МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА"

ТЕХНОЛОГІЇ ПРОЄКТУВАННЯ ПРОГРАМНИХ СИСТЕМ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання курсового проєкту
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальності 123 – «Комп'ютерна інженерія»

Обговорено і рекомендовано
на засіданні кафедри
інформаційних та комп'ютерних
систем
Протокол № 5
від 30 квітня 2023 р.

ЧЕРНІГІВ 2023

Технології проєктування програмних систем. Методичні вказівки до виконання курсового проєкту для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 123 – «Комп'ютерна інженерія». Частина I. / Укл.: Казимир В.В., Пріла О.А., Хижняк А.В., Чорноног О.А. – Чернігів: НУ «Чернігівська політехніка»,, 2023. – 23 с., укр. мовою.

Укладачі: КАЗИМИР ВОЛОДИМИР ВІКТОРОВИЧ, доктор технічних наук, професор кафедри інформаційних та комп'ютерних систем;
ПРІЛА ОЛЬГА АНАТОЛІЇВНА, кандидат технічних наук, доцент кафедри інформаційних та комп'ютерних систем;
ХИЖНЯК АНДРІЙ ВАСИЛЬОВИЧ, старший викладач кафедри інформаційних та комп'ютерних систем;
ЧОРНОНОГ ОЛЬГА АНАТОЛІЇВНА, асистент кафедри інформаційних та комп'ютерних систем

Відповідальний за випуск: БАЗИЛЕВИЧ ВОЛОДИМИР МАРКОВИЧ, завідувач кафедри інформаційних та комп'ютерних систем Національного університету "Чернігівська політехніка", кандидат економічних наук, доцент

Рецензент: БИВОЙНО ПАВЛО ГЕОРГІЙОВИЧ, кандидат технічних наук, доцент кафедри інформаційних та комп'ютерних систем Національного університету "Чернігівська політехніка"

Зміст

Вступ..4
1 Поняття корпоративних програмних систем (систем рівня підприємства). Вимоги до них...5
2 Вимоги до архітектури та структури застосування 7
2.1 Використання патернів проєктування архітектури застосувань 7
2.2 Особливості використання архітектурних шарів 7
2.3 Класична модель розшарування системи............................8
2.4 Вертикальна та горизонтальна декомпозиція системи........9
2.5 Шар сервісів в класичній моделі розшарування..................9
2.6 Взаємодія між шарами..11
2.7 Розміщення шарів 11
2.2 Параметри продуктивності веб-застосування 13
3 Вимоги до підсистеми безпеки застосування......................15
4 Автоматизація процесів тестування, збірки та розгортання проєкту 16
5 Організація командної роботи над проєктом 17
6 Структура пояснювальної записки 18
7 Порядок захисту та оцінювання роботи над проєктом......19
Додатки 21
Додаток А - Протокол захисту курсового проєкту 22
Рекомендована література………………………………………23
Інформаційні ресурси…………………………………………….23

Вступ

Курс «Технології проєктування програмних систем» присвячений вивченню сучасних підходів до проєктування і реалізації корпоративних програмних систем (КПС), що характеризуються складністю бізнес-логіки, а також високими вимогами до рівня надійності, продуктивності, розширюваності, масштабованості, безпеки та захищеності.

Особлива увага приділяється особливостям застосування класичних патернів проєктування та усталених практик, які використовуються для розробки КПС.

Метою виконання курсового проєкту є отримання практичних навичок проєктування і розробки програмних систем рівня підприємства, використовуючи загальноприйняті стандарти і підходи, а також сучасні бібліотеки та інструментальні засоби розробки КПС.

У методичних вказівках наведені вимоги до архітектурних особливостей розроблюваного в рамках курсового проєкту додатку без орієнтації на конкретні інструментальні засоби і бібліотеки. Представлені вимоги до організації командної роботи над проєктом. Курс орієнтований на використання специфікації Java EE 7, а також agile-принципів командної розробки програмного забезпечення.

Методичні вказівки можуть використовуватися при виконанні дипломної роботи та проходженні переддипломної практики.

1 Поняття корпоративних програмних систем (систем рівня підприємства). Вимоги до них

Корпоративна програмна система – це програмна система, яка оперує великими обсягами даних за допомогою певних бізнес-правил, орієнтована на конкретний вид діяльності підприємства та від її функціонування часто залежить діяльність підприємства.

Корпоративні системи мають ряд особливостей: вони істотно відрізняються від вбудованих систем, систем керування, телекомунікаційних систем, операційних систем тощо. Прикладом може бути порівняння корпоративної системи із програмною системою сфери телекомунікацій. З певної точки зору, корпоративні системи є набагато простішими, ніж телекомунікаційне програмне забезпечення, оскільки не вирішують проблем забезпечення багатопоточного функціонування, інтеграції апаратних і програмних компонентів. Однак, в іншому КПС є значно складнішими, оскільки передбачають обробку даних (що зазвичай великі за обсягом та передбачають довготривале зберігання) з використанням бізнес-правил, що дуже часто змінюються.

