GWT Showcase Tutorial

Version 1.0

- 05 May 2013

Table of Contents

AIM OF THIS DOCUMENT

The aim of this document is to provide the implementation details of the Appverse Web GWT showcase application. We will show you how to tackle basic and important aspects that most of the applications require such as: i18N, data validation and errors management, security, logging, styling, etc.

WHO CAN READ THIS DOCUMENT?

Anyone interested in developing applications with Appverse Web and GWT front-end is encouraged to read this document, especially developers. However, you need to be aware that this is a quite technical document. In order to be able to follow the explanation and have a deep understanding of the technical details a good knowledge in Java basics, web development, GWT and Appverse Web back-end devolpment is required.

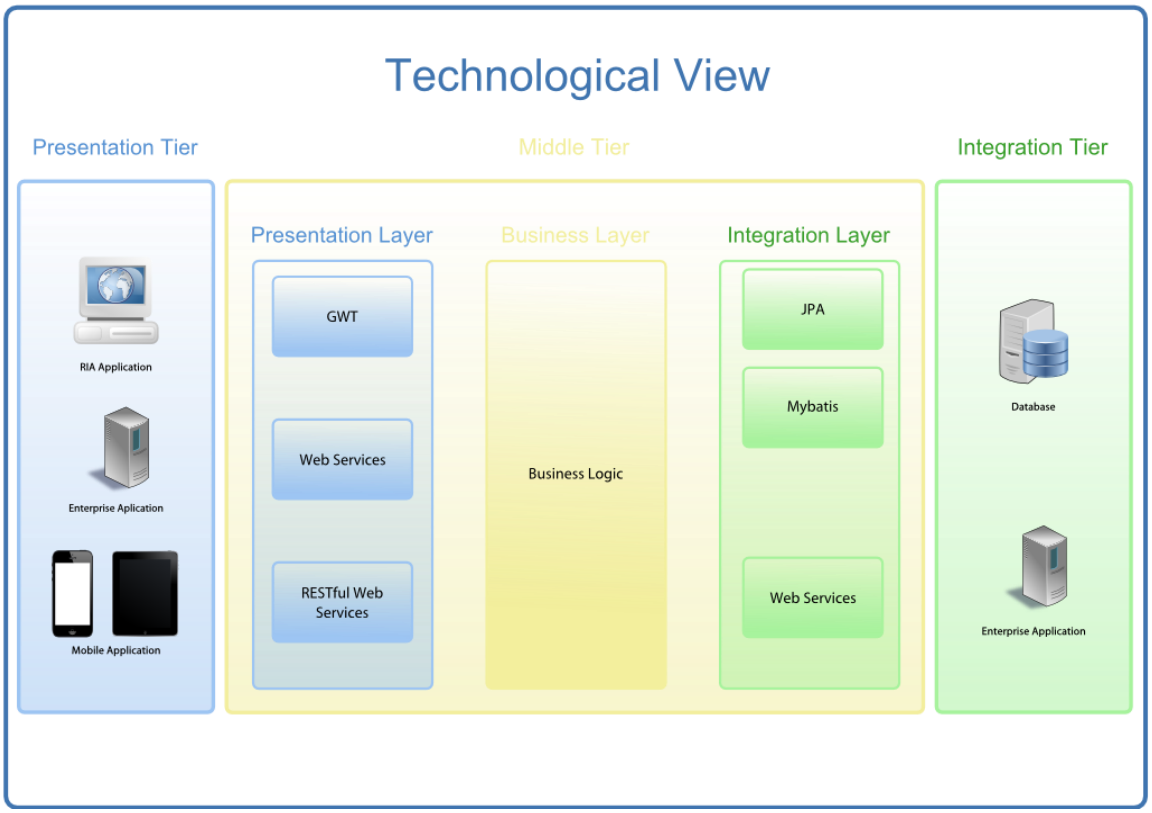
It is highly recommended to read the following Appverse Web documentation before starting this document. Some aspects will be just commented quickly as you can find further explanations here:

http://appverse.github.io/appverse-web/guide/index.htm

1.1. GWT for client-side development

Appverse Web is built using a multi-tier architecture pattern, either using a three-tier architecture or being part of a more complex or simpler N-tier architecture.

