Advanced Java

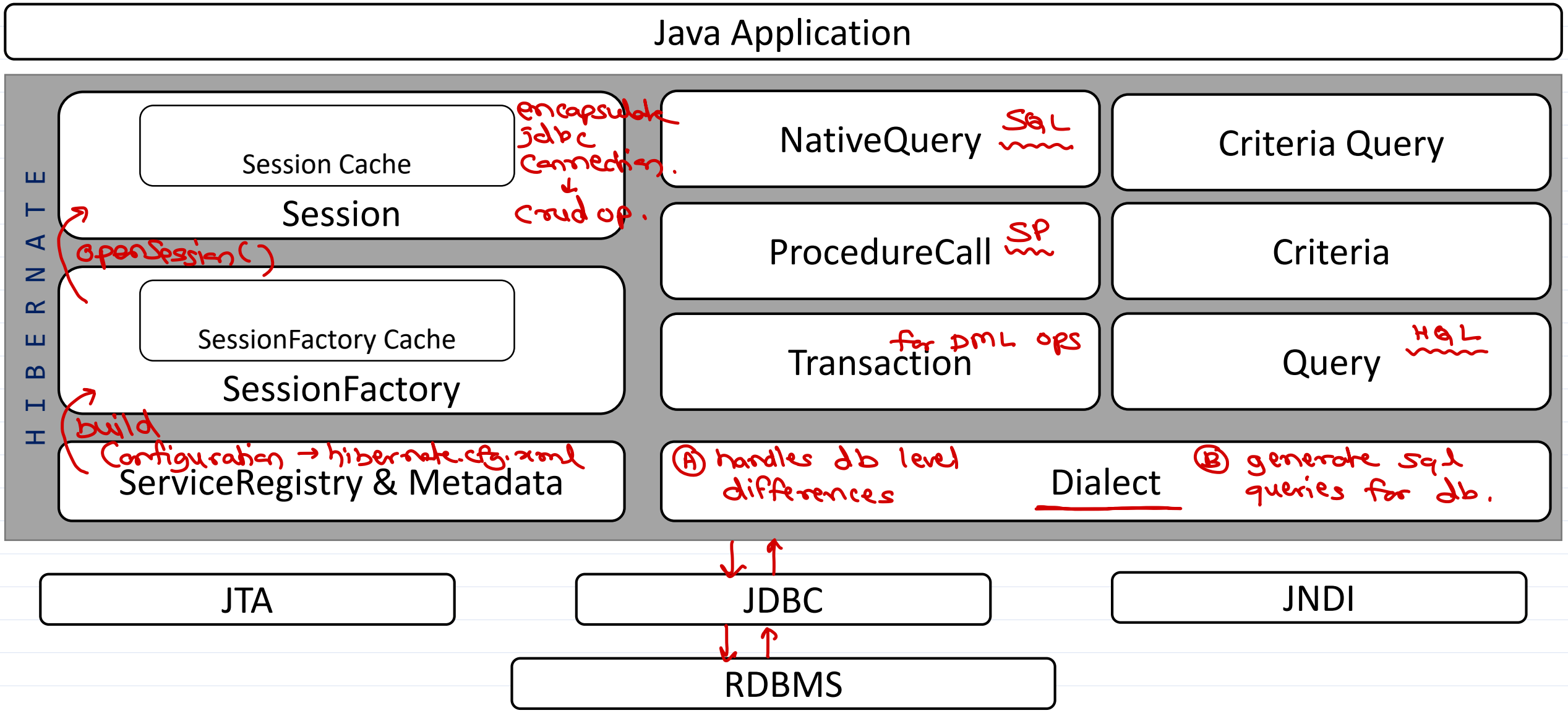
Trainer: Nilesh Ghule

Code First approach

