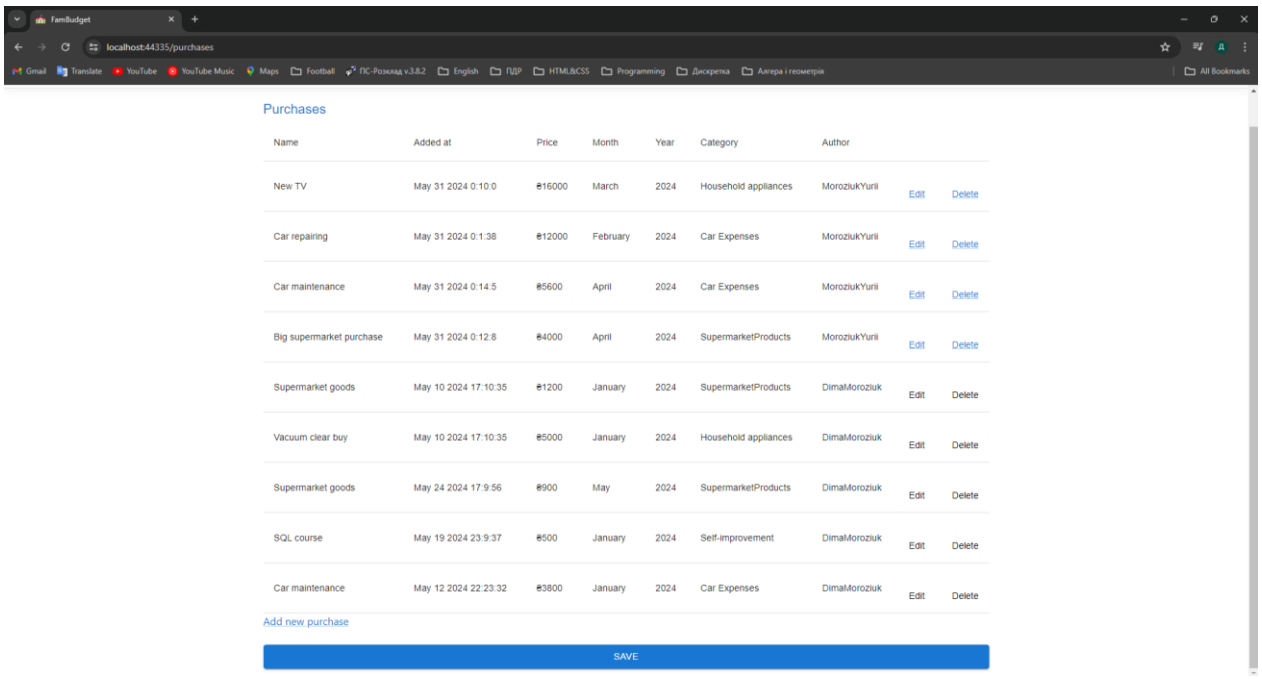


Рис. 3.5. Сторінка для перегляду покупок.

При натиску на кнопку “Add new purchase” користувачу відображається сторінка для додавання нової покупки. Покупка має свою назву, ціну, місяць та рік, у якому вона була куплена, та категорія, яка є наявною у базі даних.

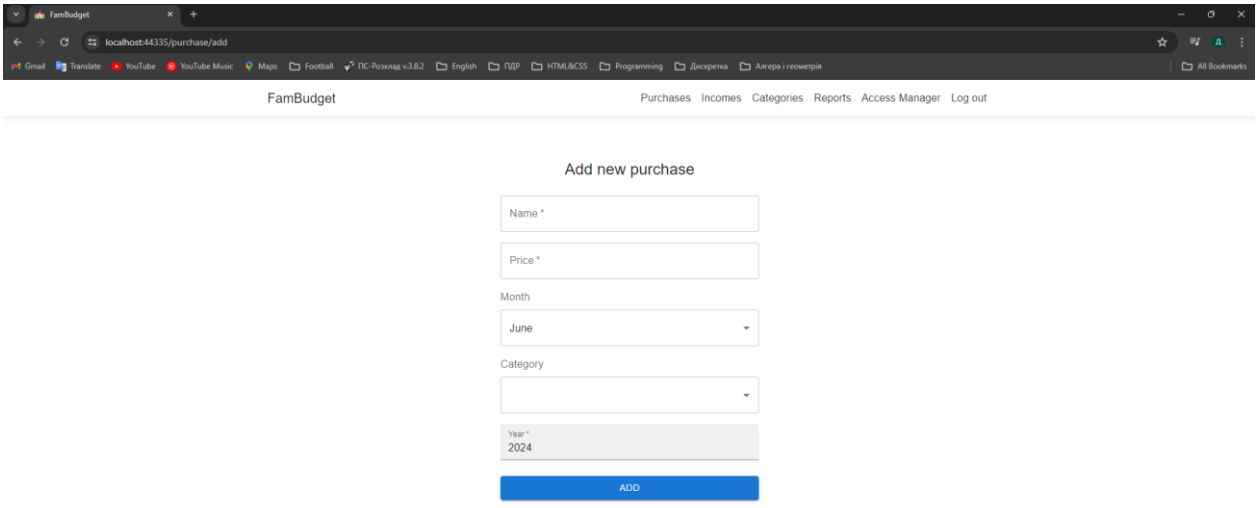


Рис. 3.6. Сторінка для додавання нової покупки

Коли користувач натискає на кнопку “Edit”, йому відображається схожа форма як і при додаванні, але вона є із заповненими даними, які вже є у цій покупці.

