ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

УДК 004.9:61 **DOI: 10.20998/2411-0558.2025.02.11**

Є. О. ДАЦОК, студ., ХНУРЕ, Харків,
О. М. ДАЦОК, канд. техн. наук, доц. ХНУРЕ, Харків,
Д. О. РУДЕНКО, канд. техн. наук, доц. ХНУРЕ, Харків

АНАЛІЗ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ORM DAPPER ТА ПРИНЦИПІВ SOLID У ПРОЄКТУВАННІ ІНФОРМАЦІЙНОЇ СИСТЕМИ МЕДИЧНОГО ПРИЗНАЧЕННЯ

У роботі висвітлено актуальні аспекти проєктування медичних інформаційних систем із урахуванням вимог масштабованості, підтримуваності та ефективної взаємодії з базами даних. Розглянуто доцільність використання мікроORM бібліотеки Dapper у поєднанні з принципами SOLID для забезпечення чистої архітектури прикладного програмного забезпечення. Проведено аналіз архітектурних підходів, структур бази даних, а також оцінено переваги прямого контролю над SQL-запитами в контексті систем медичного призначення. Окрему увагу приділено впровадженню патернів проєктування та практик Clean Code як чинників, що впливають на якість коду та гнучкість системи. Результати роботи демонструють доцільність поєднання об'єктно-орієнтованого підходу з легковаговими ORM-інструментами для створення ефективних, адаптивних та надійних медичних інформаційних систем. Іл.: 2. Табл.: 2. Бібліогр.: 10 назв.

Ключові слова: інформаційна система, медичне програмне забезпечення, SOLID, Dapper, ORM, патерни проєктування, Clean Code, SQL Server, обробка медичних даних.

Постановка проблеми. У сучасних медичних установах стрімке зростання обсягів даних, необхідність забезпечення надійного обліку пацієнтів, процедур та фінансових операцій потребують впровадження ефективних інформаційних систем. Водночас підвищуються вимоги до архітектурної якості програмного забезпечення (ПЗ): воно має бути масштабованим, підтримуваним, адаптивним до змін і водночас простим у супроводі [1]. Особливої актуальності набуває побудова таких систем із дотриманням принципів чистої архітектури, використанням об'єктно-орієнтованого проєктування та ефективної взаємодії з базами даних (БД).

Одним із викликів при розробці прикладного ПЗ для медичної сфери є вибір технологічного стеку, який би одночасно забезпечував високу продуктивність, зручність обробки даних, прозору структуру коду та можливість швидкого розширення функціоналу. Популярні ORM-інструменти, як-от Entity Framework, хоча й спрощують розробку, можуть виявитися надмірно важкими або обмежувальними в умовах високої

інтенсивності запитів і потреби точного контролю над SQL-логікою [2]. У цьому контексті актуальним є дослідження ефективності застосування мікроORM бібліотеки Dapper, яка забезпечує мінімальні непрямі витрати (overhead) при роботі з БД та дозволяє напряму реалізовувати SQL-запити.

Паралельно з цим постає потреба в дотриманні архітектурних принципів, зокрема, SOLID, як основи для побудови логічно структурованої, гнучкої та розширюваної системи. У поєднанні з шаблонами проєктування та практиками Clean Code, такі підходи сприяють створенню якісного коду, що легко тестується та змінюється без порушення існуючої функціональності.

Таким чином, проблема полягає у визначенні ефективності та доцільності застосування поєднання архітектурних принципів, патернів та легковагових засобів ORM при розробці інформаційної системи для медичного призначення, що має відповідати сучасним вимогам до ПЗ в умовах реального використання.

Аналіз літератури. У сучасних дослідженнях зростає увага до побудови гнучких та підтримуваних медичних інформаційних систем (МІС), які здатні ефективно обробляти великі обсяги даних, зберігаючи при цьому структурованість і безпеку. У роботі [1] здійснено систематичний огляд застосування корпоративної архітектури в мережах охорони здоров’я. Автори доводять, що використання таких підходів як багаторівнева структура та логічна декомпозиція систем значно підвищує інтеграційні можливості та знижує технічну заборгованість у великих медичних мережах.

Дослідження [2] доповнює попереднє, акцентуючи увагу на складності адаптації архітектурних моделей у країнах із різним рівнем розвитку ІТ-інфраструктури. Автори наводять результати огляду 53 наукових праць і визначають найчастіше застосовувані архітектурні шаблони в МІС, зокрема Service-Oriented Architecture (SOA), Microservices та Clean Architecture.

