DOI 10.31891/2307-5732-2024-345-6-17
УДК 004.021

НІКОНОВ ОЛЕГ
Київський національний університет технологій та дизайну
https://orcid.org/0000-0002-8878-4318
e-mail: nikonov.oy@knutd.edu.ua
СКІДАН ВЛАДІСЛАВА
Київський національний університет технологій та дизайну
https://orcid.org/0000-0002-8358-9759
e-mail: skidan.vv@knutd.edu.ua
ВОЛІВАЧ АНТОНІНА
Київський національний університет технологій та дизайну
https://orcid.org/0000-0002-7119-7774
e-mail: volivach.ap@knutd.edu.ua
МАМОНТОВ ВІТАЛІЙ
Київський національний університет технологій та дизайну
e-mail: vitalii.mamontov.0008@gmail.com

ЗАСТОСУВАННЯ ONION-АРХІТЕКТУРИ В РАМКАХ ПРЕДМЕТНО-ОРІЄНТОВАНОГО ПІДХОДУ

В результаті дослідження Onion-архітектури у предметно-орієнтованому підході проаналізовано ефективність використання предметно-орієнтованого проєктування на основі Onion-архітектури. Виявлено, що при розробці проєкту на основі предметно-орієнтованого підходу ключовим аспектом є створення надійної моделі предметної області. Встановлено, що процес створення цієї моделі для конкретної проблеми є ітеративною співпрацею між бізнес-експертами та технічною командою. Визначено, що під час процесу розробки програмного забезпечення, ставиться акцент на важливість предметної області майбутньої системи, збільшення гнучкості та комунікації між членами команди розробників.

Запропоновано використання предметно-орієнтованого підходу для розробки систем автоматизації бізнес-процесів, а також для розробки інтеграцій для систем автоматизації бізнес-процесів. Досліджено переваги використання предметно-орієнтованого підходу в контексті систем автоматизації бізнес-процесів.

Ключові слова: предметно-орієнтований підхід, автоматизація бізнес-процесів, інтеграція, Onion-архітектура, проєктування.

NIKONOV OLEH, SKIDAN VLADYSLAVA, VOLIVACH ANTONINA, MAMONTOV VITALII
Kyiv National University of Technology and Design

USING OF ONION ARCHITECTURE WITHIN THE SUBJECT-ORIENTED APPROACH

As a result of the Onion-architecture research in a subject-oriented approach, the effectiveness of using subject-oriented design, as well as Onion architecture, was analyzed. It was found that when developing a project based on a subject-oriented approach, the key aspect is the creation of a reliable model of the subject area. It is also established that the process of creating this model for a specific problem is an iterative collaboration between business experts and the technical team. It was determined that during the software development process, emphasis is placed on increasing its flexibility, facilitating communication between members of the development team and emphasizing the importance of the subject area of the future system.

It is proposed to use a subject-oriented approach for the development of business process automation systems, as well as for the development of integrations for business process automation systems. The advantages of using a subject-oriented approach in the context of business process automation systems have been researched.

The conducted studies allow us to assess the advantages and challenges of using a subject-oriented approach during the development of business process automation systems and integrations for them, which is important for development teams. The obtained results are the basis for future research and effective implementation of business process automation systems and integrations for them. For future research, it is suggested to focus on the prospects for the development of a subject-oriented approach, in particular, on taking into account modern trends and forecasting possible changes in the business environment.

Looking ahead, it is recommended that future research delves deeper into the evolving landscape of subject-oriented approaches. Specifically, attention should be directed towards staying abreast of modern trends in technology and business practices. By forecasting potential changes in the business environment, development teams can proactively adapt their strategies, ensuring the continued relevance and effectiveness of subject-oriented approaches in the realm of business process automation. This forward-looking approach will contribute to the ongoing refinement and optimization of systems, aligning them with the dynamic needs of the contemporary business.