For this showcase, we are going to focus on the three-tier architecture traditionally used for web development:



We will not give an extense explanation of N-tier architectures as this is out of the scope of this document. If you want further details regarding this matter and how Appverse Web supports this, please read the following section of the document:

(ADD LINK HERE).

The point we would like to make with the previous picture is that GWT fits perfectly in the Presentation Layer of the Middle Tier for RIA Applications and Presentation Tier for RIA Applications development. The Presentation Tier is responsible of physically data rendering, providing a visual representation of the data feed by the middle tier into presentation flavour specifics and reacts to the user interaction calling to the middle tier in order to perform transactions and build a response. From a logical point of view, the Presentation Layer is providing support to the Presentation Tier to achieve its goal. Having said that, it is important to note that GWT offers a client-server architecture with the possibility of keeping state in the client. JavaScript code running in the client side belong to the Presentation Tier whilst GWT services (both RESTFul JSON or GWT RPC services) belong to the Presentation Layer in the Middle Tier.

GWT is one of the possible flavours for the Presentation Layer. For instance, you could develop an application based on RIA paradigm using GWT (GWT flavoured) instead of JSF2 (JSF2 flavoured) depending on your requirements. You would need to take into account that JSF 2 and GWT are completely different technologies with its advantages and disadvantages.As you can see in the picture

above, the Business and Integration layers are not dependant on the technology used in the presentation layer. This allows us to use GWT as a mere implementation of the presentation layer that delegates on the Business Layer (completely agnostic of the Presentation Layer) in order to provide the service layer to the client GWT-based GUI by means GWT RPC or RESTful JSON.

1.2. Why GWT ?

It is out of the scope of this document to compare different web technologies or frameworks that allow you to develop rich internet applications.

We believe that depending on the requirements, GWT could be a perfect technology to develop a Rich Internet Application (RIA) based on a client-server model. It could be a good alternative to other technologies such as JSF2. However, as it has been commented earlier, each technology has its advantages and disadvantages. For instance, GWT presents a pure client – server architecture with the client (user browser) being able to keep status and making GWT RPC calls or JSON based calls to the service layer in the server side. Thus, you can make most of your services stateless. On the other hand, the fact that GWT follows a client – server architecture obliges you to be aware of the data serialization process between server and client. Architectures based on JSF2 are completely different in the sense that it is not a client – server architecture. In this case, all the processing it is performed in the server side, like traditional servlets or JSP that return directly the HTML code to the browser. In this regard, JSF2 makes you to be aware about the fact that status has to be kept between HTTP requests – even though that JSF2 lifecycle is precisely designed to make this almost transparent for you. On the other hand, with JSF2 you will not experience serialization issues as all the processing is carried out in the same JVM. In summary, these are some things to take into account in order to choose JSF2 or GWT: history management required by the application, if the HTML code has to be crawlable by search engines, scalability, status to be kept by the application, richness of the UI components offered by the plain technology or third parties that you might use, new components development effort, developers skills and so on.

For RIA (and web development in general) we recommend you to develop your presentation layer using either Google Web Toolkit or JSF2.

It is higly recommendable to use a third party GWT UI component framework to develop the user interface. The reason is that the components they offer are richer than plain GWT components. Appverse Web team has choosen Sencha GXT 3 as UI component framework. This does not mean that you could not use Appverse Web with plain GWT components or another UI component framework of your choice.

1.3. Why Sencha GXT 3?

Developing and maintaining GWT components takes time and effort. For this reason, we recommend you to use third party libraries as long as the components fit your application needs. If not, you can always develop a new component or extend an existing one.

There are several GWT GUI components libraries both for free or for a fee.

