

Технологія створення програмних продуктів



Лекція 5. Проектування програмних систем.

Проектування програмних систем

Проектування ПЗ – це процес розробки, який виконується після етапу аналізу і формування вимог. Завдання проектування полягає у перетворенні вимог до системи у вимоги до ПЗ та побудові архітектури системи.

Архітектура системи – це структурна схема компонентів системи, що взаємодіють між собою через інтерфейси.

Основна умова **побудови архітектури системи** – це **декомпозиція** системи на **компоненти** або **модулі**, а також:

- визначення цілей і перевірка їх виконуваності;
- визначення вхідних і вихідних даних;
- ієрархічне подання абстракції системи та приховування тих деталей, які будуть відпрацьовані на наступних рівнях.

Загальні рекомендації до проектування архітектури системи

- У першу чергу архітектура повинна включати загальний опис системи;
- Архітектура повинна містити підтвердження того, що при її розробці були розглянуті альтернативні варіанти, і обґрунтовувати вибір остаточної організації системи;
- Архітектура повинна визначати основні компоненти програми;
- Архітектура повинна чітко визначати відповідальність кожного компонента;
- Архітектура повинна ясно визначати правила комунікації для кожного компонента;
- Архітектура повинна визначати основні класи програмного додатка, їх області відповідальності й механізми взаємодії з іншими класами;

Загальні рекомендації до проектування архітектури системи

- Архітектура повинна описувати інші варіанти, що розглядалися, організації класів і обґрунтовувати підсумковий варіант;
- Архітектура повинна описувати основні види формату файлів і таблиць;
- Прямий доступ до даних, зазвичай, слід надавати тільки одній підсистемі або класу; виключення можливі при використанні класів або методів доступу, що забезпечують доступ до даних, що контролюються абстрактним чином;
- Архітектура повинна визначати високорівневу організацію й зміст усіх використовуваних баз даних;
- Архітектура, що залежить від специфічних бізнес-правил, повинна визначати їх і описувати їхній вплив на проект системи;

Загальні рекомендації до проектування архітектури системи

- Архітектура повинна описувати головні елементи формату Веб-сторінок, GUI, інтерфейс командного рядка тощо;
- Архітектура повинна бути модульною, щоб GUI можна було змінити, не заціпивши при цьому бізнес-правил і модулів програми, що відповідають за виведення даних;
- Архітектура повинна включати план керування обмеженими ресурсами, такими як з'єднання з базами даних, потоки та дескриптори;
- Архітектура повинна визначати підхід до безпеки на рівні проекту додатка й на рівні коду;
- Архітектура повинна містити оцінки продуктивності й пояснювати, чому розробники архітектури вважають ці показники досяжними;

Загальні рекомендації до проектування архітектури системи

- Архітектура повинна описувати, як система буде реагувати на зростання числа користувачів, серверів, мережних вузлів, записів у базах даних, транзакцій тощо;
- Якщо деякі дані або ресурси будуть загальними для розроблюваної системи й інших програм або обладнань, в архітектурі потрібно вказати, як це буде реалізовано;
- Архітектура повинна визначати схему зчитування даних: упереджене зчитування, зчитування із затримкою або за вимогою;
- При розробці архітектури системи слід вказати очікуваний рівень її відмовостійкості;
- Архітектура повинна підтверджувати, що система є технічно здійсненною;


Загальні рекомендації до проектування архітектури системи

- Якщо план передбачає застосування існуючого коду, тестів, форматів даних тощо, архітектура повинна пояснювати, як повторно використані ресурси будуть адаптовані до інших архітектурних особливостей, якщо це буде зроблено;
- Архітектура повинна чітко описувати стратегію змін;
- В архітектурі повинні бути відображені стратегії, які дозволяють програмістам не обмежувати наявний у них вибір завчасно;
- «Інтернаціоналізацією» називають реалізацію в програмі підтримки регіональних стандартів. «Локалізацією» називають переклад інтерфейсу програми й реалізацію в ній підтримки конкретної мови. Архітектура повинна пояснювати, який варіант обрано і чому.

Стандартизований підхід до проектування

Розробка автоматизованих систем виконується на основі стандарту ГОСТ 34.601–90, що регламентує стадії та етапи процесу розробки програмних систем з врахуванням їх особливостей і засобів об'єднання підсистем. Даний стандарт забезпечує:

1. *концептуальне проектування, яке полягає в побудові концептуальної моделі, уточненні та узгодженні вимог:*
 - джерела надходження даних від замовника, який несе відповідальність за їх достовірність;
 - об'єкти системи та їх атрибути;
 - види організації даних;
 - інтерфейси з потенційними користувачами системи для надання їм допомоги при формулюванні цілей і функцій системи;
 - методи взаємодії користувачів з системою для забезпечення швидкості реакції системи;



Стандартизований підхід до проектування

2. *архітектурне проектування*, яке полягає у визначенні головних структурних особливостей створюваної системи;
3. *технічне проектування* – це відображення вимог, визначення завдань і принципів їх реалізації в середовищі функціонування системи;
4. *детальне робоче проектування* – полягає у визначенні алгоритмів вирішення завдань, побудові баз даних і програмного забезпечення системи.

Загальносистемний підхід до проектування архітектури

При такому підході передбачають, що створювана архітектура складається з чотирьох рівнів, які містять:

- *системі компоненти*, що встановлюють інтерфейс з обладнанням та апаратурою;
- *загальносистемні компоненти*, що встановлюють інтерфейс з універсальними системами комп'ютерів;
- *специфічні бізнес-компоненти* (компоненти певної проблемної області) – являються складовими компонентами програмних систем і призначені для вирішення різних задач (наприклад, бізнес-задач);
- *прикладні програмні системи* – реалізують конкретні завдання окремих груп споживачів інформації з різних предметних областей (офісні системи, системи бухгалтерського обліку тощо), можуть використовувати компоненти нижніх рівнів.

Загальносистемний підхід до проектування архітектури

Результат проектування – архітектура та інфраструктура, що містить набір об'єктів, з яких можна формувати деякий конкретний вигляд архітектурної схеми для конкретного середовища виконання системи.

Логічна структура проектованої системи – це композиція об'єктів і готових програмних продуктів, що виконують відповідні функції системи. Композиція ґрунтується на наступних положеннях:

- кожна підсистема повинна відображати вимоги та спосіб їх реалізації;
- змінні функції виділяться в підсистеми так, щоб для них прогнозувалися зміни вимог та окремі об'єкти;
- зв'язок об'єктів здійснюється через інтерфейс;
- кожна підсистема повинна виконувати мінімум послуг або функцій і мати фіксовану множину параметрів інтерфейсу.

