

Programació

UT10.2. Patró MVC

El que ja saps

- Un programa rep dades (per mitjà d'un usuari, fitxer o BD) els processa i presenta uns resultats per pantalla o els emmagatzema.
- L'usuari introdueix les dades i el programa les presenta mitjançant una interfície gràfica o bé per consola.
 - En una aplicació hi ha que distingir entre:
 - La **Interfície Gràfica de Usuari(GUI)**
 - El processat de dades
 - El emmagatzematge de els dades

Interfície gràfica de usuari (GUI)

- Un bon disseny de la GUI és vital.
- Deu fer que el programa siga:
 - *Intuïtiu*
 - *Fàcil d'utilitzar*
- **Un programa amb un processat de dades òptim però incòmode serà rebutjat per l'usuari**
- **En el nostre cas, la vista la componen un conjunt de pàgines HTML / JSP**



USABILITAT

Processat de dades

- És important independitzar el processat del GUI.
- El processat de dades deu ser eficaç i eficient.
- Un programa amb un bon GUI però que processa malament les dades serà també rebutjat.

- En el nostre cas, el **controlador de l'aplicació** seran els **Servlets**



Estructura de les dades

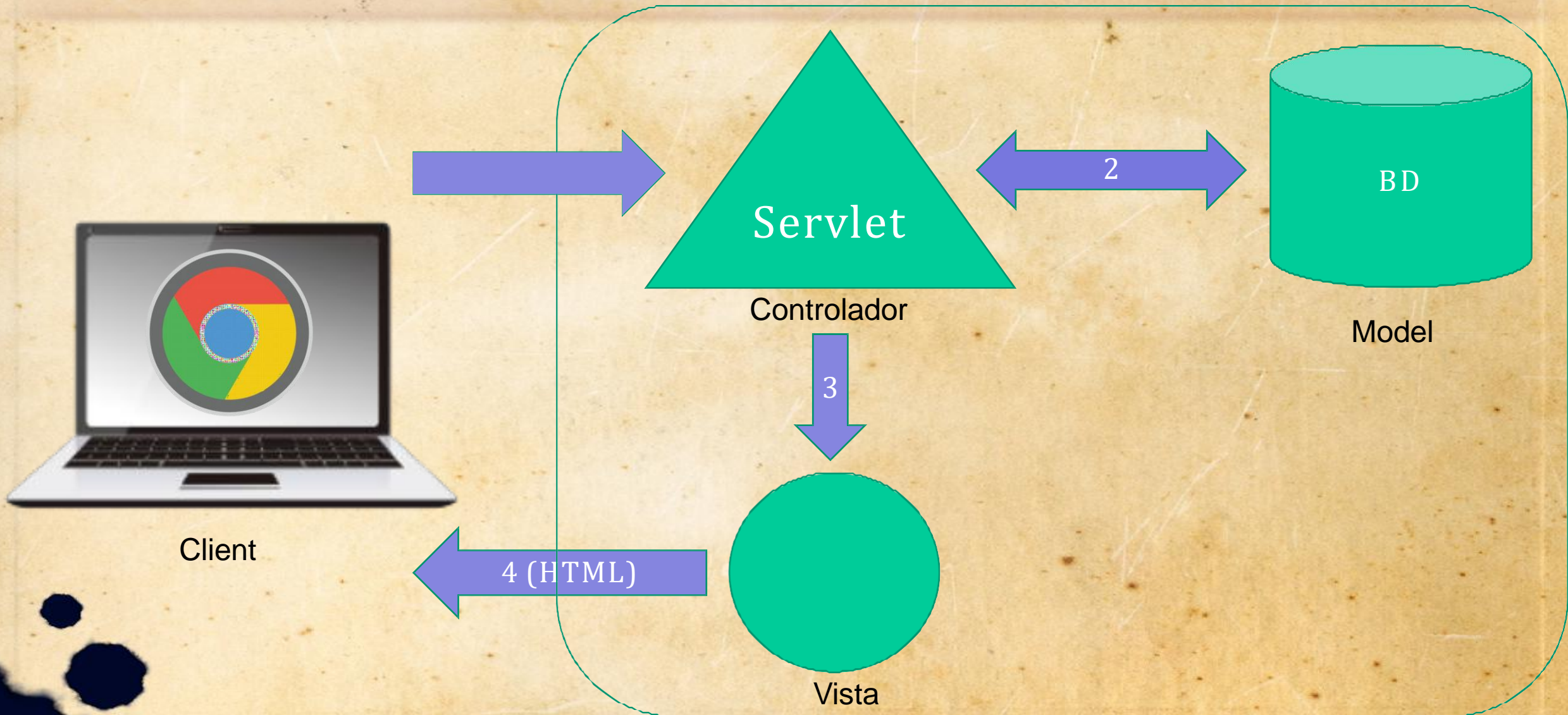
- El programa ha de ser capaç d'emmagatzemar directa o indirectament la informació.
- Diferents suports d'emmagatzematge com ara fitxers o bases de dades.
- Avantatges de l'emmagatzematge en BD
 - Millora l'organització.
 - Facilita la manipulació.
 - Integritat de les dades.



