Distributed Systems (H0N08A)

Report: Java EE

Bart van Bommel *(r0620692)*
Wouter Mertens *(r0746877)*

November 19, 2020

1. Outline the different tiers of your application, and indicate where classes are located The project is divided in three tiers: the client, the ejb and the lib tier. The client is where all the functionality is located that the user will need to connect to and query the main server / database. In this case the main.java file. The lib tier is where all the shared data types are located. Classes which will be used by both the client and server are found here. In the case of the sessions, only an interface is provided as these are objects on server side and the client needs only communicate with them to start its methods. On the ejb tier, all data that is used on server side is found. This includes the sessions, the rental companies and the cars.

2. Why are client and manager session beans stateful and stateless respectively? The client sessions are stateful as they will hold data while a client is using it. The name of the client and a list of quotes that have yet to be confirmed are stored in the session. The manager session holds not state as it will only query the database or create new objects that need to be stored. Nothing is saved in the session itself. Stateless classes also increase scalability as they can be reassigned to another user without having to wipe their contents.

3. How does dependency injection compare to the RMI registry of the RMI assignment? Since EJBs are managed by the container, they are a lot less work to manage in bigger applications. RMI requires you to build everything from the ground up, while EJBs give you a more pre-packaged approach, which can be more robust in the end.

4. JPQL persistence queries without application logic are the recommended approach for retrieving rental statistics. Can you explain why this is more efficient? JPQL statements are not limited to a single entity instance, operations can be defined over sets of entities, and sets of entities can be returned. This is because JPQL does not query the database tables directly, but queries JPA entities mapped to underlying database tables. Entity instances retrieved by a JPQL query also automatically become managed, this means the state of the instances associated with the retrieved instances will also be taken care of.

5. How does your solution compare with the Java RMI assignment in terms of resilience against server crashes? While Java RMI has no pre-built protections for server crashes, EJB has some. The sessions in EJB will be lost as the states will not be restored. Although this matters little for stateless sessions. The EJB entities will survive the crash as the states are automatically saved in the database and then restored when the bean is re-instantiated.

6. How does the Java EE middleware reduce the effort of migrating to another database engine? Java EE allows to define a source for a database engine and accesses these database through SQL queries. As such any database engine that supports SQL can be accessed and easily assigned using the Java EE middleware.

7. How does your solution to concurrency prevent race conditions? Our solution uses bean managed transactions. The use of transactions means that all changes are commited only when all the required transactions can push through. i.e. if a transactions also requires another transaction, but the contained transaction fails, then the containing transaction will also fail and a rollback is initiated. This prevents race conditions as no change is made while the method is still working on its sub transactions and any new method that would invoke a change still reads the database as if it was unchanged.

8. How do transactions compare to synchronization in Java RMI in terms of the scalability of your application? Transactions in Java EE are a lot easier to use than synchronization in RMI, this because of the transaction attribute. This attribute allows to manage the functionality and behaviour of the transaction. For example, we can declare that a transaction should fail if another transaction that is called from withing the containing transaction fails. In RMI this kind of behaviour was not supported. RMI's synchronization prevents multiple users from changing the same object at the same time. It does not, however, allow the same functionality of roll backs in case another transaction failed, which could lead to scalability issues.

9.How do you ensure that only users that have specifically been assigned a manager role can open a ManagerSession and access the manager functionality? A @RolesDeclared and @RolesAllowed annotation has been added to the ManagerSession. With the annotation spanning the whole class, this means that any functionality inside of the class will not be available to any user which does not have the needed role.

10. Why would someone choose a Java EE solution over a regular Java SE application with Java RMI? Java EE already comes with transaction and role functionality and prevents the programmer from having to implement his own version of these algorithms. This allows us to rely on knowledge from experts before us, prevents us from making mistakes in the implementation of these algorithms and provides a form a quality assurance.