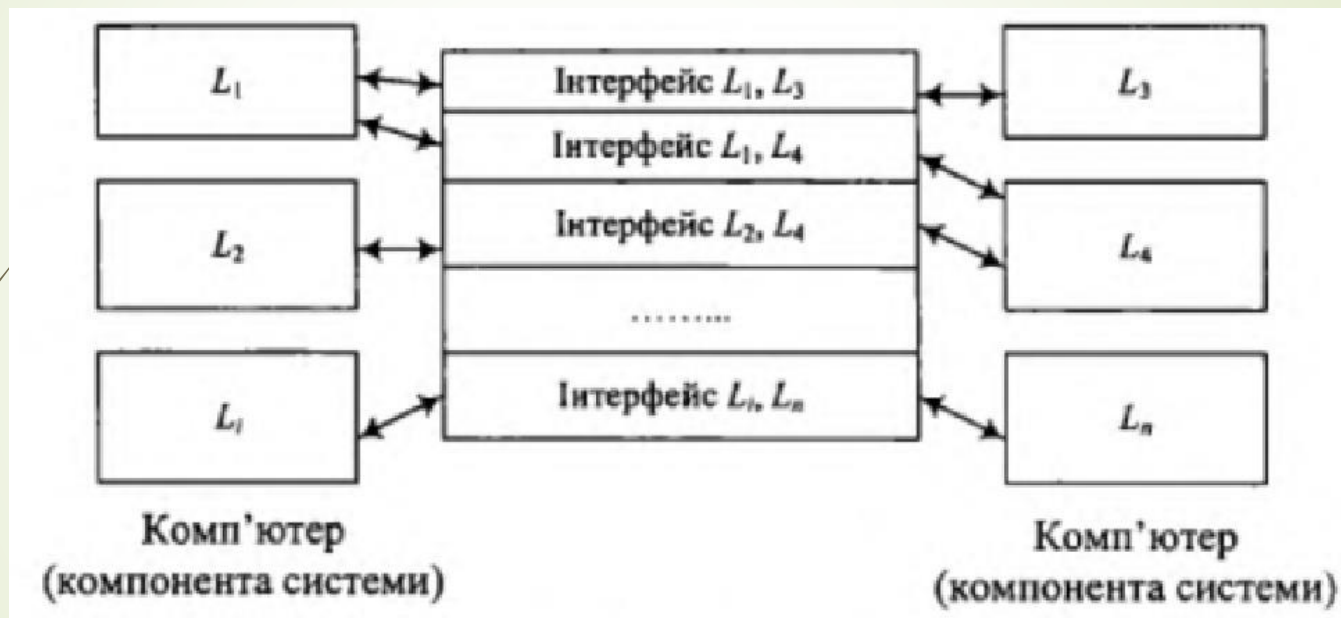


Архітектурна схема може бути:

1. розподілена;
2. клієнт–серверна;
3. багаторівнева.

Розподілена схема забезпечує взаємодію компонентів системи, розташованих на різних комп'ютерах через стандартні механізми виклику RPC (Remote Procedure Calls), RMI (Remote Method Invocation), які реалізуються проміжними середовищами (COM/DCOM, CORBA, Java та ін.). Взаємодіючі компоненти можуть бути неоднорідними, написаними на різних мовах програмування (C, C++, Pascal, Java, Basic, Smalltalk тощо), які допускаються в проміжному середовищі системи CORBA.

Зв'язок між мовами, що реалізують різні компоненти, через інтерфейси

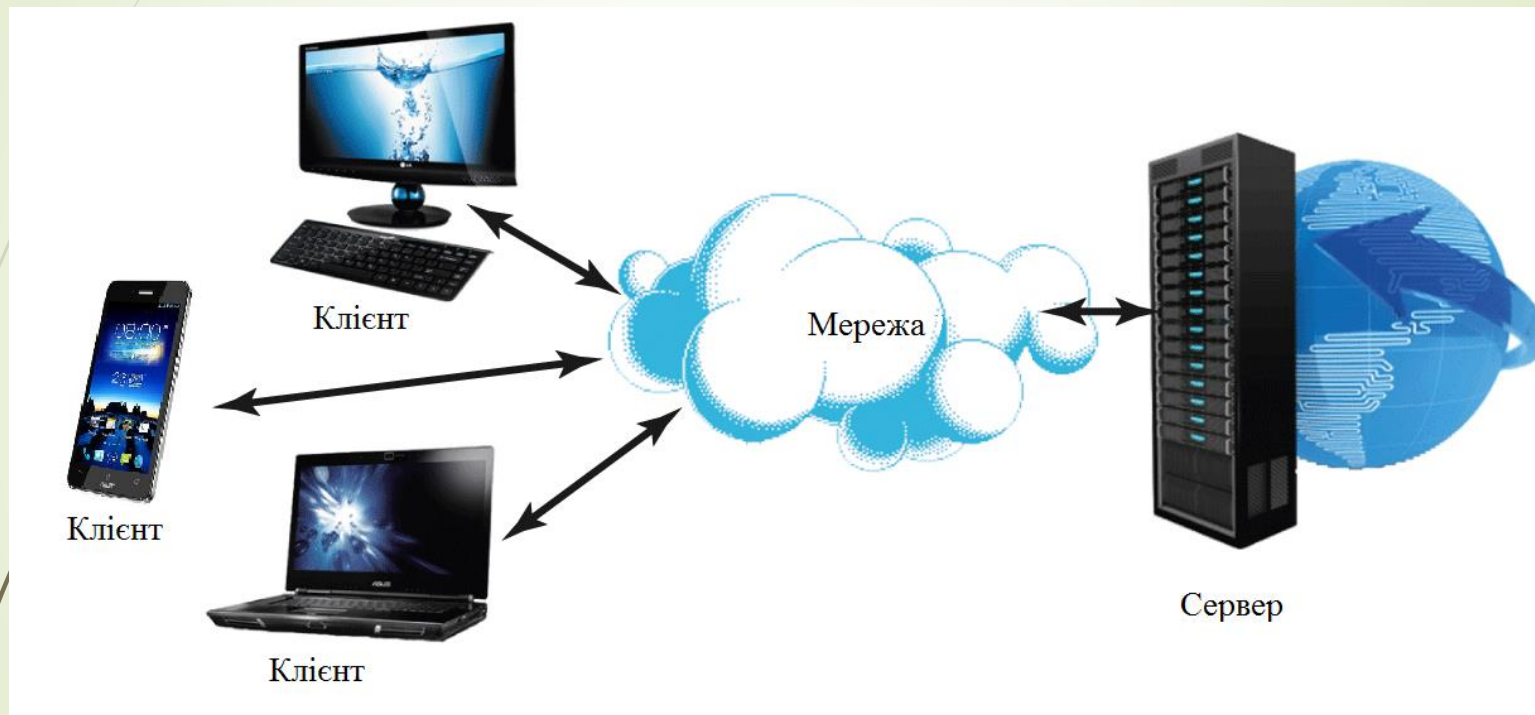


Архітектура «клієнт–сервер»

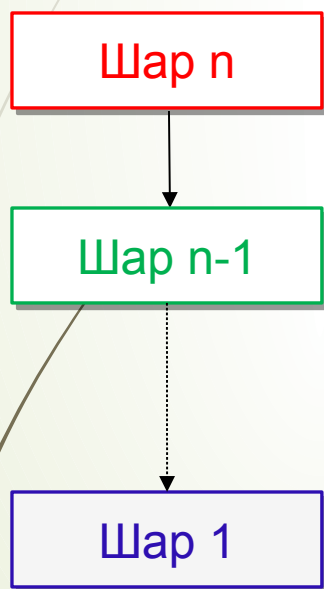
При реалізації *архітектури «клієнт–сервер»* сервер управляє ресурсами та надає до них доступ, а клієнт їх використовує. Архітектура заснована на розподілених об'єктах, які містять ресурси, та видають послуги іншим об'єктам:

1. серверна частина (зберігання та обробка інформації);
2. клієнтська частина (робочий інструмент користувача);
3. мережа, яка забезпечує взаємодію (обмін інформацією) між клієнтом і сервером.