У сфері роботи з БД значну увагу приділено порівнянню ORM-інструментів. В [3] наведено експериментальний аналіз ефективності Dapper, Entity Framework Core та інших рішень у .NET 6. Згідно з отриманими результатами, Dapper демонструє найвищу продуктивність при виконанні CRUD-операцій, що є критичним у медичних системах, де швидкість запиту напряму впливає на якість обслуговування.

У роботі [4] обґрунтовується перевага Dapper у проєктах із високою інтенсивністю звернень до БД. Автори наголошують на тому, що при

правильному використанні шаблонів, таких як Repository та Unit of Work, мікроORM може бути не лише швидким, але й архітектурно чистим рішенням. У цьому контексті також згадується важливість дотримання принципів SOLID для підтримуваності коду.

Серія публікацій [5 – 7] присвячена детальному аналізу принципів SOLID в архітектурі застосунків. В огляді [5] пояснюються практичні приклади реалізації кожного з принципів в умовах реального проєкту на C#, включаючи структурування репозиторіїв, інверсію залежностей та ізоляцію бізнес-логіки. Автори статей [6, 7] підкреслюють, що SOLID-архітектура дозволяє ефективно розвивати систему в умовах частих змін, що є типовим для галузі охорони здоров'я, де нормативні вимоги оновлюються регулярно.

Отже, огляд літератури підтверджує, що поєднання мікроORM Dapper із принципами SOLID та сучасними архітектурними шаблонами (Clean Architecture, MVC) забезпечує високу ефективність, масштабованість та підтримуваність МІС. Це створює надійне підґрунтя для подальшого розвитку інформаційних систем у медицині, зокрема в напрямку персоналізованої медицини та інтеграції зі сторонніми сервісами.

Мета статті – аналіз ефективності використання мікроORM бібліотеки Dapper та принципів SOLID у процесі проєктування та реалізації інформаційної системи медичного призначення. Дослідження спрямоване на обґрунтування доцільності вибору зазначених інструментів і архітектурних підходів з урахуванням вимог до масштабованості, підтримуваності та продуктивності ПЗ в умовах медичної сфери.

Роль архітектури в МІС.

МІС забезпечують зберігання, обробку та доступ до клінічних даних, що вимагає високої надійності, масштабованості, супроводжуваності та дотримання стандартів інформаційної безпеки. Архітектура ПЗ відіграє визначальну роль у структурній організації компонентів, забезпеченні інтеграції із зовнішніми сервісами (eHealth, LIS, телемедицина) та адаптації до нормативних змін.

Недоліки монолітних архітектур – зокрема низька гнучкість і складність модифікації – спричинили перехід до багаторівневих та чистих архітектур (Layered, Clean Architecture), які передбачають логічне розділення відповідальностей між ізольованими модулями, зменшення зв'язаності та підвищення тестованості.

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

У реалізованій системі "Стоматологічна клініка" застосовано три ключові рівні.

Presentation Layer: відповідає за інтерфейс взаємодії з користувачем; повністю ізольований від бізнес-логіки, що спрощує модифікацію UI.

Business Logic Layer: реалізує доменні сценарії (візити, облік послуг тощо) згідно з принципами SOLID, що підвищує підтримуваність та адаптивність.

Data Access Layer: забезпечує доступ до реляційної бази даних (SQL Server) через Dapper – мікроORM, який забезпечує пряме управління SQL-запитами з мінімальним накладом.

Взаємодія між рівнями реалізована через інтерфейси, що відповідає принципу інверсії залежностей, сприяє модульності та полегшує тестування. Така структура відкриває перспективи переходу до мікросервісної архітектури з незалежними компонентами.

Гнучкість архітектури дозволяє швидко адаптувати систему до нових вимог: додавання типів процедур, інтеграція з API, впровадження електронного рецепта тощо. Це безпосередньо впливає на якість обслуговування, зменшення ризику помилок і оперативність прийняття рішень.

Отже, багаторівнева архітектура в поєднанні з об’єктно-орієнтованим підходом, шаблонами проєктування та принципами SOLID формує технічно стійку, адаптивну та масштабовану основу для сучасних МІС.

Обґрунтування вибору ORM-рішення та Dapper у МІС.

У контексті розробки МІС критично важливим є вибір інструменту доступу до БД, що забезпечуватиме високу продуктивність, контроль над запитами та надійність обробки чутливих даних. Традиційно для .NET-платформи використовується Entity Framework Core (EF Core), однак у реалізованому проєкті було обрано легковагову мікроORM Dapper.