Arquitectura Model Vista Controlador (MVC)

- És un patró de disseny a l'hora de crear aplicacions dinàmiques.
- Encara que no és obligatori seguir-lo, donada la seua claredat, és el patró més usual de desenvolupament d'aplicacions web.
- Es basa en la separació de la part lògica, la part visual i l'accés a les dades d'una aplicació.

Esquema MVC



Passos per el desenvolupament

- No hem d'oblidar que en tot cicle de vida d'una aplicació les fases són:
 - Anàlisi
 - Disseny
 - Implementació
 - Proves
 - Manteniment
- En la fase de **Disseny**, s'hauria de tenir clar quin és el **Model (BD)** i la **Vista** de l'aplicació (normalment recolzat sobre un document d'especificació que naix de l'**Anàlisi**).

Model de l'aplicació

- La base de dades és el pilar fonamental.
- No només la pròpia **base de dades** ho conforma.
- Una aplicació Java basada a POO farà ús d'objectes (Beans) que emmagatzemaran o recuperaran les dades d'aquesta. El més habitual és crear **una classe per cada taula**.
- Per exemple:
 - Classe Client (a la BD hi haurà una taula itv_clients)
 - Classe Vehicle
 - ...etc...

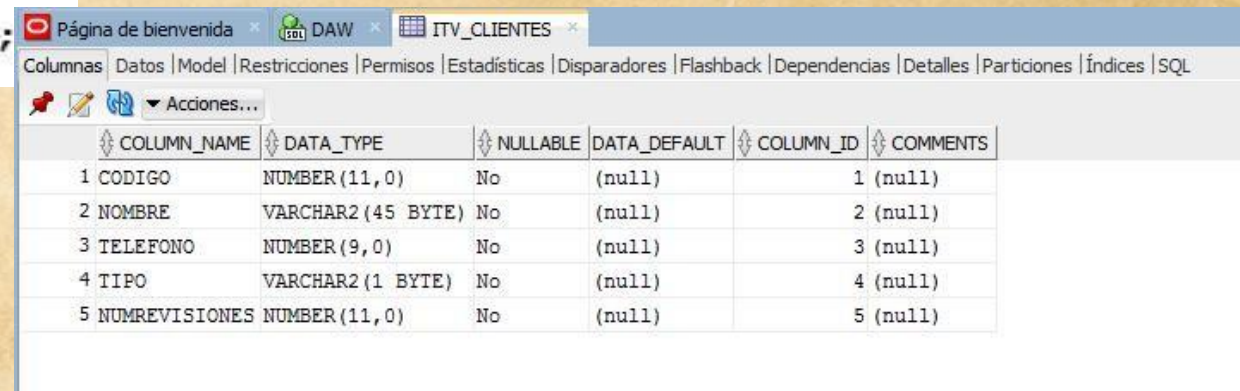
Model de l'aplicació - Beans

- Normalment dites classes contindran un atribut per cada columna de la taula de la base de dades.

```
public class Cliente {  
  
    private int codigo;  
    private String nombre;  
    private int telefono;  
    private TipoCliente tipo;  
    private int numRevisionesRealizadas;  
}
```