Keywords: subject-oriented approach, business process automation, integration, Onion architecture, design

Постановка проблеми

Розробка програмного забезпечення є невід'ємною частиною сучасного технологічного розвитку, її основною метою є оптимізація та автоматизація процесів у визначених предметних областях. Проте цей процес часто ускладнюється контекстуальними обмеженнями, що зумовлені особливостями та специфікою самої предметної області [1-4].

У контексті розробки систем автоматизації бізнес-процесів одним з головних викликів є глибоке розуміння предметної області. Для вирішення цієї проблеми часто застосовують предметно-орієнтовані моделі, які забезпечують детальне дослідження сутності завдань та їхніх особливостей. У випадку великих програмних рішень та складних галузей важко одразу врахувати всі аспекти предметної області, що ускладнює побудову відповідної архітектури системи.

Проблема полягає в тому, що розробники архітектури часто не мають достатнього глибокого розуміння предметної області, що ускладнює створення ефективного продукту. У таких випадках для зменшення складності та підвищення гнучкості програмного продукту застосовується модельно-орієнтований підхід, який дозволяє краще структурувати систему, знижуючи рівень зв'язку компонентів та полегшуючи надалі їх адаптацію й модифікацію. Такий підхід є особливо важливим у високонавантажених системах, де кожен елемент потребує ретельної інтеграції й оптимізації.

Постановка проблеми зводиться до необхідності створення високоякісної моделі програмного забезпечення, яке буде точно відповідати поставленим бізнес-цілям, взаємодії між розробниками та експертами предметної області, чітким визначенням бізнес-процесів та узгодженням технічного рішення. Такий підхід дозволяє мінімізувати ризики непорозумінь, підвищити ефективність розробки та забезпечити відповідність кінцевого продукту потребам користувачів.

Аналіз останніх досліджень і публікацій

Onion-архітектура успішно впроваджується в різних галузевих проєктах, зокрема у веб-додатках, мікросервісній архітектурі та у додатках для мобільних платформ [5-12]. Застосування Onion-архітектури в рамках предметно-орієнтованого підходу набуває все більшої популярності у розробці програмного забезпечення, особливо в складних системах, де необхідна висока модульність і гнучкість. У прикладних дослідженнях ця архітектура розглядається як спосіб вирішення проблем надмірної залежності компонентів, що спрощує масштабування та тестування систем [5-10].

В роботі проведено дослідження щодо застосування Onion-архітектури у предметно-орієнтованому підході та аналіз ефективності використання предметно-орієнтованого проєктування.

Формулювання цілей статті

Метою статті є дослідження застосування Onion архітектури у предметно-орієнтованому підході для моделювання систем автоматизації бізнес-процесів з метою оцінки їх технічних переваг.

Виклад основного матеріалу дослідження

Предметно-орієнтоване проєктування (Domain-Driven Design, далі DDD) це набір принципів і схем, що допомагають розробникам створювати системи об'єктів. Правильне застосування DDD призводить до створення програмних абстракцій – моделей предметних областей. Такі моделі включають складну бізнес-логіку, що усуває проміжок між реальними умовами бізнесу та кодом. Варто зазначити, що DDD не має певної техніки або методів. Це набір правил, які дозволяють приймати правильні (ефективні) дизайнерські рішення, що суттєво прискорює процес розробки програмного забезпечення в різних тематичних областях.

Слід зауважити, що до початку роботи над проєктом усі залучені сторони формують єдині вимоги, що залежать від специфіки бізнесу та переваг команди, можуть бути представлені у вигляді глосарію та структурних діаграм, зрозумілих усім учасникам проєкту, або у вигляді розгорнутої документації. Структурну схему взаємозв'язку між учасниками проєкту наведено на рис (рис. 1) [5-10]. З метою уникнення непорозуміння всі учасники проєкту (розробники, аналітики, експерти предметної галузі, менеджери), використовують «єдину мову», встановлену на початку проєкту.

