h2 database (embedded database)

=> This database setup would be created on the ram once we start the application
   and it will be destroyed once the application stops.

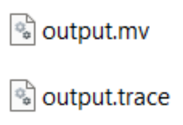
=> It would be available through the ui request as "/h2-console".

Database Configuration
=====================
=> By default, Spring Boot configures the application to connect to an in-memory store with the username sa and an empty password.
=> However, we can change those parameters by adding the following properties to the "application.properties" file:

```
application.properties
=====================
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
```

By design, the in-memory database is volatile, and results in data loss after application restart.
We can change that behavior by using file-based storage.
To do this we need to update the spring.datasource.url property

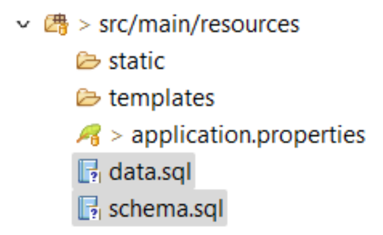
```
application.properties
====================
spring.datasource.url=jdbc:h2:file:D:/output
```



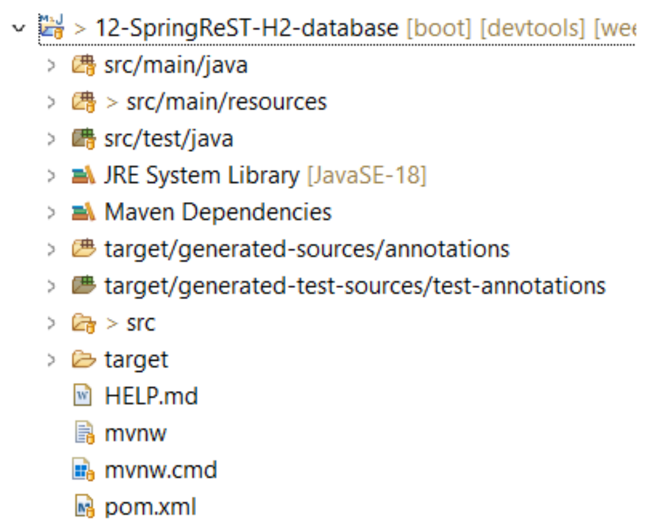
output.mv
output.trace

Note: With this setup, if we run the application, springboot will not create a table to carry out CRUD operations

To make the table to be created, inform spring boot to use a file called "schema.sql"
It is also possible to store some data during the application startup, so we use a file called "data.sql".

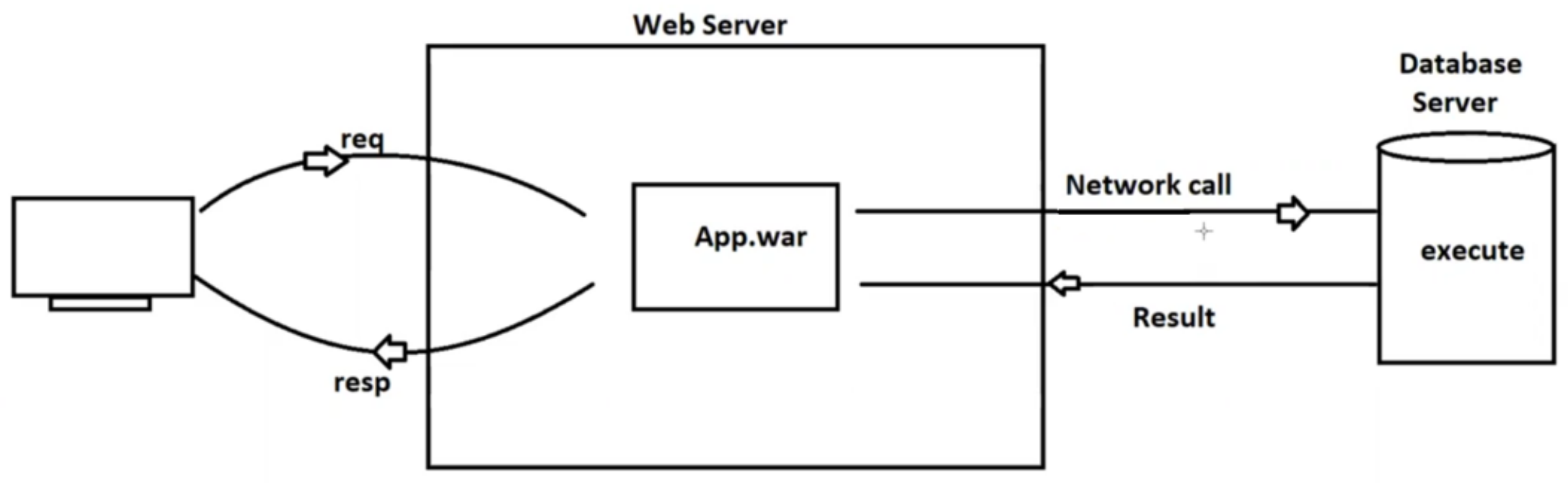


```
src/main/resources
    static
    templates
    application.properties
    data.sql
    schema.sql
```