En certs camps, caldrà adaptar expressament la informació emmagatzemada a la base de dades amb el tipus de dades de Java

- S'encarreguen d'emmagatzemar o rebre la informació directa de la BD



Columnas	Datos	Model	Restricciones	Permisos	Estadísticas	Disparadores	Flashback	Dependencias	Detalles	Particiones	Índices	SQL
Acciones...												
COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS							
1 CODIGO	NUMBER(11,0)	No	(null)	1 (null)								
2 NOMBRE	VARCHAR2(45 BYTE)	No	(null)	2 (null)								
3 TELEFONO	NUMBER(9,0)	No	(null)	3 (null)								
4 TIPO	VARCHAR2(1 BYTE)	No	(null)	4 (null)								
5 NUMREVISIONES	NUMBER(11,0)	No	(null)	5 (null)								

Model de l'aplicació

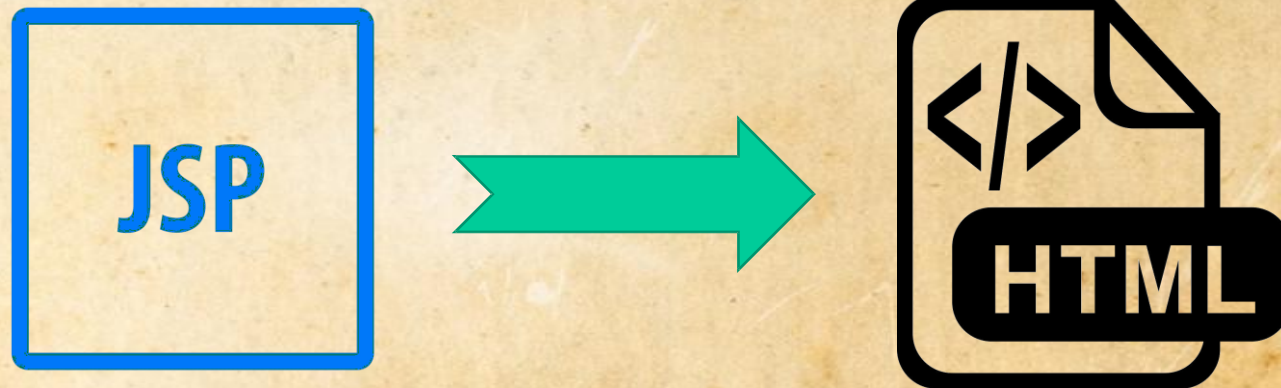
- A més, seran també part del model de l'aplicació aquelles classes que establiran contacte amb la base de dades a través de consultes SQL i que faran ús dels Beans esmentats anteriorment.
- Per exemple:
 - Classe ClientDAO (consultarà/escriurà a itv_clients)
 - Classe VehicleDAO
 - etc...
- Dites classes **consultaran contínuament a la base de dades** amb qualsevol petició d'informació.

Controlador de l'aplicació

- En este cas, les classes que formaran part del controlador de l'aplicació seran principalment els **Servlets**.
- Si el codi del Servlet quedara molt extens o es necessitara reutilitzar codi es podrien fer ús de classes addicionals que formarien també part del controlador.