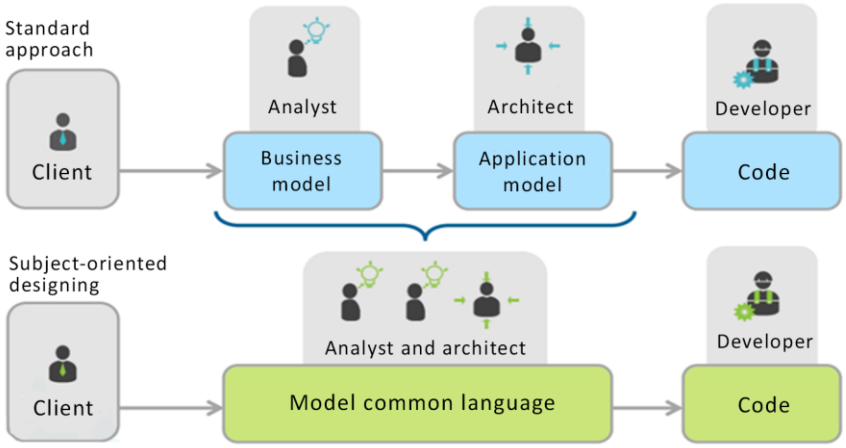

Рис. 1. Структурна схема взаємозв'язку між учасниками проєкту

В ході проведеного аналізу ефективності використання «єдиної мови» встановлено, що однією з її важливих переваг є розроблення зрозумілого та читабельного програмного коду. Це сприяє більш ефективному співробітництву між членами команди, полегшує підтримку програмного продукту та прискорює процес виявлення й виправлення помилок. На початку проєкту розробник надає структурним одиницям коду (класам, методам, змінним тощо) імена, що відповідають прийнятій термінології. Такий підхід робить код зрозумілим усім учасникам проєкту предметної області. У разі потреби, наприклад, аналітик або експерт може переглянути код і перевірити правильність обчислення певних значень, відповідність логіки

обробки даних або виконання інших операцій. Такий підхід сприяє прозорості процесу розробки проєкту та спрощує виявлення й виправлення помилок.

Проведене дослідження показало, що методологія DDD забезпечує доступність повної та цілісної інформації про предметну область і бізнес-процеси для всіх учасників проєкту. На початковому етапі взаємодії між усіма сторонами, залученими до проєкту, створюється модель предметної області. В процесі моделювання предметна область розділяється на окремі підобласті, серед яких центральну роль відіграє смислове ядро. Смислове ядро визначає ключові аспекти, що мають найвищий пріоритет для бізнесу та формує основний конкурентний потенціал проєкту.

За результатами проведеного аналізу виділено наступні переваги використання предметно-орієнтованого підходу:

1. Полегшення комунікації між розробниками та клієнтами за рахунок створення «єдиної мови», яка базується на моделі домену, комунікації з клієнтом (експертом у предметній області), що спрощує роботу протягом усього життєвого циклу розробки. Це дозволяє уникнути непорозумінь і покращити узгодженість між технічними та бізнес-вимогами.

2. Підвищення гнучкості програмного забезпечення, яке базується на принципах об'єктно-орієнтованого аналізу і дизайну, забезпечуючи модульність та інкапсуляцію компонентів. Завдяки цьому програмне забезпечення є гнучким і легко адаптується до змін у бізнес-вимогах.

3. Акцент на предметній області, а не на інтерфейсі вказує на те, що DDD фокусується на точному відображенні предметної області через модель домену, що створюється за допомогою експертів. Це забезпечує точну репрезентацію бізнес-процесів і завдань, а не надмірну увагу до UI/UX, яка може відволікати від суті проблеми.

До недоліків використання предметно-орієнтованого підходу віднесено:

1. Висока залежність від експертної оцінки домену. Для успішного впровадження та досягнення бажаних результатів DDD потребує глибоку експертизу предметної області.

2. Обмежена придатність для простих доменів. DDD підходить переважно для проєктів зі складною предметною областю. Для простих доменів застосування DDD є недоцільним.

3. Збільшення тривалості та вартості розробки. Використання DDD вимагає більше часу на аналіз і моделювання, що може призводити до довготривалості проєкту та більших витрат. Такий підхід є малоефективний для короткострокових проєктів або тих, що мають низьку складність домену.