We have choosen Sencha GXT 3 for our showcase because:

- It is developed and supported by Sencha. This ensures there is a company improving, evolving and fixing the product, providing new components and giving support in forums. Sencha GXT 3 is widely used and there is a big community.
- It offers both commercial and open source licenses. Open source license type is "GNU GPL LICENSE v3"
- It offers a rich showcase of components providing client side JSR-303 validation integration which helps to develop rich Internet applications
- It supports theming, separating component behaviour and appearance

2.1. Application aspects covered by this showcase

The own showcase application shows a list with the different application aspects that it is covering:

- GWT and Spring 3.x integration
- Maven project layout, dependencies and plugins setup
- Appverse Web three layer architecture
- In-memory database setup
- Maven plugin setup to generate database scripts from JPA annotated beans
- Spring security integration
- I18N
- JSR-303 field validation in client
- JSR-303 server side validation
- Errors management
- Remote pagination and sorting
- GWT History management

3.1. Project GitHub repository

The showcase application is based on standard Maven layout. This mean you can download the project from GitHub repository and set up the project in your favourite IDE.

https://github.com/Appverse/appverse-web.git

3.2. Configured servlet containers

For this showcase we have set up Maven configuration to be able to deploy and run the showcase in two modes:

- Using Eclipse GWT SDK (it uses an embedded Jetty servlet container)
- External Tomcat 7 servlet container with Maven deploy plugin setup (requires a external Tomcat 7 servlet container to be set up).

Note: Take into account that by default we specify Java version 1.7 in pom.xml to compile the source code.

3.2.1. Running the application in GWT development mode (using Eclipse IDE)

The root Maven module contains an Eclipse Launch configuration called "Start GWT Showcase Dev Mode" to run or debug the application in GWT development mode.

Please note that you need to have GWT plugin correctly setup in your Elipse environment to be able to start the project using the provided Eclipse launch configuration.

In order to start the project in development mode, just run the following Eclipse launch configuration in run or debug mode:

Appverse-web-showcases-gwt -> Start GWT Showcase Dev Mode.launch

3.2.2. Running the application in an external Tomcat 7 servlet container (production mode)

This option requires an external Tomcat 7 servlet container and to configure a user and password to be able to do remote deployments using Maven plugins by the Tomcat Manager URL. How to do this is out of the scope of this document.

In order to build, deploy and run your application in a Tomcat 7 servlet container, please run the next Maven goals:

```
clean package tomcat7:deploy-only
```

or, if you prefer, replace the default goal with:

```
<defaultGoal>clean package tomcat7:deploy-only</defaultGoal>
```

Once Tomcat servlet container has started please enter the following URL in your browser to access to the application:

http://127.0.0.1:8888/admin.html

4.1. Maven modules

The project follows a standard Maven multimodule layout.

The fact that we have splitted up the project in different modules allows to have backend and frontend projects clearly separated (corresponding to server and cliend sides). The items that need to be known both for backend and front end sides are placed in a "shared" project (for instance GWT RPC interfaces and presentation model).

Let us take a closer look at the three modules:

- **appverse-web-showcases-gwt**: is the root Maven project. Contains the setup that is common to all modules and the modules definition itself.
- **appverse-web-showcases-gwt-backend**: correspond to the Java server-side part of the project. Comprises presentation, business and integration backend layers.
- **appverse-web-showcases-gwt-frontend**: correspond to the GWT front-end part of the project. Comprises client side presentation layer.
- **appverese-web-showcases-gwt-shared**: contains the Java classes that both front-end and back-end need to know, basically interfaces that the client side needs to know to call back-end services (by means RPC or Rest/JSON) and the presentation model.

The server side provides a service layer to the GWT client side. GWT supports asynchronous calls to the server (by means GWT RPC or Rest/JSON). This allows the GWT client to focus just in the GUI making asynchronous calls to the server side service layer to obtain / manipulate data and modifying DOM so that the browser is able to show changes in the page without reloading it. The fact that is a client – server model allows you to keep state in the client.

The point of having this separation between client and server (service layer) is that the service layer would be reusable in the event of migrating from one frontend technology to another or even reused in a multichannel or multifrontend application. The only service layer that could be tied to a specific frontend is the Presentation Service layer (only if necessary). This is the key point that allows you to keep your business and integration layer completely agnostic and reusable regardless the frontend flavour.

4.2. Back end project

Let us review the back end project structure that provides the service layer to the client as it has been commented earlier.

In the next sections we are going to review the structure for:

- Java packages (src/main/java)
- Application resources (src/main/resources)
- Java test packages (src/test/java)
- Test resources (src/test/resources)
- Webapp (src/main/webapp)

We will just comment the structure of the project quickly. We will not go in detail for every file.

