**Salesforce Tosca Automation**

After spending some time to understand the tool , how its build. Below are my recommendations with regards to how Automation structure should look and why it is so, will explain as well.

API Automation in Tosca:

**Where to get Salesforce API collection**

Salesforce provides a set of APIs we can use . You can refer APIs via workbench : <https://workbench.developerforce.com/insert.php> or Postman via github <https://github.com/forcedotcom/postman-salesforce-apis>

**Which Calls to Use**

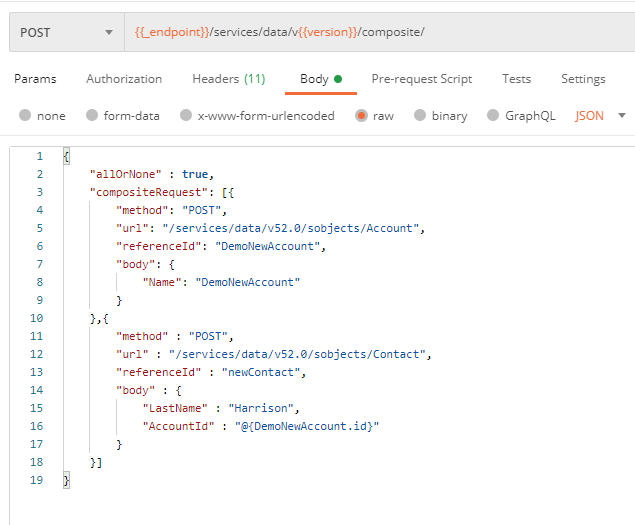
Call I will recommend for usage should be Composite call and Rest calls.

**Reason For Using specific calls**

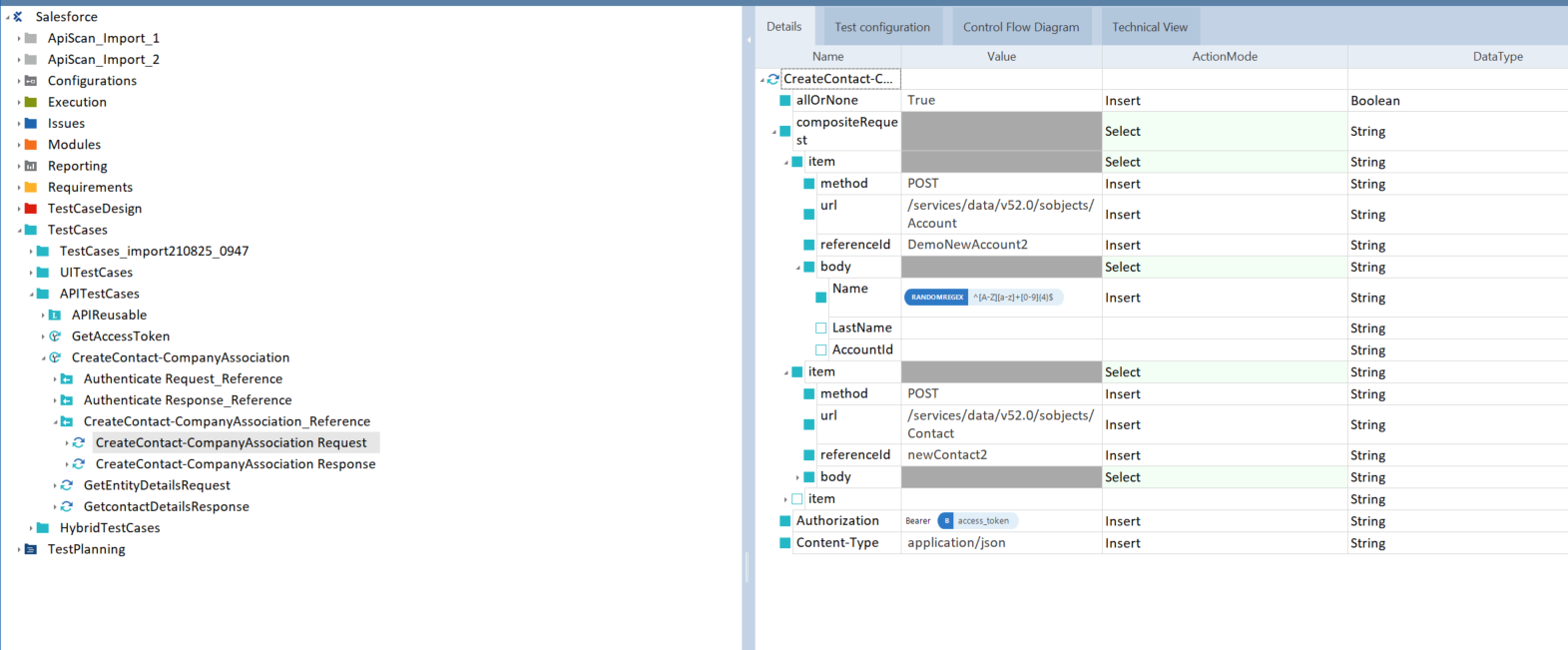
Composite call can encapsulate more than one API call and Create/Update/build relationsips across Salesforce entities. (Below call Created Account and Contact, also associates Account to Contact) .