В ході аналізу [5-10] було встановлено, що традиційна архітектура програмного забезпечення складається з трьох рівнів (рис. 2). Ця структура забезпечує базову організацію проєкту, однак у складних системах може бути менш ефективною у порівнянні з сучасними підходами, такими як DDD.

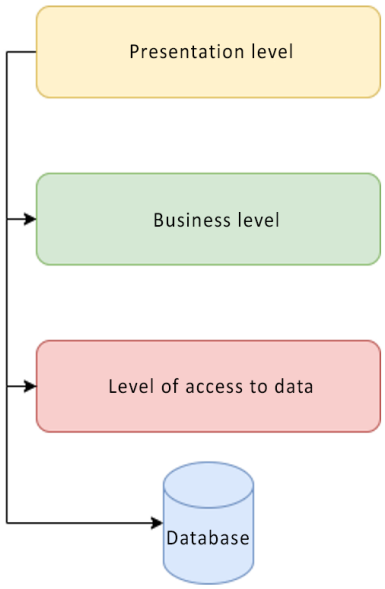

Рис. 2. Трирівнева архітектура програми

Як видно з рисунку, кожен рівень архітектури залежить від нижчих рівнів та загальних інфраструктур таких, як фреймворки. Така залежність забезпечує взаємозв'язок між компонентами системи та їхню комунікацію, що є важливим для узгодженої роботи всієї системи.

Основною проблемою є налагодження взаємодії між інтерфейсом користувача, бізнес-логікою та доступом до даних. Оскільки без бізнес-логіки інтерфейс користувача працювати не буде, а без доступу до даних бізнес-логіка є не функціональною. Зміни в доступі до даних вимагають налаштування бізнес-рівня архітектури, що призводить до великого ризику для стабільних систем. Для уникнення цих проблем авторами запропоновано застосовувати предметно-орієнтоване проєктування з використанням Onion архітектури. Такий підхід дозволяє зменшити залежності між рівнями, ізолювати бізнес-логіку від змін у даних чи інтерфейсі користувача та забезпечити високу гнучкість і масштабованість системи. Ця архітектура

організовує компоненти системи у вигляді шарів, де центральне місце займає доменна модель, що відповідає за бізнес-логіку. При цьому, зовнішні шари відповідають за інфраструктуру, доступ до даних та інтерфейс користувача, взаємодіють з бізнес-логікою через чітко визначені контракти. Така структура забезпечує високу гнучкість, спрощує масштабування та підтримку системи, а також мінімізує ризики, пов'язані зі змінами в окремих компонентах.

Головна мета Onion архітектури полягає у впорядкуванні та контролюванні зв'язків між структурними елементами системи таким чином, щоб вони завжди спрямовувалися до центру. Основне правило цієї архітектури полягає в тому, що усі залежності між компонентами мають бути спрямовані до центру, де знаходиться доменна модель, яка є ядром системи.

Onion архітектура поділяється на наступні рівні, кожен з яких має чітко визначену роль, а саме:

- інтерфейс користувача (UI) забезпечує взаємодію користувача із системою;
- тести (Tests) – частина системи, що відповідає за перевірку роботи її компонентів, включаючи юніт-тести, інтеграційні тести тощо;
- інфраструктура (Infrastructure) відповідає за взаємодію з базами даних, зовнішніми системами, API тощо;
- служби додатків (Application Services) реалізують логіку взаємодії між інфраструктурою та бізнес-логікою;
- служби домену (Domain Services) містять бізнес-правила, які не можуть бути віднесені до конкретних об'єктів доменної моделі;
- модель домену (Domain Model) – ядро системи, яке містить бізнес-логіку та правила предметної області.

Такий підхід забезпечує високу модульність, дозволяє ізолювати бізнес-логіку від змін в інфраструктурі та інтерфейсі користувача, а також спрощує тестування та підтримку системи. Приклад структури Onion архітектури наведено на рис. 3 [5-8].

