Рекомендації щодо проектування багаторівневих застосувань

Рекомендації щодо проектування багаторівневих застосувань

Зміст

Огляд

В цій лекції відображається загальна структура застосувань з точки зору логічного групування компонентів в окремі рівні, взаємодіючі один з одним і іншими клієнтами та застосуваннями.

Розбиття на рівні виконується відповідно логічному поділу компонентів і функціональності і не враховує фізичного розміщення компонентів.

Рівні можуть розміщуватися як на різних вузлах, так і на одному.

Буде розглянуто, як розділяти застосування на логічні частини, як вибирати відповідну функціональну компоновку застосування і як забезпечити підтримку застосуванням безлічі типів клієнтів.

Огляд

Важливо розуміти різницю між рівнями та вузлами.

Рівні (Layers) описують логічне угруповання функцій і компонентів в застосуванні, тоді як вузли (tiers) описують фізичний розподіл функцій і компонентів по сервісам, комп'ютерам, мережам або віддаленим місцям розміщення. Незважаючи на те, що і для рівнів, і для вузлів застосовується одна і та ж термінологія (представлення, бізнес, сервіси та дані), слід пам'ятати, що тільки вузли мають на увазі фізичний поділ.

Розміщення декількох рівнів на одному комп'ютері (одному вузлі) - досить звичайне явище. Термін вузол використовується в застосуванні до схем фізичного розподілу, наприклад, двовузловий, тривузловий, n-вузловий.

Логічний розділ на рівні

Незалежно від типу застосування, що проектується і того, чи є у нього призначений для користувача інтерфейс або воно є сервісним застосуванням, яке просто надає послуги (не плутайте з рівнем сервісів застосування), його структуру можна розкласти на логічні групи програмних компонентів.

Ці логічні групи називаються рівнями.

Рівні допомагають розділити різні типи завдань, які здійснюються цими компонентами, що спрощує створення дизайну, що підтримує можливість повторного використання компонентів. Кожен логічний рівень включає ряд окремих типів компонентів, згрупованих в підрівні, кожен з підрівнів виконує певний тип завдань.

Логічний розділ на рівні

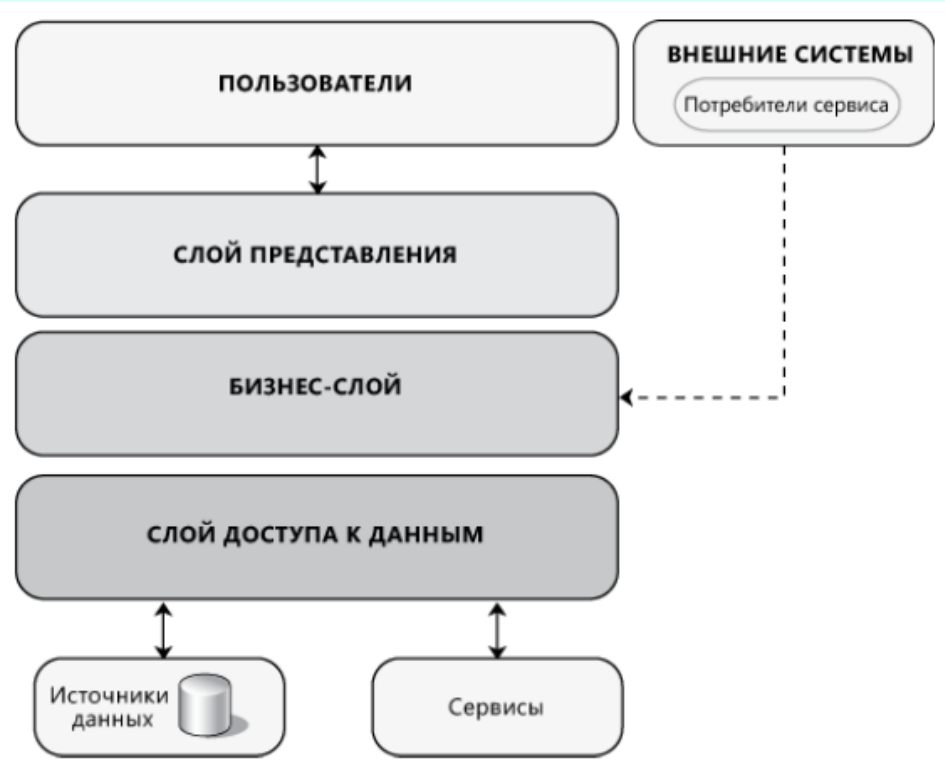
Визначаючи універсальні типи компонентів, які присутні в більшості рішень, можна створити схему програми або сервісу і потім використовувати цю схему як ескіз створюваного дизайну.