Vista de l'aplicació

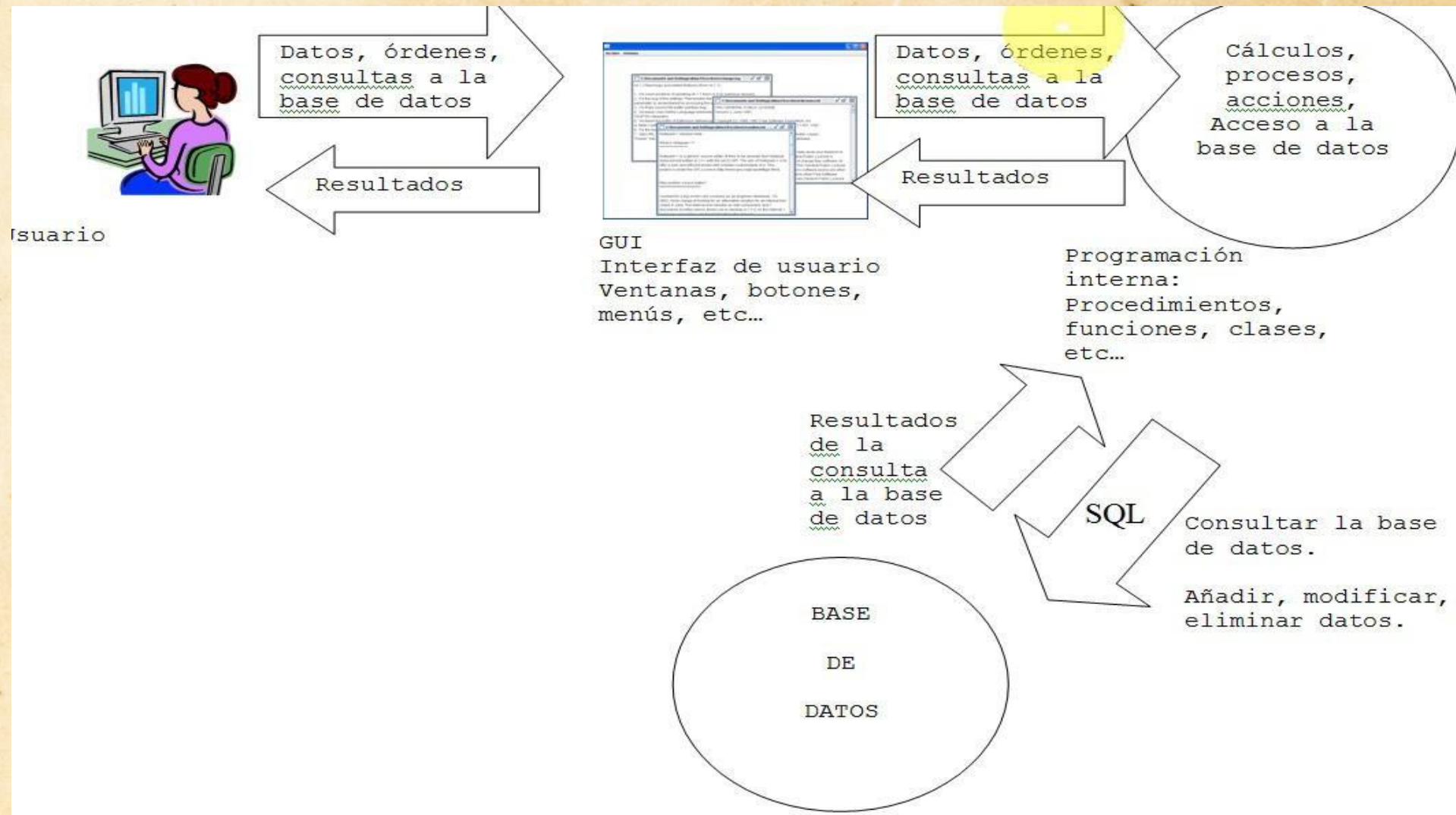
- En una aplicació web la vista la conformaran les pàgines HTML
- En el nostre cas, les pàgines JSP acabaran generant les pàgines HTML.



Reorganització de classes

- Per tenir molt més organitzat el codi, quan una aplicació es basa en este patró, és important reorganitzar les classes en **tres paquets** fonamentals:
 - Model
 - Vista
 - Controlador