Отже, основними аспектами розробки корпоративних додатків є:

1) Корпоративні програми зазвичай включають необхідність довготривалого (іноді протягом десятиліть) зберігання даних. Дані часто здатні "пережити” кілька поколінь реалізацій прикладних програм, призначених для їх обробки, апаратних засобів, операційних систем. Упродовж цього часу структура даних може піддаватися численним змінам з метою додавання нових даних без впливу на старі. У разі внесення змін до програмно-апаратного забезпечення, дані не знищуються, а переносяться в нове середовище. Обсяг даних, як правило, є великий і завдання маніпуляції цими даними виростає в одну з основних функцій програмної системи.

2) Значна кількість користувачів звертається до даних паралельно, що обов’язково необхідно враховувати для забезпечення продуктивності системи при визначенні нефункціональних вимог. Зрозуміло, що більшість завдань з виконання паралельних низькорівневих транзакцій вирішуються на рівні диспетчера транзакцій систем керування базами даних, проте управління рівнем ізоляції бізнес-транзакцій виконується на рівні прикладного застосування та суттєво впливає на продуктивність. Високий рівень ізоляції транзакцій підвищує надійність інформаційної системи, усуває ймовірність виникнення конфліктних ситуацій, проте значно знижує продуктивність системи.

3) Велика кількість ролей користувачів застосування передбачає необхідність реалізації різних варіантів екранного інтерфейсу та можливість подання даних в різних формах, зручних для різних груп користувачів.

4) Бізнес-логіка корпоративних систем складна та характеризується постійним внесенням змін. Це, наприклад, різні умови оплати кредиту для клієнтів з урахуванням різних критеріїв. Велика кількість таких умов роб-

лять бізнес-логіку програмної системи складною, заплутаною і такою, що складно піддається сприйняттю. Для порівняння, операційної системи має строгу логіку реалізації.

5) Некоректне функціонування системи або невчасна реалізація функціоналу та впровадження системи в експлуатацію означає значні фінансові затрати.

До числа корпоративних додатків відносяться, наприклад, системи бронювання квитків, програми забезпечення фінансових процесів, системи торгового, бухгалтерського обліку, збереження та обробки медичних карт пацієнтів, системи документообігу і т.і. Корпоративними додатками не є засоби обробки тексту, керування пристроями, телекомунікаційні системи, операційні системи, компілятори, ігри і т.д.

Зазначені особливості корпоративних програмних систем визначають такі нефункціональні вимоги до них: надійність, захищеність, масштабованість, гнучкість інтеграції та супроводження.

2 Вимоги до архітектури та структури застосування

Термін «архітектура програмної системи» не має однозначного визначення. Крім того, існує більше одного способу опису архітектури та ступінь важливості кожного з них змінюється впродовж життєвого циклу застосування.

Два класичних визначення терміна «архітектура» представлено нижче:

1) ідентифікація головних компонент системи та способів їх взаємодії;

2) вибір рішень, які інтерпретуються як основоположні та не підлягають зміні в майбутньому.

Перший пов'язаний з поділом системи на найбільш великі складові частини. У другому випадку маються на увазі конструктивні рішення, які не підлягають зміні після їх прийняття. Якщо пізніше виявляється, що прийняті рішення підлягають зміні, то вони виключаються з "архітектури" застосування.

2.1 Використання патернів проєктування архітектури застосувань

Основа паттерна (типового рішення) – підхід, досить загальний і ефективний для подолання проблем реалізації певного типу. Типові рішення – не наукові відкриття, вони є великою мірою узагальненням результатів, накопичених у відповідній області. Тому доречно говорити про пошук типового рішення, аніж про його винахід.

Обираючи архітектуру застосування та шаблони реалізації, слід брати до уваги особливості конкретної системи. Обране типове рішення найчастіше вимагає модернізації для конкретного випадку.

При проєктуванні архітектури застосування ефективним є застосування типових підходів до виділення основних компонентів, архітектурних шарів і схеми взаємодії між ними.

2.2 Особливості використання архітектурних шарів

Концепція шарів (layers) – одна з загальновживаних, що використовується розробниками програмного забезпечення для розділення складних систем на прості частини. Відрізняють поняття горизонтальної та вертикальної декомпозиції систем.

Шар вищого рівня зазвичай «знає» про особливості реалізації шару нижнього рівня та може користуватися сервісами нижнього шару. Однак нижні шари не знають про особливості реалізації верхніх шарів та не можуть звертається до їх сервісів. Визначають поняття чіткого та нечіткого розшарування системи. Чітке розмежування передбачає, що кожний проміжний шар "приховує" нижній шар від верхнього: наприклад, шар 4 користується послугами шару 3, який звертається до шару 2, але шар 4 не знає

про існування шару 2. Поняття нечіткого розмежування передбачає можливість звернення із даного шару до сервісів будь-якого нижнього шару.

Розподілення системи на шари надає ряд переваг:

- окремий шар можна розглядати як єдине самодостатнє ціле;
- можна вибирати альтернативну реалізацію базових шарів;
- залежності між шарами можна звести до мінімуму;
- кожен шар є вдалим кандидатом на стандартизацію.

Розшарування архітектури має й певні недоліки:

1) шари здатні інкапсулювати особливості реалізації застосування, однак модифікація одного шару зазвичай пов'язана з необхідністю внесення каскадних змін в інші шари. Наприклад, поле, додане в таблицю бази даних, зазвичай має бути відтвореним в графічному інтерфейсі та знайти відповідне відображення в кожному проміжному шарі.

