Date: 28-07-18(Installing and Setup of SAP Hybris and Eclipse workspace)

* Installed hybris v5.7
* Apache ant home directory was set C:\hyb5.7\hybris\bin\platform\apache-ant-1.9.1
* Replaced -Xms256m to -Xmx1024m in eclipse.ini
* Set up Eclipse to follow the hybris coding conventions by downloading the four files from Trail ~ Eclipse Configuration Files and copying into<YourWorkspace>/.metadata/.plugins/org.eclipse.core.runtime/.settings (if one or more files by the same name(s) already exist in the target folder, simply overwrite them). Start/Restart Eclipse.
* Open a command line window and navigate to C:\hybris\hybris\bin\platform and execute command ./setantenv.bat(this command sets the ant environment to the one Installed in the hybris setup)
* Go to your path Hybris->bin->platform>ant clean and Press Enter
* After completion of the previous process, some folders will be automatically added to the “bin” folder
* We don't require all the extensions of the commerceSuite as of now, so we will configure to use only the hybris platform and few other extensions.
* Edit hybris/config/localextensions.xml and copy over the following content:

<!-- ext-platform -->

<extension name="admincockpit" />

<extension name="backoffice" />

<extension name="cockpit" />

<extension name="hmc" />

<extension name="mcc" />

<extension name="platformhmc" />

<extension name="mediaconversion" />

* Go to your path Hybris->bin->platform>ant all and Press Enter
* Add platform, config to Eclipse workspace
* Now go to your path hybris->bin->platform>hybrisserver.bat and Press Enter
* Open a browser and go to <http://localhost:9001>
* Click on the link Initialize

**Adding a custom project to the System**

* Now add <extension name="cuppy"/> in hybris/bin/custom folder.
* Also add the extension in eclipse YourWorkspace
* Stop the server if it is running.
* Rebuild hybris, either in the console (ant all) or in Eclipse.
* Use hybrisserver.bat or ./hybrisserver.sh to start the server.
* Be sure to do a Refresh all in the Eclipse project explorer view (so Eclipse sees the files that were generated by the ant build).
* The project should now compile successfully.
* NOTE: you should refresh the project in the Eclipse after rebuilding to reflect the changes made to the system.
* Open a browser and go to http://localhost:9001 and Go to Platform/Update. Scroll down and indicate the data to be imported as described below.

https://wiki.hybris.com/pages/viewpage.action?pageId=294094002

* Import sample data into the Cuppy Extension as shown in above link.
* Click on Update
* Done for today

Date: 29-07-18(Creating a new Cuppytrail extension)

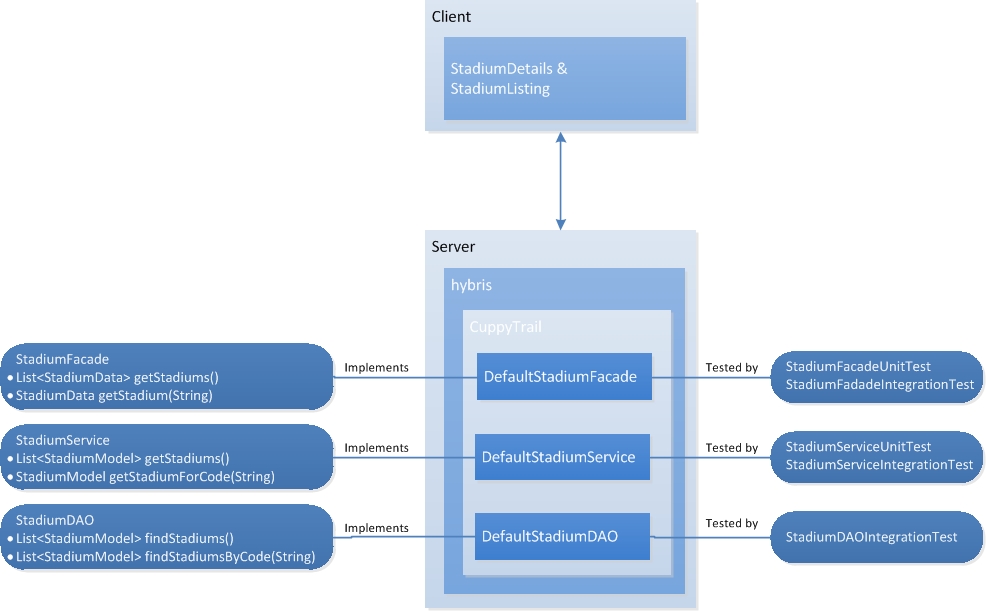
To ensure that the Cuppy extension's code remains consistent and to ensure that trainees understand how to create their own extensions, we will create a separate extension called CuppyTrail to offer the extended functionality.