Схема «клієнт–сервер»

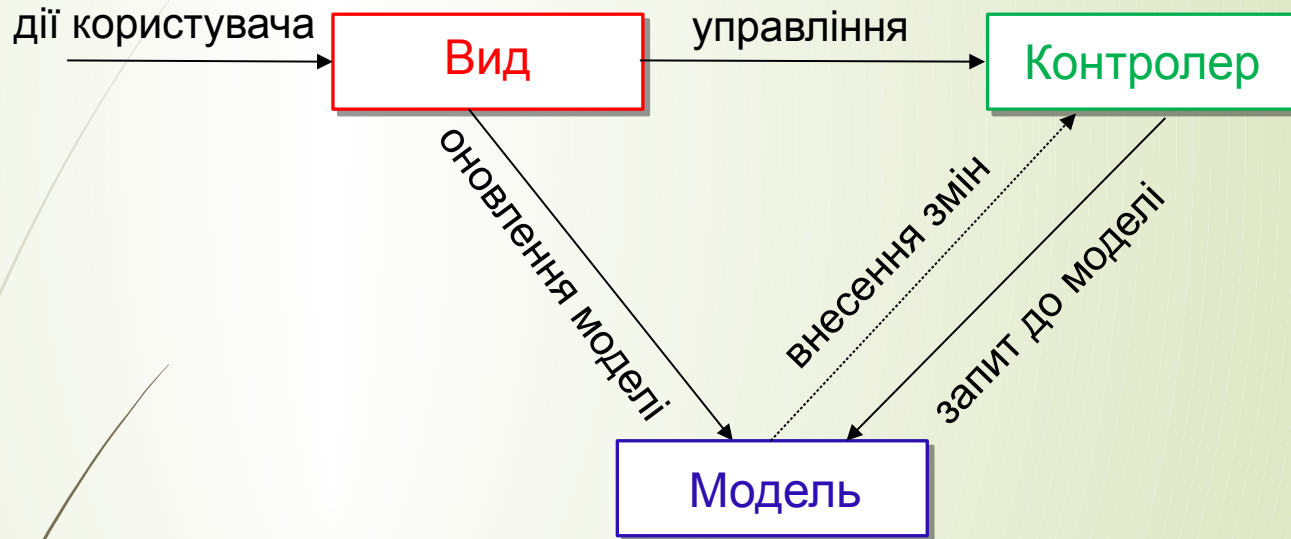


Багаторівневий шаблон



Цей шаблон використовується для структурування програм, які можна розкласти на групи деяких підзадач, що перебувають на певних рівнях абстракції. Кожен шар надає служби для наступного, більш високого шару.

Шаблон MVC

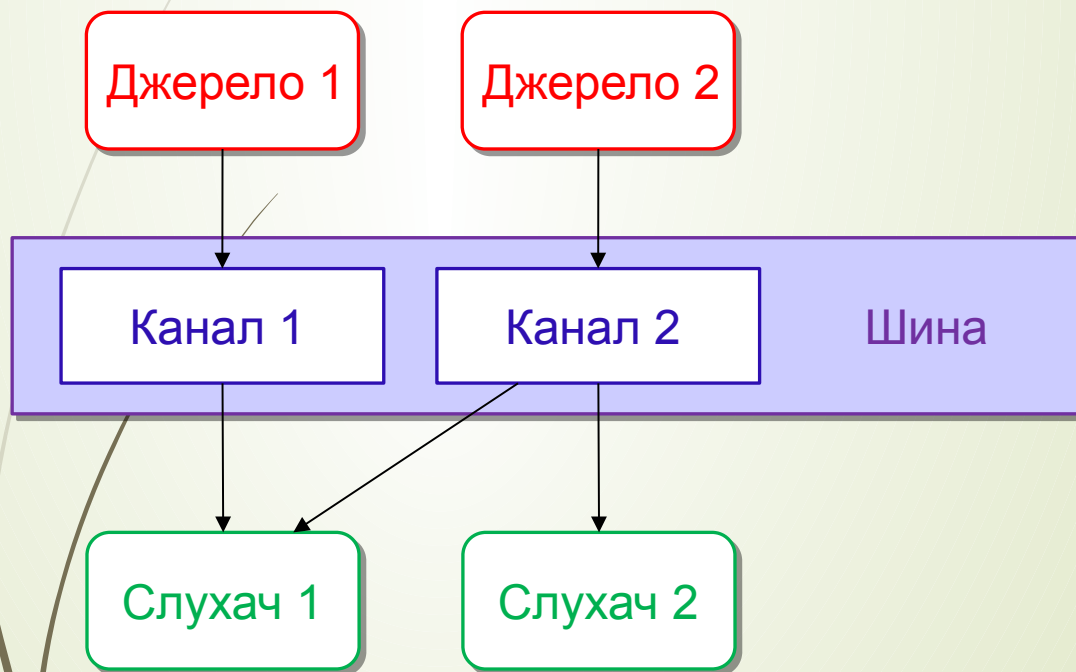


Він розділяє інтерактивні прикладні програми на 3 частини:

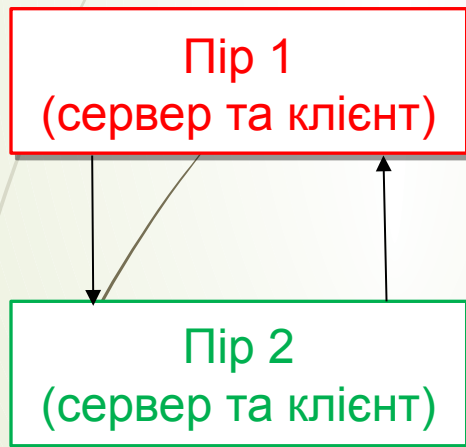
1. модель - містить ключові дані і функціонал;
2. вид - показує інформацію користувачеві;
3. контролер - займається обробкою даних від користувача.

Це робиться з метою розмежування внутрішнього представлення інформації від способів її подання і приймання від користувача. Ця схема ізолює компоненти і дозволяє ефективно реалізувати повторне використання коду.

Шаблон шина подій



Цей шаблон, в основному, взаємодіє з подіями і складається з 4 основних компонентів: джерело події, прослуховувач події, канал і шина подій. Джерела розміщують повідомлення для певних каналів на шині подій. Прослуховувачі підписуються на певні канали. Прослуховувачі отримують повідомлення про появу повідомлень, розміщених на каналах з їх підписки.