2) Наявність надлишкових шарів знижує продуктивність системи. При передачі об’єктів від шару до шару вони зазвичай потребують перетворення з одного вигляду до іншого. Однак, незважаючи на це, інкапсуляція нижніх шарів дозволяє досягти істотних переваг. Наприклад, оптимізація шару транзакцій зазвичай призводить до підвищення продуктивності верхніх шарів.

Найважче при використанні архітектурних шарів – це визначення меж відповідальності кожного шару.

2.3 Класична модель розшарування системи

Класична архітектура передбачає наявність 3 основних шарів: шар представлення (presentation), шар бізнес-логіки (домен, domain) та шар інтеграції (джерело даних, data source).

До основних функцій шару представлення відносяться відображення інформації та інтерпретація команд, що вводяться користувачем, з перетворенням їх у відповідні запити в контексті домену (бізнес-логіки) та джерела даних.

Шар джерела даних реалізує підмножину функцій, що забезпечують взаємодію застосування зі сторонніми системами, які реалізують функції, що використовуються додатком. Код цієї категорії несе відповідальність за моніторинг транзакцій, взаємодію з іншими застосуваннями, обмін повідомленнями та ін. Для більшості корпоративних додатків основна частина низькорівневої логіки джерела даних зосереджена в коді СУБД та відповідному провайдері доступу до БД.

Логіка домену (бізнес-логіка або логіка предметної області) описує основні функції застосування, призначені для досягнення поставленої перед ним мети. До таких функцій відносяться обчислення для даних, перевірка даних та обробка запитів, що надходять від шару представлення, а також передача інформації шару джерела даних.

Класична архітектура корпоративної програмної системи, що характеризується складністю бізнес-логіки, представлена на рисунку 1.

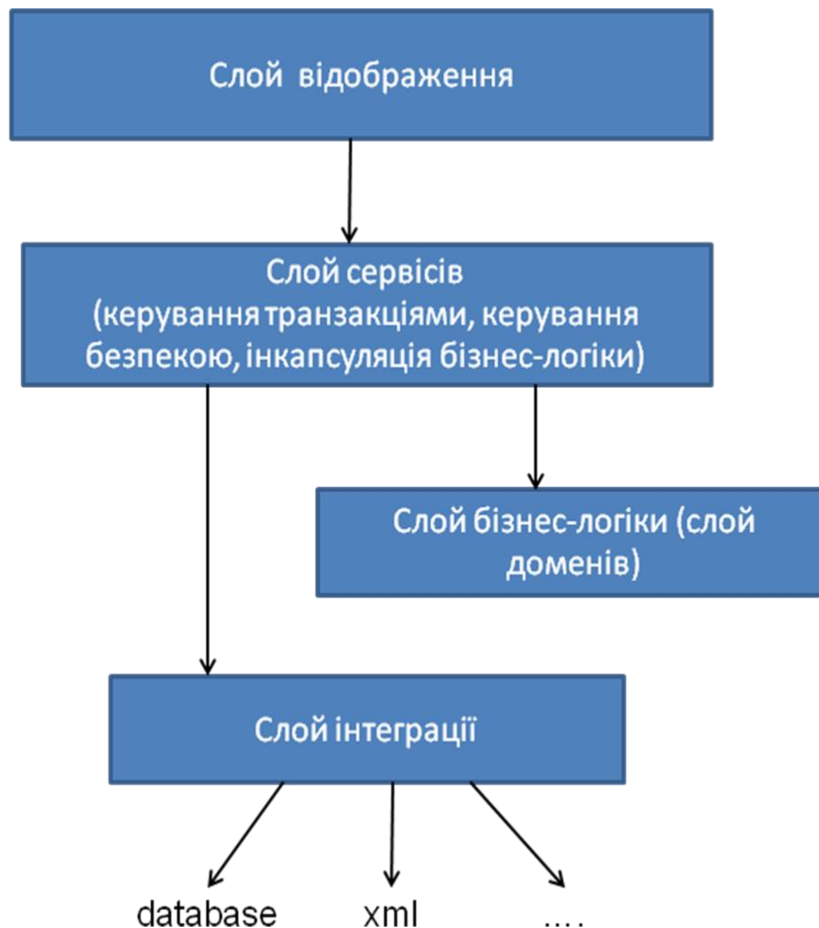

Рисунок 1 – Класична архітектура корпоративної програмної системи

Модель трьох шарів – не єдина, існують альтернативні схеми "розшарування" коду. Хоча три основні шари – відображення, бізнес-логіка і джерело даних – можна виявити в будь-якому корпоративному додатку, спосіб їх поділу залежить від ступеня складності цього додатка. Форма розшарування може бути довільною, але в будь-якому корпоративному додатку шари повинні бути ідентифіковані.

Часто застосування має декілька варіантів реалізації певного шару.

2.4 Вертикальна та горизонтальна декомпозиція системи

Архітектура застосування повинна відображати не тільки вертикальну, але і горизонтальну декомпозицію системи. Вертикальна декомпозиція передбачає виділення архітектурних шарів, в той час як горизонтальна – визначення функціональних модулей в межах одного архітектурного шару. Вертикальна декомпозиція не відображає функціональних особливостей і предметної області застосування, в більшості випадків є стандартна для багатьох корпоративних програмних застосувань.