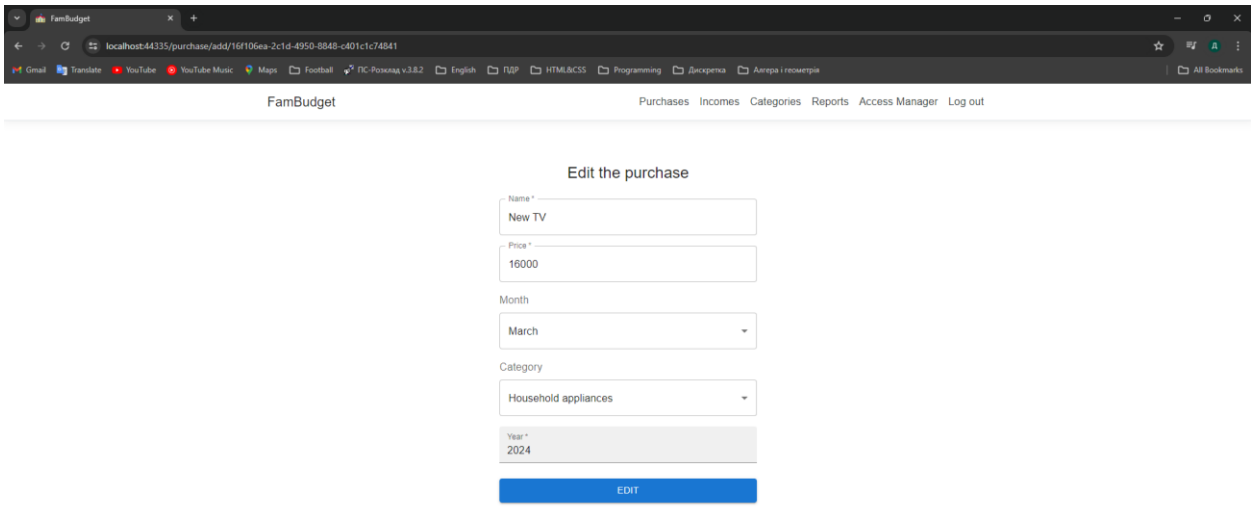


Рис. 3.7. Сторінка для редагування покупки.

Пункт меню Incomes містить список доходів у вигляді таблиці та має схожу функціональність, як і покупки:

- кнопки редагування, видалення рядочків, які мають такі ж правила клікабельності, як і покупки в залежності від доступу до доходів інших користувачів.
- видалення доходу видаляє миттєво з таблиці рядок, але не з бази даних, для видалення з бази даних потрібно натиснути кнопку збереження “Save”.
- Кнопка додавання нового доходу “Add new Income”.

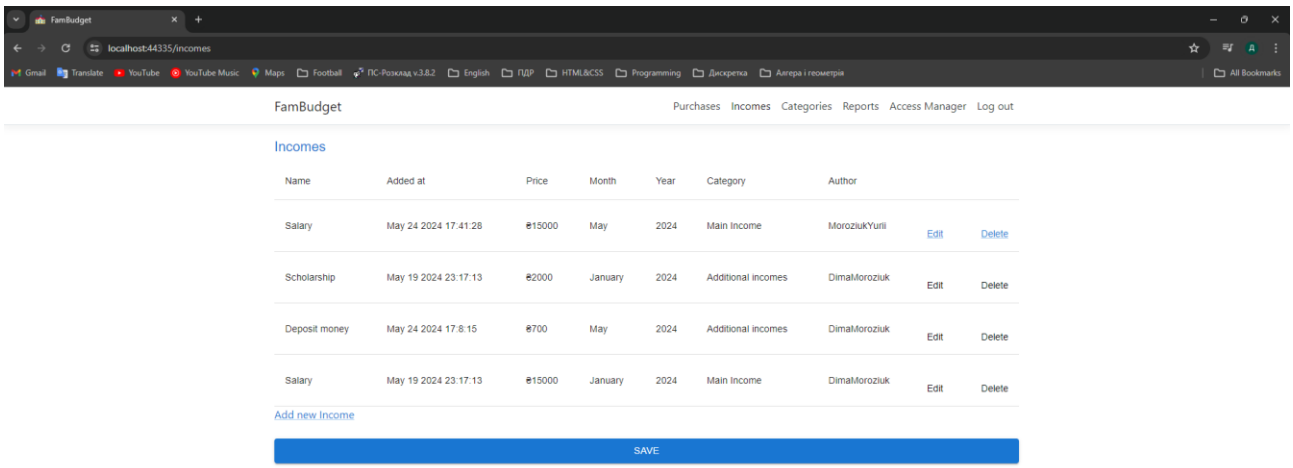


Рис. 3.8. Сторінка для перегляду доходів

При натиску на кнопку “Add new Income” відкривається сторінка для додавання нового доходу, яка містить назву, суму, місяць та рік, у якому дохід було отримано, та категорію.

