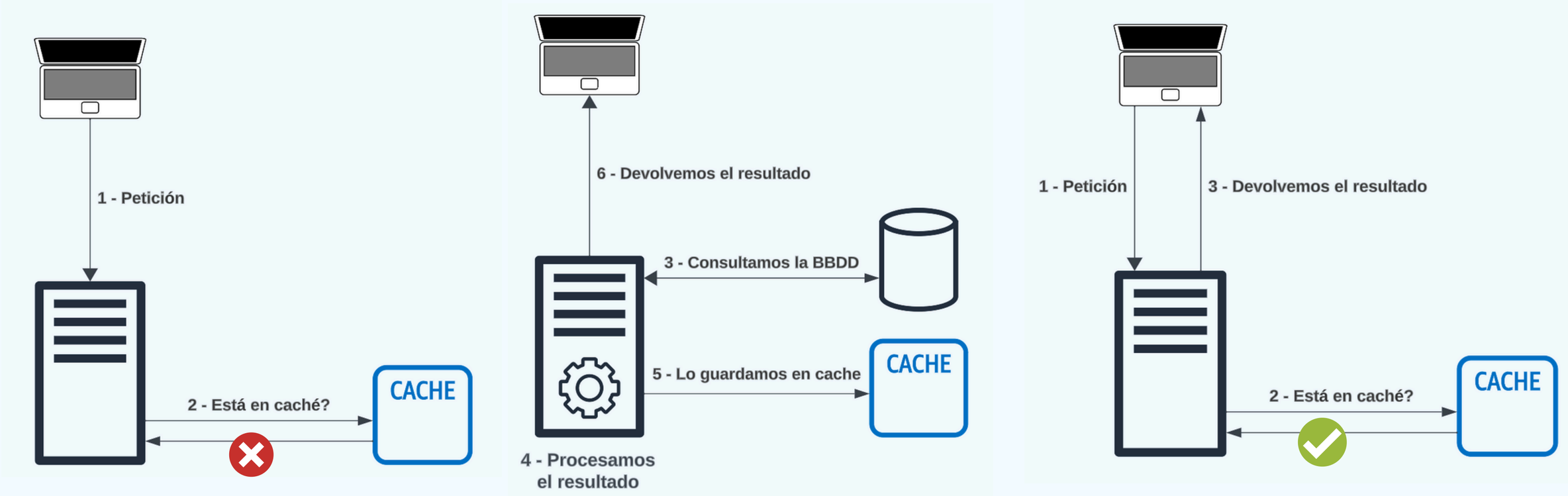
CONCEPTOS CLAVE DISEÑO SISTEMAS A GRAN ESCALA