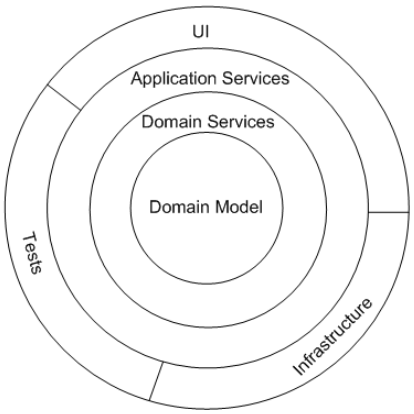

Рис. 3. Onion архітектура

До переваг використання Onion архітектури в розробці проєкту віднесено:

- Onion архітектура чітко розділяє додаток на шари з різними функціями, що забезпечує збереження принципу єдиної відповідальності (SRP). Це полегшує розробку, модифікацію та тестування кожного шару окремо.
- Кожен шар є окремим модулем, що сприяє легкості у підтримці та розширенні проєкту. При цьому, можливість змінювати або вдосконалювати один шар без впливу на інші покращує гнучкість системи.
- Ізольованість шарів дозволяє розробникам тестувати кожен шар окремо, що значно підвищує якість і надійність тестування. Наприклад, доменний шар можна тестувати незалежно від інфраструктури.
- Шарова структура дозволяє різним командам працювати над своїми частинами проєкту, незалежно одна від одної. Це сприяє ефективній співпраці, особливо у великих командах, де завдання можуть бути розподілені за відповідними рівнями.

Прикладом використання Onion архітектури у предметно-орієнтованому підході може служити розроблений прототип інтеграції CRM системи Creatio з Google Classroom (рис. 4).

Ядром інтеграції будуть слугувати моделі предметної області, а саме курс та контакт (студент та викладач). Моделі відображатимуть основні сутності, що взаємодіють в межах системи Google Classroom, забезпечуючи зв'язок між різними компонентами, такими як курси, завдання та користувачі. Моделі курсів будуть містити інформацію про назву, опис, учасників та інші важливі атрибути, а моделі контактів – дані про студентів та викладачів, їх ролі в курсах і взаємодію з завданнями.

На рівні сервісів будуть використовуватися сервіси автентифікації для безпечного доступу до Google Classroom API, а також сервіси для відправки запитів до Google Classroom, які здійснюватимуть комунікацію з Google API для отримання та оновлення даних про курси, студентів, завдання та оцінки. Ці сервіси

надаватимуть функціональність для створення, редагування та отримання інформації, що необхідна для роботи додатка.

Рівень інфраструктури займатимуть сервіси Creatio, які будуть забезпечувати інтеграцію з CRM-системою для управління користувачами, курсами та завданнями, а також виконуватимуть синхронізацію з зовнішніми системами через API. Крім того, інфраструктура включатиме модульні тести, що дозволяють перевірити коректність роботи окремих компонентів системи, таких як сервіси автентифікації, запити до Google Classroom та інтерфейси взаємодії з Creatio. Тести будуть забезпечувати високий рівень покриття і гарантувати, що всі частини системи працюють правильно як у ізольованому вигляді, так і в інтегрованому середовищі.

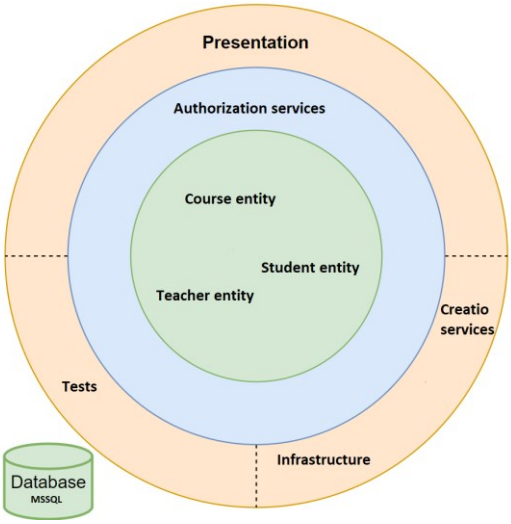

Рис. 4. Onion архітектура Google classroom інтеграції

Back-end буде реалізований за допомогою мови програмування C#, а фреймворки ASP.NET. Ext.js, та Angular будуть використані в якості засобу для реалізації front-end.