Поділ програми на рівні, що виконують різні ролі і функції, допомагає максимально підвищити зручність і простоту обслуговування коду, оптимізувати роботу програми при різних схемах розгортання і забезпечує чітке розмежування галузей застосування певної технології або прийняття певних проектних рішень.

Рівень представлення, бізнес-рівень та рівень даних

На найвищому і найбільш абстрактному рівні логічне представлення архітектури системи може розглядатися як набір взаємодіючих компонентів, згрупованих у рівні.

Рівень представлення, бізнес-рівень та рівень даних



Логічне уявлення архітектури багаторівневої системи

На рис. 1 показано спрощене високорівневе уявлення рівнів і їх взаємовідносини з користувачами, іншими застосуваннями, що викликають сервіси, що реалізовані в бізнес-рівні застосування, джерелами даних, такими як реляційні бази даних або Веб-сервіси, що забезпечують доступ до даних, і зовнішніх або віддалених сервісів, які використовуються застосуванням.

Рівень представлення, бізнес-рівень та рівень даних

Застосування може складатися з ряду базових рівнів.

Типовий трирівневий дизайн включає наступні рівні:

Рівень представлення.
Даний рівень містить орієнтовану на користувача функціональність, яка відповідає за реалізацію взаємодії користувача з системою, і, як правило, включає компоненти, що забезпечують загальний зв'язок з основною бізнес-логікою, інкапсульованою в бізнес-рівні.

Рівень представлення, бізнес-рівень та рівень даних

Бізнес-рівень.
Цей рівень реалізує основну функціональність системи і інкапсулює пов'язану з нею бізнес-логіку. Зазвичай він складається з компонентів, деякі з яких надають інтерфейси сервісів, доступні для використання іншими учасниками взаємодії.

Рівень доступу до даних.
Цей рівень забезпечує доступ до даних, що зберігаються в рамках системи, і даних, що надаються іншими мережевими системами. Доступ може здійснюватися через сервіси. Рівень даних надає універсальні інтерфейси, які можуть використовуватися компонентами бізнес-рівня.

Сервіси та рівні

У першому наближенні рішення, засноване на сервісах, можна розглядати як набір сервісів, що взаємодіють один з одним шляхом передачі повідомлень.

Концептуально ці сервіси можна вважати компонентами рішення в цілому.

Однак кожен сервіс, утворений програмними компонентами, як будь-яке інше застосування, і ці компоненти можуть бути логічно згруповані в рівень представлення, бізнес-представлення і представлення даних.

Інші застосування можуть використовувати сервіси, не замислюючись про спосіб їх реалізації.

