DHIS2 Module Development Tutorial

By Dennis Banga

Introduction

DHIS 2 is an open source web based information system that collects, manages, visualizes and explores data.
Its source code is hosted on Github. The core server-side code is found in the dhis2-core repository and the client-side apps are found in individual app repositories.

In this tutorial you will learn how to create a simple DHIS2 web module that performs CRUD operations.

This tutorial covers the presentation layer(web module), service layer and store/persistence layer operations.

Prerequisites for DHIS2 Development

- ➔ Understanding of the Git technology.(Tutorials-point-Git)
- ➔ Understanding and skill of using java programming language.(javatpoint-java-tutorial)
- ➔ Understanding of Spring Framework(Inversion of Control(IoC), Aspect oriented Programming, JDBC, Transaction Management and MVC) -(javatpoint-spring-tutorial)
- ➔ Understanding of Hibernate(Mapping and queries)-(javapoint-hibernate-tutorial, object-relational_mapping_and_hibernate, mappings_and_queries_with_hibernate)
- ➔ Understanding of Maven Project management tool (javatpoint-maven-tutorial, software_project_management_and_maven.pdf)
- ➔ Understanding of Struts2 Web development framework,and velocity- templating language -(web_frameworks_and_struts2.pdf, struts-2-tutorial)
- ➔ Understanding of Junit -unit tests framework (unit_testing_and_junit), and Java servlets - like Tomcat and Jetty)

Audience

This tutorial assumes you understand the above listed frameworks and technologies, and would like to contribute to the core development of DHIS2.

NB: It is not a step by step guide.

PART 1 : Setting Up and DHIS2 Code Structure

Setting Up the Development Environment

- ➔ Download and install Java SDK 8, Git and Maven.
- ➔ Setup up an integrated development environment of your choice. (Recommended
- ➔ IDEs-IntelliJ, Netbeans, and Eclipse)
- ➔ Clone the source code from Github. The core source code repo is dhis2-core.
- ➔ To build the source code with Maven navigate to the /dhis-2 directory and invoke mvn install Then navigate to the /dhis-2/dhis-web directory and invoke mvn install -U. (To reduce build time, add these maven options -DskipTests=true -DuseWarCompression=false)
- ➔ Each project in the /dhis-2/dhis-web directory is an individual web module. The dhis-web-portal project is an assembly of all the individual web modules. All of these modules can be started by invoking mvn jetty:run The web application can then be accessed at http://localhost:8080.