Do not worry if you do not understand exactly what a configuration file is for. The important thing is that you are able to understand the project structure and we will go in detail in further sections.

4.2.1. Java packages

Following standard Maven layout, we place Java packages in src/main/java folder.

The table below shows the showcase application packages structure. This is the structure we recommend you to follow in your projects. This is the structure you would obtain if you used an Appverse Web archetype to generate a project.

All subpackages are included into main project package:

- Main project package: org.appverse.web.showcases.gwtshowcase

4.2.1.1. Backend subpackages structure

Let us review the subpackages structure for backend module.

Package	Subpackage	Description
[MAIN].backend.services	presentation.impl.live	This package holds all presentation services live implementations. Presentation Services are the only "backend" classes that might be dependant on frontend technology.
	business	This package holds all business services interfaces.
	business.impl.live	This package holds all business services live implementations. Business Services are Presentation Layer agnostic.

Package	Subpackage	Description
	integration	This package holds all integration services interfaces
	integration.impl.live	This package holds all integration services live implementations
[MAIN].backend.model	business	This package holds all business model objects
	integration	This package holds all integration model objects
[MAIN].backend.converters	p2b	This package holds all presentation to business converters. Converters work in both ways, converting model objects from presentation layer to business and the other way around.
	b2i	This package holds all business to integration converters. Converters work in two directions, converting business model objects from business layer to integration and the other way around.

Backend subpackages structure

4.2.2. Application resources

Following standard Maven layout, we place resources in src/main/resources.

Let us review the resources folder structure:

Folder	Description	Contents
dozer	Contains Dozer mappings setup used for b2i and p2b converters.	b2i-bean-mappings.xml Business to integration dozer converters mappings.
		p2b-bean-mappings.xml Presentation to business dozer converters mappings
Log4j	Log4j setup	log4j.properties
META-INF	Directory where the different providers require their setup	orm.xml JPA mapping file

Folder	Description	Contents
	to be stored	persistence.xml JPA file
		persistence-dll.xml File used by Appverse Web "appverse-web-tools-jpa-ddl-generator" Maven plugin. This plugin use this configuration file in order to generate the database Data Definition Language (DDL) scripts automatically from the annotated JPA entities.
properties	Application property files	db.properties Database connection info. The real values are over-ridden in this file by Maven depending on the current Maven profile. This makes distribution for different environments easier.
spring	Spring framework config files	application-config.xml Spring context configuration file. Includes specific database config and security files.
		database-config.xml Default HSQLDB in-memory database out of the box setup.
		security-security.xml Spring security specific setup file
sql	Contains automatically generated Data Definition Language (DDL) scripts by Appverse Web "appverse-web-tools-jpa-ddl-generator" Maven plugin.	

Resources folder structure

4.2.3. Java test packages

TODO: There are not test in the showcase yet

4.2.4. Test resources

TODO: There are not test in the showcase yet

4.2.5. Webapp folder

Following Maven standard layout, web application sources are stored in /src/main/webapp folder.

Let us review the structure:

Folder	Description	Contents
[ROOT FOLDER]	This is the standard Maven layout for web application sources.	Login page and logo image.
/WEB-INF	Standar WEB-INF directory for web applications	web.xml Deployment descriptor file
		dispatcher-servlet.xml Spring MVC dispatcher servlet

Webapp folder strcuture

4.3. Front end project

4.3.1. Modular structure and patterns

The front end architecture follows Google recommendation of using a R-MVP (Reverse MVP) pattern implementation supported with Command pattern and Dependency Injection. Besides the advantatges that this approach offers regarding manteinance, reusability and testing, we have designed a standard project structure that will provide organization taking into account the usage of these patterns.

We use MVP4g framework which offers a good implementation of the MPV pattern and offers dependency injection using GIN:

http://code.google.com/p/mvp4g

4.3.2. About GWT modules

A glimpse about GWT modules organization in general:

- A GWT module is an individual bundle of configuration settings needed in your GWT project.
- Your GWT project might have several modules.
- When the GWT compiler compiles a module includes all the specific setup, static components (images, etc), localized files (constants, messages) and the generated JS code corresponding to this specific module.
- Every module has its own package and there is a complete separation between modules. Modules, by default, do not share resources.
- A module can inherit from another module. This will allow two modules to inherit the same module and thus share common resources and code.
- Every module has a XML module descriptor file (specifying module name, source path, inherited modules, etc).
- A module can have from none up to several Entry-Point classes. When a module is loaded onModuleLoaded() method is called for every Entry-Point.
- We can use different modules in an application to improve the performance and separate completely independent functionalities. This way we can have a two modules for two different menu options (different functionalities) and the resources are not loaded into the browser until every option is accessed. This reduces the initial loading time.