2.5 Шар сервісів в класичній моделі розшарування

При проєктуванні корпоративних систем та при виконанні лабораторних робіт та курсового проєкту, відповідно, пропонується відділення ло-

гіки застосування від «чистої» логіки домену та введення додаткового шару сервісів (або шару служб).

Проміжні шари – часто, але не обов'язково корисні. Загальний підхід: якщо який-небудь з трьох базових шарів переходить розумну межу складності, модель можна доповнити додатковим шаром, який приймає на себе надлишок функцій.

Ідея виокремлення шару служб з шару предметної області передбачає можливість відмежування логіки процесу від "чистої" бізнес-логіки. Рівень служб зазвичай охоплює логіку, яка відноситься до конкретного варіанту використання системи або забезпечує взаємодію з іншими інфраструктурами (наприклад, за допомогою механізму повідомлень).

Шар служб визначає межі застосування та API сервісів, що надаються шару представлення та іншим клієнтам застосування. Шар інкапсулює бізнес-логіку додатка, здійснює керування транзакціями та керує безпекою застосування. В даному випадку бізнес-логіку розносять по двох категоріях: логіка домену (domain logic), що має справу лише з предметною областю застосування, і логіка застосування (application logic), що визначає допоміжну логіку додатку (наприклад, надсилання повідомлень користувачам або стороннім застосуванням). Такий підхід є ефективним, оскільки розміщення допоміжної логіки додатку в "чистих" класах домену небажано, тому що, по-перше, класи домену допускають меншу ймовірність повторного використання, якщо вони реалізують специфічну логіку застосування та залежать від тих чи інших прикладних інструментальних пакетів, та, по-друге, змішування логіки обох категорій в контексті одних і тих же класів ускладнює можливість нової реалізації логіки застосування, якщо необхідність такого кроку стає очевидною. З цих причин шар служб передбачає розподілення "різної" логіки по окремих шарах, що забезпечує традиційні переваги розшарування, та забезпечує можливість гнучкого використання компонентів шару доменних об'єктів в інших додатках.

Компоненти шару служб можуть використовуватись також для розміщення логіки управління транзакціями та керування безпекою і зазвичай реалізують відповідні сервіси визначення меж транзакцій, перевірки прав доступу користувачів.

Найскладнішим при проєктуванні бізнес-логіки є, ймовірно, вибір того, що саме слід відносити до того чи іншого шару. Наприклад, якщо для web-застосування додається принципово відмінний шар, як-от, інтерфейс командного рядку. Якщо існують сервіси, які доведеться продублювати в даному випадку, то логіка домену "перетікає" в шар представлення. Або можна навести інший приклад: чи потрібно повторювати бізнес-логіку компонентів застосування при необхідності заміни реляційної бази даних XML-файлом?

Уявіть собі додаток, що відображає червоним кольором назви товарів, обсяги продажів яких зросли більш ніж на 10% в порівнянні з рівнем минулого місяця. Припустимо, що відповідна логіка розміщена безпосередньо в шарі представлення, що передбачає зіставлення рівню продажів по-

точного й минулого місяця та відповідної зміни кольору, якщо різниця перевищує заданий поріг. В даному випадку до шару відображення вводиться зайва логіка предметної області. Для належного розділення шарів потрібно реалізувати метод бізнес-логіки, що визначає факт перевищення рівня продажів певного продукта на задану величину – метод, що здійснює порівняння рівнів продажів за два місяці, та повертає відповідне значення. Надалі у коді шару представлення викликати відповідний метод бізнес-логіки та, в залежності від повернутого значення, прийняти рішення про колір відображення. В цьому випадку логіку сервісу буде розділено на дві частини: логіка виявлення необхідності зміни кольору, і, власне, зміна при відображенні.

2.6 Взаємодія між шарами

Крім необхідності поділу на шари, існують правила взаємодії між шарами: залежність бізнес-логіки і джерела даних від рівня відображення не допускається, тобто в коді застосування не повинно бути викликів функцій відображення з коду бізнес-логіки або джерела даних. Такий підхід дозволяє спростити можливість адаптації шару уявлення або заміни його альтернативним варіантом зі збереженням базових модулей застосування. Зв'язок між бізнес-логікою і джерелом даних, однак, не настільки однозначний і багато в чому визначається вибором типових рішень для архітектури джерела даних.

Іноді шари організують таким чином, щоб бізнес-логіка повністю приховувала джерело даних від уявлення. Найчастіше, однак, код уявлення може звертатися до джерела даних безпосередньо. Хоча такий варіант менш бездоганний з теоретичної точки зору, в практичному відношенні він нерідко більш зручний і доцільний: код уявлення може інтерпретувати команду користувача, активізувати функції джерела даних для вилучення відповідних порцій інформації з бази даних, звернутися до засобів бізнес-логіки для аналізу цієї інформації і здійснення необхідних розрахунків і потім відобразити результат на екрані.