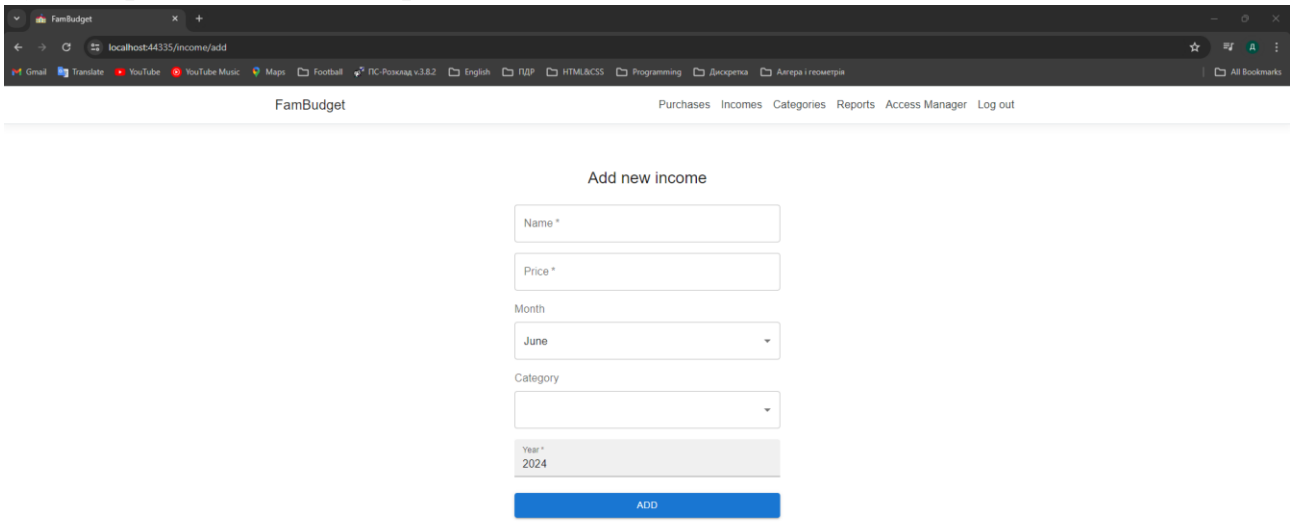


Рис. 3.9. Сторінка для додавання доходів

Коли користувач хоче відредагувати дохід, то він бачить ту ж саму сторінку, що й на додаванні, але із заповненими даними про цей дохід.

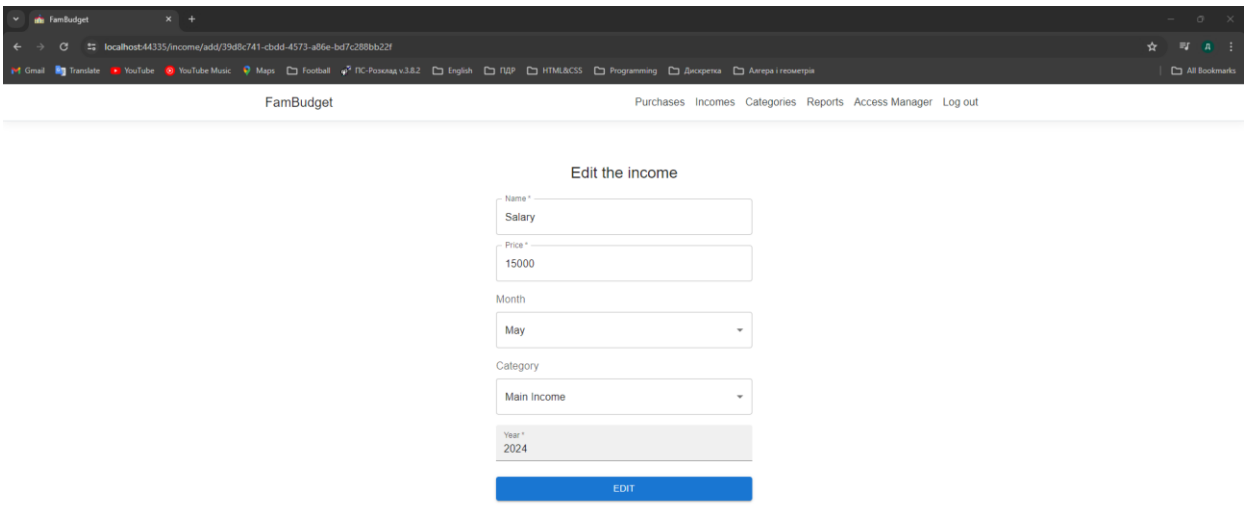


Рис. 3.10. Сторінка для редагування доходів

3.2.2. Додавання категорій та доступів

На сторінці категорій ми можемо побачити список категорій, який містить також кнопки редагування та видалення певної категорії. Видалення працює за такою ж логікою, що й на сторінці покупок та доходів.