Dapper надає можливість прямої роботи з SQL-запитами, мінімізуючи overhead та виключаючи непрозору генерацію коду. Це забезпечує високу швидкодію (в 2 – 3 рази швидше за EF Core при роботі з тисячами записів), передбачуваність виконання запитів і зручність мапінгу результатів на строго типізовані доменні об’єкти.

У системі запити до SQL Server реалізовані у вигляді параметризованих конструкцій через патерн Repository, що відповідає принципам SOLID. Це гарантує як захист від SQL-ін’єкцій, так і ізоляцію бізнес-логіки від механізмів доступу до даних. Dapper зберігає гнучкість у

реалізації складних JOIN-, фільтраційних та агрегатних операцій, що необхідно в медичних сценаріях.

Крім технічних переваг, застосування Dapper обґрунтоване також можливістю його заміни на інші засоби (EF, ADO.NET) без впливу на основну архітектуру. Це робить рішення адаптивним до змін функціональних або регуляторних вимог.

Отже, використання Dapper у складі МІС є виправданим як з точки зору продуктивності й надійності, так і з позицій підтримуваності архітектури у середовищі з високими вимогами до даних.

Порівняльна характеристика ORM-рішень у контексті МІС.

У розробці інформаційних систем значного масштабу, особливо у сфері охорони здоров'я, важливу роль відіграє вибір засобів взаємодії з БД. Object-Relational Mapping (ORM) – це клас технологій, що дозволяє перетворювати дані між реляційною моделлю та об'єктно-орієнтованим кодом. Серед найпоширеніших ORM для платформи .NET є Entity Framework (EF Core), Dapper та ADO.NET як базова альтернатива без ORM.

Вибір інструменту напряму впливає на продуктивність, гнучкість, читабельність коду, тестованість і швидкість розробки, а також на надійність обробки критичних медичних даних. У даному дослідженні проведено порівняльну характеристику згаданих технологій на основі релевантних для МІС критеріїв (табл.1).

Аналіз представленої інформації демонструє, що для МІС ключовими є продуктивність, повний контроль над SQL, гнучке моделювання запитів, тестованість і низька ймовірність логічних збоїв.

Entity Framework Core дає високу швидкість первинної розробки (автоматичне генерування моделей, LINQ, інтеграція з .NET), що знижує бар'єр входу й прискорює CRUD-операції. Однак при складних фільтраціях та великих наборах даних EF Core обмежує гнучкість і поступається за швидкодією.

Dapper забезпечує максимальний контроль над SQL, критично важливий у медичних системах, де потрібна точна й перевірювана логіка. У проєкті "Стоматологічна клініка" Dapper підвищив швидкість запитів у кілька разів, спростив журналювання доступу до чутливої інформації та, у поєднанні з патерном Repository, зберіг високу тестованість і масштабованість.

Таблиця 1

Порівняльна характеристика ORM-рішень у контексті МІС

Критерій	**Entity Framework Core**	**Dapper**	**ADO.NET (ручне написання)**
Продуктивність (запити)	Середня (≈ 1.5× повільніше за Dapper)	Висока (максимальна швидкість завдяки прямим SQL)	Висока, але ціною обсягу коду
Контроль над SQL-запитами	Обмежений (автогенерація LINQ → SQL)	Повний (ручне написання запитів)	Повний
Складність навчання / впровадження	Низька (простий API, документація)	Середня (потрібні знання SQL, мапінг вручну)	Висока (багато шаблонного коду)
Швидкість розробки MVP	Висока (швидкий старт, scaffolding)	Середня (довше, але контрольованіше)	Низька (багато рутинного коду)
Гнучкість оптимізації запитів	Обмежена (важко впливати на джойни, індекси)	Висока (SQL-підлаштований під бізнес-запити)	Висока
Ймовірність помилки в SQL	Низька (ORM перевіряє синтаксис)	Середня (написання ручне, ретельна перевірка)	Висока (легко припуститися помилки)
Ізоляція логіки від БД	Часткова	Повна через патерн Repository	Мінімальна (на рівні коду)
Тестованість (Mock/DI)	Середня	Висока (інтерфейси, легко мокати)	Висока, але з ускладненнями в підтримці
Підтримка асинхронності	Повна (async/await)	Повна (async методи + збереження продуктивності)	Повна, але потрібно реалізовувати вручну
Залежність від сторонніх пакетів	Microsoft.EntityFrameworkCore	System.Data.SqlClient + Dapper (легка)	System.Data.SqlClient (стандартна бібліотека)
Ускладнення при розширенні	Високі при нетипових випадках (custom SQL)	Мінімальні (гнучка інтеграція будь-якої логіки)	Високі (кожне нове правило — новий код вручну)
Придатність для медичних ІС	Помірна (зручно, але не завжди контрольовано)	Висока (гнучкість, контроль, швидкість критично важливі)	Обмежена (ризик помилок, важкий супровід)

ADO.NET надає найбільшу продуктивність і повний контроль, але вимагає ручного керування підключеннями, транзакціями й мапінгом, що збільшує обсяг шаблонного коду та ускладнює супровід – недоцільно для середніх і великих медичних проєктів.