We know from our basic understanding of the Architecture of the hybris Commerce Suite that:

* Web clients should communicate with hybris via a façade, using Data Transfer Objects (DTOs) to transfer data
* The façade can communicate with services in the ServiceLayer to obtain data and perform business logic
* Services in the ServiceLayer transfer the Data Entities using hybris Models
* We should separate DAO logic into separate DAOs
* All our classes should be tested (unit tests are suitable for testing classes in isolation whereas integration

tests are suitable for testing classes within a hybris "live" context)

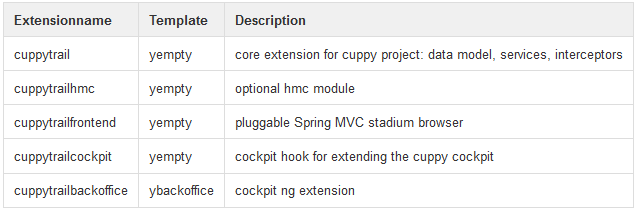
Taking all this into account, we can provide a first draft of the required solution:



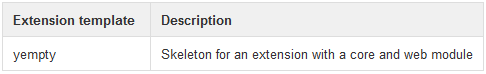
**The hybris Architecture requires a data-driven approach to software design** - i.e., a developer would start by defining the data entities, then work upwards through the layers to the frontend. Therefore, our next step will consist of creating a new Data Entity ("item") called Stadium in the hybris Data Model. We will then work upwards through the DAO, Services, Façade layers to the frontend, writing tests at each step to confirm and demonstrate correct behaviour.

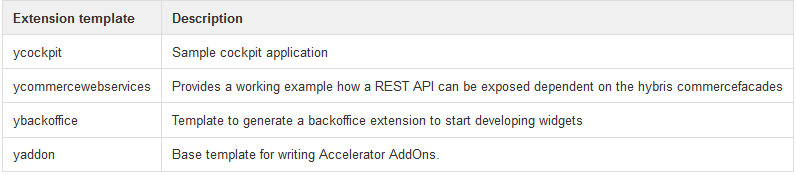
**An extension is an encapsulated piece of the hybris Commerce Suite that can contain business logic, type definitions, a web application, or hybris Management Console(hMC) configuration.**

To extend the hybris Commerce Suite, you can write a custom extension that uses existing extensions. To make this easier, hybris provides an ant-based extension generator (**extgen**) that uses a set of function-specific extension templates. We will split the **cuppytrail** project into a number of extensions to separate different functional areas, such as cockpit customization, hMC, and frontend development.



**Extension Templates**





* Now go to hybris/bin/platform> ant extgen and press Enter.
* Select **yempty** for the extension template
* Enter the name **cuppytrail** for the extension.
* Enter the package prefix **de.hybris.platform.cuppytrail**
* Append the new extension to **config/localextensions.xml**: <extension name="cuppytrail" />
* Run **ant all**
* Start the application server
* Import the generated extension into your Eclipse environment
* Open the hybris administration console <http://localhost:9001/platform/extensions>, you can see **cuppytrail** contains a core module and a web module.
* we do not need a web module for this basic extension, we will disable this in **cuppytrail/extensioninfo.xml**

<!--

<webmodule jspcompile="false"

webroot="/cuppytrail"

/>

-->

* Run **ant all**
* Run the Application Server again and check the Webroot in <http://localhost:9001/platform/extensions> has been

disabled

Note: You may come across the situation that ant is working with no errors but you get Eclipse errors like *...***Model not found**. This might refer to the fact that ant has generated classes which are not visible to Eclipse. If this happens, press F5 to refresh all projects in your workspace