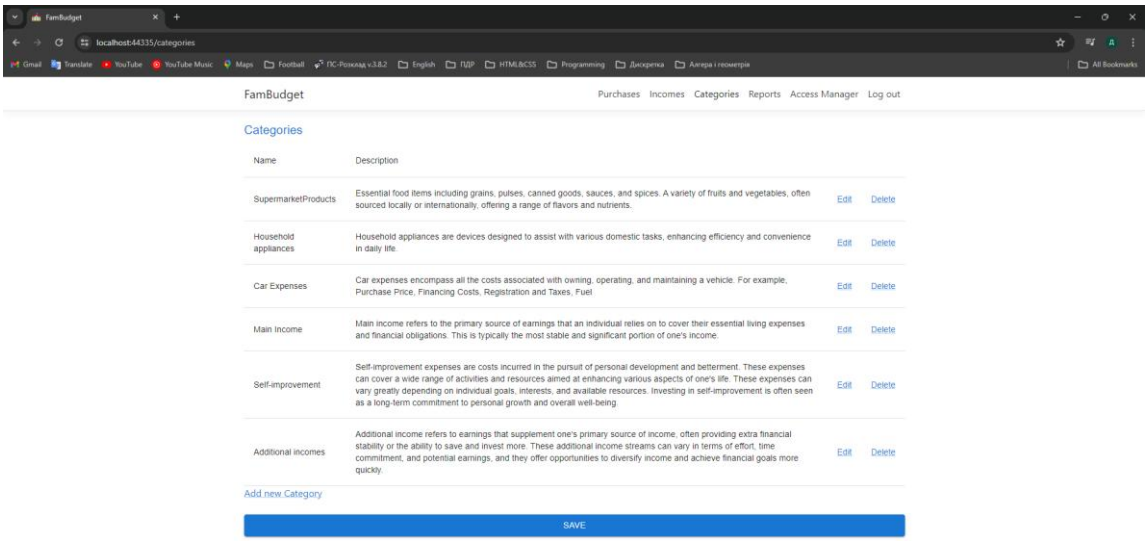


Рис. 3.11. Сторінка для перегляду категорій

Щоб додати нову категорію, потрібно натиснути на кнопку “Add new Category” та користувач буде перенесений на сторінку додавання нової категорії, яка містить назву та опис.

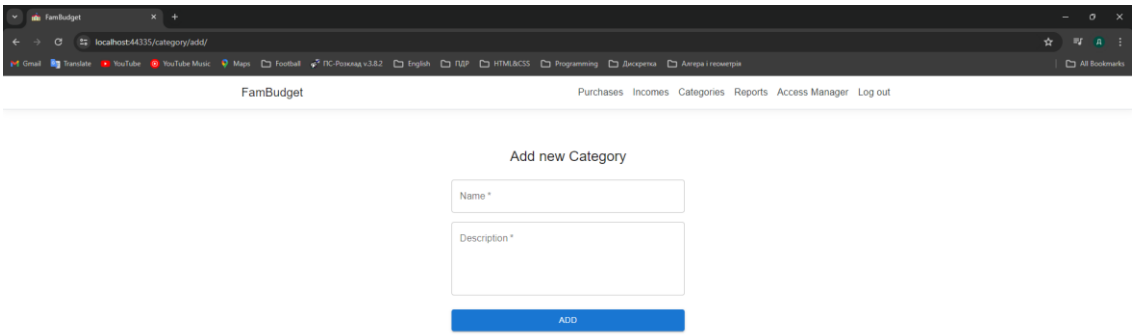


Рис. 3.12. Сторінка для додавання категорій

При редагуванні так, як і на інших сторінках, відкривається та ж сторінка, що й при додаванні, але із заповненими даними для категорії, яку модифікують.

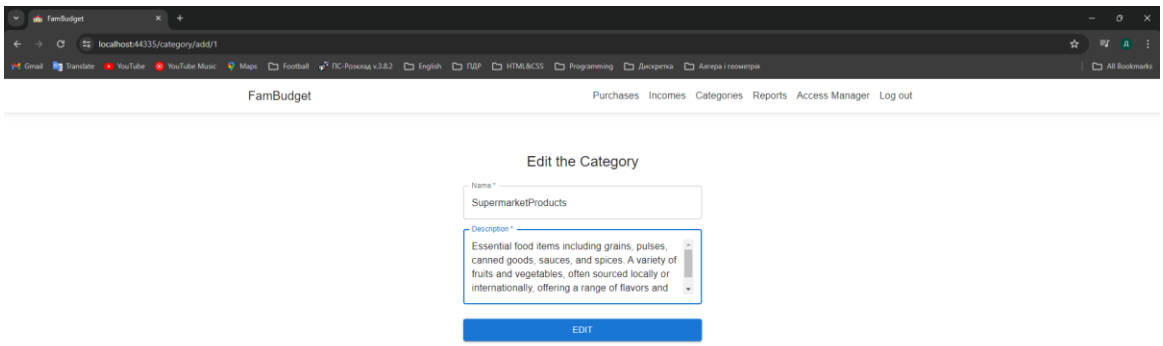


Рис. 3.13. Редагування категорії

При натиску на пункт меню “Access Manager” відкривається сторінка, де можна переглянути всіх користувачів, яким даний користувач надав доступ до своїх доходів та покупок, та рівень доступу, який був наданий.