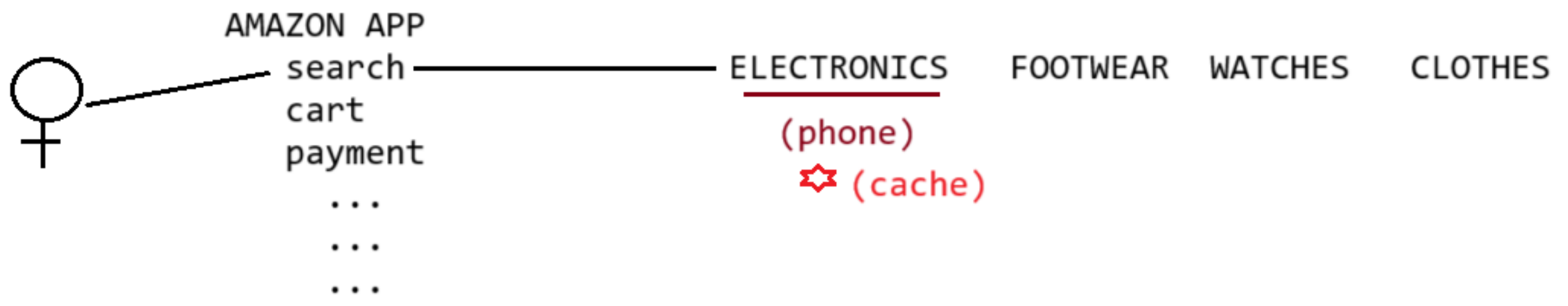


```
12-SpringReST-H2-database [boot] [devtools] [we
    src/main/java
    src/main/resources
    src/test/java
    JRE System Library [JavaSE-18]
    Maven Dependencies
    target/generated-sources/annotations
    target/generated-test-sources/test-annotations
    src
    target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

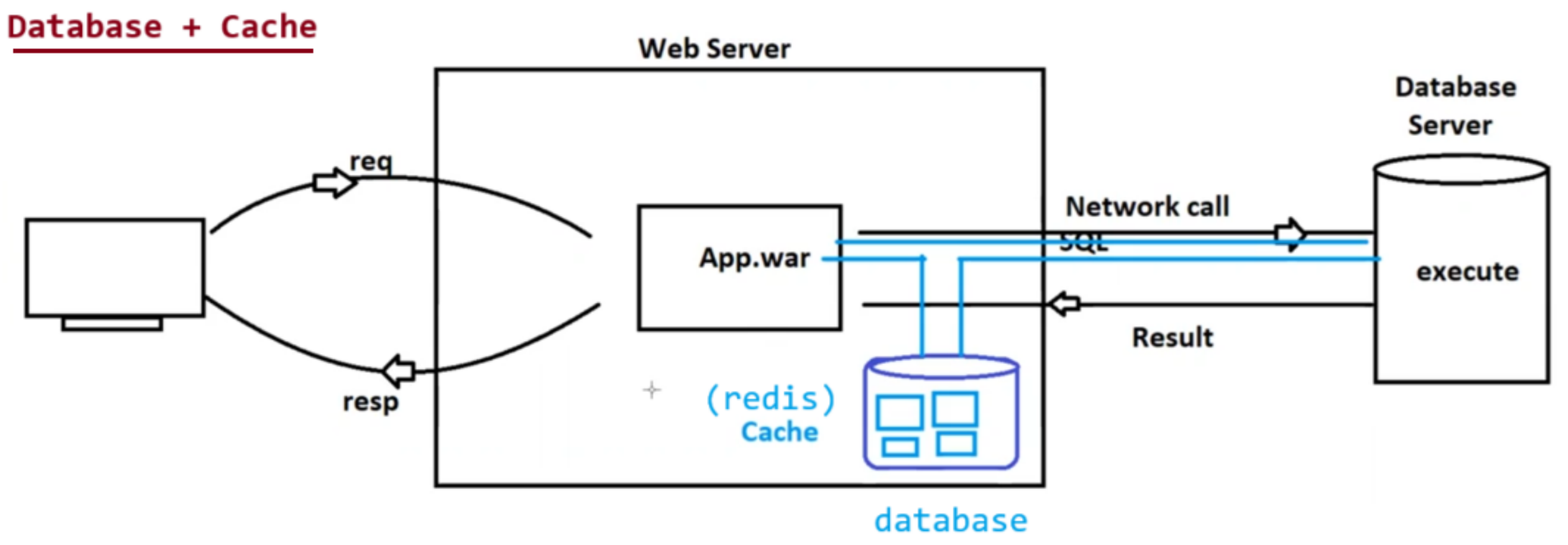
Application w/o Caching



CALL TO DB is MORE
PERFORMANCE is LOW



Database + Cache



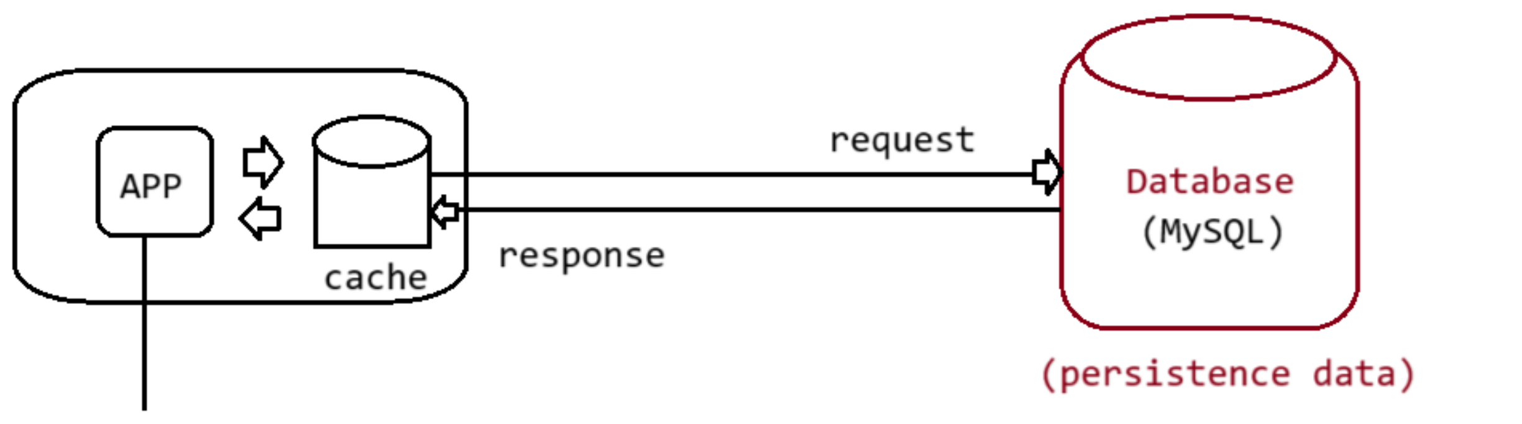
C -> save ——— No caching, it should be in db
R -> find one record | load all record ——— needs caching only for one record