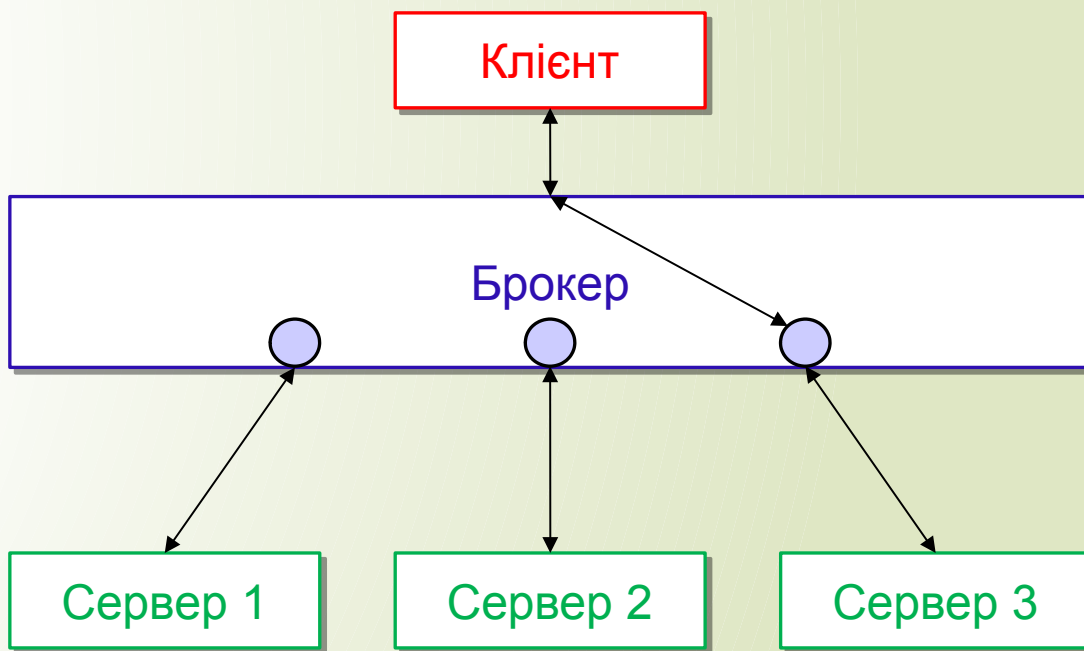


У даному шаблоні існують окремі компоненти, так звані піри. Піри можуть виступати в ролі як клієнта, що запитує послуги від інших рівноправних учасників (теж пірів), так і сервера, що надає послуги іншим пірам. Пір може бути клієнтом або сервером, або всім відразу, а також здатний з часом динамічно змінювати свою роль.

Шаблон «брокер»

Цей шаблон потрібен для структуризації розподілених систем з незв'язними компонентами. Ці компоненти можуть взаємодіяти один з одним через віддалений виклик служби. Компонент посередник відповідає за координацію взаємодії компонентів.

Сервер розміщує свої можливості (служби та характеристики) у посередника (брокера). Клієнт запитує послугу у брокера. Потім брокер перенаправляє клієнта до підходящої служби зі свого реєстру.

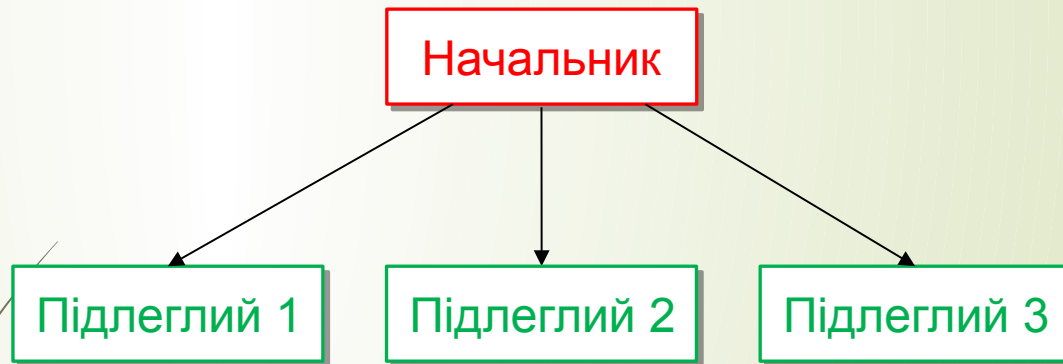


Шаблон «потік-фільтр»



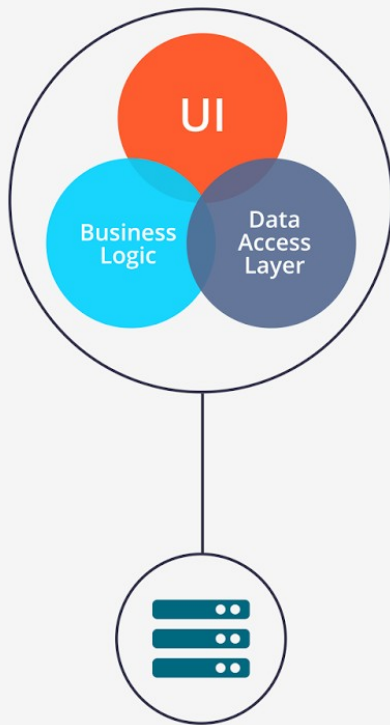
Цей шаблон підходить для систем, які виробляють і обробляють потоки даних. Кожен етап обробки відбувається всередині якогось компонента - фільтра. Дані для обробки передаються через канали. Ці канали можна використовувати для буферизації або синхронізації даних.

Шаблон «MS»

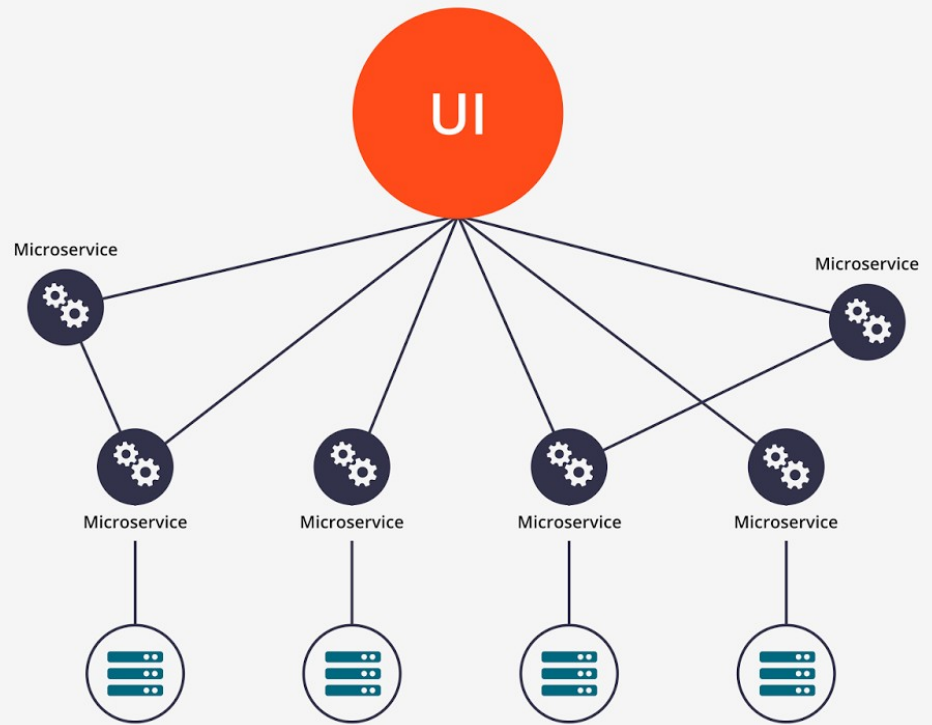


У цьому шаблоні також задіяні два учасники - ведучий і ведені. Ведучий компонент розподіляє завдання серед ідентичних ведених компонентів і обчислює підсумковий результат на підставі результатів, отриманих від своїх «підлеглих».