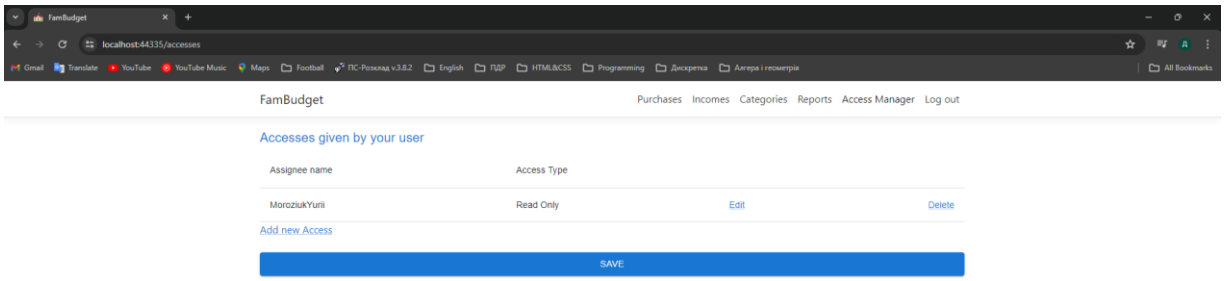


Рис. 3.14. Сторінка зі списком доступів

Сторінка додавання нового доступу має поле для введення пошти користувача, якому ви хочете надати доступ, та категорію доступу (дві категорії – лише для читання або ж з можливістю редагування).

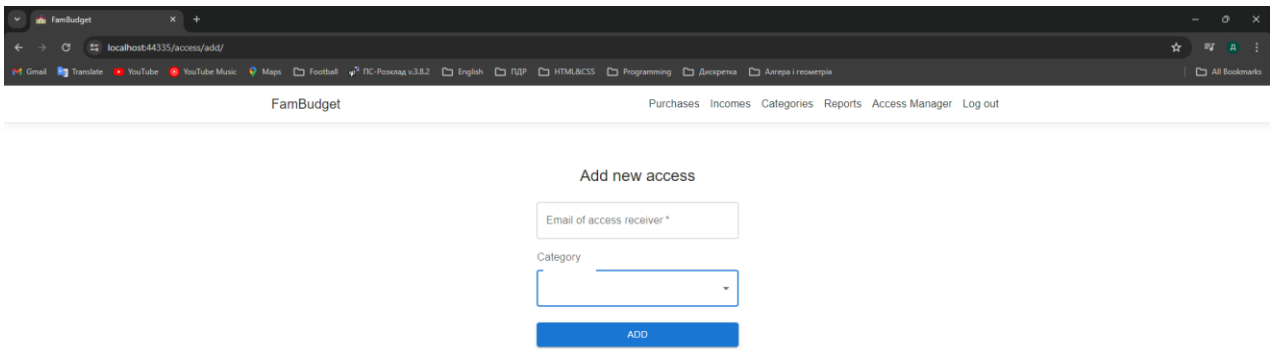


Рис. 3.15. Сторінка для додавання нового доступу.

Сторінка редагування в свою чергу виглядає так само, як сторінка додавання, але лише із заповненими даними про доступ, який користувач редагує.

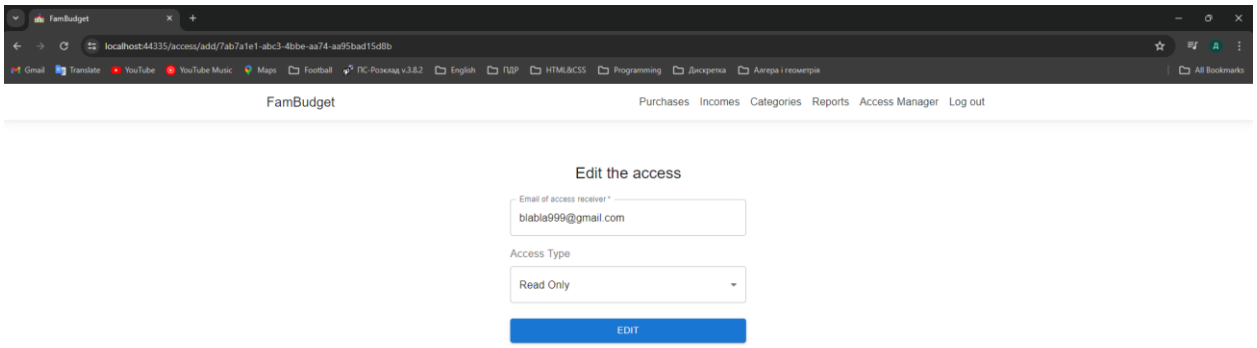


Рис. 3.16. Вигляд сторінки для редагування доступу.