2.7 Розміщення шарів

Вище йшлося про логічних шарах, тобто про розчленування системи на окремі частини. Подібне розділення корисно навіть тоді, коли всі шари функціонують на одній машині. Втім, існують ситуації, в яких відмінності в поведінці системи можуть бути обумовлені принципами її фізичної організації. У більшості випадків існує тільки два варіанти розміщення і виконання компонентів корпоративних додатків – на стороні клієнта і на стороні сервері. Найчастіше найпростішим є функціонування коду всіх верств системи на сервері, при цьому максимально спрощуються процедури виправлення помилок і оновлення версій. В цьому випадку не доводиться турбуватися про внесення відповідних змін на всіх комп'ютерах, про їх сумісність з іншими додатками і синхронізації з серверними компонентами. Загальні аргументи на користь розміщення будь-яких шарів на комп'ютері

клієнта полягають в підвищенні швидкості реагування (responsiveness) додатку і в забезпеченні можливості локальної роботи.

Шар джерела даних краще завжди розташовувати на сервері. Виняток становить випадок, коли функції сервера дублюються в коді "дуже товстого" клієнта для забезпечення засобів локального функціонування системи. При цьому передбачається, що зміни, що вносяться до окремих джерел даних на клієнтській машині і на сервері, підлягають синхронізації за допомогою механізму реплікації.

Рішення про те, де повинен функціонувати шар уявлення, здебільшого залежить від переваг у виборі типу призначеного для користувача інтерфейсу. Застосування інтерфейса товстого клієнта автоматично тягне за собою необхідність розміщення шару відображення на клієнтській машині. Використання Web-інтерфейсу означає, що логіка уявлення зосереджена на сервері. Можливий також змішаний варіант розміщення логіки подання – частково на сервері, частково на клієнті.

Розщеплення безлічі бізнес-функцій між сервером і клієнтом виглядає як найгірше рішення, оскільки в загальному випадку ускладнює ідентифікацію того чи іншого фрагмента логіки. Основна причина, що спонукає застосовувати подібну архітектуру, може полягати в тому, що клієнту необхідно володіти тільки якоюсь частиною бізнес-логіки. Головне – ізолювати цю порцію коду в окремому модулі, який не залежить від інших частин системи. Це дасть можливість активізувати код і на комп’ютері клієнта, і на сервері, якщо така потреба виникне пізніше.

Як вже було зазначено, існують різні варіанти реалізації кожного з архітектурних шарів. Існують наступні «класичні» шаблони реалізації бізнес-логіки програми: «Сценарій транзакцій», «Модуль таблиці», «Модель предметної області (домен)».

При виборі варіанта реалізації шару сервісів додатки рекомендуються до розгляду шаблони «Інтерфейс доступу до домену», «Сценарій операції».

Реалізація шару бізнес-логіки у вигляді «Моделі домену» і шару сервісів як «Інтерфейс доступу до домену» дозволяє реалізувати об'єкти бізнес-логіки «чистими», в т. ч. незалежними від реалізації шару інтеграції.

При проєктуванні і розробці компонентів додатка повинен бути застосований принцип інверсії контролю (Inversion of Control, IoC), що буде розглянутий надалі.

2.2 Параметри продуктивності веб-застосування

Вибір архітектурних рішень безпосередньо пов'язаний з аспектами продуктивності (performance) застосування. Нижче наведено параметри продуктивності програмної системи.

Час відгуку (response time) – час обробки запиту.

Швидкість реагування (responsiveness) – швидкість підтвердження запиту. Інтерактивна система, що володіє нормальним часом відгуку, але не відрізняється високою швидкістю реагування, характеризується низькою ефективністю використання. У випадку якщо, перш ніж прийняти черговий запит, система повинна повністю завершити обробку поточного, параметри часу відгуку та швидкості реагування збігаються. Якщо ж система підтверджує отримання запиту раніше, її швидкість реагування вище. Наприклад, застосування динамічного індикатора стану процесу копіювання підвищує швидкість реагування екранного інтерфейсу, хоча не впливає на час відгуку.

Час затримки (latency) – мінімальний інтервал часу до отримання відгуку на запит, який нічого не виконує («пустий» запит). Даний параметр не є характеристикою програмної системи та не може бути змінений розробником, проте при проєктуванні та розробці веб-застосувань його потрібно враховувати шляхом мінімізації кількості віддалених викликів, що реалізується за допомогою патерну «Об'єкт передачі даних» (Data Transfer Object).

Пропускна здатність (throughput) – кількість даних або операцій, що передаються або виконуються за одиницю часу. Якщо, наприклад, тестується процедура копіювання файлу, пропускна здатність може вимірюватися числом байтів в секунду. Для корпоративних додатків зазвичай мірою продуктивності є число транзакцій за 1 секунду (transactions per second – tps). Однак цей підхід не є однозначний, оскільки транзакції відрізняються ступенем складності. Для конкретної системи необхідно розглядати суміш "типових" транзакцій.

Під продуктивністю можна розуміти один з двох параметрів – час відгуку або пропускну здатність, той, що більшою мірою відповідає сервісам відповідної системи. Інколи складно визначити продуктивність системи, якщо, наприклад, використання деякого рішення підвищує пропускну здатність системи, одночасно збільшуючи час відгуку. З точки зору інтерфейсу користувача, значення швидкості реагування є більш важливим, ніж час відгуку, тому поліпшення швидкості реагування за рахунок зниження пропускної здатності або зростання часу відгуку цілком здатне підвищити продуктивність.