4.3.3. GWT modules organization recommendation

We recommend you to follow the following indications to organize your front end project:

- Use GWT modules to structure your application taking into account different functionalities that it offers

- Have a common package that all the application modules will use providing common support to the rest of the modules of the application (application constants, messages, common functionalities).
- Structure your modules in subpackages that we will call "sections", if necessary, following the same structure for all of them.

Having said that, we would like to show you the structure we propose for the "common" package and for each of the modules in your application. This is the structure we have used for this showcase.

In the next sections we still consider the main project package ("MAIN") your root package. In our case:

- Main project package: org.appverse.web.showcases.gwtshowcase

4.3.3.1. "Common" package structure

There is a "common" package (not a GWT module) with artifacts that are visible to all the GWT modules. The common package has the following structure:

Package	Description
[MAIN].gwtfrontend.common	Contains common application resources. At least global application constants, messages, injectors and utility classes: • ApplicationConstants.java • ApplicationMessages.java • ApplicationInjector.java • ApplicationGinModule.java • ApplicationImages.java • ApplicationUtils.java

Common package structure

4.3.3.2. "Common" resources

Following standard GWT layout, we will have a parallel structure in /src/main/resources in order to place the resources that the "common" package requires: property files, images, xml.

Resources folder	Description
[MAIN]/gwtfrontend	Contains all the application modules descriptors following the format: **[ModuleName.gwt.xml]**
[MAIN]/gwtfrontend/common	Contains "common" application resources files. For instance: • ApplicationConstants.properties • ApplicationMessages.properties

Common resources folder

4.3.3.3. GWT modules structure

For modules we suggest a repetitive structure so that all the modules are symetric. As it has been commented previously, your application can have as many modules as necessary.

For organizational reasons, we also suggest you divide your modules in what we call "sections". Sections are nothing else than subpackages to help you organize big modules. Small modules might not need sections (like the one we show you in this showcase). Classes or resources that are used by different sections will be placed in the root package of the module and in a package called "common". Take into account that this "common" package is shared by all the sections of the module, do not mistake this with the "common" package for the application. They are placed at different scopes.

4.3.3.3.1. Module root and "common" module packages structure

There will be a parallel structure in /src/main/resources in order to place the resources that the module root and "common" module package requires: property files, images, xml.

Package	Description
[MAIN].gwtfrontend.**[module]**	Common module resources and classes: • **[Module]**Constants.java • **[Module]**Messages.java • **[Module]**Images.java An specific Event bus for this module: • **[Module]**EventBus.java
[MAIN].gwtfrontend.**[module]**.common.commands	Interfaces of common commands used in different sections of the module. Commands encapsulate service calls to the server (by means GWT RPC or Rest/JSON) • **[CommandName]**Command.java
[MAIN].gwtfrontend.**[module]**.common.commands.impl.live	Live implementation of common commands used in different sections of the module • **[CommandName]**CommandImpl.java
[MAIN].gwtfrontend.**[module]**.common.commands.injection	Basic DI (Dependency Injection) setup using Google GIN: • **[Module]**GinModule.java • **[Module]**Injector.java
[MAIN].gwtfrontend.**[module]**.common.layout	Root folder for presenters and views corresponding to the particular module layouts (templating)
[MAIN].gwtfrontend.**[module]**.common.layout.presenters.interfaces	Interfaces of layout presenters and module layout manager
[MAIN].gwtfrontend.**[module]**.common.layout.presenters	Implementations of layout presenters and module layout manager
[MAIN].gwtfrontend.**[module]**.common.layout.	Views implementations corresponding to module layouts

Package	Description
views.impl.gxt	(templating)
[MAIN].gwtfrontend.**[module]**.history	Module history converters for GWT history support