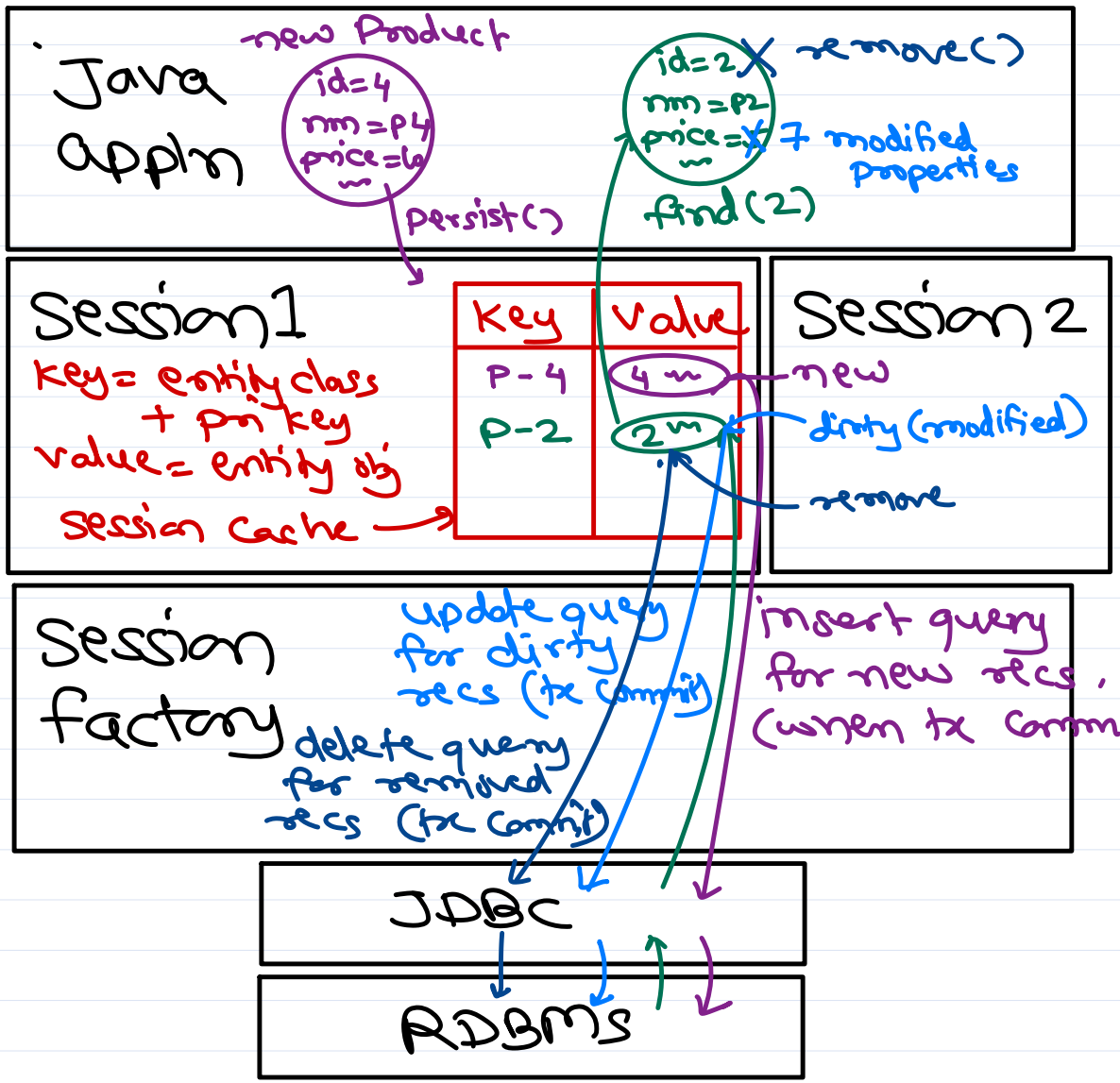
① implement all @Entity classes with desired fields & relations.