Навантаження системи (load) – кількість запитів, що одночасних надходять до системи. Параметр навантаження зазвичай вказується для визначення інших нефункціональних характеристик, наприклад, «час відгуку на запит становить 0,5 секунди для 10 користувачів і 2 секунди для 20 користувачів».

Чутливість до навантаження (load sensitivity) – вираз, що задає залежність часу відгуку від навантаження. Припустимо, що система А має час відгуку, що дорівнює 0,5 секунди для 10-20 користувачів, система В – час відгуку 0,2 секунди для 10 користувачів та 2 секунди для 20 користувачів. Отже, система А має меншу чутливість до навантаження, ніж система В. Для веб-застосувань використовується також термін погіршення (degradation) для підкреслення меншої стійкості параметрів системи.

Ефективність системи (efficiency) – питома продуктивність системи в перерахунку на кількість одиниць ресурсів, що використовуються. Наприклад, програмна систем, що використовує два процесори та має продуктивність 30 tps більш ефективна в порівнянні з системою, що має чотири аналогічних процесори та продуктивність 40 tps.

Потужність (capacity) – найбільше значення пропускної здатності або навантаження, при якому параметр продуктивності не перевищує задане критичне значення.

Здатність до масштабування (scalability) – властивість, що характеризує поведінку системи при додаванні ресурсів (зазвичай апаратних). Масштабованою прийнято вважати систему, продуктивність якої зростає пропорційно обсягу ресурсів, що використовуються. Вертикальне масштабування (vertical scalability, scaling up) – це збільшення потужності окремого сервера (наприклад, за рахунок збільшення обсягу оперативної пам'яті). Горизонтальне масштабування (horizontal scalability, scaling out) – це нарощування потенціалу системи шляхом додавання нових серверів. Здатність системи до масштабування визначається особливостями архітектури.

В курсовому проєкті при проєктуванні та реалізації застосування слід використовувати принципи проєктування розподіленої мікросервісної архітектури застосувань та відповідні технології розробки (веб-сервіси, EJB).

Також з метою забезпечення параметрів продуктивності має бути передбачена реалізація реплікації бази даних застосування (основної бази даних або бази даних користувачів) та тестування навантаження веб-застосування з використанням відповідних інструментів тестування.

3 Вимоги до підсистеми безпеки застосування

Згідно Jakarta EE 9 виділяють наступні характеристики безпеки застосування.

- Аутентифікація (Authentication) – процес ідентифікації взаємодіючих об'єктів, таких як клієнт або сервер; перевірка їх достовірності.
- Авторизація (Authorization, access control) – процес підтвердження прав користувача (або іншого об'єкта) на виконання тих чи інших дій.
- Цілісність даних (Data integrity) – засоби забезпечення неможливості підміни даних, що пересилаються по мережі, «третьою особою» (чи не джерелом інформації). Одержувач даних повинен мати можливість виявлення та скасування прийому повідомлень, які були змінені після відправки.
- Конфіденційність (Confidentiality data privacy) – забезпечення досту до захищеної інформації тільки для авторизованих користувачів.
- QoS (Quality of Service) – засоби забезпечення необхідного рівня якості обслуговування.
- Облік (Accounting, Auditing) виконуваних операцій доступу до захищених ресурсів застосування з метою оцінки ефективності забезпечення політики безпеки застосування та якості механізмів, що використовуються.

В курсовому проєкті підсистема безпеки застосування повинна бути реалізована з використанням JAAS або іншої реалізації стандарту системи інформаційної безпеки PAM. Має бути виконано налаштування реалму веб-контейнера, який містить колекцію користувачів та груп користувачів, з подальшим відображенням ролей користувачів застосування конкретних користувачам або групам. В якості реалма пропонується використання сервера LDAP.

Окремо повинні бути передбачені механізми захищеної передачі даних для певних (критичних з точки зору безпеки) ресурсів веб-застосування. Ефективним підходом є використання захищених протоколів передачі даних не для всіх ресурсів застосування, оскільки процеси шифрування/дешифрування є обчислювально ресурсоємними.

4 Автоматизація процесів тестування, збірки та розгортання проєкту

Реалізація кожного шару застосування повинна супроводжуватися відповідними автоматизованими модульними тестами.

Важливим є забезпечення незалежного тестування компонентів застосування. Наприклад, з метою уникнення неявного тестування компонентів шару інтеграції при тестуванні компонентів шару сервісів, рекомендовано використання mock-об'єктів для компонентів шару інтеграції. Обов'язковим є використання принципу інверсії контролю, що забезпечує незалежність компонентів та спрощує процес тестування, оскільки дозволяє легко заміняти реальну реалізацію сервісів на тестову реалізацію компонентів у вигляді заглушок або mock-об'єктів.

Крім модульних, повинні бути сплановані та реалізовані інтеграційні тести. Для написання модульних і інтеграційних тестів рекомендується використання бібліотеки JUnit 5.

Для оцінки параметрів продуктивності додатка, має бути проведено тестування навантаження. Для тестування навантаження рекомендується використання інструментарію Apache JMeter.

