Distributed Systems (H0N08A)

Report: Java EE

Ruben Kindt *(r0656495)*
no team partner

November 11, 2020

1. Outline the different tiers of your application, and indicate where classes are located.

We have the client tier with the Main.java and some testing classes are. This runs normally (in a distributed setting on the client's machine).
As a second tier we have the business tier which implements all the business level logic, like the session management (ManagerSession, ManagerSessionRemote, ReservationSession and ReservationSessionRemote) but also the entity classes (Car, CarRentalCompany, CarType and Reservation) and the class Quote.
As a final tier we have the database which stores all of the persistent data, where each entity class of the business tier represents a table.

2. Why are client and manager session beans stateful and stateless respectively?

The client is stateful because it has a state to remember, namely a list of quotes. The manager session does not have data/ (a state) to remember an can therefor be stateless.

3. How does dependency injection compare to the RMI registry of the RMI assignment?

In RMI the developer needs to do a manual look-up of objects whereas within Java EE the JNDI (Java Naming and Directory Interface) does all the work for the developer.

4. JPQL persistence queries without application logic are the recommended approach for retrieving rental statistics. Can you explain why this is more efficient?

JPQL persistence queries should be made without application logic to allow the client remain unaltered when changing the business logic.

5. How does your solution compare with the Java RMI assignment in terms of resilience against server crashes?

When the server crashes in the RMI assignment all the 'persistent' data would be lost. Whereas with Java EE the server (the business tier) can crash without any loss of persistent data.

6. How does the Java EE middleware reduce the effort of migrating to another database engine?

Since there is a layer between the developer and the database called the ODBC (Open Database Connectivity). This layer translates the developer's Java Persistence query language into the query language the database uses. Therefor changing database is as mush effort as configuring the ODBC.

7. How does your solution to concurrency prevent race-conditions?

This is prevented by Java EE due to the build-in support for transactions which have all the ACID properties together with the fact that all entity classes implement Serializable.

8. How do transactions compare to synchronization in Java RMI in terms of the scalability of your application?

Within Java EE it is easy to distribute the workload over multiple servers using build-in tools of Java EE therefor improving scalability. Whereas within Java RMI there are no tools provided and the developer would need to implement it himself/herself.

9. How do you ensure that only users that have specically been assigned a manager role can open a Manager Session and access the manager functionality?

By annotating the allowed role which is allowed to call particular functions or classes, which the container checks.

10. Why would someone choose a Java EE solution over a regular Java SE application with Java RMI

RMI could be a better option to Java EE in the case where no security, no transactions, no persistent data and no scalability is needed due to the simpler model.