Understanding DHIS2 Structure

1. Overall Architecture

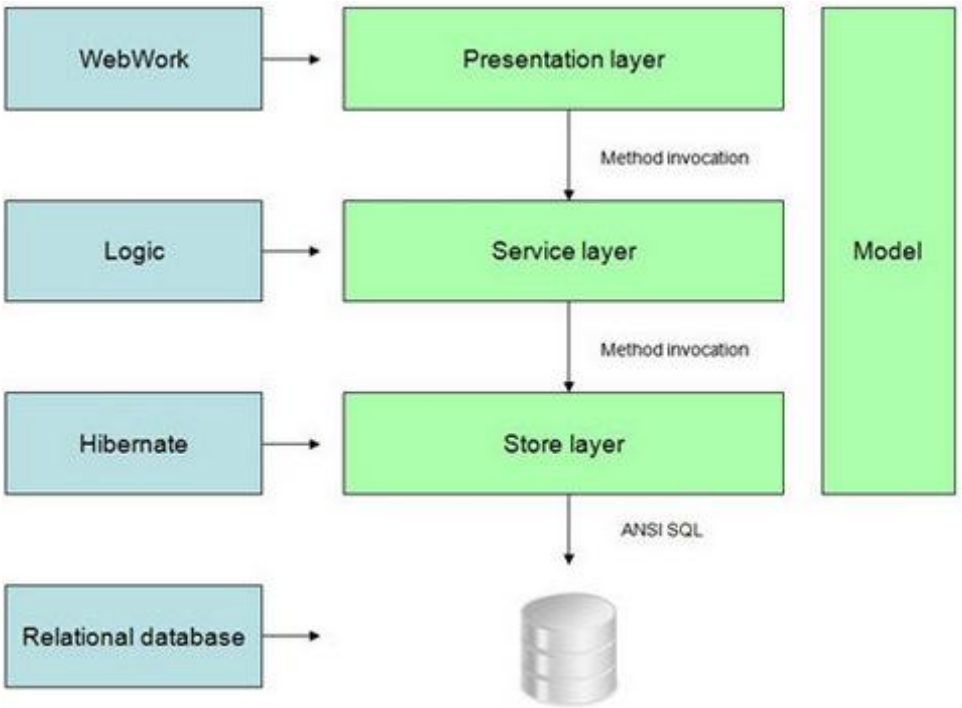


Fig. Overall DHIS2 Architecture (source - dhis2.org)

2. Project Structure

DHIS2 is made up of Maven projects, and web modules. The root POM is located in /dhis-2 and contains project aggregation for all projects excluding the /dhis-2/dhis-web folder. The /dhis-2/dhis-web folder has a web root POM which contains project aggregation for all projects within that folder.