Обов'язковою вимогою є організація автоматичної зборки та розгортання компонентів застосування з використанням будь-якої з існуючих систем зборки. Зборка та розгортання проєкту має виконуватись незалежно від середовища розробки проєкту та оточення та має включати етап автоматичного запуску модульних і інтеграційних тестів. В якості системи зборки java-застосувань рекомендується використання системи Apache Maven.

5 Організація командної роботи над проєктом

Склад команди, розподіл ролей. Курсовий проєкт виконується групами по 3-4 людини з виділенням наступних ролей в команді: менеджер, розробник і розробник тестів (тестувальник).

Оскільки проєкт носить навчальний характер, завдання учасників, що мають різні ролі в проєкті, будуть перетинатися. Наприклад, тестувальник повинен бути зайнятий не тільки в плануванні та написанні тестів, але й в розробці модулів системи. При цьому повинен залишатися пріоритет завдань, характерних для певної ролі.

Розподіл робіт повинно бути виконано таким чином, щоб кожен член команди взяв участь в розробці / тестуванні компонентів (модулів) кожного шару додатки.

Завдання менеджера проєкту:

- проєктування архітектури додатку;
- планування робіт, розподіл і призначення завдань;
- проєктування користувальницького інтерфейсу;
- формування фінальної документації;
- розробка і тестування модулів системи (20%).

Завдання розробника:

- проєктування;
- розробка модулів системи (60%);
- модульне тестування власних модулів;
- документування.

Завдання тестувальника:

- Проєктування і реалізація інтеграційних тестів;
- Тестування інтерфейсу користувача;
- Планування і проведення навантажувальних тестів;
- Розробка модулів системи (20%).

Використання системи управління проєктами. Фіксація, призначення та контроль виконання завдань проєкту повинні виконуватися централізовано, з використанням системи управління проєктами. Рекомендується використання публічних зовнішніх сервісів спільної розробки проєктів.

Використання системи управління версіями. Обов'язковою умовою командної роботи над проєктом є використання репозиторію вихідного коду проєкту.

Системи безперервної інтеграції (continuous integration). Принцип безперервної інтеграції передбачає постійну та швидку збірку та тестування на різному рівні проєкту, наприклад, при кожному внесенні змін до проєкту. Існують системи автоматизації процесу зборки та тестування застосувань, що дозволяють визначити періодичність та умови запуску процесу зборки. Використання в процесі розробки курсового проєкту засобів безперервної інтеграції не є обов'язковим.

6 Структура пояснювальної записки

Пояснювальна записка повинна бути оформлена відповідно до вимог [1].

Розширене технічне завдання. Крім стандартних вимог до структури технічного завдання, необхідно подати:

1) розширені функціональні вимоги до застосування з визначенням ролей користувачів програми та варіантів використання системи для кожної ролі. У разі корпоративної програмної системи кількість призначених для користувача ролей має бути вище трьох;

2) нефункціональні вимоги до системи та механізми їх забезпечення;

3) вимоги до підсистеми безпеки;

4) передбачувані технології та інструментальні засоби розробки системи;

5) склад команди, розподіл ролей і визначення зон відповідальності кожного учасника.

Основна частина пояснювальної записки повинна включати такі результати проєктування та реалізації застосування:

- аналіз предметної області, визначення функціональних і не функціональних вимог до системи;
- проєктування інтерфейсу користувачів додатка;
- проєктування архітектури додатку, вибір архітектурних рішень;
- вибір технологій та засобів реалізації;
- планування та розподіл завдань проєкту;
- проєктування та реалізація бізнес-логіки застосування: використовувані патерни реалізації, використовувані технології, листинги доменних об'єктів з описом, листинги модульних і інтеграційних тестів, результати тестування, листинги файлів конфігурації;
- проєктування та реалізація шару інтеграції (структура аналогічна попередній);
- проєктування та реалізація шару уявлення (структура аналогічна попередній);
- реалізація підсистеми безпеки;
- механізми забезпечення продуктивності застосування та результати тестування навантаження.

Вміст вихідних файлів проєкту в додаток виносити не потрібно. У додатках можуть бути приведені результати тестування, конфігураційні файли та ін. Звіт оформляється командою в одному екземплярі.

7 Порядок захисту та оцінювання роботи над проєктом

Для захисту курсової роботи кожна команда повинна представити презентацію проєкту, а також заповнений протокол захисту курсового проєкту (див. Додаток А). Порядок презентації проєкту представлений нижче.

Підготовча робота для демонстрації проєкту:

- завантажити вихідні файли проєкту з репозиторію;
- продемонструвати структуру каталогів проєкту (вона повинна відповідати декомпозиції проєкту, обов'язкова наявність файлів: readme, authors, copyright);
- продемонструвати автоматичну зборку та розгортання проєкту з використанням будь-якої з існуючих систем зборки незалежно від середовища розробки проєкту. Зборка проєкту повинна включати етап автоматичного запуску тестів.