Отже, Dapper пропонує оптимальний компроміс між швидкістю, контрольованістю та архітектурною гнучкістю, уникаючи надмірної абстракції EF Core та трудомісткості ADO.NET, – що робить його доцільним вибором для критично важливих МІС.

Аналітичний огляд сучасних програмних рішень у сфері МІС.

Було проаналізовано три відкриті програмні рішення, які призначені для зберігання, обробки та управління медичними даними. Метою аналізу було виявлення архітектурних особливостей, засобів доступу до БД та рівня застосування принципів об’єктно-орієнтованого проєктування, а також проведення порівняння з інформаційною системою, реалізованою в межах цього проєкту (табл. 2).

Додаткові обмеження у використанні SOLID та Dapper у МІС.

У практиці МІС існують певні обмеження, зумовлені високим рівнем регуляторних вимог, неоднозначною динамікою змін функціональності та пріоритетом безпеки над гнучкістю.

Зокрема, надмірна декомпозиція за принципами SOLID (особливо S – Single Responsibility та I – Interface Segregation) у певних випадках ускладнює трасування потоку даних і логіки в системах, де критично важлива передбачуваність. За умови реалізації великої кількості дрібних сервісів, інтерфейсів і залежностей складно досягнути прозорого контролю над кожним етапом взаємодії з медичною інформацією – а саме це часто вимагається аудиторами безпеки чи при сертифікації (ISO 27799).

У випадку використання Dapper високий рівень свободи у написанні SQL-запитів несе ризики у великих командах розробників, особливо при відсутності централізованого керування запитами. Це може призвести до неконсистентного підходу до обробки даних (різні підходи до пагінації, агрегації, фільтрації); складності аудиту запитів при медичних інцидентах; важчої підтримки та ревізії безпеки порівняно з повноцінними ORM, де поведінка прозоріша.

Відсутність вбудованих механізмів відстеження змін (change tracking), які притаманні Entity Framework, ускладнює реалізацію функцій журналювання змін або аудиту – що особливо важливо в системах, де

ведеться облік доступу до історії лікування або редагування медичних записів.

Таблиця 2

Порівняльна характеристика відкритих програмних рішень у сфері МІС

Критерій	**OpenEMR**	**GNU Health**	**MediTracker**
Мова програмування	PHP	Python	Python
База даних	MySQL	PostgreSQL	SQLite / PostgreSQL
Тип ORM / доступу до БД	Прямий SQL (без ORM)	Tryton (декларативна ORM)	SQLAlchemy (ORM)
Архітектура	Частково багаторівнева	Модульна, декларативна	MVC + REST, трирівнева
Дотримання принципів SOLID	Низьке (порушення SRP, DIP)	Часткове (зв'язаність модулів)	Часткове (шаблони, ізоляція шарів)
Продуктивність	Середня	Середня / висока	Середня
Масштабованість	Обмежена через складність супроводу	Висока завдяки модульності	Обмежена (придатна для малих проєктів)
Придатність для малих проєктів	Низька (надмірна складність)	Низька (крива навчання)	Висока (простота й зрозумілість)
Основні переваги	Функціональна повнота, популярність	Модульність, аналітична гнучкість	Простота реалізації та підтримки
Основні недоліки	Legacy-код, низька модульність	Висока складність, ресурсомісткість	Менша продуктивність ORM

Особливості проєктування БД та реалізації доступу до даних за допомогою Dapper.

У процесі розробки МІС ключову роль відігравало проєктування реляційної БД, орієнтованої на структуроване та масштабоване зберігання клінічної інформації. Базова модель охоплює основні сутності – Patients, Doctors, Visits, Services – та зв'язки між ними. Схема нормалізована до третьої нормальної форми, що мінімізує надлишковість і забезпечує

цілісність завдяки коректному використанню первинних і зовнішніх ключів.