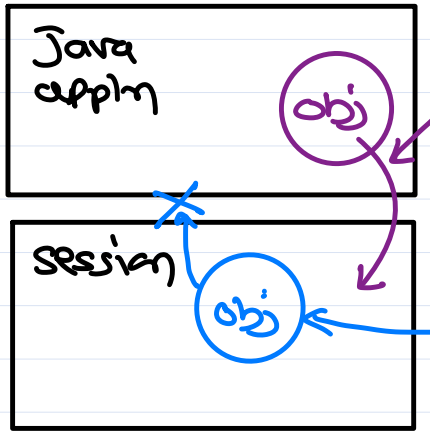
② in application.properties
spring.jpa.hibernate.ddl-auto = ?
- ⓐ none → no tables created & no validations performed.
- ⓑ validate → check if entity classes & relations matching with db tables & relations. If not matching, raise error/exception.
- ⓒ update → check if entity classes & relations matching with db tables & relations. If not matching, modify db schema (alter table ...)
- ⓓ create → create db tables as per entity classes & relations when appln starts.
- ⓔ create-drop → Same as create, but drops all created tables at end of appln.

Hibernate Architecture



Hibernate Caching





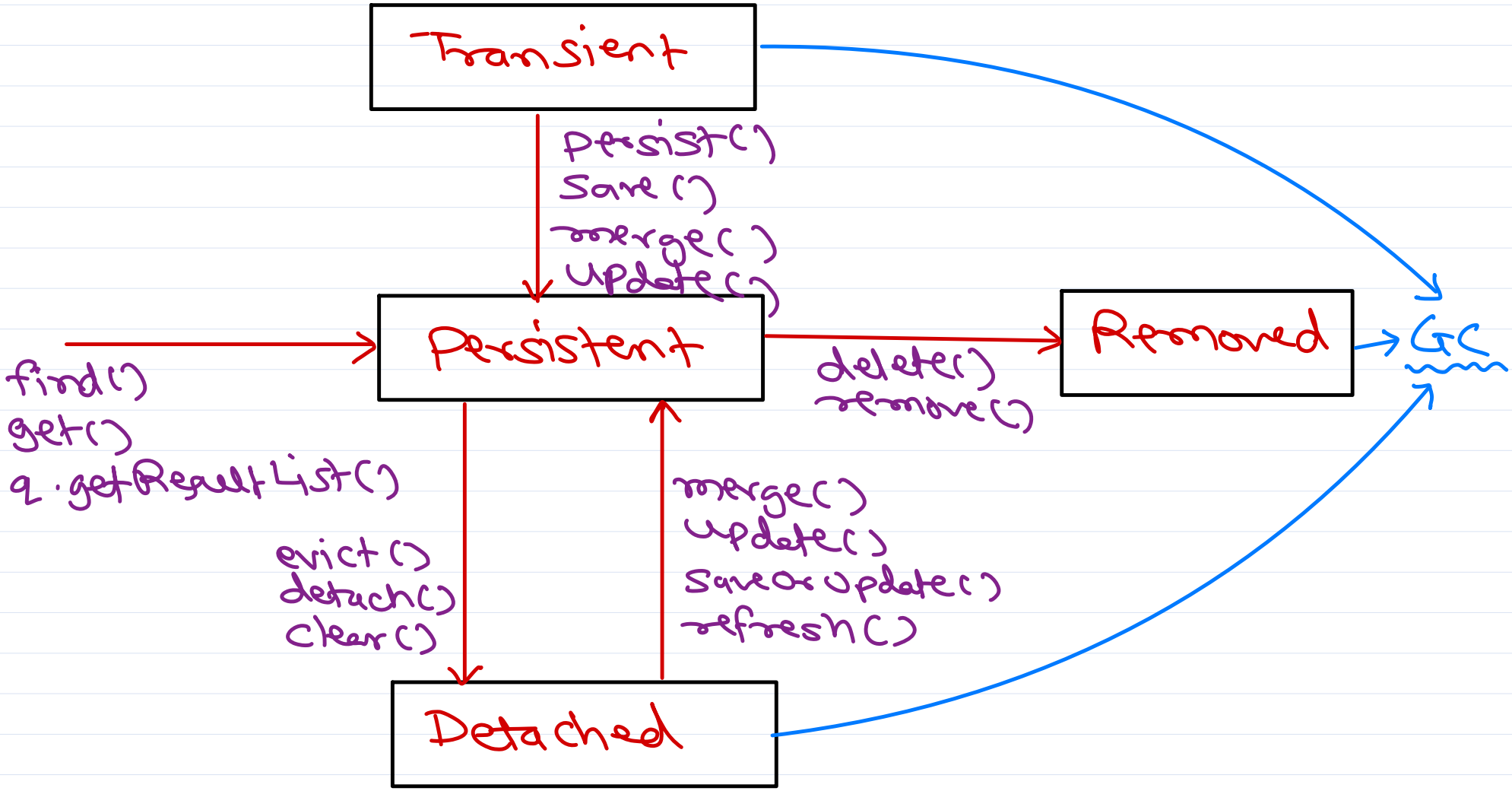
merge() - add obj into session cache & mark it as new (if not present in db) or dirty (if present in db).

