# Composite Service

Our service has root in the CompositeService-class which consists of a number of static methods.

**REST** was easy to work with and our code is both compact and easy to read. REST is intuitive, transparent and easy to debug as the error messages are standardized (HTTP errors). We quickly got the code working.

**SOAP** gave us hours of work. It was difficult and the code is not nearly as clear and clean as our REST code. The main problem is with the debuggability: As SOAP is a more complex way to communicate with servers, it also has more strict requirements, and (unfortunately) less clear error messages.

In the end, we realized that modern IDEs for a great part can generate the most SOAP boilerplate code for you, making the task easy.

We prefer REST, as it also allows for looser coupling and is not tied to certain protocols and formats in the same way as SOAP.

When turning our application into a web service, we used the simple jws library and Endpoint.publish (painless).

## Testing

We use JUnit as our testing framework, to test the general functionality. As the functionality of our web service is quite simple, our primary test makes sure we can successfully get XML correctly. When this has been assured, the other tests either create or delete elements, verifying their effect with the data-getting method.

**Testing with marshalling**We created the classes Task and TaskList with JAXB annotation. Our class JaxbUtils makes it easy to convert these to and from XML.   
We use marshalling in our Unit tests and when we merge data in service 3, from service 1 and 2 .

Task also has our own equals- and hashCode methods. These are used for testing. When we create a task on one of the servers, and we get the task again, we marshal both XML instances of the task into object instances, and compare them using their equals-method.