Мікросервіси



Monolithic Architecture



Microservice Architecture

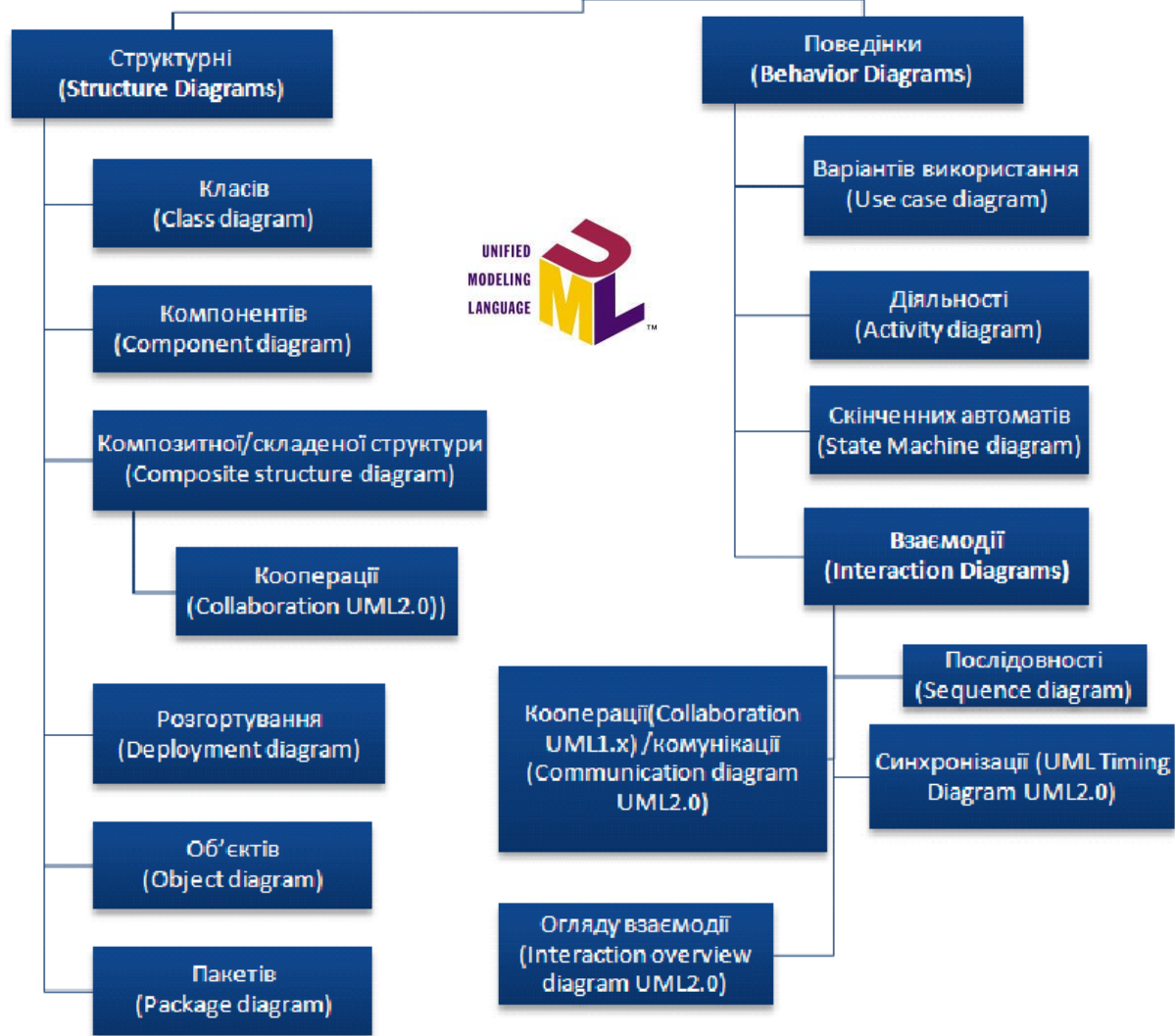
Проектування за допомогою



UML (уніфікована мова моделювання) – є графічною мовою для візуалізації, опису параметрів, конструювання й документування різних систем, зокрема і програм. Основними типами діаграм для візуалізації моделі є:

- діаграма варіантів використання (use case diagram)
- діаграма класів (class diagram)
- діаграма станів (statechart diagram)
- діаграма послідовності (sequence diagram)
- діаграма кооперації (collaboration diagram)
- діаграма компонентів (component diagram)
- діаграма розгортання (deployment diagram)

Діаграми UML



Зв'язки в



Для зв'язування сутностей у моделях UML визначено чотири типи *відносин*.

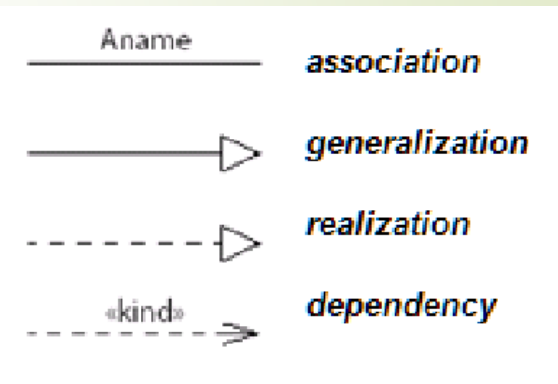
Асоціація (association) – структурне відношення, що описує сукупність змістовних або логічних зв'язків між об'єктами.

Узагальнення (generalization) – це відношення, при якому об'єкт–нащадок (child) може бути підставлений замість об'єкта–батька (parent). При цьому відповідно до принципів об'єктно–орієнтованого програмування нащадок успадковує структуру і поведінку свого батька.

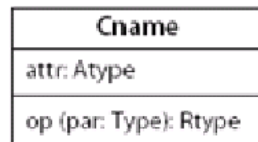
Реалізація (realization) є семантичним відношенням між класифікаторами, при якому один класифікатор визначає зобов'язання, а інший гарантує його виконання. Відношення реалізації спостерігаються у двох випадках:

- між інтерфейсами, реалізують класами або компонентами;
- між варіантами використання, реалізують кооперації.

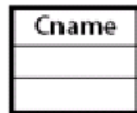
Залежність (dependency) – це семантичне відношення між двома сутностями, при якому зміна однієї з них, незалежної, може вплинути на семантику іншої, залежної.