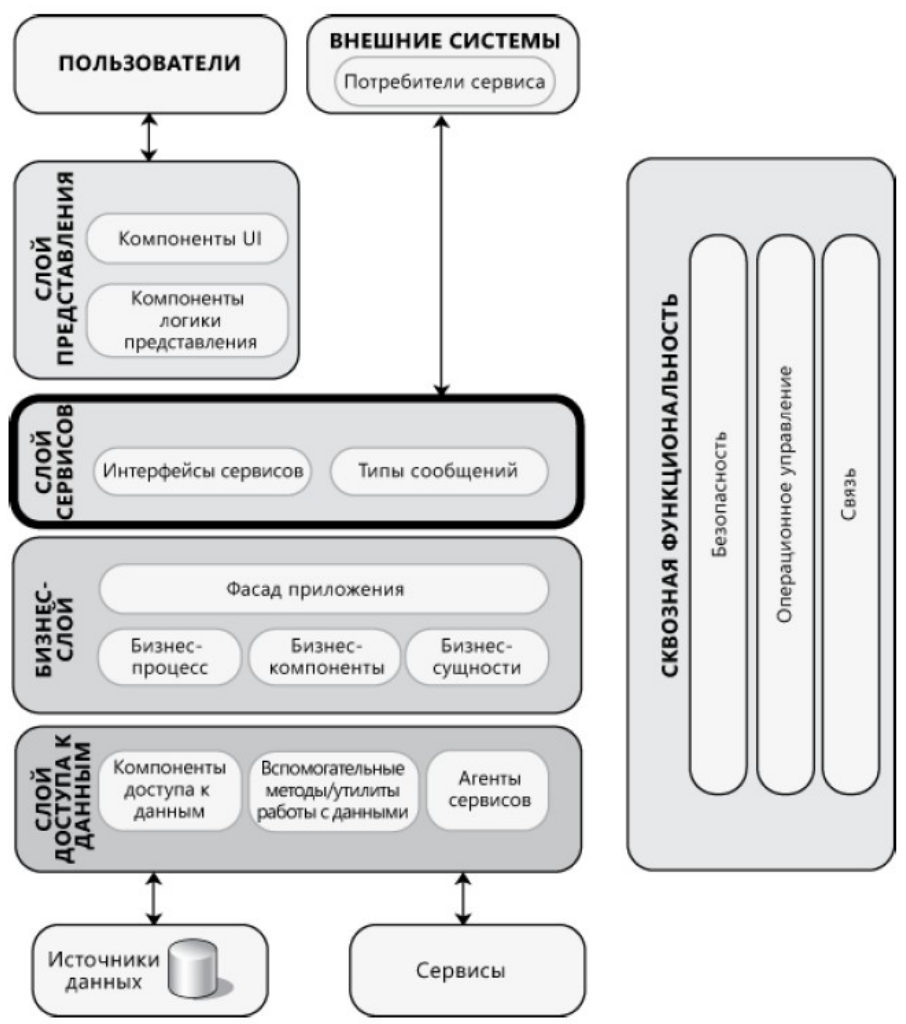
Принципи багаторівневого дизайну в рівній мірі застосовуються і до рішень, що засновані на сервісах.

Рівень сервісів

Звичайним підходом при створенні програми, яка повинна забезпечувати сервіси для інших додатків, а також реалізовувати безпосередню підтримку клієнтів, є використання рівня сервісів, який надає доступ до бізнес-функціональності додатку.

Рівень сервісів забезпечує альтернативне уявлення, що дозволяє клієнтам використовувати інший механізм для доступу до застосування.

Рівень сервісів



Включення рівня сервісів до застосування

Рівень сервісів

У даному сценарії користувачі можуть виконувати доступ до застосування через рівень представлення, що обмінюється даними з компонентами бізнес-рівня або безпосередньо, або через фасад застосування в бізнес-рівні, якщо методи зв'язку вимагають композиції функціональності.

Крім цього, зовнішні клієнти та інші системи можуть виконувати доступ до застосування і використовувати його функціональність шляхом взаємодії з бізнес-рівнем через інтерфейси сервісів.

Це покращує можливості застосування для підтримки безлічі типів клієнтів, сприяє повторному використанню і більш високого рівня композиції функціональності в додатках.

У деяких випадках рівень уявлення може взаємодіяти з бізнес-рівнем через рівень сервісів. Але це не є обов'язковою умовою. Якщо фізично рівень представлення і бізнес-рівень розташовуються на одному рівні, вони можуть взаємодіяти безпосередньо.

Етапи проектування багаторівневої структури

Приступаючи до проектування програми, перш за все, зосередьтеся на найвищому рівні абстракції і починайте з угруповання функціональності в рівні.