Esquema d'interacció d'una aplicació web



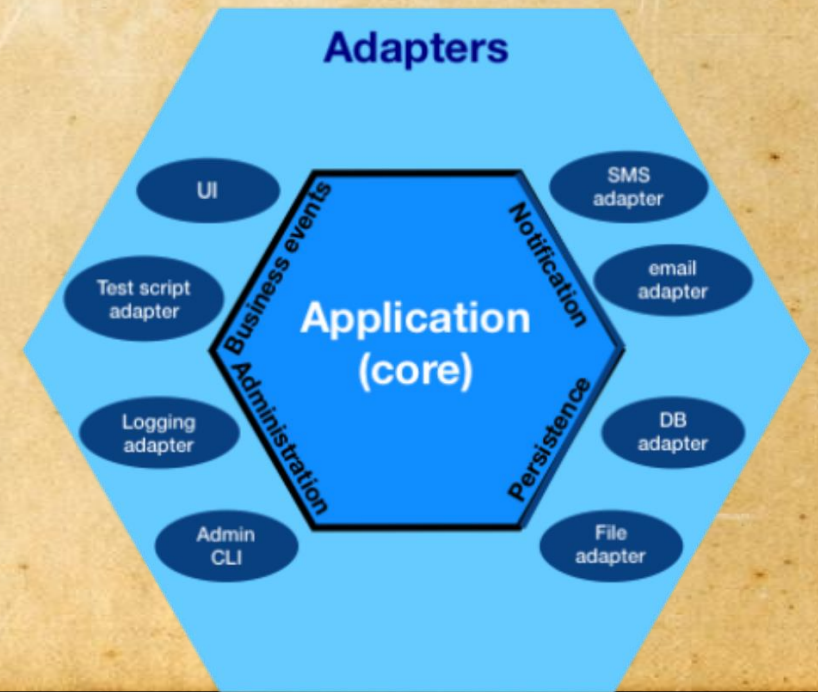
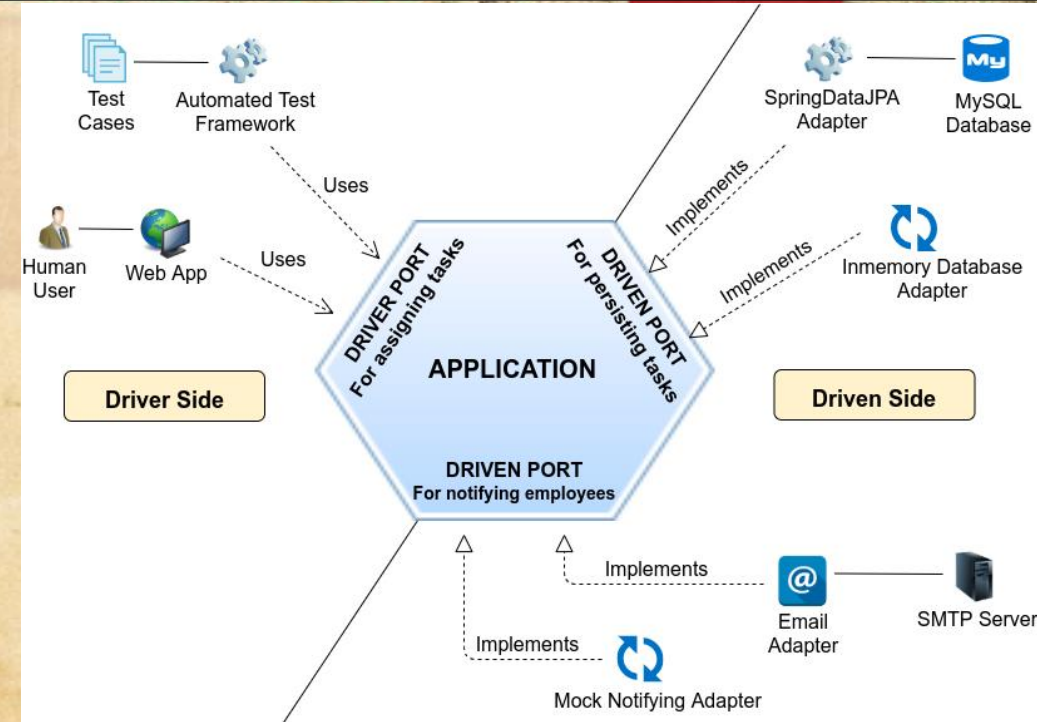


Altres architectures (extra)



Arquitectura hexagonal (ports i adaptadors)

L'Arquitectura Hexagonal, també coneguda com a “Ports and Adapters” dividix les classes d'un sistema en dos grups principal: **Classes de domini**, directament relacionades amb la lògica de negoci; i **Classes externes**, relacionades amb la interfície d'usuari i integració amb sistemes externs.