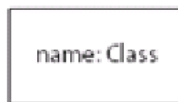
Сутності в



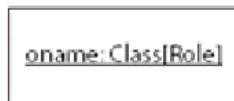
клас
class



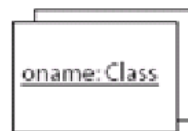
активний клас
active class



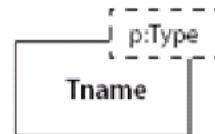
роль
role



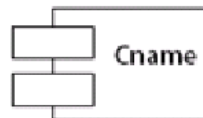
об'єкт
object



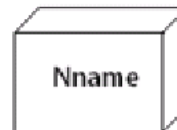
мультимоб'єкт
multiobject



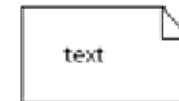
шаблон
template



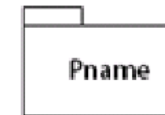
компонент
component



вузол
node



примітка
note



пакет
package



інтерфейс
interface

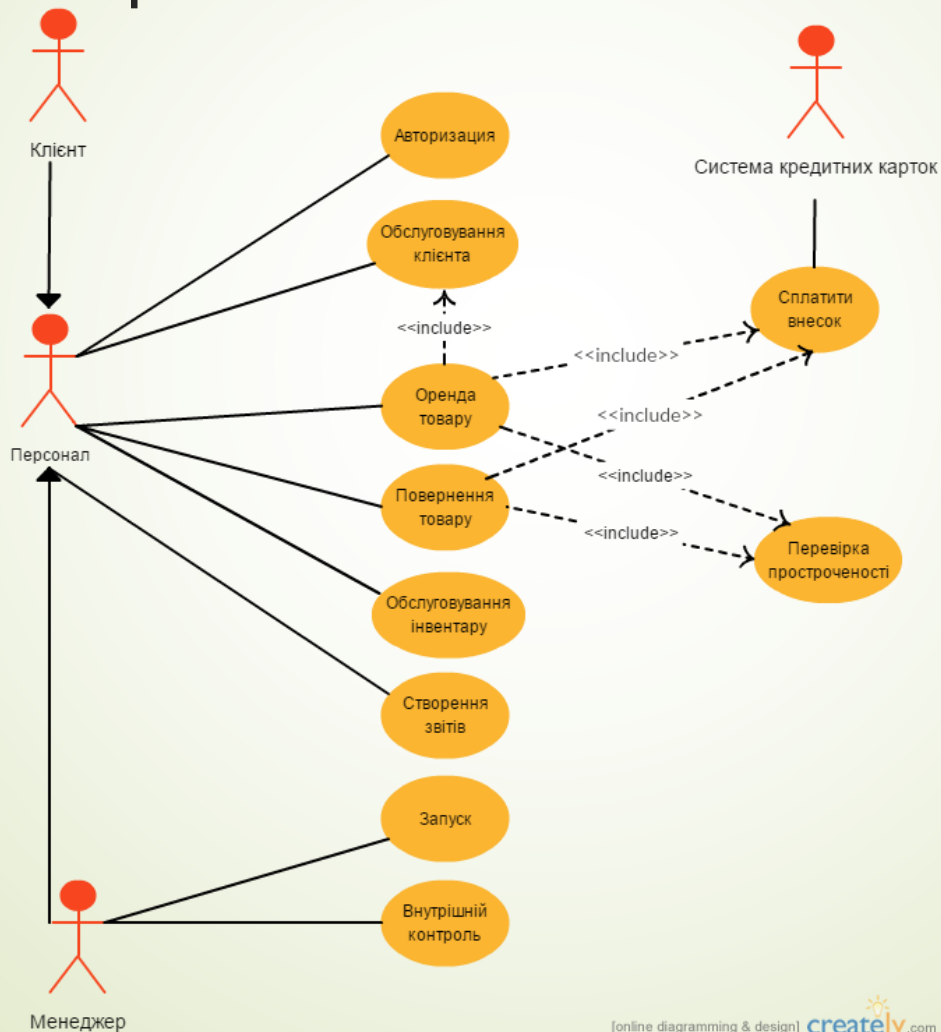


взаємодія
collaboration

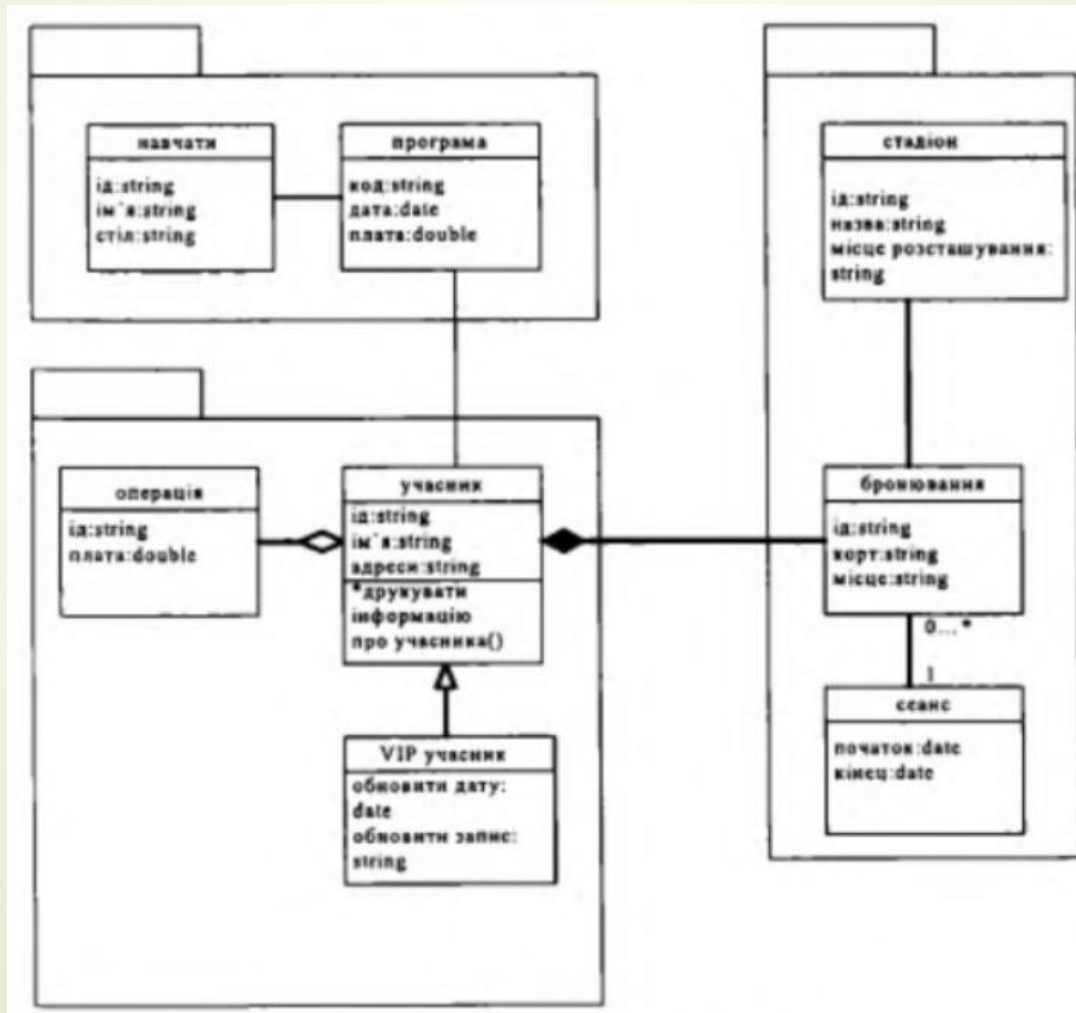
{ expression }

розширення
