Project Documentation

Setting up a running instance

To set up a running instance with Docker installed, follow these steps:

1. Import the XSD into the project directory.
2. Use "*gradlew build*" in the project directory.
3. Use "*gradlew composeUp*" to build and start the Docker Container.
4. "*docker-compose stop*" shuts the Docker Container down.

In case step 3 does not work, you can use these alternatively:

1. Move the WAR from *build/libs* to the *Docker/web* and rename it *application.war* (this is done by "*composeUp*", even if it fails).
2. In Docker, use "docker-compose build" to build the Docker Container.
3. Use "docker-compose up -d" to start the Docker Container.

To set up a running instance for Eclipse, follow these steps:

1. Import the XSD into the project directory.
2. Use "gradlew eclipse" in the project directory.
3. Import the Project into your Workspace.
4. When using Tomcat 8.5.X, change the server version in the Tomcat configuration (*org/apache/catalina/util/ServerInfo.properties*) file located in the *catalina.jar* to 8.0.0.

How to use the software

The HTTP-server is reachable on port 8081, if you use the default docker-compose settings.
Navigation can be done by clicking on the links at the top of every page.
A user can define new tests on the "Test-Definition" page; inputs will be validated and errors reported.
On the "Test-Liste" page, the user can view all defined test. The user can execute, edit or delete any of those tests.
Executing the test will send the defined request to the OSST and show the response. Editing the test will work just like defining a test. Deleting the test will cause all responses to be deleted as well.
The "Antwort-Liste" page will show all responses given by the OSST along with some additional information regarding validation. The user can also view the response.
"Alle Tests ausführen" executes all defined tests and gives a statistic on the validity of the responses received.

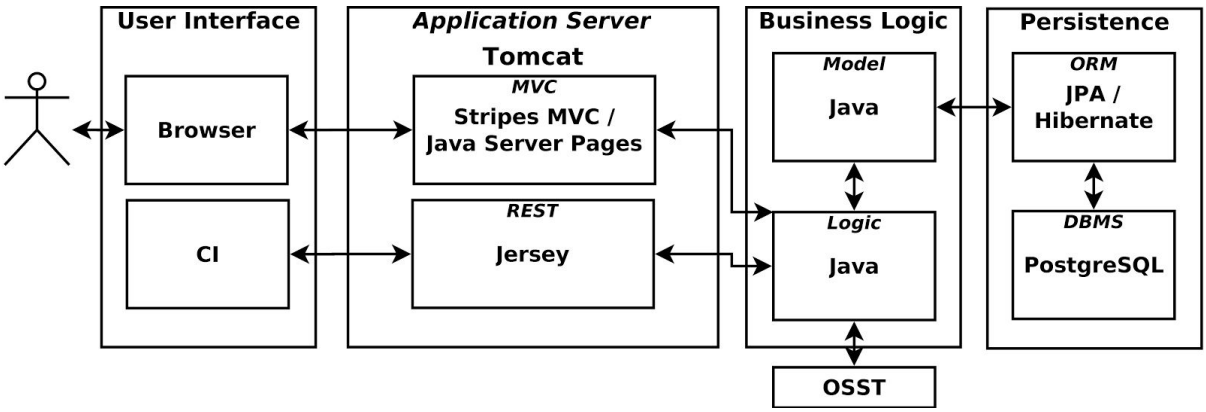
The REST functionality can be accessed via */rest/test*. All responses are given in JSON format.

- **/{id}**: Gives information about the test definition with the given id. (GET)
- **/list**: Gives information about all test definitions. (GET)
- **/{id}/execute**: Executes a test and gives information about the response. (POST)
- **/executeAll**: Executes all tests and gives information about all responses. (POST)

The URL of the OSST server can be set with the URL-Parameter *osstUrl* (default is *localhost:8082/osst*).

Architecture and components

The Architecture and components are easy-to-handle. Following we will present the schematic diagram.



Stripes is an MVC framework which defines actions and events, which can be handled by Action Beans. An Action Bean backs one or more particular views by providing data (fetched from database or other services) to a JSP file. It does not rely on XML configuration files, which makes for a less steep learning curve compared to other popular MVC frameworks like Struts or Spring.
Jersey is the reference implementation of the JAX-RS specification, which enables programmers to define RESTful services via plain Java objects and annotations.
Both of the aforementioned frameworks reside in a Tomcat servlet container. It is possible to switch to another servlet container like Jetty or a full-fledged application server like Glassfish.
Note: Stripes MVC and Jersey both include mechanisms to implement REST-interfaces. In perspective, the architecture can probably be scaled even more simply. In addition to that Hibernate is able to connect with any database you prefer. This makes the product fast and flexible. All frameworks (Stripes, Jersey, JPA) use the Java programming language, so the source has a consistent system structure.

Programmable APIs

Beyond the scope of the REST interface (which is explained above in “How to use the software”), the system does not expose any programmable APIs. Validation of XML properties is performed via XPath. It is currently not possible to add custom-defined XPath validation rules via the user interface, although this could be a useful feature for later versions.

Database and Persistence

The system uses JPA with Hibernate as a persistence provider for ORM and database abstraction. In its current state, the system uses the PostgreSQL RDBMS, although in principle, any persistence provider and any compatible database can be used. It is important to note that one annotation used in code to delete unidirectional many-to-one relationships ("@OnDelete") is specific to Hibernate and would have to be replaced with an alternative solution. To change details of the configuration, refer to the *persistence.xml* file located in the *xsd* subdirectory in the project.
The project relies on the JAXB/XJC plugin *HyperJAXB3* to generate JPA-annotated classes out of the interface description XSD-file. The classes are dynamically generated at build time. During the same process, the JPA configuration file (*persistence.xml*) stored in the *xsd* subdirectory is processed to generate JPA specific configuration for generated classes.
By default, the server connects via the JDBC URL *jdbc:postgresql:///localhost:5432/chaostesting*. In our predefined Docker environment, it connects to the database container instead of localhost. The database connection parameters can be overridden either via Java properties or system environment variables. If a variable exists in both Java properties and environment variables, the Java property will be preferred. The following properties can be overridden:

- **JPA_DB_URL**: A fully qualified JDBC URL for the system to connect to.
- **JPA_DB_USER**: The username to connect to the DBMS with.
- **JPA_DB_PASSWORD**: The password to connect to the DBMS with.

An open topic is how the system should deal with updates to the data model and the changes that it brings to the relational schema.

Bill of Materials

Component	Version
Apache Tomcat	8.5.4
Java	8
Java EE	7
Hibernate	5.2.2.Final
PostgreSQL	9.5.3
Docker	1.12.0
PostgreSQL JDBC Driver	9.4.1209 JDBC 42
Stripes MVC Framework	1.6.0
Jersey	2.32.2
HyperJAXB3	0.6.1

Links

GDocs version of this document: https://goo.gl/R4epP5