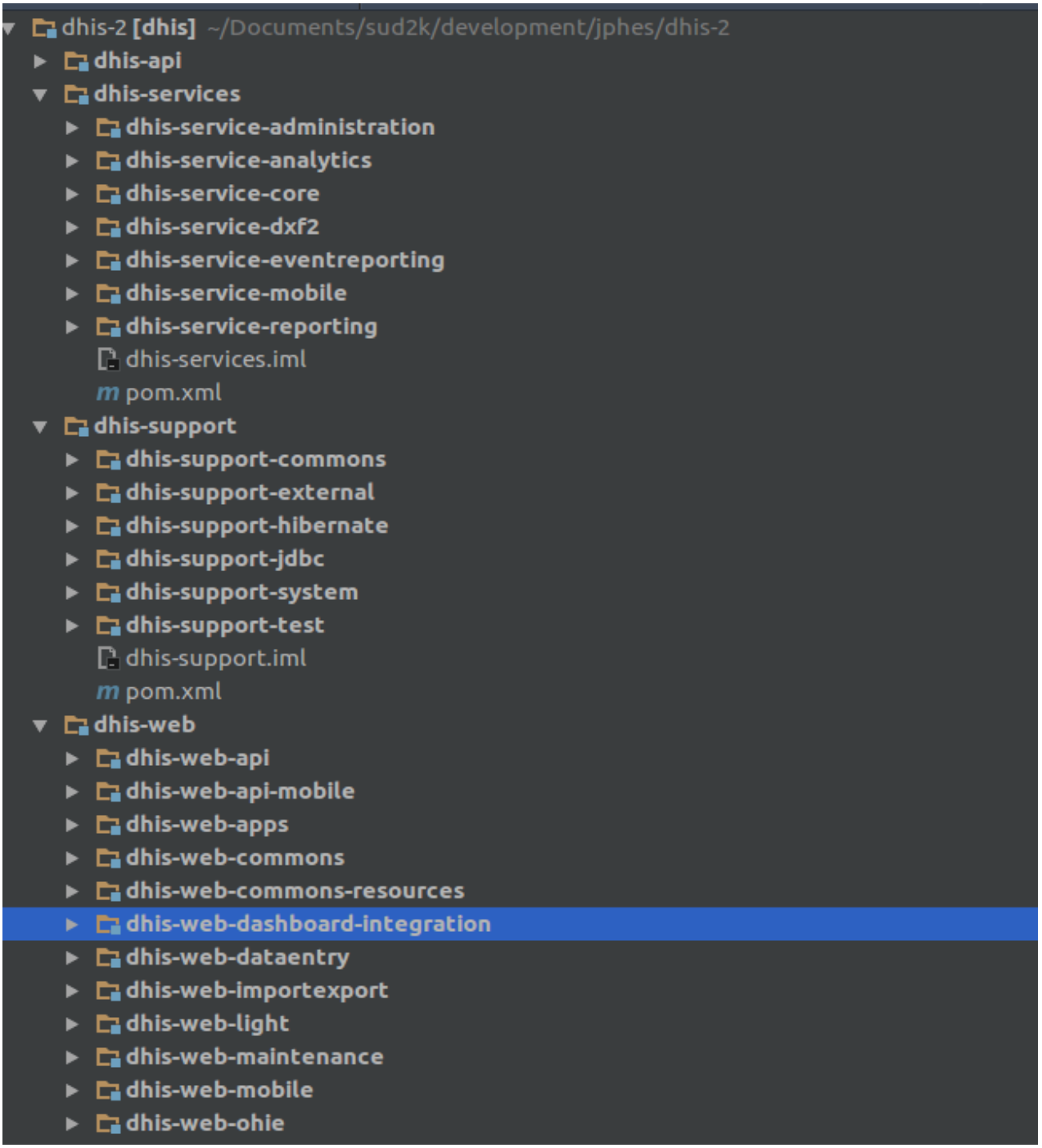


Fig. Project Structure

3. Project Dependencies

Dependencies between the projects are structured in five layers. The support modules provide support functionality for the core and service modules, related to Hibernate, testing, JDBC, and the file system. The core module provides the core functionality in the system, like persistence and business logic for the central domain objects. The service modules provide business logic for services related to reporting, import-export, mapping, and administration. The web modules are self-contained web modules. The portal is a wrapper web module which assembles all the web modules. Modules from each layer can only have dependencies to modules at the same layer or the layer right below.