У розширеній структурі (рис. 1) проміжна таблиця VisitServices реалізує зв'язок "багато-до-багатьох" між візитами та послугами, що відображає реальні сценарії лікування й дозволяє формувати запити зі складними JOIN-конструкціями для побудови аналітичних звітів.

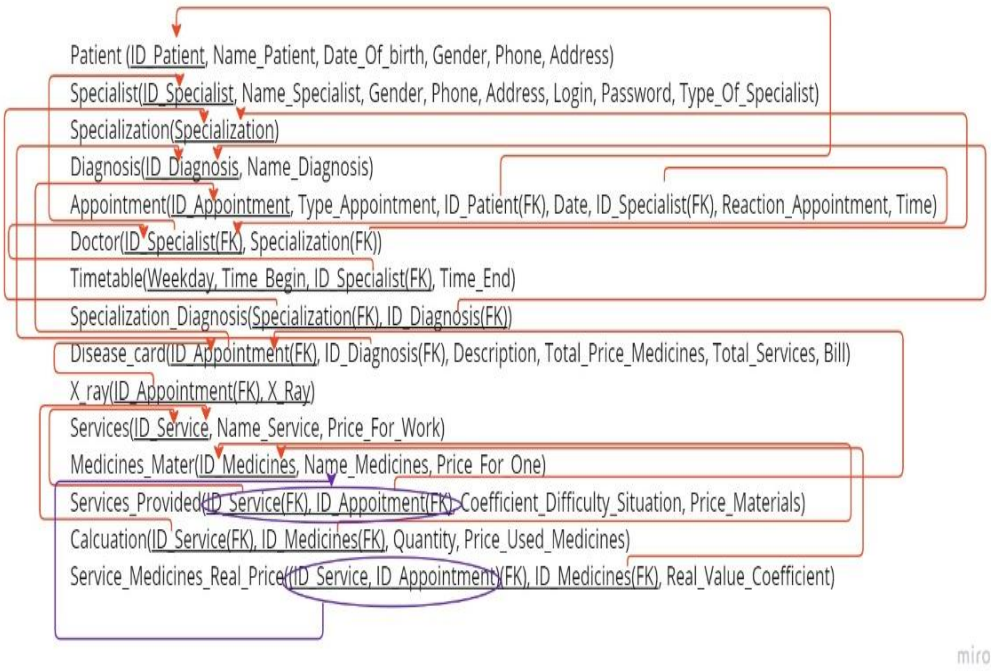

Рис. 1. Логічна структура бази даних інформаційної системи медичного призначення

Для доступу до даних використано Dapper – мікроORM, що забезпечує високу продуктивність і повний контроль над SQL. Запити мапляться на строго типізовані класи, що спрощує тестування й логування. Усі запити реалізовані в репозиторіях за відповідними інтерфейсами (наприклад, IVisitRepository) у межах патерну Repository. Вони параметризовані, що гарантує безпеку та гнучкість при фільтрації медичних даних. Поєднання нормалізованої БД, Dapper і принципів SOLID забезпечує стабільність, продуктивність та адаптивність системи до змін і інтеграції з іншими підсистемами.

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

Реалізація принципів SOLID у доступі до даних МІС.

У створеній системі "Стоматологічна клініка" особливу увагу приділено архітектурі доступу до даних, яка реалізована з урахуванням принципів SOLID. Це забезпечило гнучкість, масштабованість, високу підтримуваність, можливість модульного тестування та спрощену інтеграцію нових функціональних компонентів.

Загальна структура реалізації репозиторіїв, інтерфейсів та спадкування від базового класу зображена на UML-діаграмі (рис. 2).

S – Single Responsibility Principle реалізовано на рівні кожного окремого репозиторію: кожен клас відповідає лише за логіку однієї сутності [8]. Наприклад, PatientRepository оперує виключно таблицею пацієнтів, не зачіпаючи логіку інших доменів.

O – Open/Closed Principle дотримано завдяки застосуванню інтерфейсів для репозиторіїв. Поведінку класів можна розширювати без зміни вихідного коду, наприклад, реалізуючи додаткові методи у нових класах на основі IAppointmentRepository.

L – Liskov Substitution Principle підтримується через базовий клас Repository, який інкапсулює логіку підключення до БД [9]. Конкретні реалізації можуть взаємозамінно використовуватися в рамках абстрактних контекстів.

I – Interface Segregation Principle забезпечено поділом інтерфейсів за предметними областями (IPatientRepository, ISpecialistRepository, IServiceRepository тощо). Це дозволяє реалізовувати лише релевантні методи, уникаючи перевантажених залежностей [10].