CACHÉ

Daniel Blanco Calviño

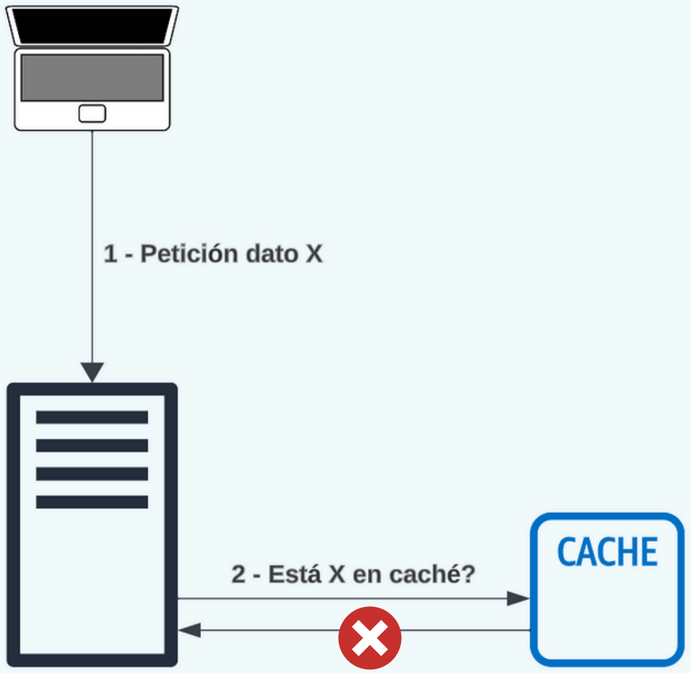
CACHÉ

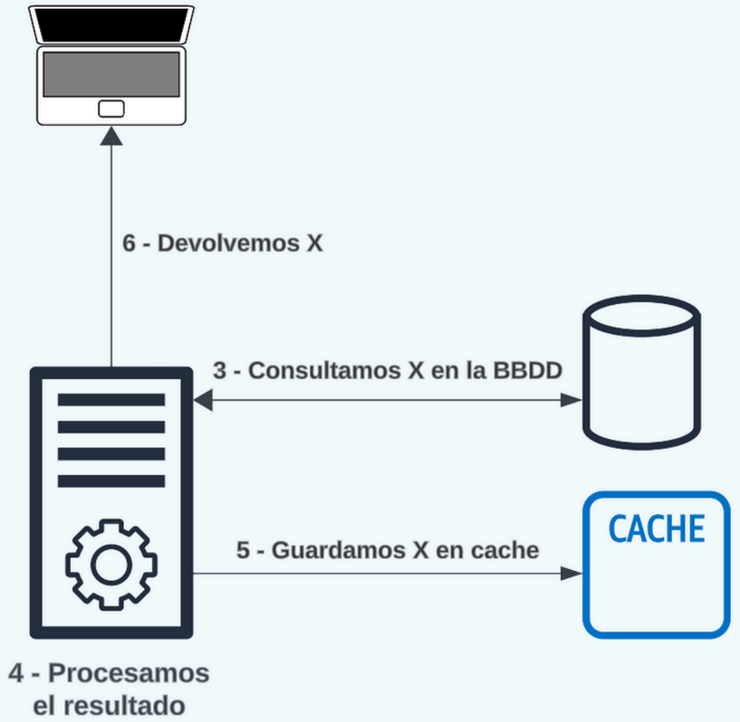
- Área de **almacenamiento temporal** utilizada para almacenar el **resultado de peticiones frecuentes y costosas.**
  - Peticiones posteriores serán mucho más rápidas.



CACHE-ASIDE

1. El cliente realiza una petición.
2. Si la caché no tiene el dato, se consulta la BBDD.
3. Se actualiza la caché con la información obtenida.
4. Se devuelve el resultado al cliente.