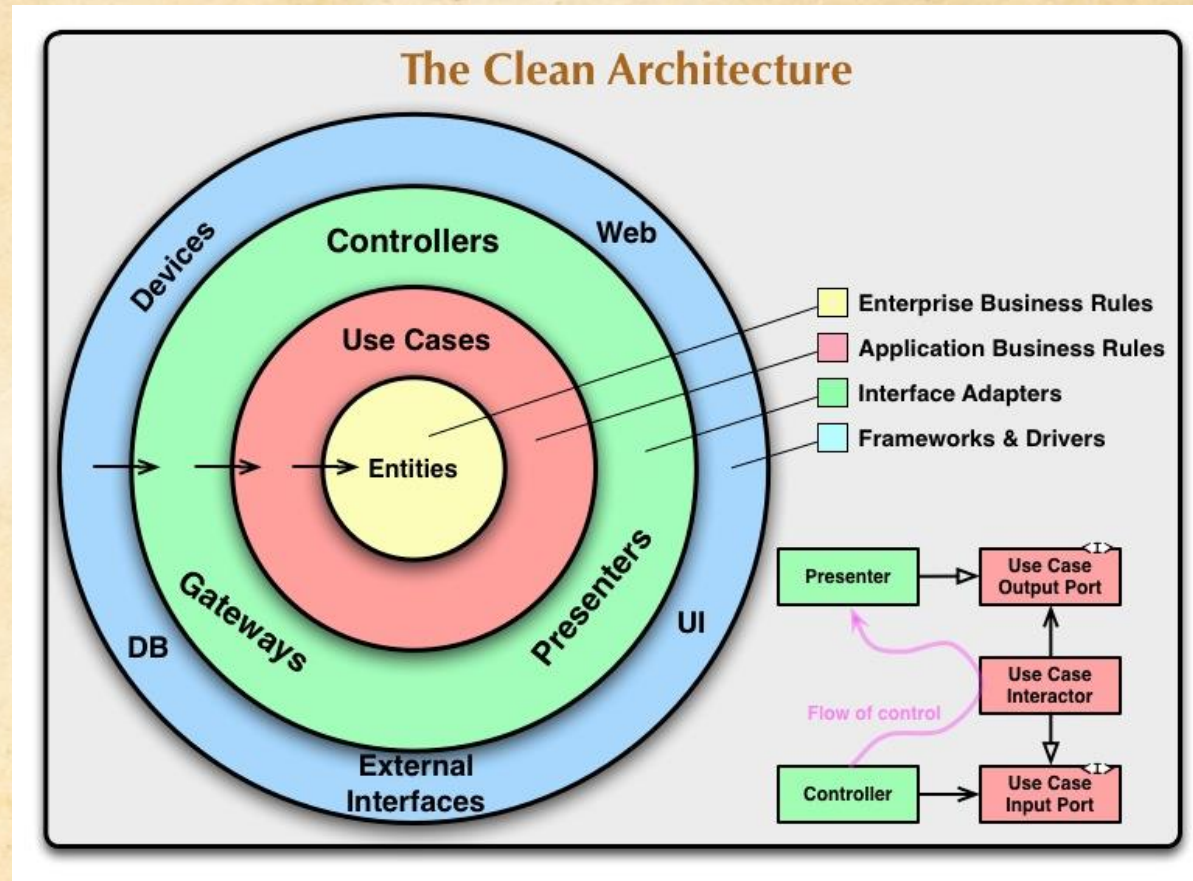


Arquitectura neta (clean architecture)

L'arquitectura neta combina els principis de l'arquitectura hexagonal, l'arquitectura de ceba i altres variants. Proporciona nivells addicionals de detall del component, que es presenten com a anells concèntrics. Aïlla adaptadors i interfícies amb la regla estricta que **les dependències només han d'existir des d'un anell extern cap a un anell intern i mai al revés.**

També segueix el “**Domain Driven Design**” que consisteix a entendre i modelar el domini del problema que estàs intentant resoldre. Un domini és simplement l'àrea específica de coneixement o activitat en la qual s'aplicarà el programari. Per exemple, en un sistema de gestió d'una biblioteca, el domini seria la gestió de llibres, préstecs, usuaris, etc.