На рис. 5 наведено діаграму класів системи Google Classroom інтеграції.

.

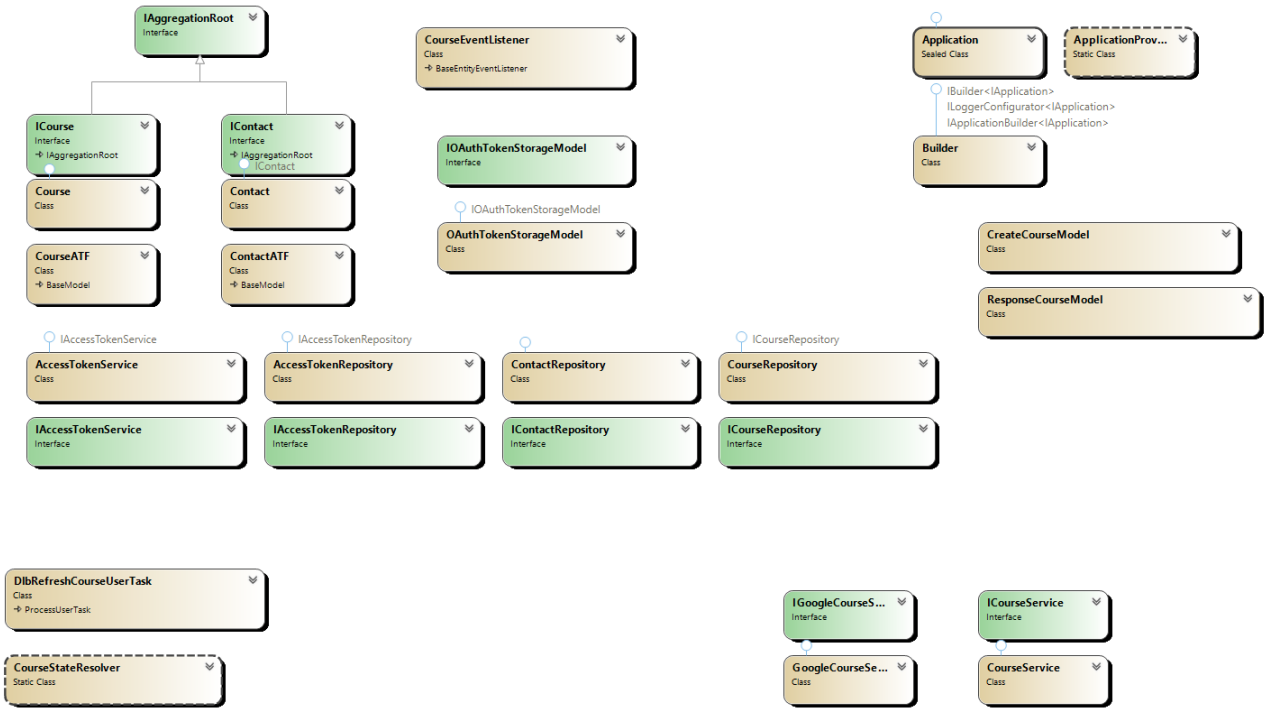

Рис. 5. Діаграма класів Google classroom інтеграції

Діаграма класів для інтеграції з Google Classroom описує основні компоненти системи, що забезпечують доступ до даних про курси, користувачів та завдання через API, а також сервіси автентифікації

та інтеграції з зовнішніми системами, такими як Creatio. Вищезазначені компоненти працюють разом, забезпечуючи гнучку і масштабовану архітектуру для управління курсами та оцінками.