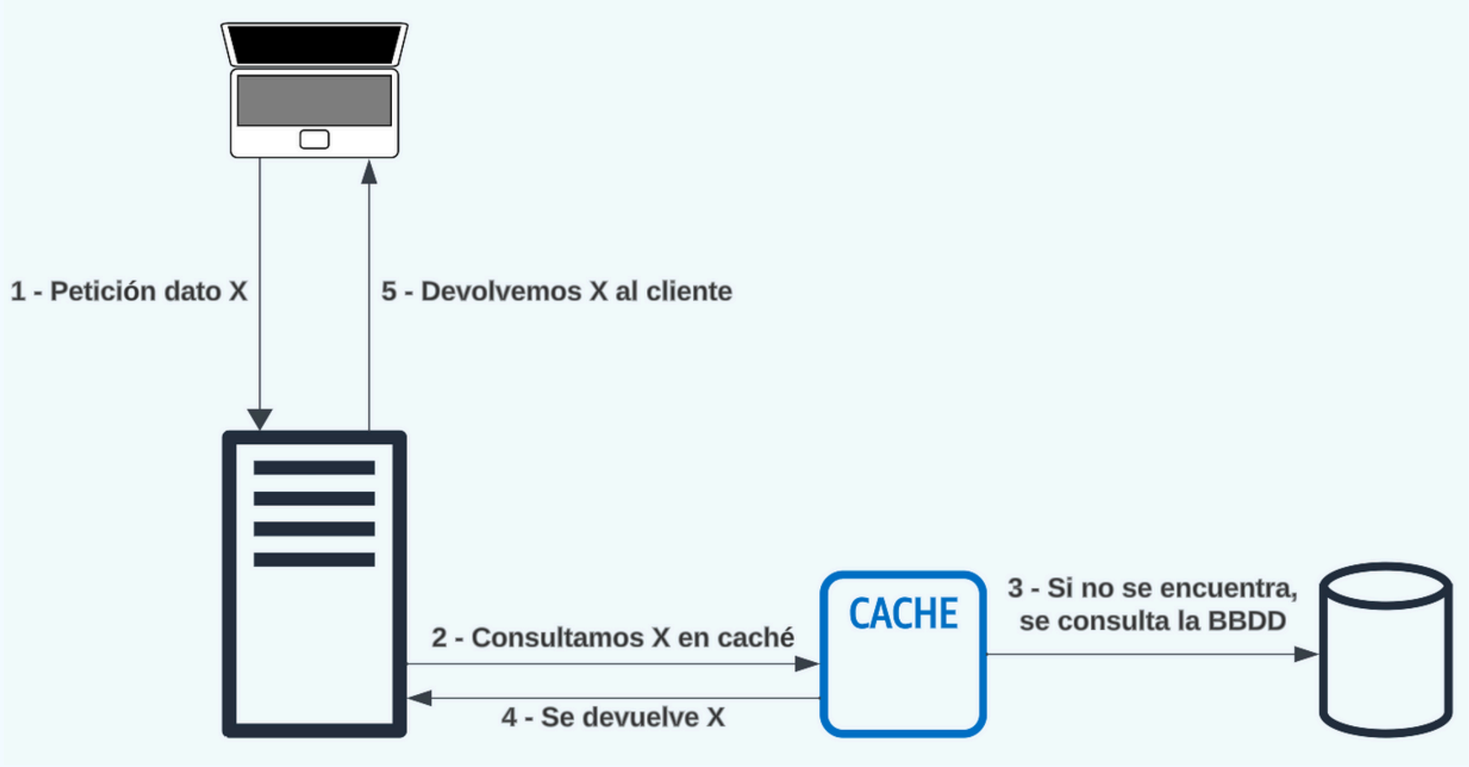
PROS Y CONTRAS CACHE-ASIDE

- Ventajas:
  - Reduce el tiempo de lecturas posteriores.
  - **Minimiza el riesgo de inconsistencia** entre la caché y el almacenamiento principal.
  - Muy buena para sistemas **read-heavy.**
  - La caché y la BBDD están desacopladas.

- Desventajas:
  - Caché inconsistente cuando se realicen escrituras.
  - **Latencia adicional** durante la recuperación de datos del almacenamiento principal.
  - La aplicación es la encargada de realizar el trabajo extra cuando no encontramos la información en la caché.

READ-THROUGH

1. El cliente realiza una petición.
2. Si la caché no tiene el dato, consulta directamente la BBDD y actualiza la información.
3. Se devuelve el dato a la aplicación de forma transparente.
4. Se devuelve el resultado al cliente.



