Distributed Systems (H0N08A)

Report: Java EE

Dries Janse *(r0627054)*
Steven Ghekiere *(r0626062)*

November 19, 2019

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

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

Both the manager and client session beans are server-side (EJB) Enterprise Java Bean components. The component is a session bean, because it represents a session with the client. The client session is a stateful session bean, this means that it retains state between method invocations. The (conversational) state consists of the values of the instance variables. The client session is stateful because it keeps the conversational state, it contains the name of the renter and the list of quotes of this renter. The manager session is a stateless session bean, this means that it does not retains the conversational state. In our application there is no need to make the manager session stateful because all instances of the bean are equivalent. This allows the EJB container to assign an instance to any manager.

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

The RMI registry of the RMI assignment is a simplified name service that allows clients to get a reference to a remote object. The @EJB annotation can be applied on fields or methods to inject dependencies. First, the EJB client container will perform a JNDI lookup to locate the dependency. JNDI stands for Java Naming and Directory Interface, it allows distributed application to look up services in an abstract way. With JEE we don't have to explicitly instantiate, with the "new" keyword, the objects of which we need a reference. JEE provides these injection mechanisms. We only have to declare the needed resources with the annotations (@EJB) that denotes the injection point to the compiler. The container will then provide an instance of the required resource at runtime. The advantage of using dependency injection is that it simplifies our code because the container automatically provides these instances.

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

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

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

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

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

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

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