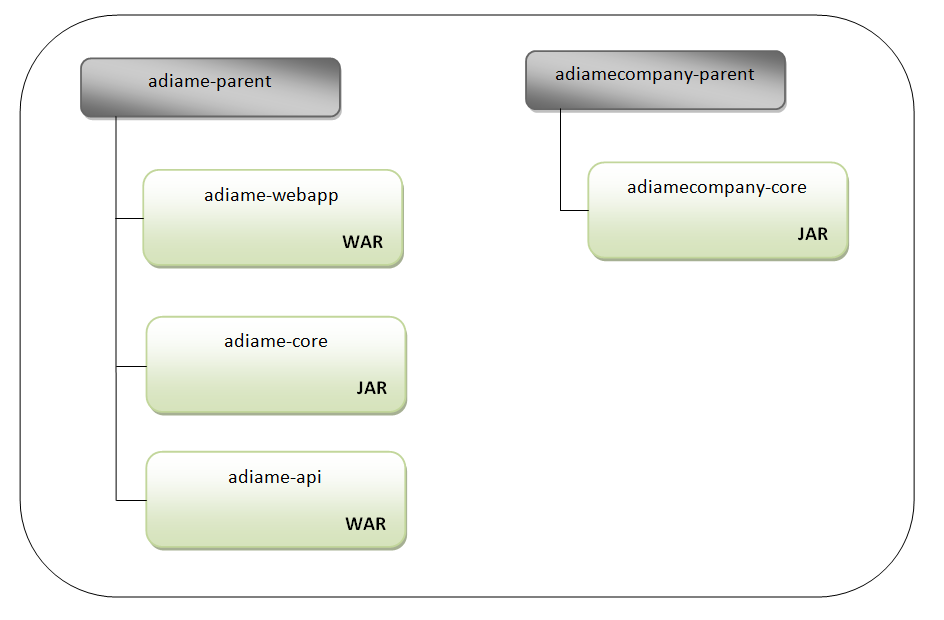
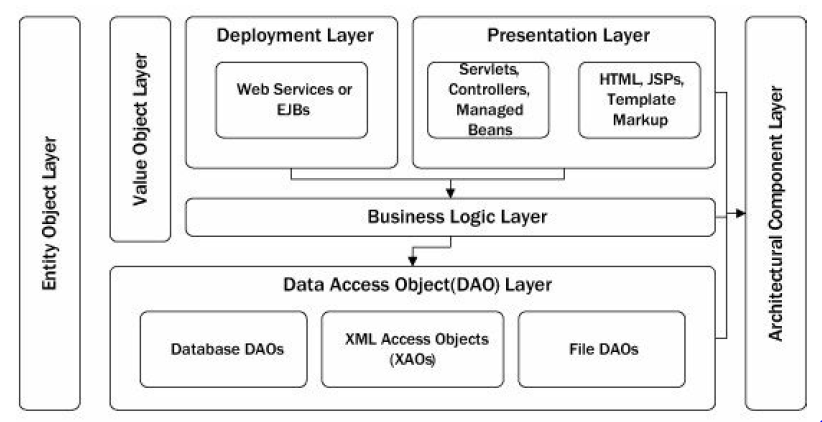
**Defining the Adiame project structure with Maven**

Here, we will focus on defining, with Maven, the project structure we need for our application (Adiame).

* The application project adiame-parent will have three modules. Two of them will be packaged as web archives (war): the main web application (adiame-webapp) and the REST API (adiame-api). One of them will be packaged as a jar dependency (adiame-core).
* The company-specific project adiamecompany-parent will have only one sub-module (adiamecompany-core), which will be packaged as jar.

**ADIAME Architecture**



1. adiame-webapp

|  |  |
| --- | --- |
| **Layer** | **Contents** |
| Presentation layer | Css3, javascript 1.8,bootstrap 3.3.6, html5, xhtml(html5 written as XML), jsf 2.2 |

1. adiame-api

|  |  |
| --- | --- |
| **Layer** | **Contents** |
| deployment layer | Web service: RESTful (jax-rs) and WSDL Bindings (jax-ws). Implementation Apache CXF. |

1. adiame-core

|  |  |
| --- | --- |
| **Layer** | **Contents** |
| Business logic layer + DAO layer | Pojo (Value objects) , ejb 3 (stateless, stateful, singleton, MDB), jpa entities (Entity objects), Apache Commons(reusable Java components) |

1. adiamecompany-core

|  |  |
| --- | --- |
| **Layer** | **Contents** |
| Architectural component layer | Java utilities classes, Apache Commons(reusable Java components) |

**ADIAME package structure**

|  |  |
| --- | --- |
| com.adia.adiame.dao | Data access object layer |
| com.adia. adiame | Business logic layer |
| com. adia. adiame.entity | Entity objects |
| com. adia. adiame.vo | Value objects |
| com. adia. adiame.ui | Presentation layer |
| com. adia. adiame.util | Architectural component layer |
| com. adia. adiame.services | Deployment layer/web services |

**ADIAME Architecture Strategies**

**Logging Strategy**

* Use com. adia. adiame.util.Logger for all logging. Do not use System.out.println().
* Do not log error messages directly in application code. Rely on the application architecture (e.g., error handling servlet filter) to log.
* Warnings, informational, and debug messages (i.e., errors not severe enough to warrant the throwing of an exception but that would be useful to a developer fixing bugs) can be logged anywhere from any layer.
* When using logging to output information, use logger message formatting features, instead of employing custom formatting logic, to reduce application code and the risk of producing derivative exceptions.
* Do not use the general logging facility as a transaction log.

**Exception-handling Strategy**

* Use Validate from Apache Commons Lang to check all method arguments on public methods and constructors.
* Always include enough information in the message of the exception to duplicate the condition in a testing environment.
* All application exceptions should extend ContextedRuntimeException from Apache Commons Lang.
* All try/catch blocks in the business logic layer or the data access layer should not interfere with the throwing of unchecked exceptions. Throw the exception to the caller instead.

**Asynchronous Task Strategy**

* Extend com. adia. adiame.util.AbstractBatchJob for all asynchronous tasks. Do not spawn asynchronous threads individually.
* Do not log your own exceptions; the architecture will ensure that any exceptions you throw are properly logged, along with specifics about the job and any execution parameters used.
* Do not manage your own transactions; a commit will be executed on successful completion and a rollback will be issued should an exception be generated.

**Configuration Guidelines**

* Define all configuration values as properties in class com. adia. adiame.util.Configuration.
* Ensure that logic to set the configuration value is present in class com. adia. adiame.util.Configurator. If possible, ensure that the configurator assigns a default value for the property in case it’s not specified in a particular runtime environment.
* Throw an exception in the configurator should the configuration value not be set to an allowed value. Please include the invalid value and a blurb about what valid values are in the error message.

**CACHING STRATEGIES**

Example:

private static ExpiringCache<Long, String> cache = new ExpiringCache<Long, String>();

.....

String value = cache.get(key);

if (value == null) {

// Put code to look up the value

cache.put(key, value);

}

**TESTING GUIDELINES AND STRATEGY**