U -> update one record ——— one record operation should happen in cache
D -> delete one record ——— and also it should be reflected in "Main memory(Actual DB)"

RedisCache
==========
*) Problem:
If No.of Network calls Between Server(App) and Database are increased then that results application performance down.
(Which takes more time to execute all N/w calls)

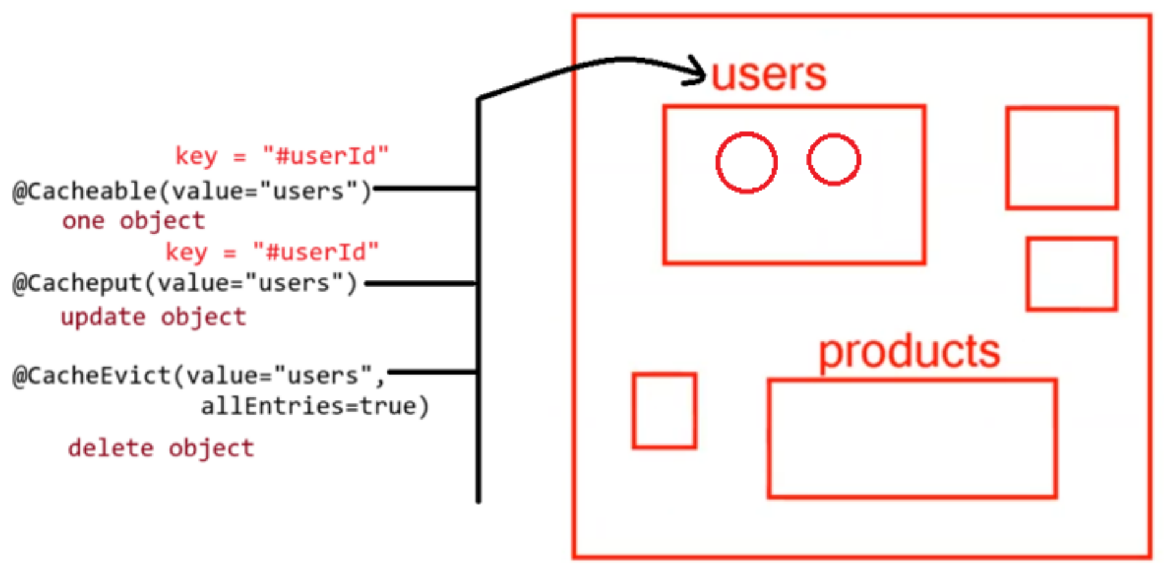
*) Cache : It is a process of storing data at server side to reduce no.of network calls for commonly accessed data.

Like Top 50 Emails, Top 30 user posts, commonly searched mobiles,..etc

-> Cache Exist at server side.
-> Cache is a also one type of database.
-> Cache reduces network calls from 100% to 80%/90%/99%...etc
-> Cache can store any type of objects(products, Inbox...etc)
-> Cache is handled by Operations (getOne/updateOne/deleteOne) ie Cache and DB must be in Sync.
-> Cache should never be used to store all DB Data.
   (dont use for findAll() and save() operations)



SpringBoot with redisDB(used for caching) and MySQL DB(persistence data)



Redis Cache

Annotation support given by springboot to work for Caching
  @EnableCaching : To Activate Cache Concept (add at Main class)

  @Cacheable   : Store object in cache [find/get]
  @CachePut   : Modify existed object in cache [update]
  @CacheEvict   : Remove existed object from cache [remove/delete]