Загалом, використання Onion-архітектури підвищує ефективність процесу розробки, тестування, підтримки і масштабування програмного забезпечення, зокрема в багатошарових застосунках. Однак важливо враховувати, що вона може вимагати більше часу та зусиль на початковій стадії розробки, а її переваги виявляються особливо корисними у великих проєктах і на довгостроковій перспективі.

Проведені дослідження показали переваги використання предметно-орієнтованого підходу в контексті систем автоматизації бізнес-процесів, а саме:

1. Адаптація до бізнес-потреб дозволяє створювати систему, яка максимально точно відповідає потребам конкретного бізнесу. Процеси та правила, які розробляються в системі, відображають реальні бізнес-процеси, що сприяє максимальній ефективності автоматизації.

2. Продуктивність розробки дозволяє розробникам створювати програмне забезпечення швидше і ефективніше. Використовуючи моделі, які відображають предметну область, розробники можуть швидко створювати та підтримувати систему.

3. Зниження ризиків і помилок дозволяє створювати системи, які більш точно відповідають бізнес-вимогам. Такий підхід зменшує ризики неправильної реалізації та виникнення помилок в системі.

4. Зменшення залежності від технічних деталей дозволяє розробникам зосередитися на моделюванні предметної області, не вдаючись у деталі технічної реалізації. Це сприяє збереженню вищого рівня абстракції та полегшує підтримку системи в разі змін технологічного стеку.

5. Підвищена взаємодія з бізнес-користувачами дозволяє представити бізнес-процеси та правила в інтуїтивно зрозумілій формі, що сприяє більш ефективному спілкуванню між розробниками та бізнес-користувачами.

6. Покращення спроможностей аналітики та звітності дозволяє легко створювати звіти та аналітичні інструменти, оскільки дані в системі структуровані та легко доступні.

Onion-архітектура допомагає уникнути витоку даних між шарами, оскільки кожен шар має чітко визначені межі доступу. Це особливо важливо для бізнес-процесів, які обробляють конфіденційні або особисті дані. Розділення шарів допомагає зберегти безпеку даних і запобігти несанкціонованому доступу.

Висновки

Отже, застосування Onion-архітектури для розробки систем автоматизації бізнес-процесів та їх інтеграції є ефективним рішенням. Такий підхід забезпечує чітке розділення відповідальності між компонентами системи, спрощує масштабованість і модифікацію, підвищує якість програмного забезпечення, сприяє інтеграції з іншими системами та гарантує безпеку даних. Це дозволяє ефективно впроваджувати автоматизацію бізнес-процесів в проєктах та залишатися конкурентоспроможними в різних сферах діяльності.

В результаті дослідження:

1. Виявлено, що модель предметної області є важливою відправною точкою під час розробки проєкту на основі предметно-орієнтованого підходу, при цьому розробка надійної моделі домену для певної проблеми є ітеративним підходом співпраці між бізнес-експертами та технічною групою.

2. Onion-архітектура демонструє високу ефективність у роботі з предметно-орієнтованими моделями завдяки спрямованим нагору залежностям сутностей, що знижує ризики та сприяє стабільності системи.

3. Досліджено, що під час розробки за рахунок «єдиної мови» підвищується гнучкість програмного забезпечення, полегшується комунікація між командою розробників, робиться акцент на предметну область майбутньої системи, поліпшується процес розроблення документації.

4. Використання Onion архітектури в предметно-орієнтованому підході дозволяє покращити читабельність коду, зменшити залежності між модулями й полегшити розширення програми, що впливає на підтримку та масштабування проєкту.

Перспективи подальших досліджень: вдосконалення методів і засобів створення та перевірки моделей предметної області; розробка адаптивних підходів до інтеграції Onion-архітектури з іншими архітектурними стилями; розробка оптимальних підходів до забезпечення масштабованості та продуктивності програмних рішень, розроблених на основі Onion-архітектури.

Результати роботи підтверджують практичну цінність застосування Onion-архітектури у вирішенні складних задач предметно-орієнтованого проєктування, забезпечуючи високу якість і ефективність систем автоматизації.