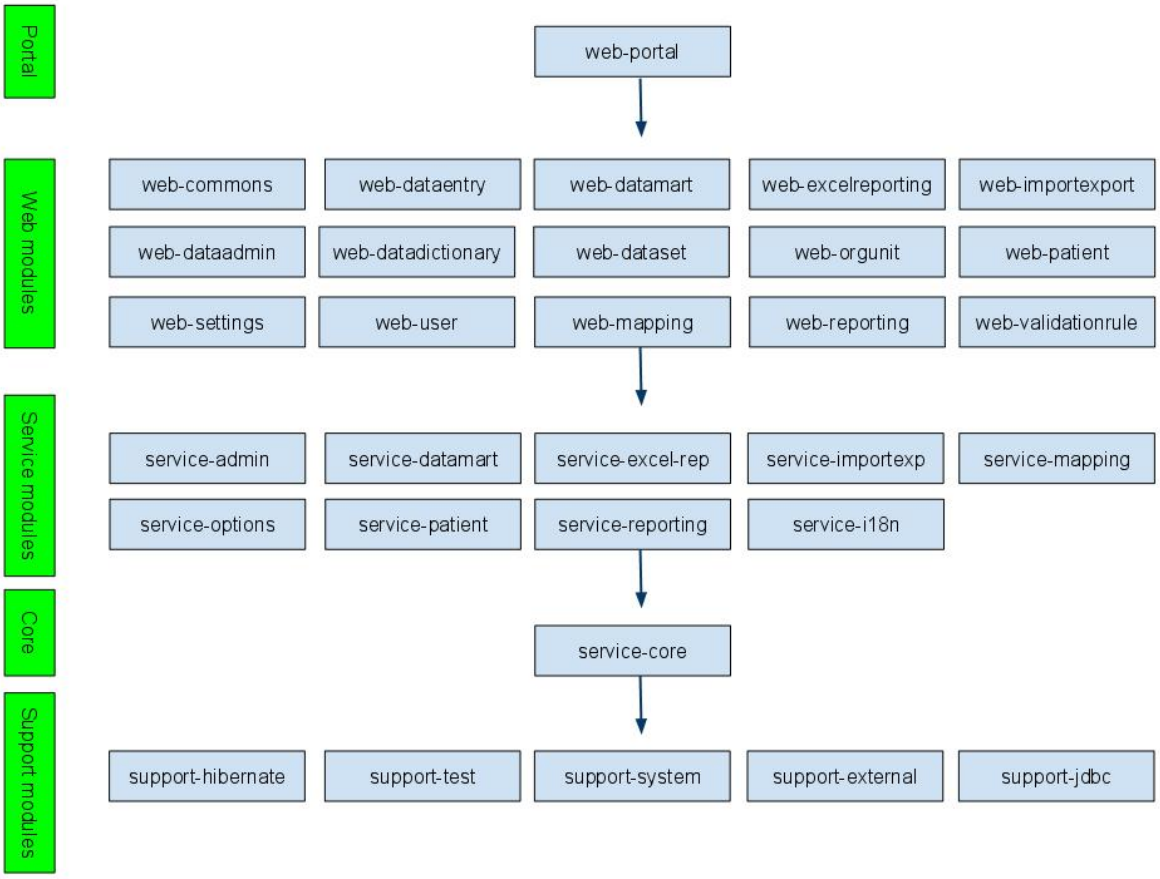
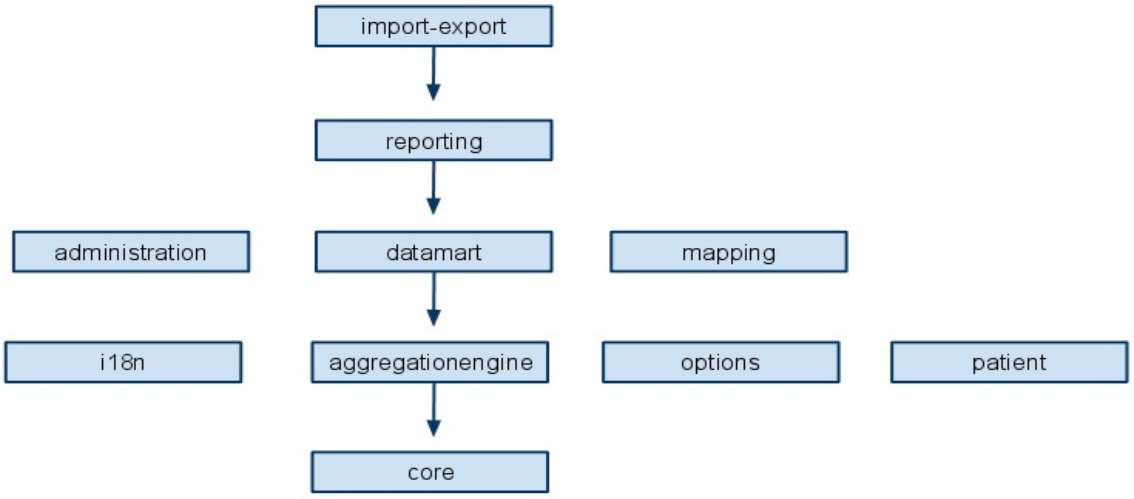


Fig. Project dependencies (source- dhis2.org)



Internal Service structure. (source- dhis2.org)

4. Persistence Layer

The persistence layer is based on Hibernate in order to achieve the ability to run on any major DBMS. Hibernate abstracts the underlying DBMS away and let you define the database connection properties in a file called hibernate.properties.
DHIS2 uses Spring-Hibernate integration, and retrieves a SessionFactory through Spring's LocalSessionFactoryBean. This LocalSessionFactoryBean is injected with a custom HibernateConfigurationProvider instance which fetches Hibernate mapping files from all modules currently on the classpath. All store implementations get injected with a SessionFactory and use this to perform persistence operations.
Most important objects have their corresponding Hibernate store implementation. A store provides methods for CRUD operations and queries for that object, e.g. HibernateDataElementStore which offers methods such as *addDataElement(DataElement), deleteDataElement(DataElement), getDataElementByName(String)*, etc.