Далі слід визначити відкритий інтерфейс для кожного рівня, який залежить від типу створюваного додатка. Визначивши рівні і інтерфейси, необхідно прийняти рішення про те, як буде розгортатися додаток. Нарешті, вибираються протоколи зв'язку для забезпечення взаємодії між рівнями і вузлами додатка. Незважаючи на те, що структура і інтерфейси, які розробляються можуть змінюватися з часом, особливо в разі застосування гнучкої розробки, проходження цих етапів гарантовано забезпечить розгляд всіх важливих аспектів на початку процесу.

Етапи проектування багаторівневої структури

Зазвичай при проектуванні використовується наступна послідовність кроків:

Крок 1 – Вибір стратегії поділу на рівні
Крок 2 – Вибір необхідних рівнів
Крок 3 – Ухвалення рішення про розподіл рівнів і компонентів
Крок 4 – З'ясування можливості згортання рівнів
Крок 5 – Визначення правил взаємодії між рівнями
Крок 6 – Визначення наскрізної функціональності
Крок 7 – Визначення інтерфейсів між рівнями
Крок 8 – Вибір стратегії розгортання
Крок 9 – Вибір протоколів зв'язку

Крок 1 – Вибір стратегії розділення на рівні

Поділ на шари представляє логічне розподіл компонентів додатка по групах, які виконують певні ролі і функції.

Використання багатошарового підходу може підвищити зручність обслуговування програми та спростити його масштабування, якщо необхідно підвищити продуктивність.

Неправильне поділ на рівні (занадто мало або занадто багато) може лише ускладнити застосування, приводячи до зниження загальної продуктивності, зручності обслуговування і гнучкості.

Визначення відповідного рівня деталізації при поділі програми на рівні - критично важливий перший крок у визначенні стратегії поділу на рівні.

Крок 1 – Вибір стратегії розділення на рівні

Слід врахувати, чи застосовуються шари виключно для логічного поділу або для забезпечення фізичного поділу в разі потреби.

Перетин кордонів рівнів, особливо кордонів між фізично віддаленими компонентами, обумовлює виникнення витрат і зниження продуктивності.

Однак загальне підвищення масштабованості і гнучкості програми може бути набагато сильнішим аргументом в порівнянні з падінням продуктивності.

Крок 1 – Вибір стратегії розділення на рівні

Крім того, поділ на шари може спростити оптимізацію продуктивності окремих шарів без впливу на суміжні шари.

У разі логічного поділу взаємодіючі рівні застосування будуть розгортатися на одному вузлі і виконуватися в одному процесі, що дозволить застосовувати більш продуктивні механізми зв'язку, такі як прямі виклики через інтерфейси компонентів. Однак щоб скористатися перевагами логічного поділу на рівні і гарантувати гнучкість в майбутньому, слід серйозно і ретельно підійти до питань забезпечення інкапсуляції і слабкого зв'язування між рівнями.

Крок 1 – Вибір стратегії розділення на рівні

Для шарів, розгорнутих на різних вузлах (різні фізичні комп'ютери), взаємодія із суміжними рівнями буде відбуватися по мережі, і необхідно забезпечити, щоб вибраний дизайн підтримував відповідний механізм зв'язку, який буде враховувати затримку зв'язку та забезпечувати слабке зв'язування між шарами.

Визначення того, які рівні застосування ймовірніше всього будуть розгортатися на різних вузлах, а які на одному, також є важливою частиною стратегії поділу на рівні.

Для забезпечення гнучкості необхідно гарантовано забезпечити слабку зв'язаність рівнів. Це дозволяє використовувати переваги більшої продуктивності при розміщенні рівнів на одному вузлі і в разі необхідності розгортати їх на безлічі вузлів.

Крок 1 – Вибір стратегії розділення на рівні

Застосування багаторівневого підходу може дещо ускладнити дизайн і збільшити тривалість підготовчого етапу розробки, але в разі правильної реалізації істотно поліпшить обслуговуваність, розширюваність і гнучкість програми.
Зіставте переваги, що забезпечуються можливістю повторного використання і слабким зв'язуванням при поділі на рівні, з негативними наслідками їх застосування, такими як зниження продуктивності і підвищення складності.