Module root and common package structure

4.3.3.3.2. Module sections

Every module can have as many sections as necessary in order to structure the project properly. Every section will have exactly the same structure:

Package	Description
[MAIN].gwtfrontend.**[module]**.**[section]**.commands	Command interfaces: • **[CommandName]**Command.java
[MAIN].gwtfrontend.**[module]**.**[section]**.commands.impl.live	Command implementations: • **[CommandName]**CommandImpl.java
[MAIN].gwtfrontend.**[module]**.**[section]**.presenters.interfaces	Interfaces that all view implementations of a particular presenter need to accomplish. All the view interfaces include an inner presenter interface so that the views are able to invoke methods over its presenter according to Reverse MVP implementation: • **[PresenterName]**View.java
[MAIN].gwtfrontend.**[module]**.**[section]**.presenters	Presenter implementations: • **[PresenterName]**Presenter.java
[MAIN].gwtfrontend.**[module]**.**[section]**.views.impl.gxt	View implementations: • **[ViewName]**ViewImpl.java : View implementation • **[ViewName]**ViewImpl.ui.xml : UiBinder declarative layout
[MAIN].gwtfrontend.**[module]**.**[section]**.editors.impl.gxt	Editor views using GWT Editor framework: • **[BeanToEditName]**Editor.java : Editor implementation • **[BeanToEditName]**Editor.ui.xml: UiBinder declarative layout

Module section structure

4.4. Shared project

4.4.1. Java packages

Let us review the subpackages structure for the shared module.

Package	Description
[MAIN].backend.services.presentation	This package holds all presentation services interfaces the client needs to call a server side servide by means GWT RPC or Rest/JSON.
[MAIN].backend.model	This package holds all presentation model objects known by client and server side and used to transfer data between them.
[MAIN].backend.constants	This package holds common constants classes known by client and server side

Shared subpackages structure

4.4.2. Resources

Resources folder	Description
[MAIN]/gwtfronted	This package holds the Shared GWT module file descriptor (Shared.gwt.xml). It is necessary to specity the GWT source paths for the corresponding subpackages in the shared project.

Shared resources folder

5.1. Root maven module project configuration

The GWT showcase application follows a standard Maven project layout.

Let us review the most important things that you should know.

5.1.1. Configured servlet containers

In previous sections we have explained you how to build, deploy and run the application either in an embedded Jetty container (provided by GWT SDK for Eclipse) or external Tomcat 7 container.

Let us review the plugins configuration:

5.1.1.1. Java compilation version in Maven Compiler Plugin

By default, the project Java compilation version is 1.7. If you needed to change it, the maven-compiler-plugin allows you to specify the "source" and "target" compiler options:

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
                <source>1.7</source>
                <target>1.7</target>
                <showWarnings>true</showWarnings>
                <showDeprecated>true</showDeprecated>
                <encoding>UTF-8</encoding>
        </configuration>
</plugin>
```

5.1.1.2. Tomcat 7 container

The Maven setup is:

```
<plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.1</version>
        <configuration>
                <username>${tomcat.manager.username}</username>
                <password>${tomcat.manager.password}</password>
                <server>tomcat</server>
                <path>/${app.context.root}</path>
                <url>${tomcat.manager.url}</url>
                <update>true</update>
        </configuration>
</plugin>
```

The setup is quite straightforward. The only thing you need to take into account is that you need the tomcat manager url, username and password to be properly defined in an active Maven profile, for instance:

```
<properties>

  <tomcat.manager.username>admin</tomcat.manager.username>

  <tomcat.manager.password>admin</tomcat.manager.password>

  <tomcat.manager.url>http://localhost:8080/manager/text/deploy</tomcat.manager.url>
```

```
</properties>
```

5.1.2. Appverse Web JPA DLL generator plugin

Appverse Web JPA DLL generator plugin generates DLL database scripts automatically when you build your project if you are using EclipseLink as a JPA provider for your choosen platform.

This tool is very useful especially for development when you are working with an in-memory database as this ensures that your database is always kept up-to-date when you make changes in your JPA annotated model objects.

The script is generated in the configured "ddlOutputDir" and by default when you use an in-memory database this is used to initialize your database.