3.2.3. Статистика та звіти

Як тільки користувач заповнив власні доходи та покупки, отримав доступи від інших користувачів, він має можливість отримати статистику та аналізувати свій бюджет. Для цього використовується ще один пункт меню – “Reports”.

На цій сторінці доступні 3 таби, перша з них – “Excel report”, тут можна скачати ексель файл зі статистикою про всі покупки та доходи згруповані по місяцях для певного року.

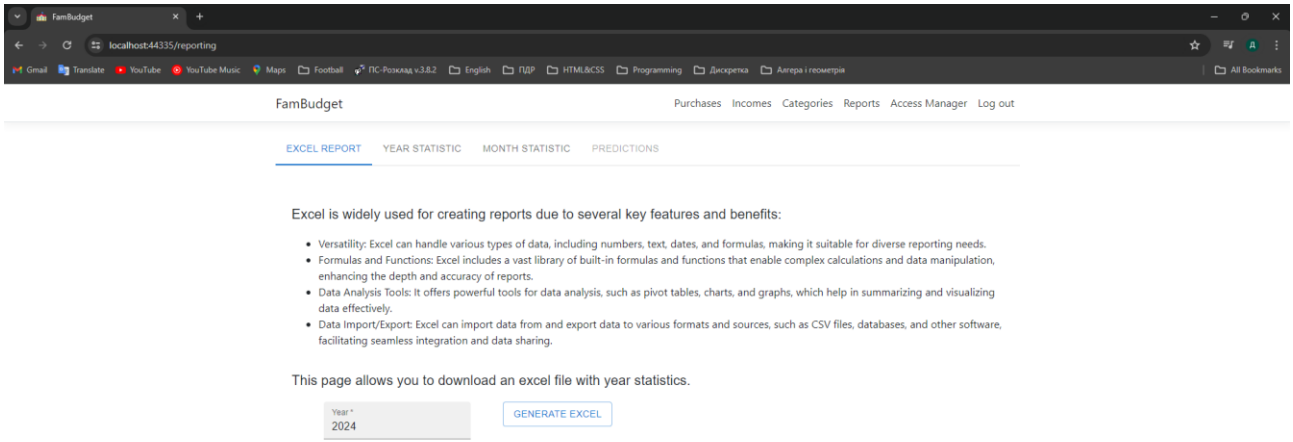
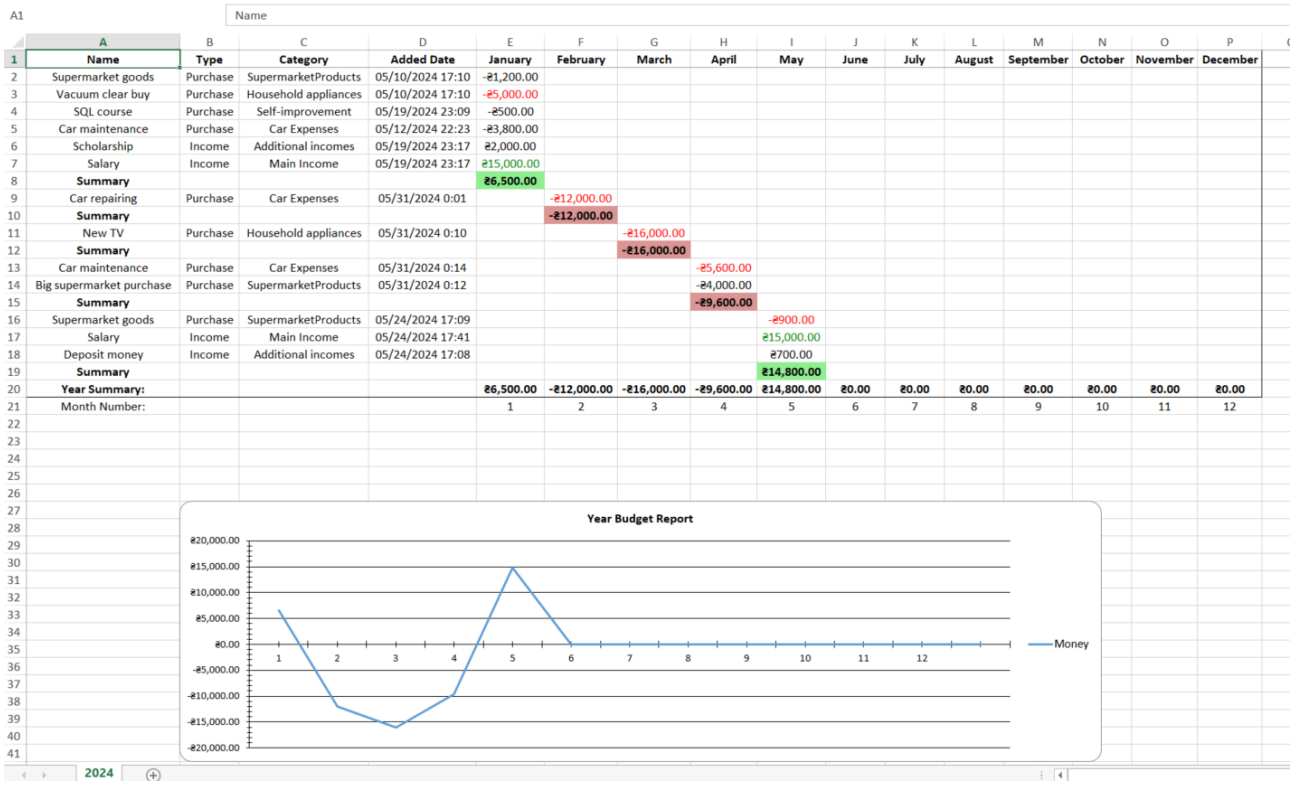