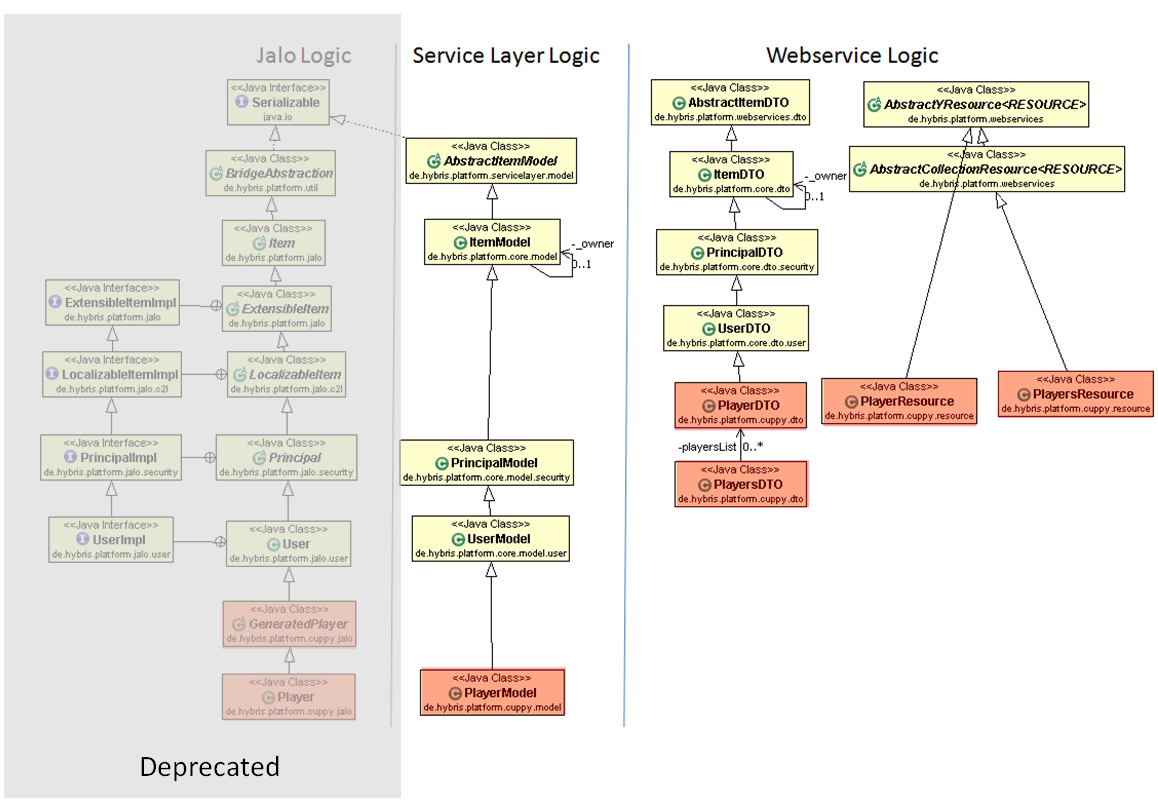
Date: 29-07-18(New Data Model)

The data model is fundamental to an extension and defines the "problem domain" or structure of your application. **Business logic is always located on top of the data model**. This step shows how a data model depending on another extension is defined.

**Background on Data Models**

Data Models are defined in the <extension>-items.xml file of each extension. Essentially data entities (called "items" in hybris) are defined with itemtype elements, whereas relations between items are defined with relation elements. For example, open **cuppy/resources/cuppy-items.xml**and have a closer look at the numerous relation and itemtype definitions which refer to the Cuppy Data Model.

The hybris build system extracts the**\\*-items.xml** files located in all extensions and generates for every itemtype a number of Java Source files to support access to these entities from within the hybris framework (most notably via Java calls in the ServiceLayer as well as via RESTful URIs). The **red classes** in the diagram below show the files created for the **Player** itemtype in cuppy-items.xml. The classes from which they are derived are shown in **yellow**.