detach() - remove obj from session cache.

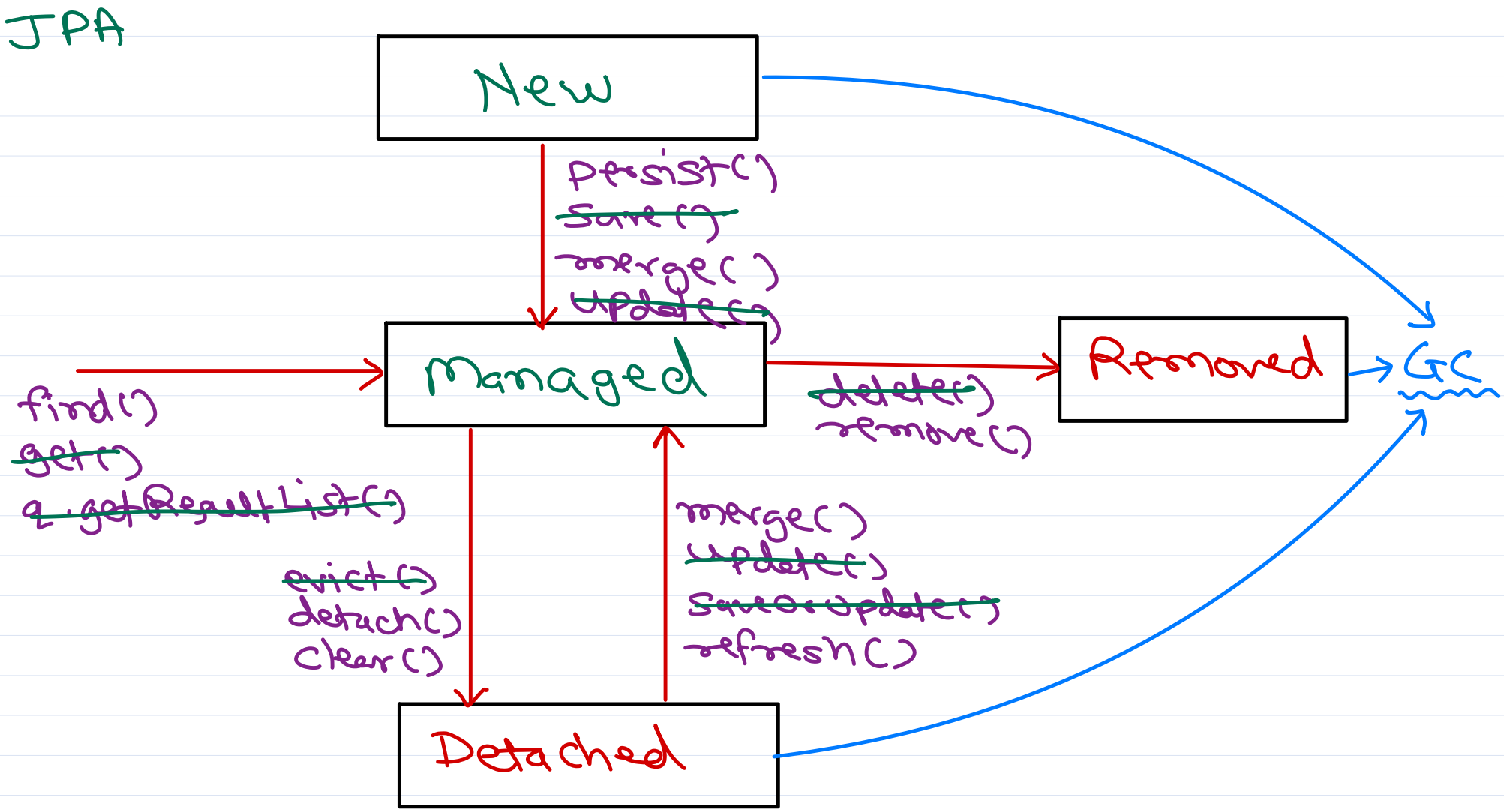
* session cache a.k.a. L1 cache:
- always enabled (cannot be disabled)
- if multiple sessions refer same row in db, they will create separate objs.