Ретельно продумайте, як розділити застосування на рівні, і як рівні будуть взаємодіяти один з одним; тим самим ви забезпечите добрий баланс продуктивності і гнучкості. Як правило, виграш в гнучкості і зручності обслуговування, що забезпечується багаторівневою схемою, набагато перевищує сумнівне підвищення продуктивності, якого можна досягти в тісно пов'язаному дизайні, що не використовує рівні.

Крок 2 – Вибір необхідних рівнів

Найпоширеніший в бізнес-додатках підхід - розподіл функціональності представлення, сервісів, доступу до даних і бізнес-функціональності по різним рівням. У деяких додатках також використовуються рівні складання звітів, управління та інфраструктури.

Уважно підходите до питання введення додаткових рівнів. Рівень повинен забезпечувати логічне угруповання взаємозалежних компонентів, яке помітно збільшує зручність обслуговування, масштабованість і гнучкість програми. Наприклад, якщо застосування не надає сервіси, можливо, окремий шар сервісів не знадобиться, тоді додаток буде включати тільки шар представлення, бізнес-рівень і рівень доступу до даних.

Крок 3 – Прийняття рішення про розподіл рівнів і компонентів

Рівні і компоненти повинні розподілятися за різними фізичними вузлами, тільки якщо в цьому є необхідність. До типових причин реалізації розподіленого розгортання відносяться політики безпеки, фізичні обмеження, спільно використовувана бізнес-логіка і масштабованість.
- Якщо компоненти представлення Веб-застосування здійснюють синхронний доступ до компонентів бізнес-рівня і обмеження безпеки не вимагають наявності кордону довіри між рівнями, розгляньте можливість розгортання компонентів бізнес-рівня і рівня представлення на одному вузлі, це забезпечить максимальну продуктивність і керованість.
- У насичених клієнтських застосуваннях, в яких обробка UI виконується на клієнтському комп'ютері, варіант розгортання компонентів бізнес-рівня на окремому бізнес-вузлі може бути обраний з міркування безпеки і для підвищення керованості.

Крок 3 – Прийняття рішення про розподіл рівнів і компонентів

- Розгортайте бізнес-сутності на одному вузлі з кодом, що їх використовують. Це може означати їх розгортання в декількох місцях, наприклад, розміщення копій на окремому вузлі представлення або даних, логіка якого використовує або посилається на ці бізнес-сутності. Розгортайте компоненти агентів сервісу на тому ж вузлі, що і код, що викликає ці компоненти, якщо обмеження безпеки не вимагають наявності кордону довіри між ними.
- Розгляньте можливість розгортання асинхронних компонентів бізнес-шару, компонентів робочого процесу і сервісів з однаковими характеристиками завантаження і введення на окремому вузлі. Це дозволить налаштовувати інфраструктуру для забезпечення максимальної продуктивності і масштабованості.

Крок 4 – Виявлення можливості згортання рівнів

У деяких випадках має сенс звернути рівні. Наприклад, в застосуванні, що має дуже обмежений набір бізнес-правил або використовує правила переважно для валідації, бізнес-логіка і логіка представлення можуть бути реалізовані на одному рівні. У застосуванні, яке просто отримує дані з Веб-сервісу і відображає їх, може мати сенс просто додати посилання на Веб-сервіс безпосередньо до рівня представлення і використовувати дані Веб-сервісу безпосередньо. В цьому випадку логічно об'єднуються рівні доступу до даних та представлення.

Угруповання функціональності в рівні є загальним правилом. У деяких випадках рівень може виступати в ролі проксі або транзитного рівня, який забезпечує інкапсуляцію або слабке зв'язування практично без надання функціональності. Але відділення цієї функціональності дозволить розширювати її в майбутньому без надання впливу або з невеликим впливом на інші рівні.

Крок 5 – Визначення правил взаємодії між рівнями

Коли справа доходить до стратегії поділу на рівні, необхідно визначити правила взаємодії рівнів один з одним. Основна мета завдання правил взаємодії - мінімізація залежностей і виключення циклічних посилань. Загальним правилом, якого слід дотримуватися в даному випадку, є дозвіл тільки односпрямованої взаємодії між рівнями:

- Взаємодія зверху вниз. Рівні можуть взаємодіяти з рівнями, розташованими нижче, але нижні рівні ніколи не можуть взаємодіяти з розташованими вище рівнями. Це правило допоможе уникнути циклічних залежностей між рівнями. Використання подій дозволить сповіщати компоненти розташованих вище рівнів про зміни в нижніх рівнях без введення залежностей.