Рис.3.17. Таба Excel Report

Сам ексель файл містить інформацію про кожну покупку та дохід у році та є графік підсумкових для місяців витрат.



Name	Type	Category	Added Date	January	February	March	April	May	June	July	August	September	October	November	December
Supermarket goods	Purchase	SupermarketProducts	05/10/2024 17:10	-₴1,200.00											
Vacuum clear buy	Purchase	Household appliances	05/10/2024 17:10	-₴5,000.00											
SQL course	Purchase	Self-improvement	05/19/2024 23:09	-₴500.00											
Car maintenance	Purchase	Car Expenses	05/12/2024 22:23	-₴3,800.00											
Scholarship	Income	Additional incomes	05/19/2024 23:17	₴2,000.00											
Salary	Income	Main Income	05/19/2024 23:17	₴15,000.00											
Summary				**₴6,500.00**											
Car repairing	Purchase	Car Expenses	05/31/2024 0:01		-₴12,000.00										
Summary					**-₴12,000.00**										
New TV	Purchase	Household appliances	05/31/2024 0:10			-₴16,000.00									
Summary						**-₴16,000.00**									
Car maintenance	Purchase	Car Expenses	05/31/2024 0:14				-₴5,600.00								
Big supermarket purchase	Purchase	SupermarketProducts	05/31/2024 0:12				-₴4,000.00								
Summary							**-₴9,600.00**								
Supermarket goods	Purchase	SupermarketProducts	05/24/2024 17:09					-₴900.00							
Salary	Income	Main Income	05/24/2024 17:41					₴15,000.00							
Deposit money	Income	Additional incomes	05/24/2024 17:08					₴700.00							
Summary								**₴14,800.00**							
Year Summary:				**₴6,500.00**	**-₴12,000.00**	**-₴16,000.00**	**-₴9,600.00**	**₴14,800.00**	**₴0.00**	**₴0.00**	**₴0.00**	**₴0.00**	**₴0.00**	**₴0.00**	**₴0.00**
Month Number:				1	2	3	4	5	6	7	8	9	10	11	12

Рис. 3.18. Приклад статистики у Excel файлі

Інша таба, яка є на сторінці звітів – “Year Statistic”, на якій користувач може побачити стовпчасту діаграму витрат або доходів за рік і також знизу вказана інформація про місяць, у якому було різниця між доходами та витратами була найкращою і найгіршою.

Рис. 3.19. Діаграма Year Statistic

Остання сторінка статистики “Month Statistic” містить кругову діаграму витрат, доходів або обох згруповано по категоріям для певного місяця. Знизу також міститься інформація про найкращу і найгіршу різницю між доходами та витратами по згрупованим по категоріях даним для певного місяця.