D – Dependency Inversion Principle реалізовано за допомогою впровадження залежностей через абстракції. Залежності типу IPatientRepository передаються через DI-контейнери, що дозволяє легко підмінювати реалізації (наприклад, на мок-об’єкти для юніт-тестів) без зміни споживача.

Така побудова сприяє централізованому доступу до даних, чіткому розмежуванню відповідальностей та відповідає вимогам безпечної розробки в МІС.

ISSN 2079-0031 (Print) ISSN 2411-0558 (Online)

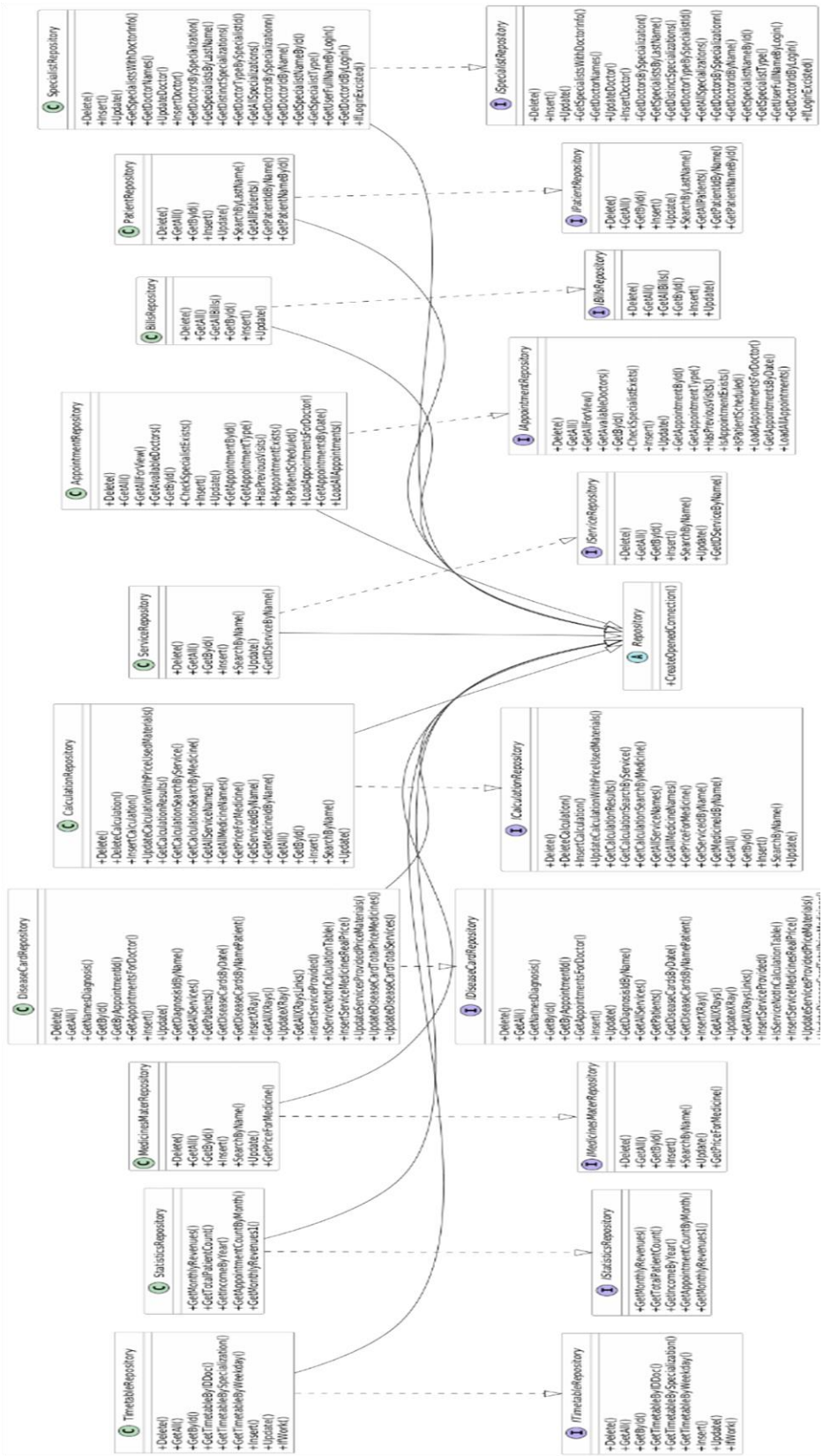

Рис. 2 UML-діаграма репозиторіїв у структурі доступу до даних

Висновки. Інтеграція принципів SOLID та мікроORM Dapper у проєктуванні медичних інформаційних систем є обґрунтованим прикладом поєднання архітектурної строгості з прикладними вимогами клінічного середовища. Проведене дослідження підтверджує, що технічні рішення повинні враховувати не лише продуктивність і масштабованість, але й нормативні, етичні та експлуатаційні особливості сфери охорони здоров'я.Використання багаторівневої архітектури у поєднанні з Dapper забезпечує контрольовану взаємодію з базами даних, високу продуктивність і гнучкість інтеграції. Дотримання принципів SOLID підвищує підтримуваність і тестованість системи, що є критичним для безперервної еволюції програмного забезпечення в медичних закладах.