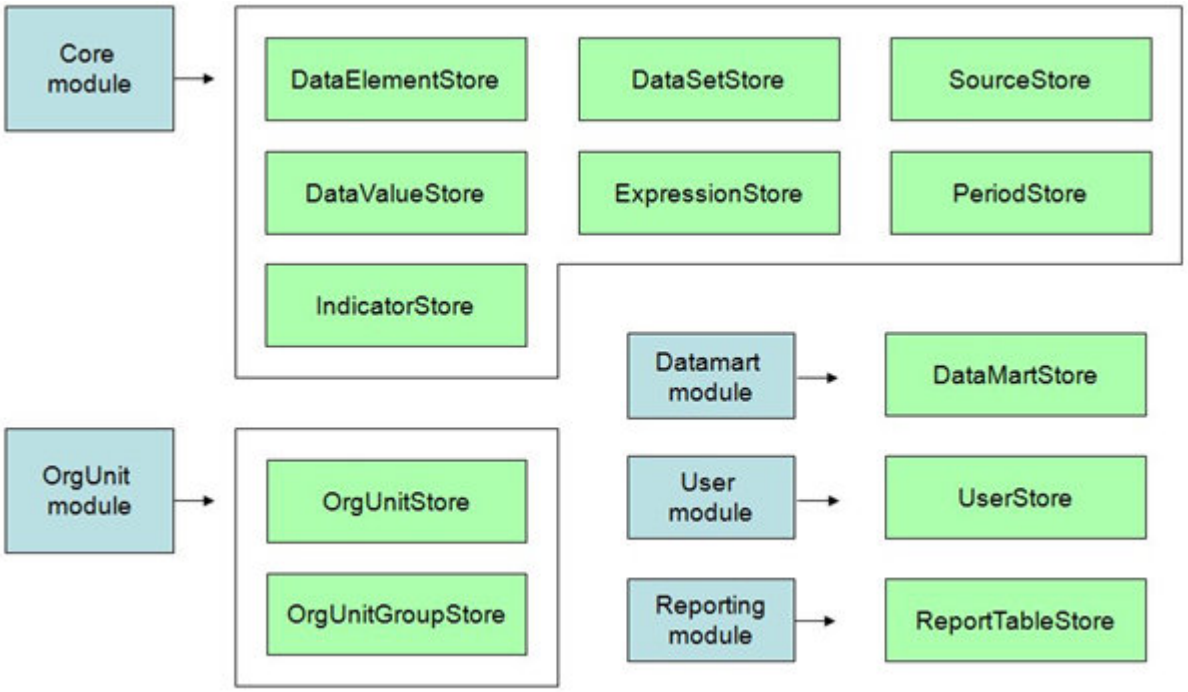


Fig. Persistence Layer

5. Business/Service Layer

All major classes, like those responsible for persistence, business logic, and presentation, are mapped as Spring managed beans. “Bean” is Spring terminology and simply refers to a class that is instantiated, assembled, and otherwise managed by the Spring IoC container. Dependencies between beans are injected by the IoC container, which allows for loose coupling, re-configuration and testability. For documentation on Spring, please refer to springframework.org.
The services found in the dhis-service-core project basically provide methods that delegate to a corresponding method in the persistence layer, or contain simple and self-explanatory logic. Some services, like the ones found in the dhis-service-datamart, dhis-service-import-export, dhis-service-jdbc, and dhis-service-reporting projects are more complex.

6. Presentation Layer

The presentation layer of DHIS2 is based on web modules which are assembled into a portal. This implies a modularized design where each module has its own domain, e.g. the dhis-web-reporting module deals with reports, charts, pivot tables, documents, while the dhis-web-maintenance-dataset module is responsible for data set management. The web modules are based on Struts and follow the MVC pattern . The modules also follow the Maven standard for directory layout, which implies that Java classes are located in src/main/java, configuration files and other resources in src/main/resources, and templates and other web resources in src/main/webapp. All modules can be run as a standalone application.

Common Java classes, configuration files, and property files are located in the dhis-web-commons project, which is packaged as a JAR file. Common templates, style sheets and other web resources are located in the dhis-web-commons-resources project, which is packaged as a WAR file. These are closely related but are separated into two projects. The reason for this is that other modules must be able to have compile dependencies on the common Java code, which requires it to be packaged as a JAR file. For other modules to be able to access the common web resources, these must be packaged as a WAR file.

6.1 Portal

DHIS2 uses a light-weight portal construct to assemble all web modules into one application. The portal functionality is located in the dhis-web-portal project. The portal solution is integrated with Struts.