* session factory cache a.k.a. L2 cache.
- by default disabled. - need to configure.
- stores entity objs in serialized form.
- Common cache across all sessions ie. multiple sessions get data from L2 Cache & create java obj in their cache.
- sped up execution - less db connectivity.

Hibernate Entity Life Cycle



~~Hibernate~~ Entity Life Cycle



Hibernate vs JPA

JDBC

1. DriverManager
2. Connection
   - CRUD ops
3. No life cycle
5. pkg: javax.sql

Hibernate

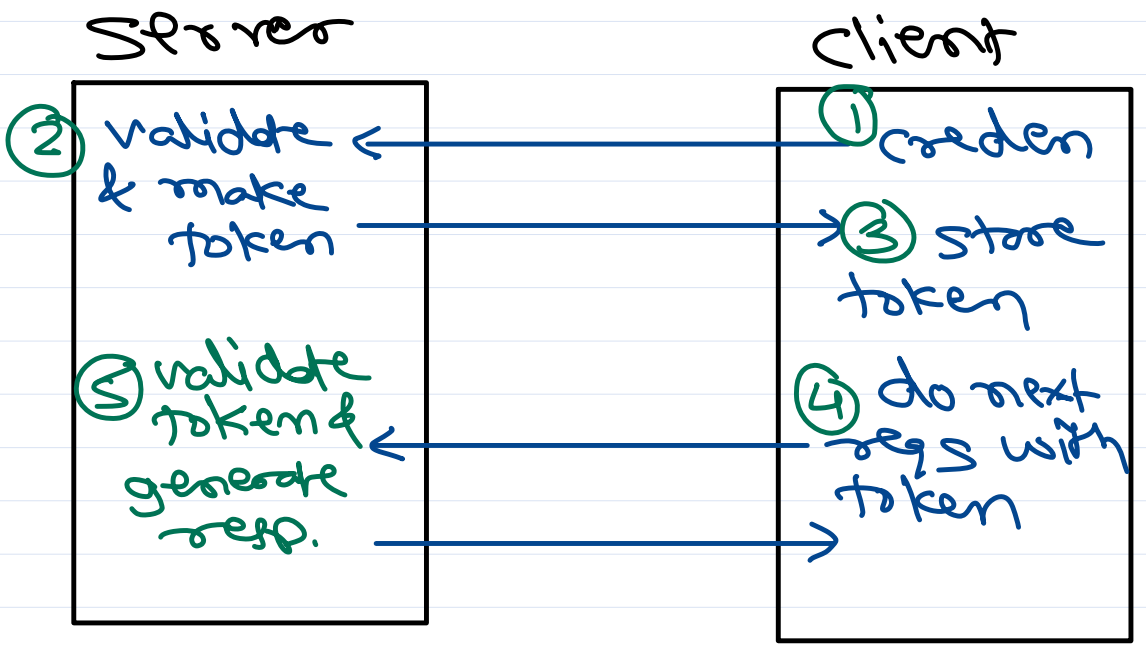
1. SessionFactory
2. Session
   - CRUD ops
3. Life cycle:
   Transient, Persistent,
   Detached, Removed
4. hibernate.cfg.xml
5. pkg: org.hibernate
6. CRUD methods:
   save(), saveOrUpdate(),
   delete(), evict(),
   get() & load(), ...