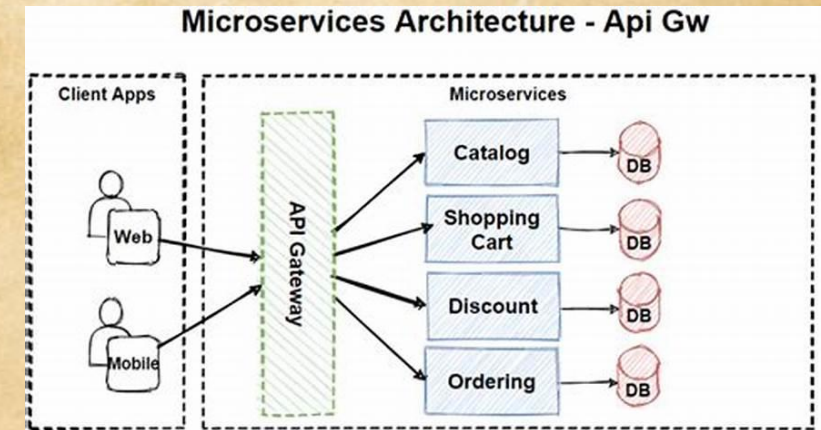
Arquitectura neta (clean architecture)



Microservicis

Els microservicis són un conjunt d'aplicacions de programari dissenyades amb un abast limitat que treballen juntes per a formar una solució més gran. Cada microservici té capacitats mínimes per a crear una arquitectura molt modularitzada. Característiques principals dels microservicis:

- Són xicotets i independents
- Poden desplegar-se i escalar independentment
- Es comuniquen mitjançant APIs ben definides
- Suporten programació políglota



Components típics d'una arquitectura de microservicis:

- **Gestió/orquestració:** Responsable de col·locar servicis en nodes (normalment Kubernetes)
- **API Gateway:** Punt d'entrada per als clients, desacobla clients de servicis

Arquitectura Basada en Esdeveniments (EDA)

L'arquitectura dirigida per esdeveniments està dissenyada per a orquestrar el comportament entorn de la **producció, detecció, consum i reacció a esdeveniments**. Esta arquitectura permet interconnexions altament desacobrades, escalables i dinàmiques entre productors i consumidors d'esdeveniments.

Components clau de EDA:

Productors d'esdeveniments:

Generen esdeveniments

Encaminadors d'esdeveniments:

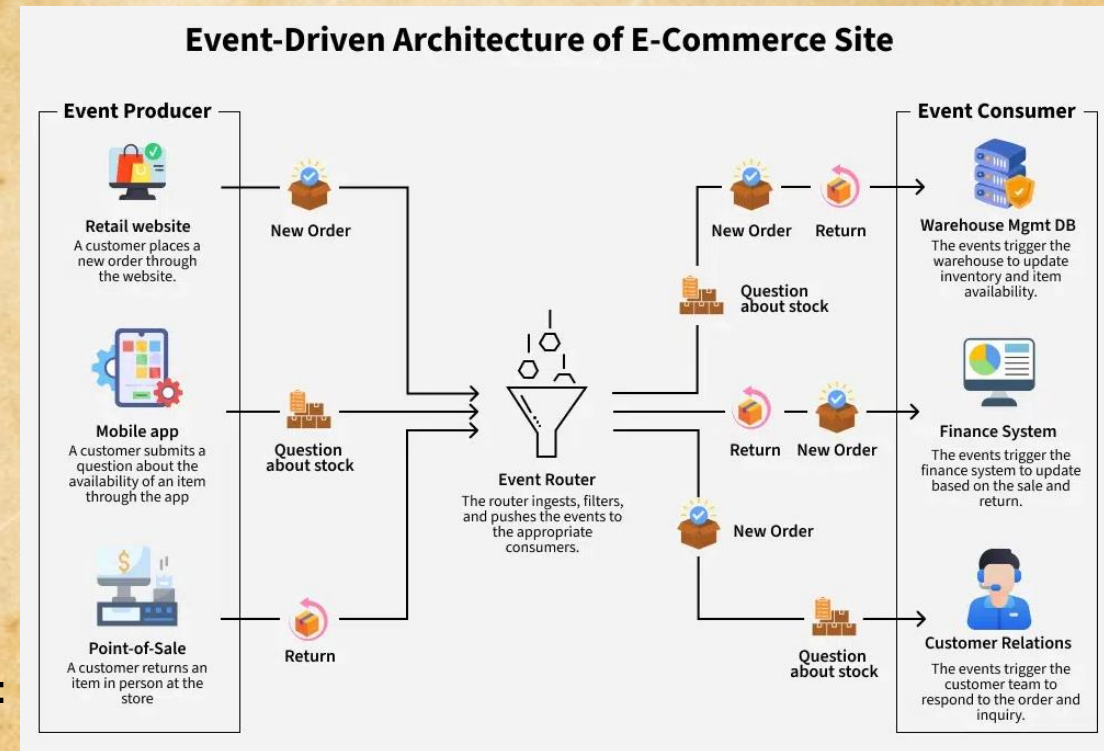
Filtren i envien esdeveniments

Consumidors d'esdeveniments:

Responen als esdeveniments

Principals encaminadors (message brokers):

Apache Kafka i RabbitMQ



Serverless

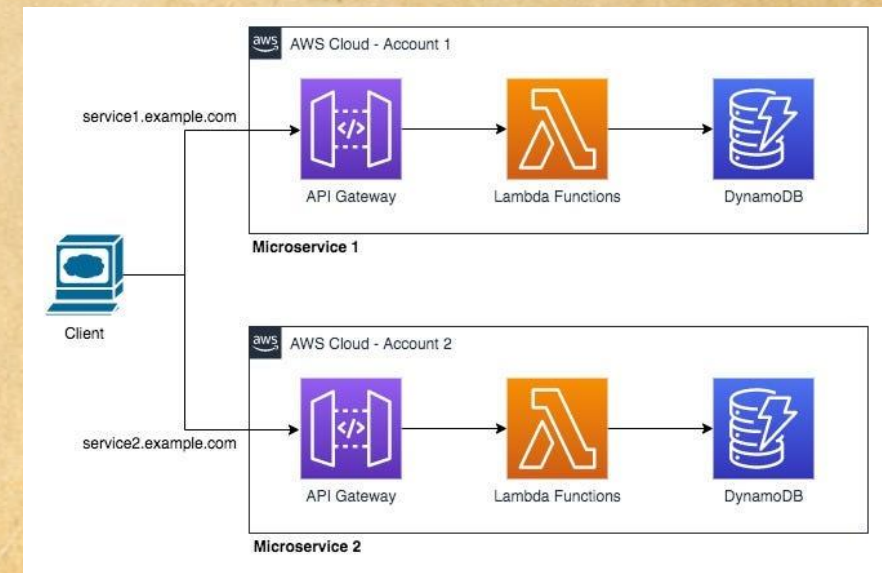
Permet als desenrotlladors crear i executar aplicacions sense gestionar la infraestructura subjacent. El proveïdor del núvol maneja tota la configuració, manteniment i escalat de la infraestructura.

Característiques principals:

- Basat en esdeveniments: les funcions s'activen per esdeveniments com a sol·licituds HTTP o càrregues d'arxius
- Model de pagament per ús
- Escalat automàtic segons la demanda
- Sense gestió de servidors

Plataformes populars:

- AWS Lambda
- Google Cloud Functions
- Azure Functions



Arquitectura reactiva

Proposa un disseny per a construir sistemes de programari sensibles, resistents i escalables que puguin fer front a reptes moderns com ara **l'alta concurrència, el processament de dades en temps real i els entorns informàtics distribuïts**. Abraça principis com la capacitat de resposta, la resiliència, l'elasticitat i la comunicació basada en missatges per garantir que els sistemes puguin reaccionar als canvis i als errors de manera eficaç.