constraint

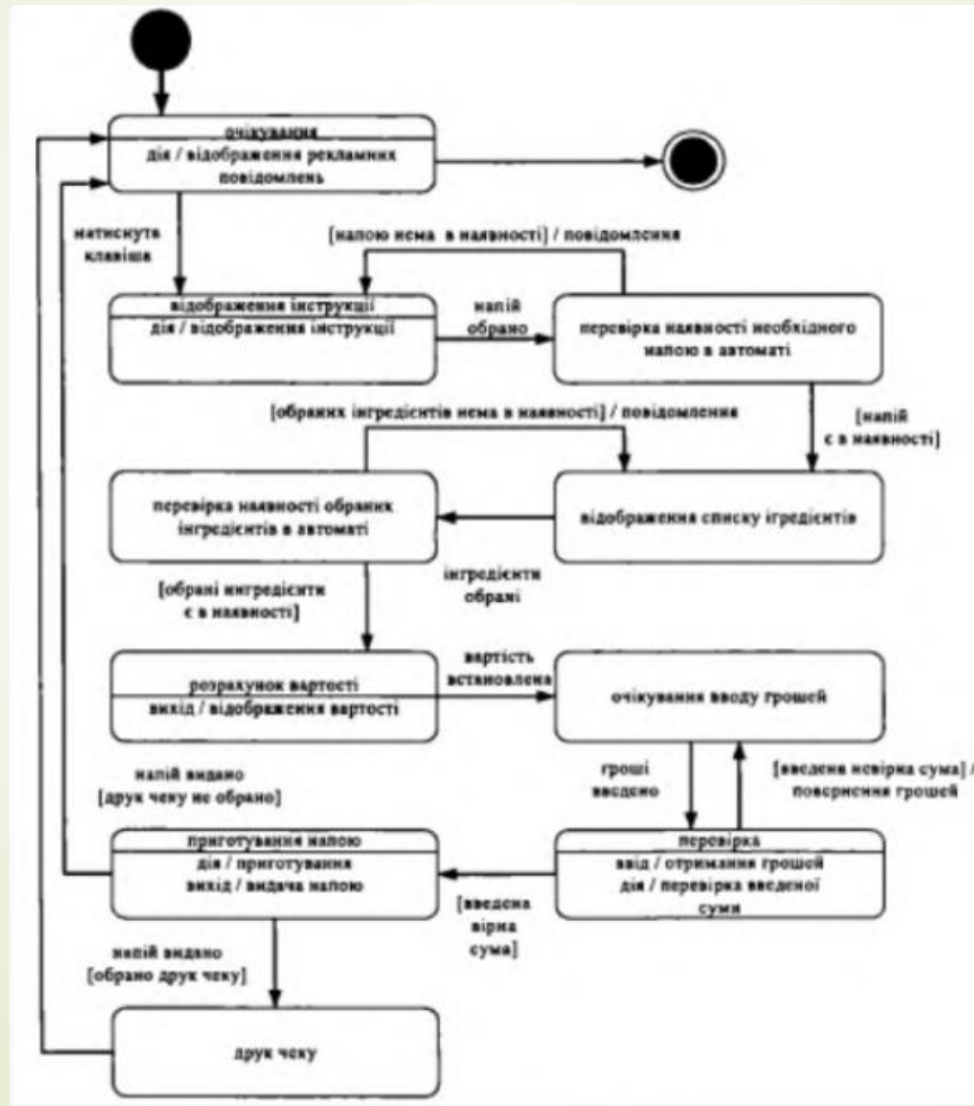
Діаграма варіантів використання



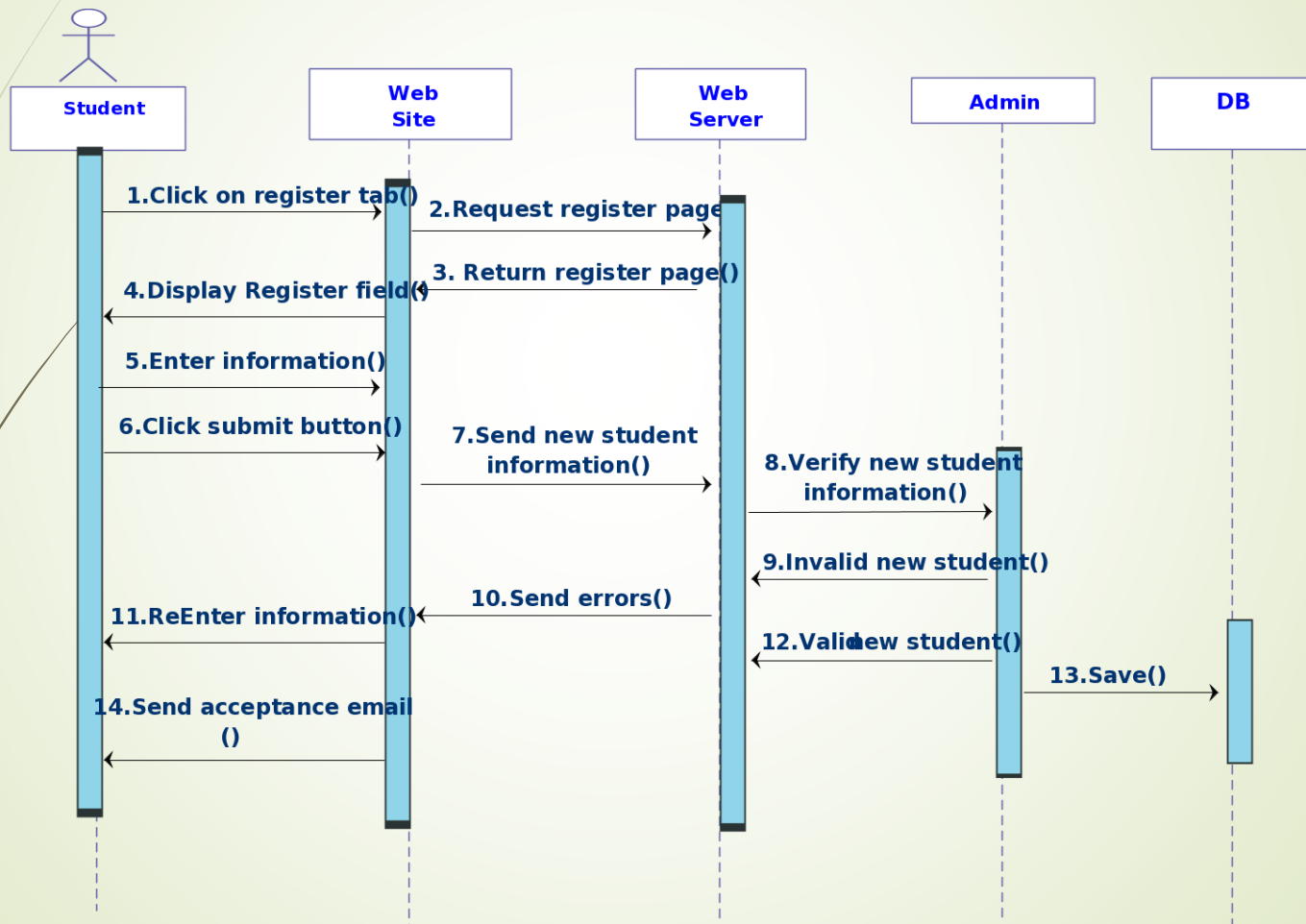
Діаграма класів



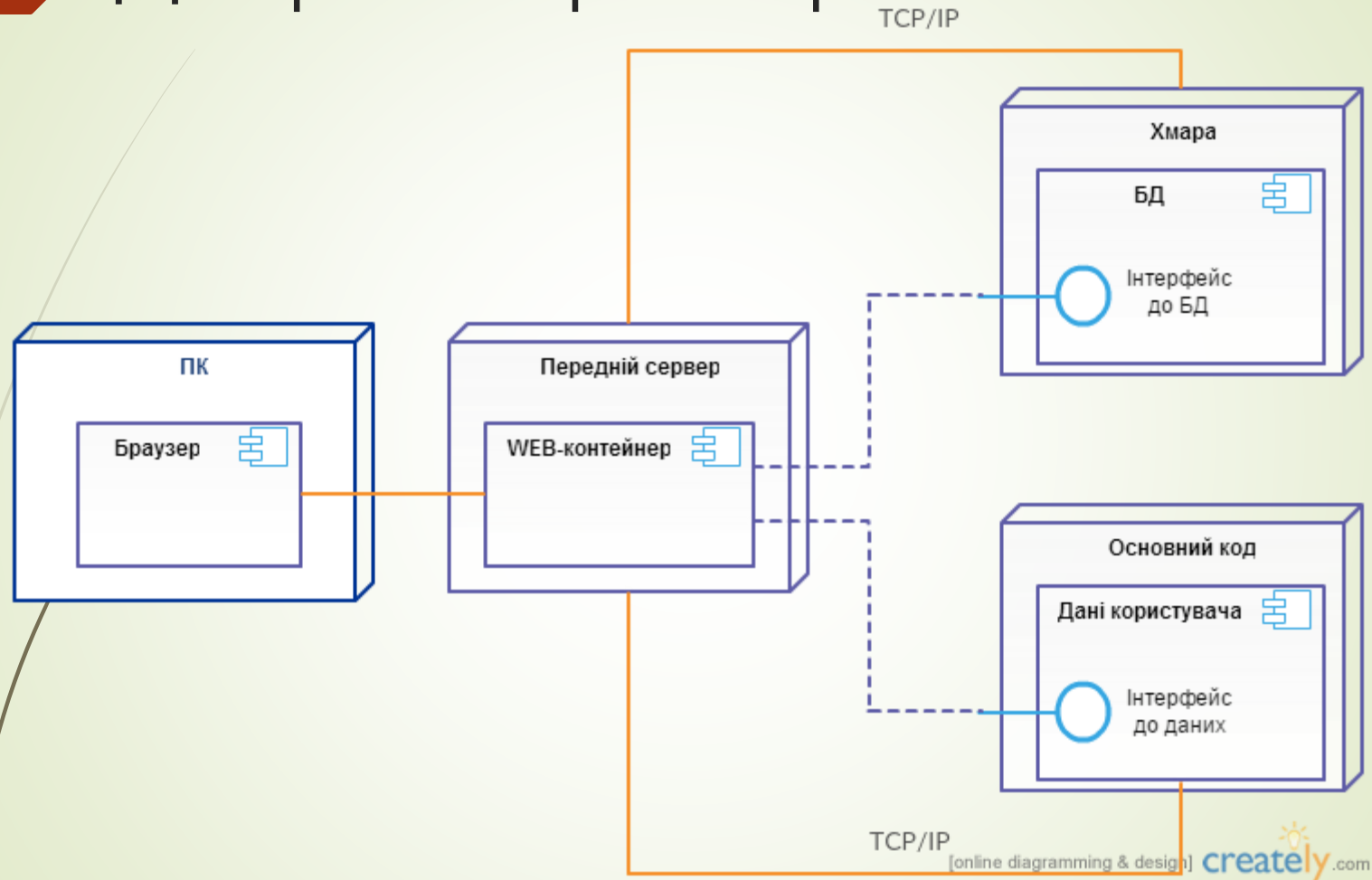
Діаграма станів



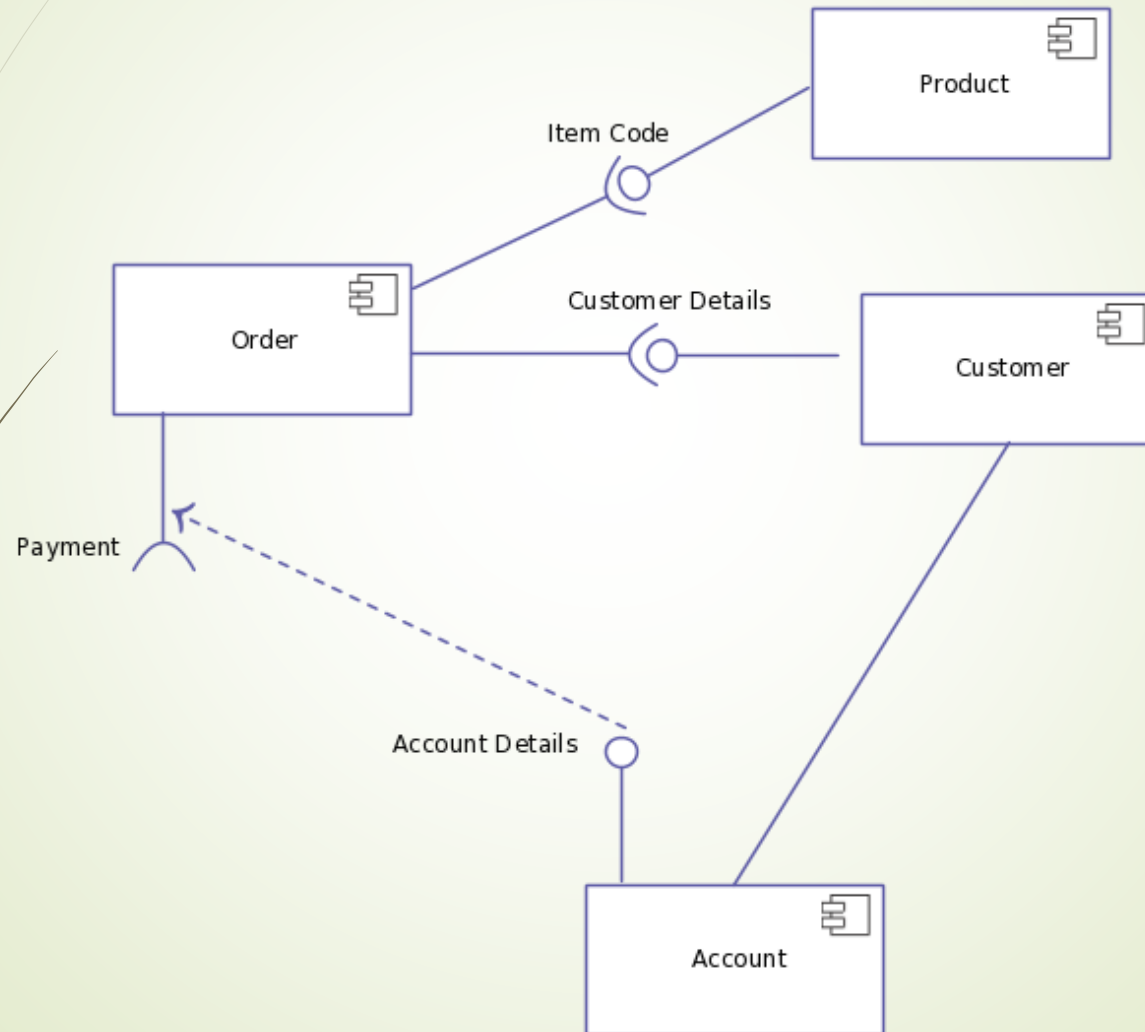
Діаграма послідовності



Діаграма розгортання



Діаграма компонентів



Діаграма кооперації