6.2 Module Assembly

All web modules are packaged as WAR files. The portal uses the Maven WAR plug-in to assemble the common web modules and all web modules into a single WAR file. Which modules are included in the portal can be controlled simply through the dependency section in the POM file in the dhis-web-portal project. The web module WAR files will be extracted and its content merged together.

6.3 Portal Module Requirements

The portal requires the web modules to adhere to a few principles:

- ➔ The web resources must be located in a folder src/main/webapp/<module-artifact-id>.
- ➔ The xwork.xml configuration file must extend the dhis-web-commons.xml configuration file.
- ➔ The action definitions in xwork.xml for a module must be in a package where the name is <module-artifact-id>, namespace is /<module-artifact-id>, and which extends the *dhis-web-commons* package.
- ➔ All modules must define a default action called *index*.
- ➔ The web.xml of the module must define a redirect filter, open-session-in-view filter, security filter, and the Struts FilterDispatcher [8].
- ➔ All modules must have dependencies to the dhis-web-commons and dhis-web-commons-resources projects.

6.4 Common look and feel

Common look and feel is achieved using a back-bone Velocity template which includes a page template and a menu template defined by individual actions in the web modules. This is done by using static parameters in the Struts/Xwork xwork.xml configuration file. The action response is mapped to the back-bone template called *main.vm*, while static parameters called page and menu refers to the templates that should be included. This allows the web modules to display its desired content and left side menu while maintaining a common look-and-feel.

PART 2: Developing the Module - JPHES

This part will cover the development of a dhis2 module from the model and service layer, then the web module.
The module is an app that lists partners, and allows adding, editing and deleting partners.

Model/POJO

To create the model/POJO for the project called Partner, It doesn't require adding another module on the dhis-api module. It only requires adding a package for that app/module under dhis-api.
Under the package, you include the project model object class, the object store interface, and object service interface. The object service and object store interfaces will be implemented at the service layer.
There are already defined abstract object classes in dhis-api, so there is no need of defining everything from scratch. Some of the abstract classes are BaseIdentifiableObject and BaseDimensionalObject.
Let's create our Partner Class, PartnerStore Interface, and PartnerService Interface.

- ➔ Navigate to this directory (dhis-api/src/main/java/org/hisp/dhis) under dhis-api module.
- ➔ Create a package/directory called partner.(name of your choice based on the your conventions). This helps in organizing your code.
- ➔ Under the partner directory, create a Partner Class, PartnerStore Interface and PartnerService Interface as follows.