JPA

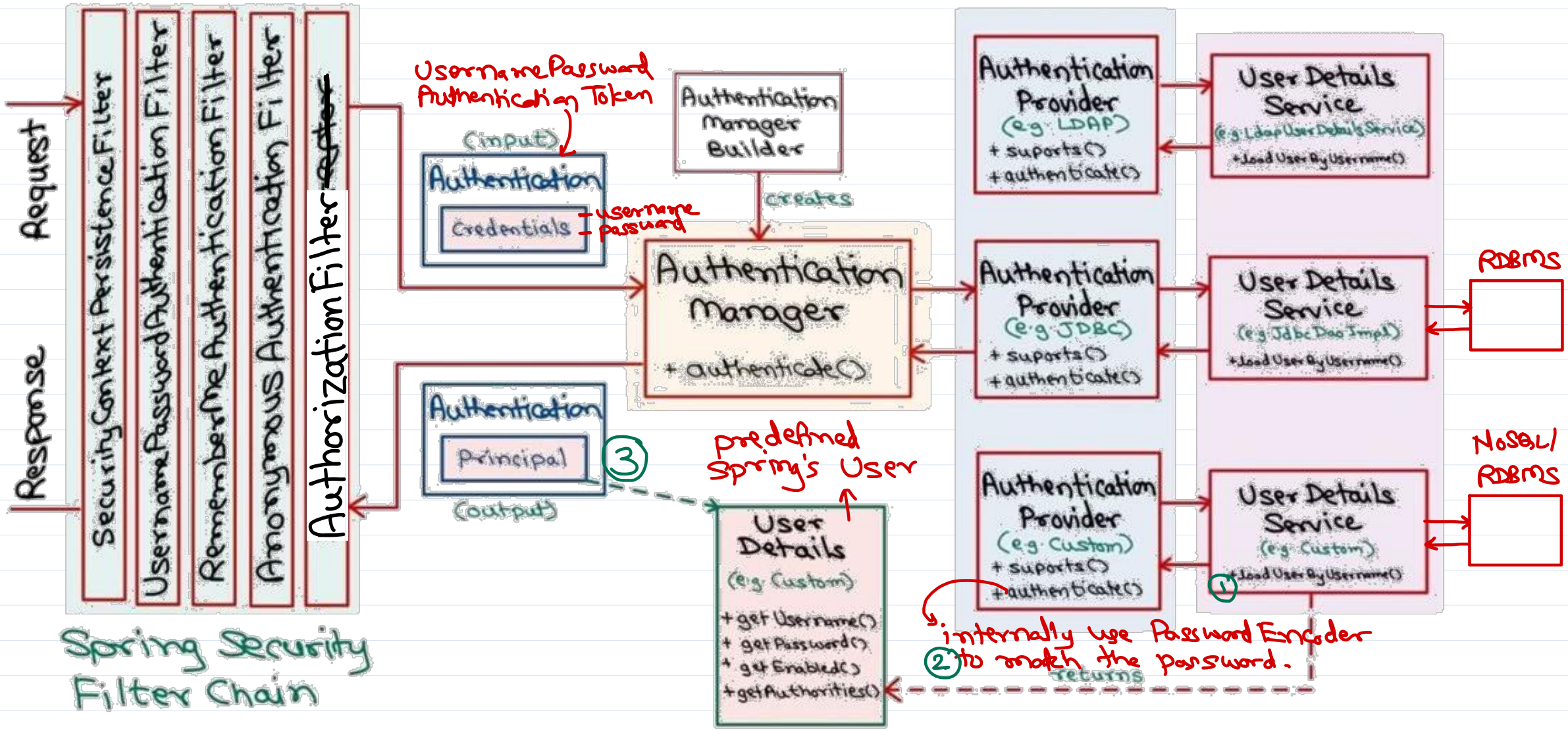
1. EntityManagerFactory
2. EntityManager
   - CRUD ops
3. Life cycle:
   New, Managed,
   Detached, Removed
4. persistence.xml
5. pkg: javax.persistence
7. CRUD methods:
   persist(), merge(),
   remove(), detach(),
   find(), ...