Література

1. Кондра А., Кунанець Н. Інтелектуальна інформаційна система електронної комерції бренду одягу // Вісник Хмельницького національного університету. Технічні науки. – 2023. – № 4 (323). – С. 159–167.

2. Bodyanskiy Y., Chala O., Filatov V., Pliss I. Neo-Fuzzy Radial-Basis Function Neural Network and Its Combined Learning. In: Zgurovsky, M., Pankratova, N. (eds) System Analysis and Artificial Intelligence . Studies in Computational Intelligence. – 2023. – V.1107. Springer, Cham. – P. 323–340.

3. Skvorchevsky A. Increasing the robustness of computer networks by using hybrid centralized-distributed topology, 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine, 2022. – P. 239–242.

4. Nikonov O., Skidan V., Volivach A., Nadopta T., Pavlenko V. Cloud System of Content Accounting with Access on OC Android and IOS. 2023 IEEE 4th KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2023. – P. 1002–1005.

5. Stenberg J. Domain-Driven Design with Onion Architecture [Electronic resource]. URL: https://www.infoq.com/news/2014/10/ddd-onion-architecture/.

6. Domain-Driven Design на практиці: матеріали вебінара. URL: https://careers.epam.ua/events/java-webinar-03022022.

7. Предметно-орієнтоване проєктування. URL: https://www.wiki.uk-ua.nina.az/Domain_driven_design.html

8. Laribee D. Best Practice – An Introduction To Domain-Driven Design. URL: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/best-practice-an-introduction-to-domain-driven-design.

9. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Boston: Addison-Wesley Professional, 2003. – 79 p.

10. Vaughn V. Implementing Domain-driven Design. – Boston: Addison-Wesley Professional, 2013. – 50 p.

11. Khononov V. Learning Domain-Driven Design. – Boston: Addison-Wesley Professional, 2021. – 144 p.

12. Nilsson J. Applying Domain-Driven Design And Patterns: With Examples in C# and .NET. – Boston: Addison-Wesley Professional, 2006. – 201 p.

References

1. Kondra A., Kunanets N. Intelektualna informatsiina systema elektronnoi komertsii brendu odiahu // Herald of Khmelnytskyi National University. Technical sciences. – 2023. – № 4 (323). – S. 159–167.

2. Bodyanskiy Y., Chala O., Filatov V., Pliss I. Neo-Fuzzy Radial-Basis Function Neural Network and Its Combined Learning. In: Zgurovsky, M., Pankratova, N. (eds) System Analysis and Artificial Intelligence . Studies in Computational Intelligence. – 2023. – V.1107. Springer, Cham. – P. 323–340.

3. Skvorchevsky A. Increasing the robustness of computer networks by using hybrid centralized-distributed topology, 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine, 2022. – P. 239–242.

4. Nikonov O., Skidan V., Volivach A., Nadopta T., Pavlenko V. Cloud System of Content Accounting with Access on OC Android and IOS. 2023 IEEE 4th KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2023. – P. 1002–1005.

5. Stenberg J. Domain-Driven Design with Onion Architecture [Electronic resource]. URL: https://www.infoq.com/news/2014/10/ddd-onion-architecture/.

6. Domain-Driven Design na praktytsi: materialy vebinara. URL: https://careers.epam.ua/events/java-webinar-03022022.

7. Predmetno-oriientovane proektuvannia. URL: https://www.wiki.uk-ua.nina.az/Domain_driven_design.html

8. Laribee D. Best Practice – An Introduction To Domain-Driven Design. URL: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/best-practice-an-introduction-to-domain-driven-design.

9. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Boston: Addison-Wesley Professional, 2003. – 79 p.

10. Vaughn V. Implementing Domain-driven Design. – Boston: Addison-Wesley Professional, 2013. – 50 p.

11. Khononov V. Learning Domain-Driven Design. – Boston: Addison-Wesley Professional, 2021. – 144 p.

12. Nilsson J. Applying Domain-Driven Design And Patterns: With Examples in C# and .NET. – Boston: Addison-Wesley Professional, 2006. – 201 p.