Distributed Systems (H0N08A)

Report: Java EE

Filipe Borralho *(r0825061)*
Razvan Roatis *(r0825322)*

November 20, 2020

Question 1 We have the client tier where we have the main class, then there is the enterprise information systems tier where our database is located and there is also the business tier where our ejb's are, those would be for example session beans like ReservationSession and ManagerSession.

Question 2 Stateless sessions are meant to do a single transaction at a time, it either fails or it works but there's no relation with the previous transactions, for example the retrieval of rental statistics. If the method fails, the manager can just ask for those again. It would be possible to do this with a stateful session but that would make the server need to be able to support a huge amount of sessions and states, which would lead into scalability problems. On the other hand the client needs to be able to call consecutive related methods, for example create multiple quotes. In this case the session must keep it's state until the client terminates so that he can somehow have a shopping cart of quotes and only finalize those quotes in the future.

Question 3 Dependency injections allow the client to use remote methods without having to lookup in a registry using string-based names. So we could say that Java EE provides a more reasonable way of accessing remote objects. We use the annotation @EJB to tell Java EE to inject the ManagerSessionRemote. We have not done the same for ReservationSessionRemote because those sessions are statefull and we need a different one for each new client. We are retrieving them by calling context.lookup(ReservationSessionRemote.class.getName()).

Question 4 JPQL persistence queries are more efficient then application logic because usually DBMS are very well optimized for queries and so it is faster to retrieve data this way. Other reason is the avoidance of transferring huge amounts of data from the database to the java code, by querying and retrieving only the necessary data a lot of memory can be saved.

Question 5 Since there is a transaction support in Java EE, this solution is more resilient against server crashes because if a transaction is interrupted in the middle it will be aborted and there will be an automatic rollback, whereas in the RMI assignment this wouldn't happen. Also since the data is stored in a persistence database even if a transaction aborts the data won't be lost.

Question 6 Java EE allows to query a database, selecting, inserting, removing and updating values. That way by opening a connection with another database it's quite simple to persist the data retrieved from another database. Though one has to keep in mind that not all engines support the exact same query syntax, for example commands such as AVG(COUNT(sth)) might not be accepted.

Question 7 Race conditions happen when 2 or more clients/managers try to access the same information on the DB before the first one as "finished". This means that, for example, if two clients try to call the method confirmQuotes and in their individual list of quotes they both have the same car then only one will be able to reserve that car so the server has to temporary delay

the second calling of the method and then rollback that second one. Therefore, we prevent race conditions by using transactions provided by Java EE.

Question 8 Transactions can be executed in parallel as long as they don't operate over the same data. Whenever a transaction starts it puts a lock on the data it's using and only when it commits or aborts is this lock lifted. With synchronization this isn't possible at all, so in terms of scalability transactions are a better option except when, for example, all clients try to reserve the same car, case in which the problems are similar.

Question 9 To ensure only managers can access a manager session we annotate the ManagerSession class with @DeclareRoles("carManager") and then annotate the methods with @RolesAllowed("carManager"). This way only users registered as carManager's in the server can call the methods declared in ManagerSession.

Question 10 Java EE is built on top of the Java SE and provides an API and runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications. In other words, EE has some extra functionalities like:

- Transaction support
- Session management per user (session state)
- Extra security with annotations like @RolesAllowed
- Persistence

These make the java EE solution much more feasible.