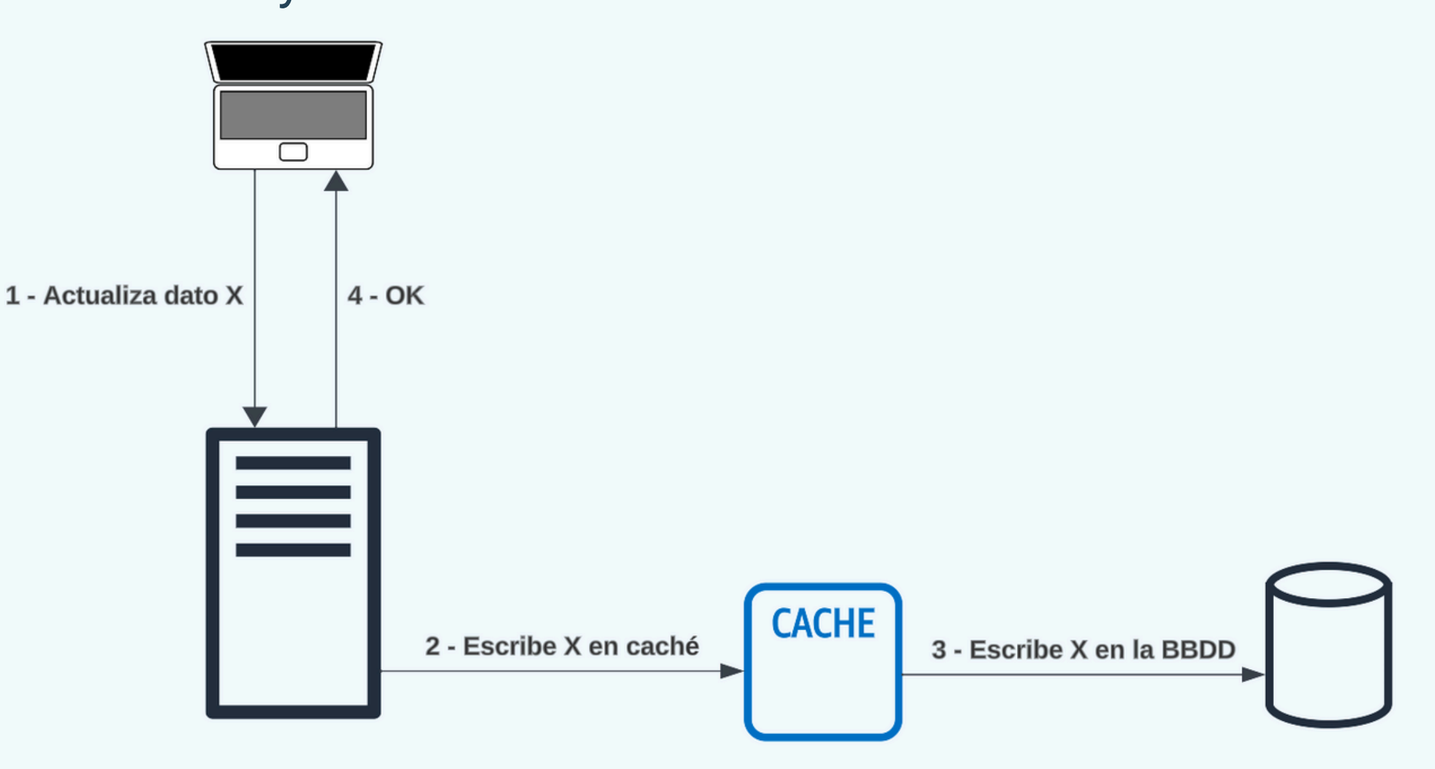
PROS Y CONTRAS READ-THROUGH

- Ventajas:
  - Reduce el tiempo de lecturas posteriores.
  - **Minimiza el riesgo de inconsistencia** entre la caché y el almacenamiento principal.
  - Muy buena para sistemas **read-heavy.**
  - Los fallos en caché y posterior consulta a la BBDD ocurren de forma **transparente para la aplicación.**

- Desventajas:
  - Caché inconsistente cuando se realicen escrituras.
  - **Latencia adicional** durante la recuperación de datos del almacenamiento principal.
  - El modelo de la **caché y el almacenamiento principal están acoplados.**

WRITE-THROUGH

1. El cliente escribe o actualiza un dato en la aplicación.
2. La aplicación actualiza la caché con la información.
3. La caché actualiza el almacenamiento principal con la información.
4. Se devuelve con éxito y se finaliza la conexión con el cliente.