Обраний підхід продемонстрував ефективність в умовах розробки MIC, відкриваючи перспективи для подальшої інтеграції з аналітичними та діагностичними модулями, орієнтованими на підтримку прийняття клінічних рішень.

Список літератури:

1. Maris A., Hoppenbrouwers S., Ravesteijn P., van Hillegersberg J. Enterprise Architecture in Healthcare Networks: A Systematic Literature Review // *Communications of the IIMA,* 2023, Vol. 21, No. 1, Article 5. URL: https://doi.org/10.58729/1941-6687.1446.

2. A Survey on the Application of Enterprise Architecture in Healthcare Systems / F. P. Alencar de Medeiros та ін. *International Journal of Enterprise Information Systems,* 2021, Т. 17, № 3. С. 1–15. URL: https://doi.org/10.4018/ijeis.2021070101 .

3. Performance Analysis of Object-Relational Mapping (ORM) Tools in .NET 6 // *Bilişim Teknolojileri Dergisi*, 2022, Vol. 15, No. 4, pp. 459–464. URL: https://doi.org/10.17671/gazibtd.1059516.

4. Performance Comparison of CRUD Methods using NET Object Relational Mappers: A Case Study / D. Zmaranda та ін. *International Journal of Advanced Computer Science and Applications,* 2020, Т. 11, № 1. URL: https://doi.org/10.14569/ijacsa.2020.0110107).

5. NS K. EF Core vs Dapper Performance in 2025: Which One Should You Choose?. *Medium*. URL: https://medium.com/@karthikns999/ef-core-vs-dapper-performance-dotnet8-2025-3a3c803cb3ff (дата звернення: 04.06.2025).

6. SOLID Principles in C#. C# Corner. URL: https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/ (дата звернення: 06.06.2025).

7. Understanding SOLID Principles in .NET. Dot Net Tricks. URL: https://www.dotnettricks.com/learn/designpatterns/solid-principles-with-real-time-examples (дата звернення: 08.06.2025).

8. Mastering SOLID Principles in C# – Beginner to Advanced. Code Maze. URL: https://code-maze.com/solid-principles/ (дата звернення: 08.06.2025).

9. SOLID Design Principles explained with examples. Refactoring Guru. URL: https://refactoring.guru/design-patterns/solid-principles (дата звернення: 10.06.2025).

10. SOLID Principles of Object-Oriented Design: Explained with C# Examples. Educative. URL: https://www.educative.io/blog/solid-principles-object-oriented-design (дата звернення: 10.06.2025).

References:

1. Maris, A, Hoppenbrouwers, S, Ravesteijn, P & van Hillegersberg, J (2023), Enterprise Architecture in Healthcare Networks: A Systematic Literature Review. in Proceedings of the 34th Annual Conferences of the International Information Management Association, Vol. 21, No. 1, Article 5. DOI: 10.58729/1941-6687.1446.
2. F. P. Alencar de Medeiros et al. (2021) A review of the application of enterprise architecture in healthcare systems / *International Journal of Enterprise Information Systems*. T. 17, № 3. C. 1-15. URL: https://doi.org/10.4018/ijeis.2021070101.
3. Güvercin, A. E., & Avenoglu, B. (2022). Performance Analysis of Object-Relational Mapping (ORM) Tools in .Net 6 Environment. Bilişim Teknolojileri Dergisi, 15(4), 453-465. https://doi.org/10.17671/gazibtd.1059516.
4. D. Zmaranda et al. (2020) Comparison of the performance of CRUD methods using NET object-relational mappers: A Case Study / *International Journal of Advanced Computer Science and Applications.* T. 11, № 1. URL: https://doi.org/10.14569/ijacsa.2020.0110107.
5. NS K. EF Core vs Dapper Performance in 2025: Which One Should You Choose? Medium. URL: https://medium.com/@karthikns999/ef-core-vs-dapper-performance-dotnet8-2025-3a3c803cb3ff (accessed 04.06.2025).
6. SOLID principles in C#. C# Corner. URL: https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/ (accessed 06.06.2025).
7. Understanding SOLID Principles in .NET. Dot Net Tricks. URL: https://www.dotnettricks.com/learn/designpatterns/solid-principles-with-real-time-examples (accessed 08.06.2025).
8. Mastering SOLID Principles in C# – Beginner to Advanced. Code Maze. URL: https://code-maze.com/solid-principles/ (accessed 08.06.2025).
9. SOLID Design Principles explained with examples. Refactoring Guru. URL: https://refactoring.guru/design-patterns/solid-principles (accessed 10.06.2025).
10. SOLID Principles of Object-Oriented Design: Explained with C# Examples. Educative. URL: https://www.educative.io/blog/solid-principles-object-oriented-design (accessed 10.06.2025).