В ході презентації проєкту необхідно відобразити такі аспекти:

- призначення розробленої системи, її основні функції;
- представити архітектуру системи, архітектурні рішення та патерни, що використовуються;
- розподіл завдань за проєктом між учасниками команди, визначення ролей і функцій кожного учасника;
- реалізація системи та використовувані при розробці технології (кожен учасник представляє результати реалізації підсистем, розробкою яких він займався);
- демонстрація системи на основі сценаріїв тестування (3 основних сценарії використання);
- демонстрація працездатності реплікації бази даних;
- надати результати тестування навантаження веб-застосування;
- організація підсистеми безпеки застосування.

Супровід презентації проєкту слайдами є бажаний.

Кожен учасник проєкту отримує індивідуальну оцінку. Початкова оцінка студента – це оцінка проєкту відповідно до визначений вимог. Внесок кожного учасника в проєкт оцінюється відповідно до наступних критеріїв:

- активність роботи з репозиторієм проєкту;
- якість реалізації та презентації своєї частини робіт за проєктом;
- якість оформлення документації;
- володіння і знання вихідного коду всього проєкту (командна розробка передбачає колективне володіння кодом, тобто кожен з учасників повинен орієнтуватися в початкових кодах всіх модулів проєкту);
- лідер проєкту може підвищувати або знижувати оцінку кожного з учасників до 10 балів.

Приклади можливих запитань: «яким чином поле Х через шар бізнес-логіки потрапляє з БД на екран користувача?»; «Поясніть тестовий метод

для тієї чи іншої функції»; «Яким чином перевіряються права користувача на виконання тієї чи іншої дії в системі?» і т.д.

Додатки

Додаток А - Протокол захисту курсового проєкту

Протокол захисту курсового проєкту з дисципліни «Технології проєктування програмних систем»

дата захисту ___.___.____
Склад команди (ПІБ, група) ______________________________
Тема роботи: ______________________________

Відповідність програмного проєкту вимогам, затвердженим в ТЗ ______
Загальна оцінка проєкта _________
Індивідуальні оцінки ______________________________

Критерії оцінювання проєкту

Управління конфігурацією проєкту
• наявність вихідних файлів проєкту в репозиторії;
• правильна структура каталогів проєкту;
• автоматизована збірка, запуск тестів і розгортання застосування.
Проєктування та реалізація
• розподіл архітектури за рівнями;
• мікросервісна архітектура застосування (бажано);
• модульне та інтеграційне тестування компонентів всіх шарів, тестування навантаження;
• реплікація БД;
• реалізація підсистеми безпеки (налаштування реалма веб-контейнера, конфігураційні файли)
Управління проєктом
• розподіл завдань, планування;
• якість оформлення документації проєкту.
Презентація проєкту
• розгортання серверів додатку;
• опис розробленої системи (призначення, функції);
• реалізація системи, технології, що використовувалися при розробці;
• демонстрація сервісів застосування на основі сценаріїв використання.

Критерії індивідуального оцінювання

• активність роботи з репозиторієм проєкту;
• якість реалізації, документації та презентації відповідних завдань;
• володіння вихідним кодом всього проєкту;
• оцінювання team-лідером роботи членів команди.

Рекомендована література

1. Методичні рекомендації щодо виконання та оформлення кваліфікаційних робіт здобувачів вищої освіти освітніх ступенів «бакалавр» і «магістр» Національного університету «Чернігівська політехніка». – Чернігів: НУЧП, 2020. – 16 с.
2. Martin Fowler. Patterns of Enterprise Application Architecture. – 2002. – 560 p.
3. Фаулер, М. Архітектура корпоративних програмних систем: пров. з англ. - М.: Видавничий дім "Вільямс", 2006. - 544 с.
4. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2017. – 432 p.
5. The Jakarta EE Tutorial Release Version 9.1. – December 2021. – [Електронний ресурс]. – Режим доступу: https://eclipse-ee4j.github.io/jakartaee-tutorial.
6. David Heffelfinger. Demystifying Microservices for Jakarta EE Developers / Payara User Guide. - 34 p.
7. Craig Walls. Spring in Action. Fifth Edition, 2019.
8. Craig Walls. Spring in Action, Sixth Edition. – [Електронний ресурс]. – Режим доступу: https://www.manning.com/books/spring-in-action-sixth-edition.
9. Bauer C., King G. Java Persistence with Hibernate. – Manning, 2016. – 610 p.
10. Baeldung. – [Електронний ресурс]. – https://www.baeldung.com/.
11. Spring | Home. – [Електронний ресурс]. – https://spring.io/.

Інформаційні ресурси

1. Jakarta Enterprise Edition Your First Cup: An Introduction to Jakarta EE, Release 8. – [Електронний ресурс]. – Режим доступу: https://eclipse-ee4j.github.io/jakartaee-firstcup/toc.html.
2. Jakarta EE Platform Specification Version 9.1 Final Release. –April 27, 2021. – [Електронний ресурс]. – Режим доступу: https://jakarta.ee/specifications/platform/9.1/jakarta-platform-spec-9.1.html.
3. Github: Jakartaee Tutorial Examples. – [Електронний ресурс]. – Режим доступу: https://github.com/eclipse-ee4j/jakartaee-tutorial-examples.
4. MicroProfile | projects.eclipse.org. – [Електронний ресурс]. – Режим доступу:https://projects.eclipse.org/projects/technology.microprofile
5. Home MicroProfile. – [Електронний ресурс]. – Режим доступу: https://microprofile.io/.