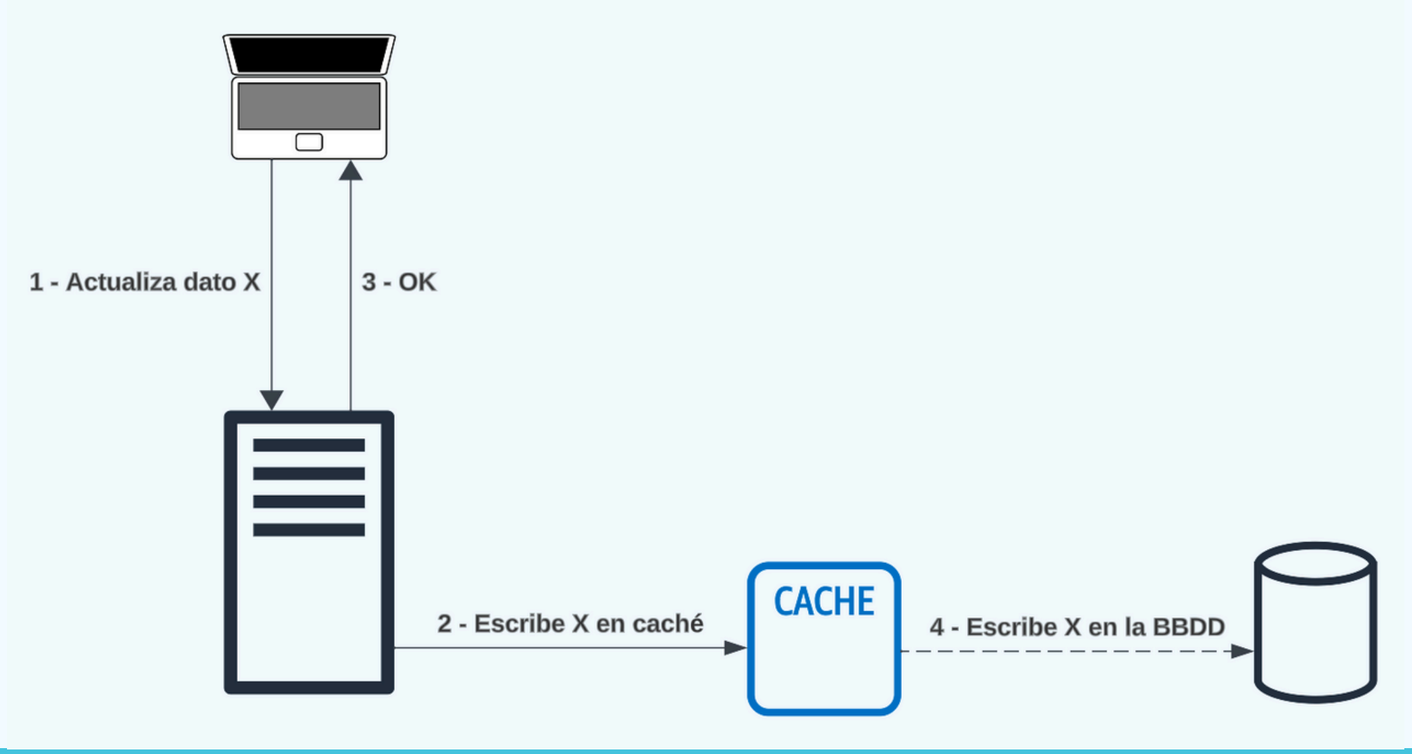
PROS Y CONTRAS WRITE-THROUGH

- Ventajas:
  - Reduce el tiempo en lecturas.
  - **Garantiza la consistencia** entre la caché y el almacenamiento principal.
    - No hay necesidad de utilizar mecanismos de invalidación.

- Desventajas:
  - **Latencia adicional** durante las operaciones de escritura.

WRITE-BEHIND / WRITE-BACK

1. El cliente escribe o actualiza un dato en la aplicación.
2. La aplicación actualiza la caché con la información y esta lo confirma al momento.
3. Se devuelve con éxito y se finaliza la conexión con el cliente.
4. La caché actualiza de forma asíncrona el almacenamiento principal con la información.



PROS Y CONTRAS WRITE-BEHIND / WRITE-BACK

- Ventajas:
  - Aumentan mucho el **rendimiento** en sistemas **write-heavy.**
  - **Tolerante a fallos en la BBDD.**
  - Puede reducir costes si las escrituras se hacen en batch.

- Desventajas:
  - **No garantiza la consistencia entre caché y BBDD.**

CONSIDERACIONES A LA HORA DE USAR CACHÉ

- Analiza dónde y cuándo es necesario.
  - Tiene un coste asociado e incluso puede resultar perjudicial.
  - ¿Realmente lo necesitas? **Observar métricas.**
  - ¿Qué **tipo de sistema** tienes? ¿Read-heavy o write-heavy? ¿La consistencia eventual es aceptable?

CONSIDERACIONES A LA HORA DE USAR CACHÉ

- Analiza dónde y cuándo es necesario.
  - Tiene un coste asociado e incluso puede resultar perjudicial.
  - ¿Realmente lo necesitas? **Observar métricas.**
  - ¿Qué **tipo de sistema** tienes? ¿Read-heavy o write-heavy? ¿La consistencia eventual es aceptable?

- Expiration Policy (aplicable a cacheado en la lectura).
  - No es recomendable mantener los datos cacheados demasiado tiempo.
  - **Establecer un TTL** (tiempo de vida) a los datos de la caché.
    - Una vez superado, el dato se elimina de la caché.
    - Si es muy alto los datos estarán desactualizados.
    - Si es muy bajo los datos se tendrán que actualizar continuamente.

CONSIDERACIONES A LA HORA DE USAR CACHÉ

- Eviction Policy (política de reemplazo)
  - Si llega un nuevo dato y se encuentra llena, debemos eliminar algún elemento.
    - **LRU (Least Recently Used)**. Se elimina el dato que se consultó hace más tiempo. Es la más popular.
    - **LFU (Least Frequently Used)**. Se elimina el dato menos consultado.
    - **FIFO (First In First Out)**. Se elimina el dato que primero se haya insertado.

CONSIDERACIONES A LA HORA DE USAR CACHÉ

- Eviction Policy (política de reemplazo)
  - Si llega un nuevo dato y se encuentra llena, debemos eliminar algún elemento.
    - **LRU (Least Recently Used)**. Se elimina el dato que se consultó hace más tiempo. Es la más popular.
    - **LFU (Least Frequently Used)**. Se elimina el dato menos consultado.
    - **FIFO (First In First Out)**. Se elimina el dato que primero se haya insertado.

- Consistencia.
  - Determinar el nivel de consistencia deseado.
  - En sistemas a gran escala con cachés y BBDD con múltiples instancias en distintas regiones del mundo, la consistencia va a ser un problema.

EJEMPLOS SISTEMAS CACHÉ

- Redis
- Memcached
- Spring Cache
- Caffeine
- AWS ElastiCache
- Microsoft Azure Cache for Redis