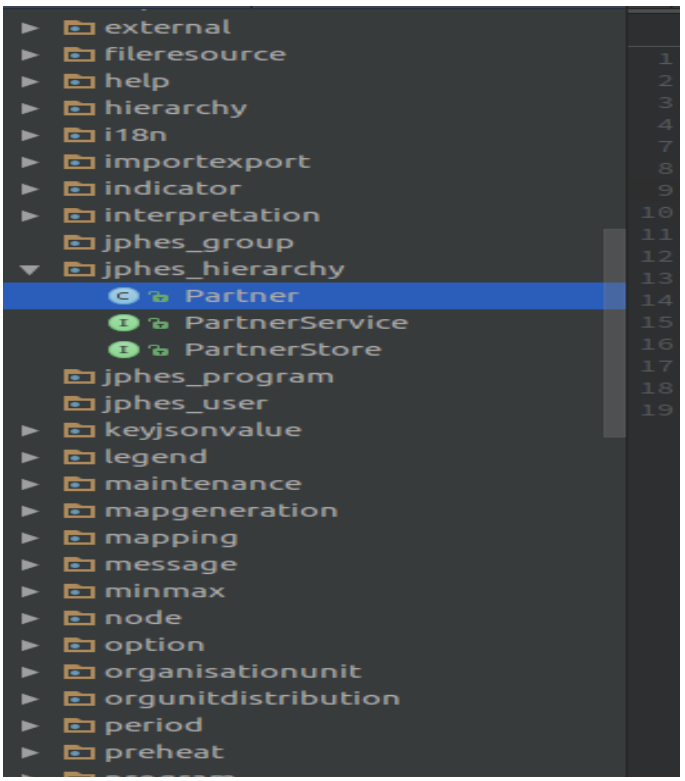
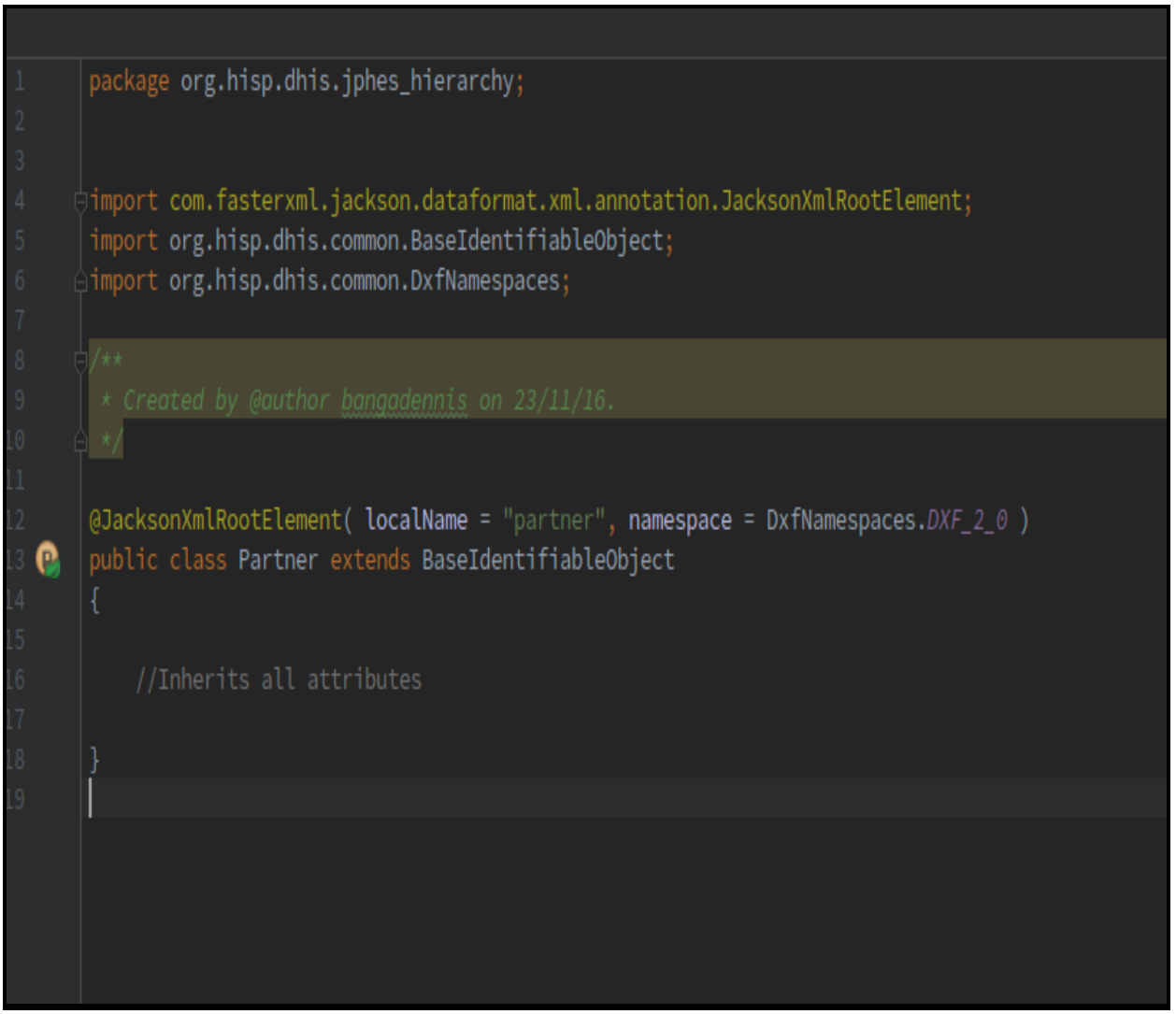


Fig. Partner directory (as jphes_hierarchy), has the Partner Class, PartnerStore and PartnerService Interfaces.