The key point to bear in mind here is that many classes are generated from itemtypes and fall into three categories:

* Model classes created for use in the ServiceLayer where partners are encouraged to develop their business logic. Below, we will specify a new Itemtype **Stadium**, and hybris will generate several files including a StadiumModel Java class that we can use.
* Webservice-related classes which are created to support CRUD logic via RESTful URIs. **These classes are only generated when the optional extension platformwebservices is included in your configuration.**We will see how the Java classes StadiumDTO(s) and StadiumResoure(s) are used in the webservice trail step.
* Low-level Jalo classes. We will not address these classes as their use is discouraged.

As depicted in the mockup, we would like to represent a new entity - Stadium - and for each Stadium a list of its matches. Matches are already defined in the **Cuppy** extension. Therefore, our main tasks are to:

* add a new item in cuppytrail-items.xml defining the new data entity Stadium.
* add a relation in cuppytrail-items.xml creating a 1-to-many relationship between Stadium and Match
* add an enum type in cuppytrail-items.xml creating a Stadium Type and Stadium Access; we are creating two new enumtypes
* add default values in cuppytrail-items.xml
* add default values to the types created

**Adding dependency between Cuppy an Cuppytrail**

As the data model will depend on cuppy's data model, set the dependency in **extensioninfo.xml**and also in the Eclipse build path:

* Tell Eclipse that the **Cuppytrail** extension depends on the **Cuppy** extension: right click on cuppytrail - build path - configure build path
* Add cuppy to Projects and then click Apply and Close.
* Tell the hybris build framework that the Cuppytrail extension depends on the Cuppy extension. Open cuppytrail/extensioninfo.xml and add **<requires-extension name="cuppy"/>** as the first element within the extension element.

**Add Stadium to cuppytrail-items.xml**

* Replace cuppytrail/resources/cuppytrail-items.xml with the following XML declarations describing the new Stadium itemtype and its attributes,

<items   xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

            xsi:noNamespaceSchemaLocation="items.xsd">

    <itemtypes>

        <itemtype code="Stadium" generate="true" autocreate="true">

            <deployment table="CuppyTrailStadium" typecode="10123" />

            <attributes>

                <attribute qualifier="code" type="java.lang.String" >

                    <persistence type="property"/>

                    <modifiers optional="false" unique="true"/>

                </attribute>

                <attribute qualifier="capacity" type="java.lang.Integer">

                    <description>Capacity</description>

                    <persistence type="property" />

                </attribute>

            </attributes>

        </itemtype>

    </itemtypes>

</items>

* Save this file run ant all.
* Refresh Eclipse workspace (all projects)

More Information regarding the XML declarations used can be found at: <https://wiki.hybris.com/pages/viewpage.action?pageId=294094110>

**Relations**

* Define a new relationship StadiumMatchRelation between Stadium and Match using the relation tag
* Define a new relationship StadiumMatchRelation between Stadium and Match using the relation tag

<relations>

    <relation code="StadiumMatchRelation" localized="false" generate="true" autocreate="true">

       <sourceElement type="Stadium" qualifier="stadium" cardinality="one" />

       <targetElement type="Match" qualifier="matches" cardinality="many"/>

    </relation>

</relations>

* the relation is a one-to-many relation. which means that the mapping will be done by an additional column on the many side, i.e. the table Match

**Enumtypes**

* Add the following element before the relations element: **cuppytrail/resources/cuppytrail-items.xml**

<enumtypes>

<enumtype code="StadiumType" autocreate="true" generate="true" dynamic="false">

<value code="openair"/>

<value code="enclosed"/>

</enumtype>

<enumtype code="StadiumAccess" autocreate="true" generate="true" dynamic="true">

<value code="road"/>

<value code="rail"/>

<value code="plane"/>

</enumtype>

</enumtypes>

**Default values**

* Add the attribute **StadiumType** to the list of two existing **Stadium** attributes and define the default value to be openair :   **cuppytrail/resources/cuppytrail-items.xml**

<attribute type="StadiumType" qualifier="StadiumType">

<persistence type="property"/>

<defaultvalue>em().getEnumerationValue("StadiumType","openair")</defaultvalue>

</attribute>

**Rebuild the Project**

* Save this file and run **ant all**.
* Refresh Eclipse workspace (all projects)
* Refresh the platform project in Eclipse and navigate to **platform/bootstrap/gensrc/de/hybris/platform/cuppytrail/model/StadiumModel.java**