```
        <plugin>
                <groupId>org.appverse.web.tools.jpaddlgenerator</groupId>
                <artifactId>appverse-web-tools-jpa-ddl-generator</artifactId>
                <version>${appverse.ddl-generator.version}</version>
                <executions>
                        <execution>
                                <goals>
                                        <goal>generate-schema</goal>
                                </goals>
                                <phase>prepare-package</phase>
                                <configuration>
                                        <ddlOutputDir>target/appverse-web-showcases-jsf2-
${project.version}/WEB-INF/classes/sql</ddlOutputDir>
                                </configuration>
                        </execution>
                </executions>
                <dependencies>
                        <dependency>

        <groupId>org.appverse.web.framework.modules.backend.core.persistence</groupId>
                                <artifactId>appverse-web-modules-backend-core-
persistence</artifactId>
                                <version>${appverse.framework.version}</version>
                        </dependency>
                </dependencies>
        </plugin>
```

5.1.3. Appverse Web dependencies

In order to setup Appverse Web dependencies, we include the following code in "dependencyManagement" section:

```
        <dependencyManagement>
                <dependencies>
                        <!-- Appverse dependencies (BOM) -->
                        <dependency>
                                <groupId>org.appverse.web.framework.poms</groupId>
                                <artifactId>appverse-web-masterpom</artifactId>
                                <version>${appverse.framework.version}</version>
                                <type>pom</type>
                                <scope>import</scope>
                        </dependency>
                </dependencies>
                ….
        </dependencyManagement>
```

This "Bill of materials" setup adds default Appverse Web managed dependencies.

On the other hand, in the “dependencies” section we will need to add the Appverse Web dependencies that our project needs. For GWT we have separate maven modules for front-end and back-end projects. Each module dependencies are explained in next sections.

5.2. Backend maven module project configuration

Take into account that the fact that dependency versions are "managed" by a BOM ("bill of materials") avoids the need of specifyng the version of the dependencies to be included as the BOM will determine the right one. Please, do not specify any version in Appverse Web dependencies in order to guarantee right depencendy setup.

In this case, Appverse Web dependencies for backend project are:

```
<dependencies>
…
   <dependency>
        <groupId>org.appverse.web.framework.modules.backend.core.persistence</groupId>
        <artifactId>appverse-web-modules-backend-core-persistence</artifactId>
   </dependency>
   <dependency>
        <groupId>org.appverse.web.framework.modules.backend.frontfacade.gwt</groupId>
        <artifactId>appverse-web-modules-backend-frontfacade-gwt</artifactId>
   </dependency>
   <dependency>
        <groupId>org.appverse.web.showcases.gwt.gwtshared</groupId>
        <artifactId>appverse-web-showcases-gwt-gwtshared</artifactId>
        <version>${project.version}</version>
   </dependency>
   <dependency>
        <groupId>org.appverse.web.showcases.gwt.gwtfrontend</groupId>
        <artifactId>appverse-web-showcases-gwt-gwtfrontend</artifactId>
        <version>${project.version}</version>
        <classifier>GWT-Frontend-Resources</classifier>
        <scope>provided</scope>
        <type>zip</type>
   </dependency>
  …
</dependencies>
```

Please take into account that back-end module has a dependency of the front-end module. The reason is that the artifact resulting of the front-end module processing contains the static content (html, JS, images) resulting of GWT compilation that has to be included in the web application.

5.2.1. HSLQ DB dependency

We include in the project POM a convenient HSQL DB setup that it is very useful in development and continuos integration test environments. Using an in-memory database in these environments make development environments lighter and running continuos integration tests faster.

This is the dependency inclusion:

```
<dependencies>
..
        <dependency>
                <groupId>org.hsqldb</groupId>
                <artifactId>hsqldb</artifactId>
                <scope>runtime</scope>
        </dependency>
..
</dependencies>
```

5.3. Shared maven module project configuration

Shared module requires front end module dependency:

```
<dependencies>
..
                <dependency>
                        <groupId>org.appverse.web.framework.modules.frontend.gwt.api</groupId>
                        <artifactId>appverse-web-modules-frontend-gwt-api</artifactId>
                </dependency>
..
</dependencies>
```