Статтю представив д-р техн. наук, проф. Національного технічного університету "Харківський політехнічний інститут" В.І. Носков.

Надійшла (*received*) *15.03.2025*

Datsok Yevheniia, student
Kharkiv National University of Radio Electronics
Nauky Ave. 14, Kharkiv, Ukraine, 61166
Tel: +421 952 155 697, e-mail: yevheniia.datsok@nure.ua
ORCID ID: 0009-0008-5101-5217

Datsok Oleh, Cand.Sc.Tech, Associate Professor
Kharkiv National University of Radio Electronics
Nauky Ave. 14, Kharkiv, Ukraine, 61166

Tel: (057) 7021-364, e-mail: oleh.datsok@nure.ua
ORCID ID: 0000-0003-4489-3819

Rudenko Diana, Cand.Sc.Tech, Associate Professor
Kharkiv National University of Radio Electronics
Nauky Ave. 14, Kharkiv, Ukraine, 61166
Tel: (057) 7021-419, e-mail: diana.rudenko@nure.ua
ORCID ID: 0000-0002-1792-5080

УДК 004.9:61

Аналіз ефективності використання ORM DAPPER та принципів SOLID у проєктуванні інформаційної системи медичного призначення / Дацок Є.О., Дацок О.М., Руденко Д.О. // Вісник НТУ "ХПІ". Серія: Інформатика та моделювання. – Харків: НТУ "ХПІ". – 2025. – № 2 (14). – С. 157 – 171.

У роботі висвітлено актуальні аспекти проєктування медичних інформаційних систем із урахуванням вимог масштабованості, підтримуваності та ефективної взаємодії з базами даних. Розглянуто доцільність використання мікроORM-бібліотеки Dapper у поєднанні з принципами SOLID для забезпечення чистої архітектури прикладного програмного забезпечення. Проведено аналіз архітектурних підходів, структур бази даних, а також оцінено переваги прямого контролю над SQL-запитами в контексті систем медичного призначення. Окрему увагу приділено впровадженню патернів проєктування та практик Clean Code як чинників, що впливають на якість коду та гнучкість системи. Результати роботи демонструють доцільність поєднання об’єктно-орієнтованого підходу з легковаговими ORM-інструментами для створення ефективних, адаптивних та надійних медичних інформаційних систем. Іл.: 2. Табл.: 2. Бібліогр.: 10 назв.

Ключові слова: інформаційна система, медичне програмне забезпечення, SOLID, Dapper, ORM, патерни проєктування, Clean Code, SQL Server, обробка медичних даних.

UDC 004.9:61

Analysis of effectiveness of using ORM DAPPER and SOLID principles in the design of a medical information system / Datsok Y., Datsok O., Rudenko D // Herald of the National Technical University "KhPI". Series of "Informatics and Modeling". – Kharkov: NTU "KhPI". – 2025. – № 2 (14). – P. 157 – 171.

The paper highlights the relevant aspects of designing medical information systems, taking into account the requirements of scalability, maintainability, and effective interaction with databases. The expediency of using the Dapper microORM library in combination with SOLID principles to ensure a clean application software architecture is considered. Architectural approaches, database structures are analyzed, and the advantages of direct control over SQL queries in the context of medical systems are evaluated. Particular attention is paid to the implementation of design patterns and Clean Code practices as factors affecting code quality and system flexibility. The results of the work demonstrate the feasibility of combining an object-oriented approach with lightweight ORM tools to create efficient, adaptive, and reliable medical information systems. Illustration: 2. Tabl.: 2. Bibliography: 10 titles.

Keywords: information system, medical software, SOLID, Dapper, ORM, design patterns, Clean Code, SQL Server, medical data processing.