JWT

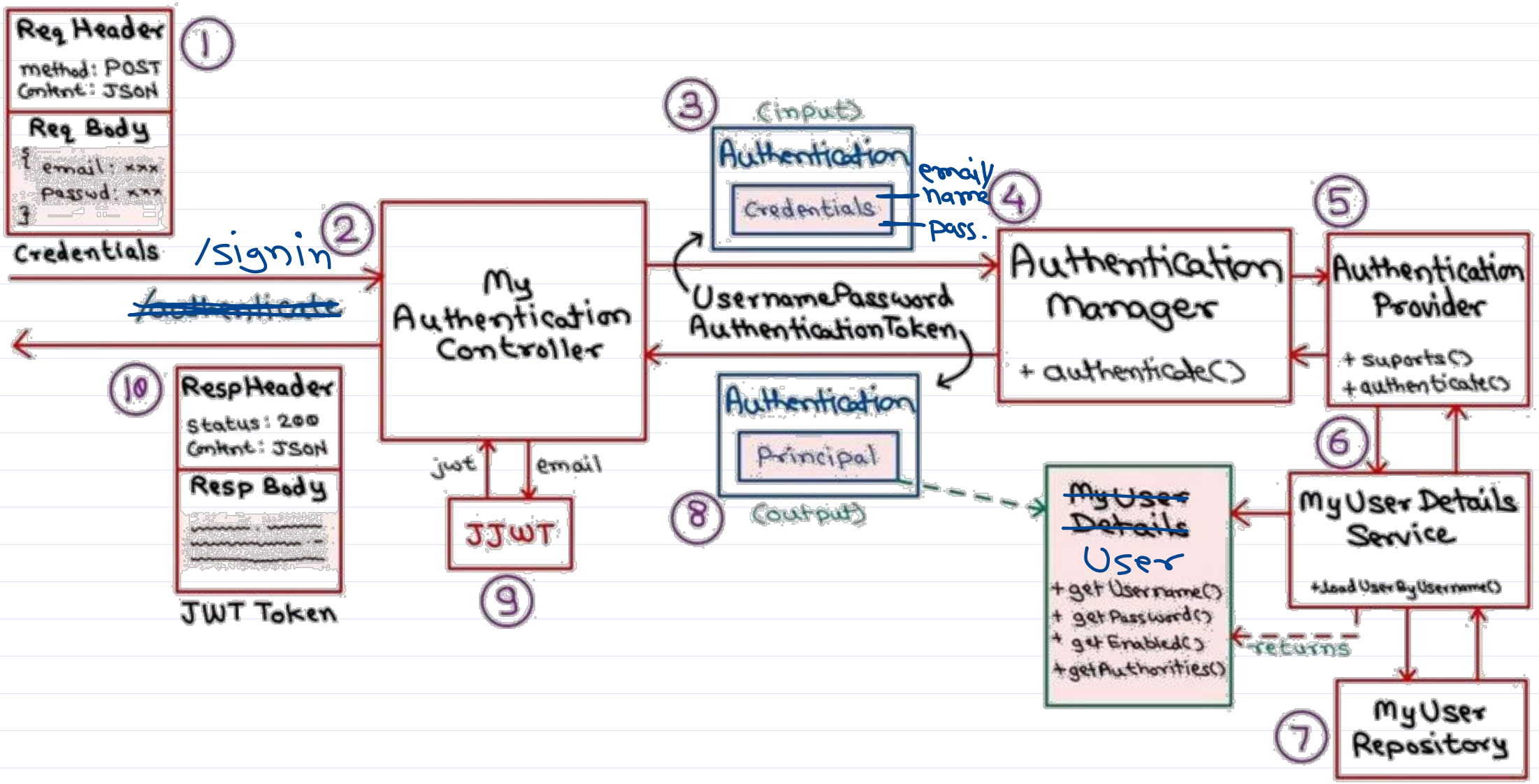
* Used for REST api authentication.
* REST apis are stateless.
  - Each req should include all input details that it needs (cannot rely on prev request inputs – not stored).
  - Each req should have user identity.
* JWT token is used to store user identity. It can be carried with each request (like cookies).
* JWT token has 3 parts
  - (A) Header → enc algo
  - (B) Payload (data) → data to store in token (id)
  - (C) Signature → secret key.
* JWT can be read by client, but can't be tampered/modified.