Крок 5 – Визначення правил взаємодії між рівнями

- Сувора взаємодія. Кожен рівень повинен взаємодіяти тільки із рівнем, розташованим безпосередньо під ним. Це правило забезпечить суворий поділ, при якому кожен рівень знає тільки про рівень відразу під ним. Позитивний ефект від цього правила в тому, що зміни в інтерфейсі рівня будуть впливати тільки на рівень, розташований безпосередньо над ним. Застосовуйте цей підхід при проектуванні програми, яка передбачає розширюватися новою функціональністю в майбутньому, якщо хочете максимально скоротити вплив цих змін; або при проектуванні програми, для якого необхідно забезпечити можливість розподілу на різні вузли.
- Вільна взаємодія. Більш високі рівні можуть взаємодіяти з розташованими нижче рівнями безпосередньо, в обхід інших рівнів. Це може підвищити продуктивність, але також збільшить залежність. Інакше кажучи, зміни в нижньому рівні можуть впливати на кілька розташованих вище рівнів. Цей підхід рекомендується застосовувати при проектуванні програми, яка гарантовано буде розміщуватися на одному вузлі (наприклад, самодостатня насичена клієнтська програма), або при проектуванні невеликого застосування, для якого внесення змін, які зачіпають безліч рівнів, не зажадає великих зусиль.

Крок 6 – Визначення наскрізної функціональності

Визначившись з рівнями, необхідно звернути увагу на функціональність, що охоплює всі рівні. Таку функціональність часто називають наскрізною функціональністю. До неї відноситься протоколювання, валідація, аутентифікація і управління винятками. Важливо виявити всі наскрізні функції програми і по можливості спроектувати для кожної з них окремі компоненти.
Такий підхід допоможе забезпечити кращу можливість повторного використання і обслуговування.
Уникайте змішування такого загального коду з кодом компонентів рівнів, щоб рівні і їх компоненти викликали компоненти наскрізної функціональності тільки для виконання таких дій, як протоколювання, кешування або аутентифікація. Оскільки ця функціональність повинна бути доступна для всіх рівнів, спосіб розгортання компонентів наскрізної функціональності повинен забезпечувати це, навіть якщо рівні фізично розміщуються на різних вузлах.
Існують різні підходи до реалізації наскрізної функціональності, від загальних бібліотек, таких як Enterprise Library групи patterns & practices, до методів Aspect Oriented Programming (AOP), в яких код наскрізний функціональності вставляється прямо в відкомпільований файл за допомогою метаданих.

Крок 7 – Визначення інтерфейсів між рівнями

Основна мета при визначенні інтерфейсу рівня - забезпечити слабке зв'язування між рівнями. Це означає, що рівень не повинен розкривати внутрішні деталі, від яких може залежати інший рівень. Замість цього інтерфейс рівня повинен бути спроектовано так, щоб звести до мінімуму залежності шляхом надання відкритого інтерфейсу, що приховує деталі компонентів рівня. Таке приховування називається абстракцією. Існує безліч способів реалізувати її. Пропонуємо підходи, які можуть використовуватися для визначення інтерфейсу рівня:

- Абстрактний інтерфейс. Може бути визначено за допомогою абстрактного базового класу або інтерфейсу, який виступає в ролі опису типу для конкретних класів. Цей тип визначає загальний інтерфейс, що використовується для взаємодії з цим рівнем.