Also Json provides a standard structure which can be across testcases

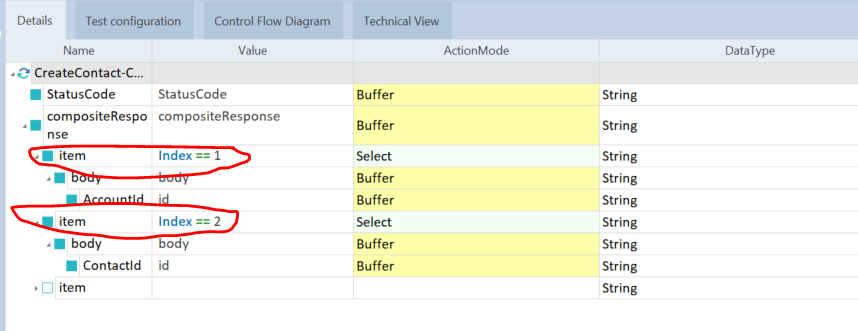
**Postman:**



**Tosca:**



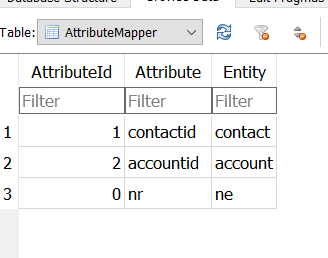
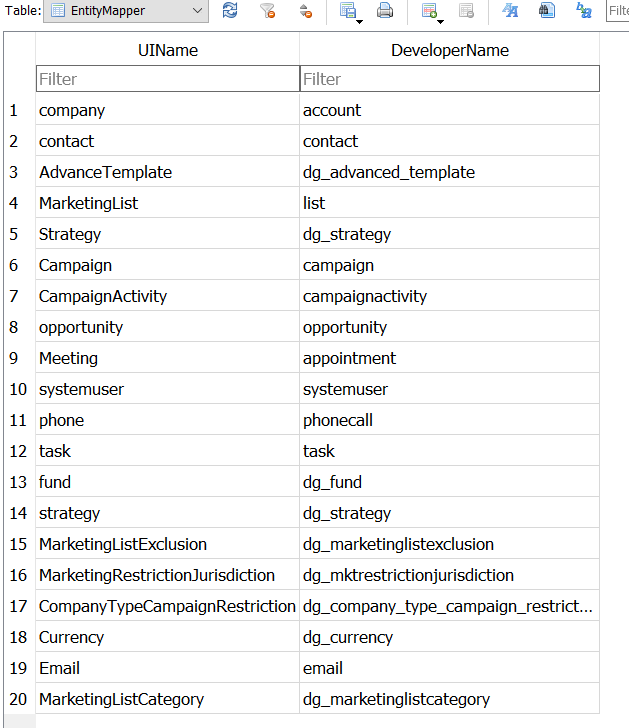
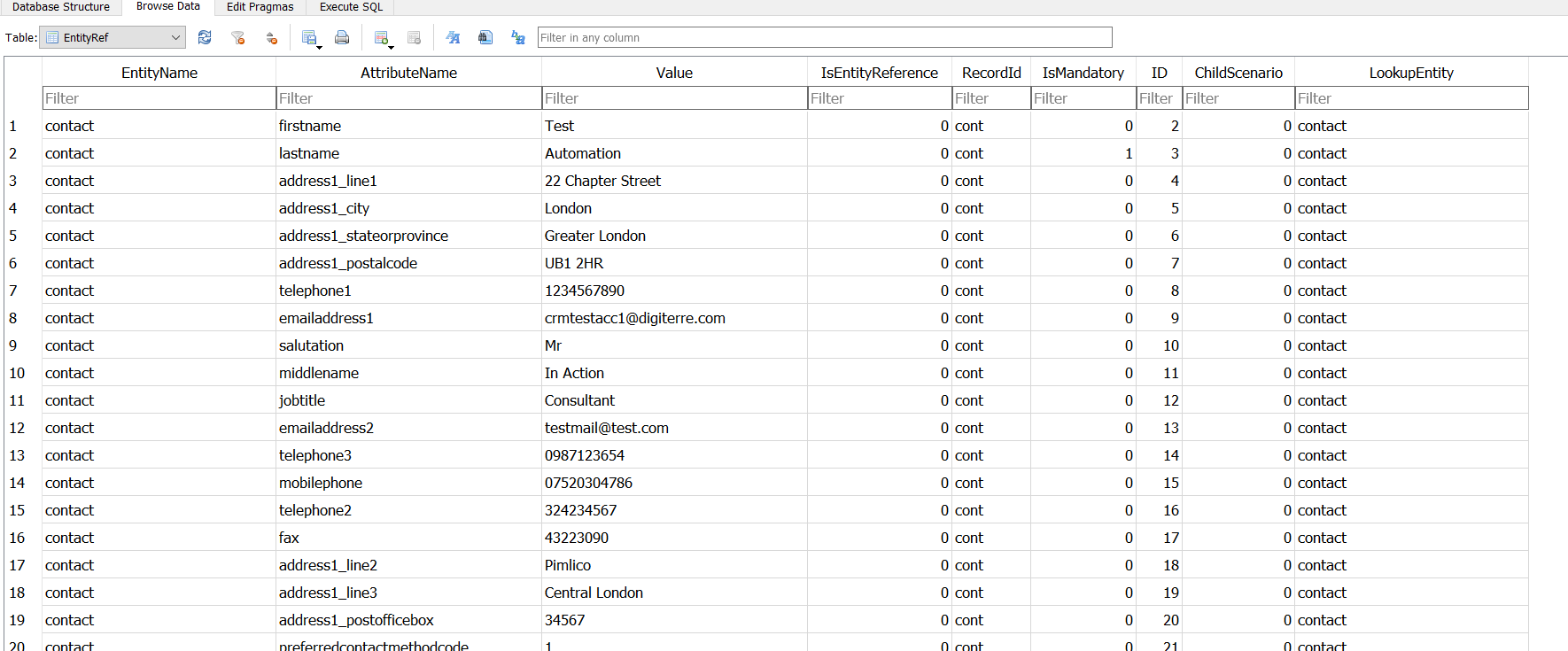
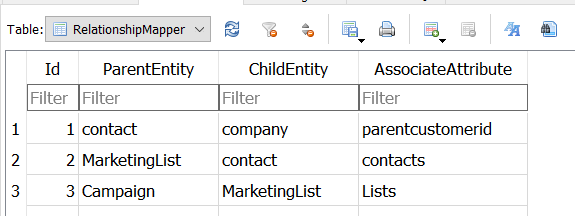
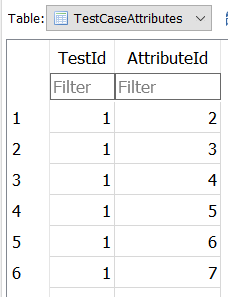
**Tool Limitation**

1. As much composite API call will make stuff generic but due to tool limitation , each Object(Contact,Account etc) will need a different Restapi step , due to different fields present in Object. As the framework will evolve API steps will increase. To keep check on steps, make sure all API calls for Object should be saved with all fields. As much it is cumbersome to start with but will give benefits in long term.
2. Multiple records returned via API had to be hardcoded with Index to retrieve record. Didn’t find any way to save it in a Object type via loop. 
3. Also realised all the UI objects are not captured with scanning , had to Zoom all tagged objects to find object(especially frontend messages as they seem to exist in Tags like <div>)

**TestData Management**

For Test Data management I will recommend using lightweight SQLite and we can fire SQL queries with Joins. Below is the structure I used for DynamicsCRM.

Tables:

1. **AttributeMapper** table (Defines primary key for each object type) 
2. **EntityMapper** (Maps UI Name with Developer/API Name for Object Type) 
3. **EntityRef**(Object Type and fields related to it) 
4. **RelationshipMapper**(Related object type for building relationship across Objects) 
5. **TestCaseAttributes(Join between TestCases table and AttributeId from EntityRef table)** 
6. TestCases(Readable TestCase name with uniqueId) 