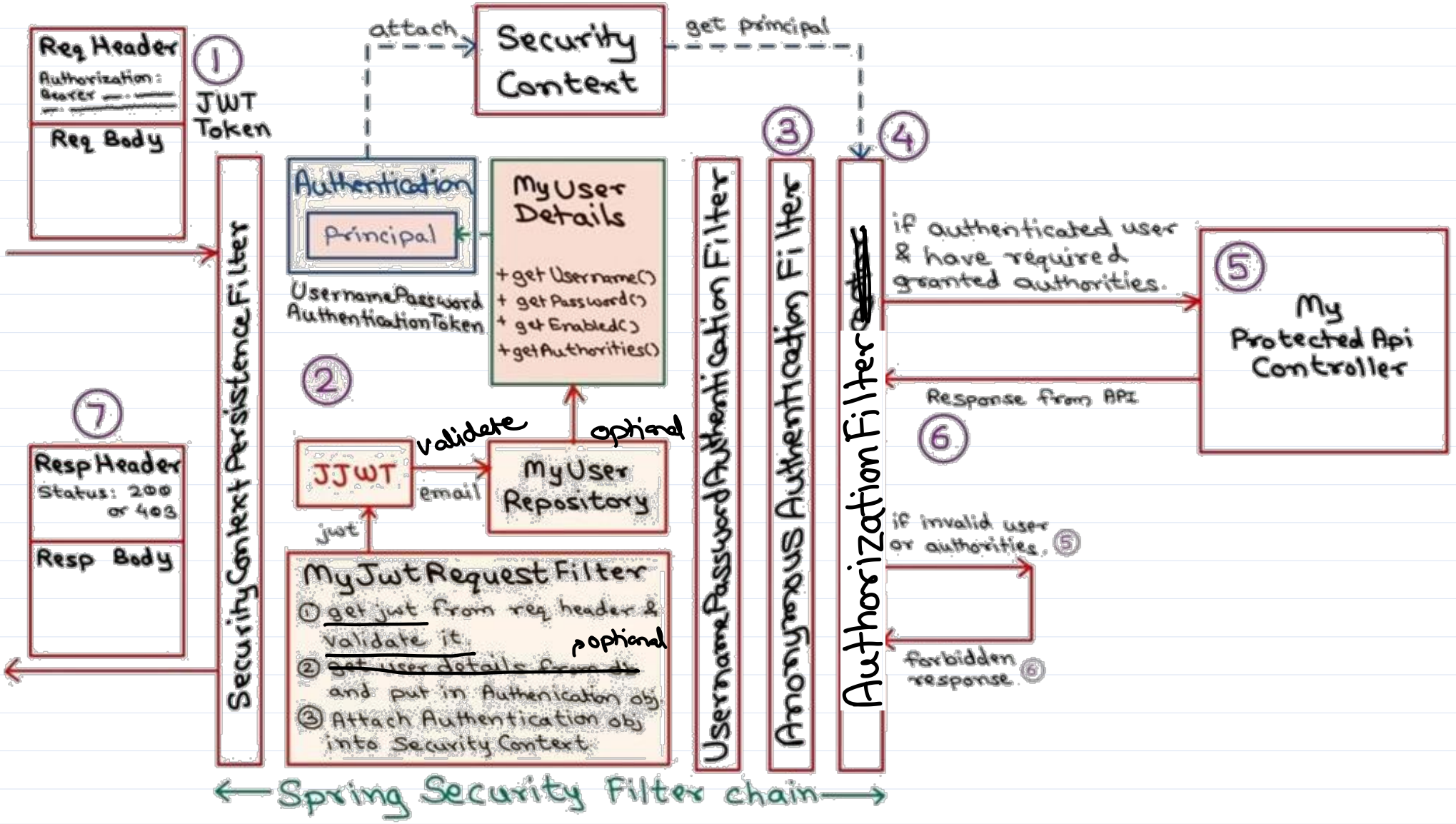
Spring Web Security



Spring REST Security (1)



Spring REST Security (2)



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>