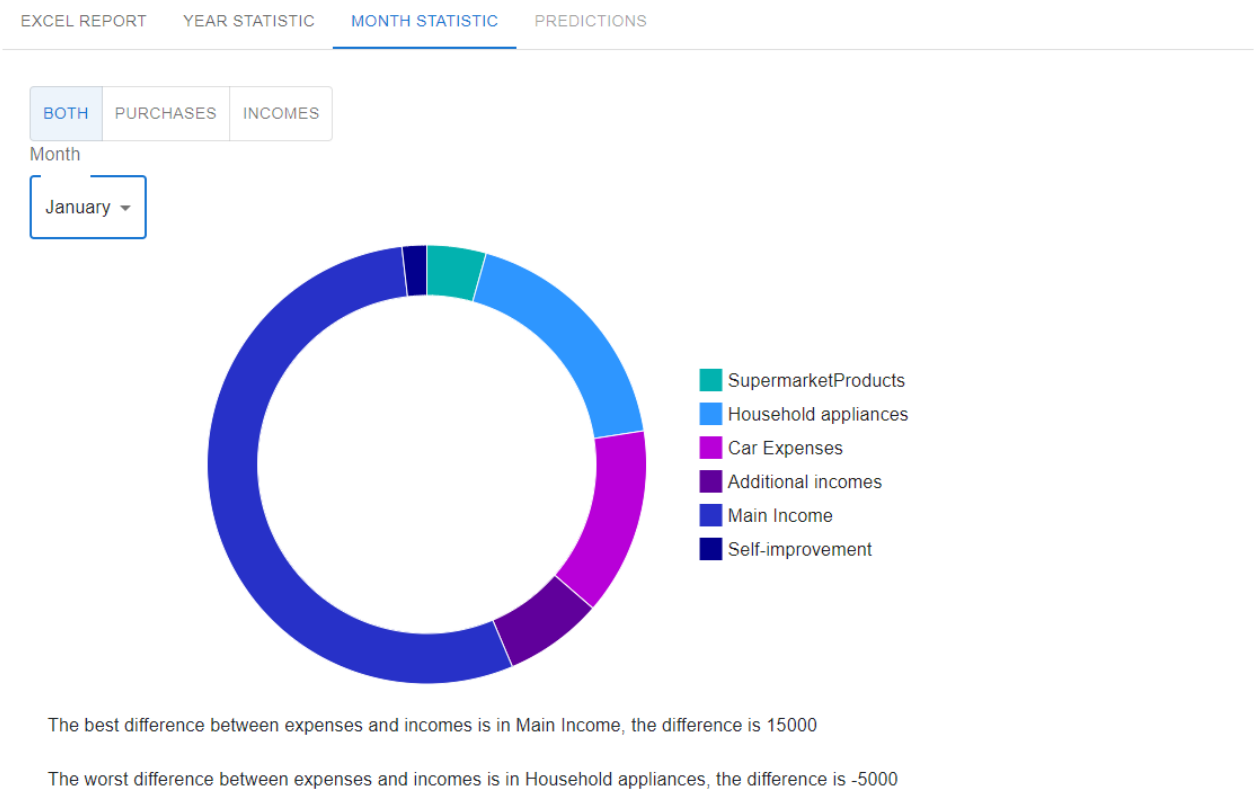


Рис. 3.20. Діаграма Month Statistic

ВИСНОВКИ

Отже, у результаті створення даного програмного забезпечення користувачі отримують хороший додаток для розподілу сімейного бюджету. За допомогою цього вебресурсу вони будуть мати можливість додавати категорії, доходи, витрати, аналізувати річний або місячний бюджет, ділитись власними покупками та витратами з іншими членами родини, що покращить фінансову комунікацію сім'ї.

У порівнянні з іншими аналогічними програмними забезпеченнями дана програма має перевагу у вигляді поширення власних покупок і доходів з іншими членами сім'ї. Аналогічні ресурси зосереджені на аналізі лише власного бюджету. Функціонал, який дозволяє ділитись власними витратами та доходами, є важливим у даного типу програмах, адже він дозволяє аналізувати бюджет всієї родини загалом, а не лише окремого члена сім'ї.

Для створення даного вебресурсу були використані сучасні технології для веброзробки: фреймворк ASP.NET з моделлю Web API для розробки серверної частини, бібліотека React для розробки клієнтської частини. Також під час розробки враховувались принципи програмування, які допоможуть у майбутньому легко підтримувати вебресурс та доповнювати його новим функціоналом. Переглянути код ресурсу можна в репозиторії GitHub за посиланням https://github.com/DimaMoroziuk/FamBudget.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ACID Properties Overview. Medium : вебсайт. URL: https://medium.com/@lorenzouriel/acid-properties-overview-83aeae4b333d (дата звернення 31.05.2024)
2. Adam Freeman Pro ASP.NET Core 6: Apress, 2022 p., 1286 c.
3. Alex Banks, Eve Porcello Learning React: Modern Patterns for Developing React Apps 2nd Edition O'Reilly, 2020 p., 300 c.
4. Dependency injection. Microsoft : вебсайт. URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/dependency-injection (дата звернення 20.05.2024)
5. Description of the database normalization basics. Microsoft : вебсайт. URL: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description (дата звернення 30.05.2024)
6. Entity Framework Core. Microsoft : вебсайт. URL: https://learn.microsoft.com/uk-ua/ef/core/ (дата звернення 18.05.2024)
7. Introduction to Identity on ASP.NET Core. Microsoft : вебсайт. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio (дата звернення 21.05.2024)
8. Layered Architecture with ASP.NET Core, Entity Framework Core and Razor Pages. Medium : вебсайт. URL: https://medium.com/aspnetrun/layered-architecture-with-asp-net-core-entity-framework-core-and-razor-pages-53a54c4028e3 (дата звернення 25.05.2024)
9. Mastering SOLID Principles: A Comprehensive Guide for Software Engineers. Medium : вебсайт. URL: https://medium.com/@GetInRhythm/mastering-solid-principles-a-comprehensive-guide-for-software-engineers-da53b054c9e1 (дата звернення 26.05.2024)

10. Tutorial: Create a web API with ASP.NET Core. Microsoft : вебсайт. URL: https://learn.microsoft.com/uk-ua/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0&WT.mc_id=dotnet-35129-website&tabs=visual-studio (дата звернення 15.05.2024)
11. What Are Single Page Applications and Why Do People Like Them So Much? bloomreach : вебсайт. URL: https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application (дата звернення 28.05.2024)