Крок 7 – Визначення інтерфейсів між рівнями

- Загальний тип проектування. Багато шаблонів проектування визначають конкретні типи об'єктів, які представляють інтерфейс на різних рівнях. Ці типи об'єктів забезпечують абстракцію, яка приховує деталі, що стосуються рівня. Наприклад, шаблон Table Data Gateway визначає типи об'єктів, які представляють таблиці в базі даних і відповідають за реалізацію SQL-запитів, необхідних для доступу до даних. Сутності, працюють з об'єктом, нічого не знають про SQL-запити або деталі того, як об'єкт підключається до бази даних і виконує команди. Багато шаблонів проектування базуються на абстрактних інтерфейсах, але в основі деяких з них лежать конкретні класи. Більшість шаблонів, такі як Table Data Gateway, добре задокументовані в цьому відношенні. Загальні типи проектування слід застосовувати, якщо необхідно спосіб швидко і просто реалізувати інтерфейс рівня або при реалізації шаблону проектування для інтерфейсу рівня.

Крок 7 – Визначення інтерфейсів між рівнями

- Інверсія залежностей. Це такий стиль програмування, при якому абстрактні інтерфейси визначаються поза або незалежно від рівнів. Тоді рівні залежать не один від одного, а від загальних інтерфейсів. Шаблон Dependency Injection є типовою реалізацією інверсії залежностей. При використанні Dependency Injection контейнер описує зіставлення, що визначають як знаходити компоненти, від яких можуть залежати інші компоненти, і контейнер може створювати і запроваджувати ці залежні компоненти автоматично. Підхід з інверсією залежностей забезпечує гнучкість і може допомогти в реалізації модульного дизайну, оскільки залежності визначаються конфігурацією, а не кодом. Також такий підхід максимально спрощує тестування, тому що дозволяє вводити тестові класи на різні рівні дизайну.

Крок 7 – Визначення інтерфейсів між рівнями

- Заснований на обміні повідомленнями. Замість взаємодії з компонентами інших рівнів безпосередньо через виклик їх методів або доступу до властивостей можна використовувати зв'язок за допомогою обміну повідомленнями для реалізації інтерфейсів і забезпечення взаємодії між шарами. Існує кілька рішень для обміну повідомленнями, такі як Windows Communication Foundation, Веб-сервіси і Microsoft Message Queuing, які підтримують взаємодію через фізичні кордони і кордони процесів. Основна відмінність підходу на основі повідомлень в тому, що для взаємодії між рівнями використовується загальний інтерфейс, що інкапсулює всі деталі взаємодії. Заснований на обміні повідомленнями підхід рекомендується застосовувати при реалізації Веб-додатків та опису інтерфейсу між рівнем представлення і бізнес-рівнем, який повинен підтримувати безліч типів клієнтів, або якщо потрібно підтримувати взаємодію через фізичні кордони і кордони процесів.

Крок 7 – Визначення інтерфейсів між рівнями

Для реалізації взаємодії між рівнем представлення Веб-застосування та рівнем бізнес-логіки рекомендується використовувати підхід на основі повідомлень. Якщо бізнес-рівень не зберігає стану між викликами (іншими словами, кожен виклик між рівнем представлення і бізнес-рівнем представляє новий контекст), можна передавати дані контексту разом із запитом і забезпечити загальну модель обробки виключень і помилок на рівні представлення.

Крок 8 – Вибір стратегії розгортання

Коли необхідно вибрати найбільш відповідне рішення розгортання для застосування, корисно спочатку розглянути загальні шаблони. Тільки повністю розібравшись з різними схемами розгортання, можна переходити до конкретних сценаріїв, вимогам і обмеженням безпеки, щоб визначитися з найбільш підходящим шаблоном або шаблонами.

Крок 9 – Вибір протоколів зв’язку

Фізичні протоколи, що використовуються для зв'язку між рівнями або вузлами, грають основну роль в забезпеченні продуктивності, безпеки і надійності застосування. Вибір протоколу зв'язку має ще більше значення для розподіленого розгортання. Якщо компоненти розміщуються фізично на одному вузлі, часто можна покластися на пряму взаємодію цих компонентів. Але якщо компоненти і рівні розгорнуто фізично на різних серверах і клієнтських комп'ютерах, як це відбувається в більшості сценаріїв, необхідно продумати, як забезпечити ефективний і надійний зв'язок між компонентами цих рівнів.

Рекомендації щодо проектування багаторівневих застосувань

Дякую за увагу