Partner Class

Has attributes and methods relating to my Partner Object. For instance, name, description, code, created and last updated.
(Based on the object, it can extend the BaseIdentifiableObject or BaseDimensionalObject class.)



```java
package org.hisp.dhis.jphes_hierarchy;


import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlRootElement;
import org.hisp.dhis.common.BaseIdentifiableObject;
import org.hisp.dhis.common.DxfNamespaces;

/**
 * Created by @author bangadennis on 23/11/16.
 */

@JacksonXmlRootElement( localName = "partner", namespace = DxfNamespaces.DXF_2_0 )
public class Partner extends BaseIdentifiableObject
{

    //Inherits all attributes

}
```

PartnerService Interface

Has the definition of the CRUD methods that can be performed on the Partner object.

PartnerService

```java
package org.hisp.dhis.jphes_hierarchy;

import java.util.List;

/**
 * Created by @author bangadennis on 24/11/16.
 */


public interface PartnerService
{
    String ID = PartnerService.class.getName();

    int savePartner(Partner partner);

    Partner getPartner(String uid);

    Partner getPartnerByName(String name);

    List<Partner> getPartners(String name);

    List<Partner> getAllPartners();

    List<Partner> getPartnersBetween( int first, int max );

    List<Partner> getPartnersBetweenByName( String name, int first, int max );

    int getPartnerCount();

    int getPartnerCountByName(String name);

    void updatePartner(Partner partner);

    void deletePartner(Partner partner);

}
```

Fig. PartnerService Interface

PartnerStore Interface

Defines the data access methods.
extends the GenericIdentifiableObjectStore<T> interface that has defined the generic data access methods.

```java
package org.hisp.dhis.jphes_hierarchy;

import ...

/**
 * Created by @author bangadennis on 24/11/16.
 */
public interface PartnerStore extends GenericIdentifiableObjectStore<Partner>
{
    List<Partner> getAllPartners();

    Partner getPartnerByName(String name);


}
```

Fig. PartnerStore Interface.

Service layer

- ➔ Under the service layer, the DefaultPartnerService class that implements the PartnerService Interface , and the HibernatePartnerStore class that implements the PartnerStore interface are defined with the method implementations.
- ➔ On the (dhis-services/dhis-service-core/src/main/java/org/hisp/dhis) add the partner directory/package.(for my case, it is jphes_hirerachy)
- ➔ Under the partner directory create the DefaultPartnerService class. Add a directory called hibernate then create the HibernatePartnerStore class. See the below structure.

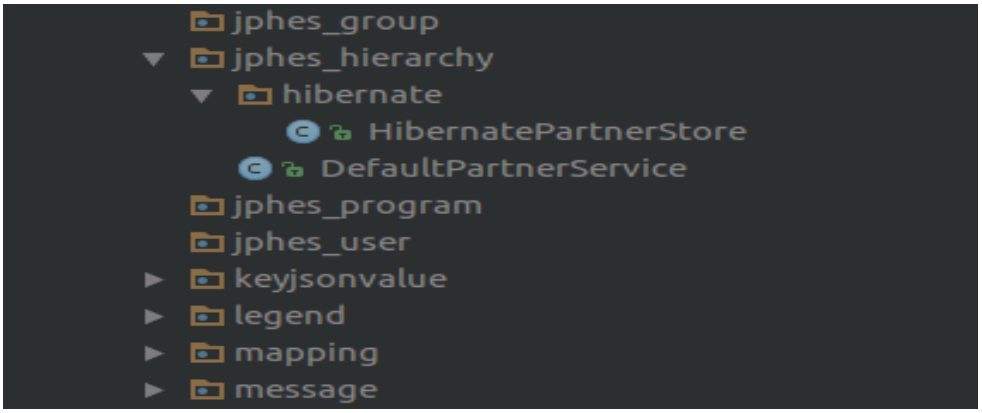


Fig. Partner service directory.

DefaultPartnerService Class

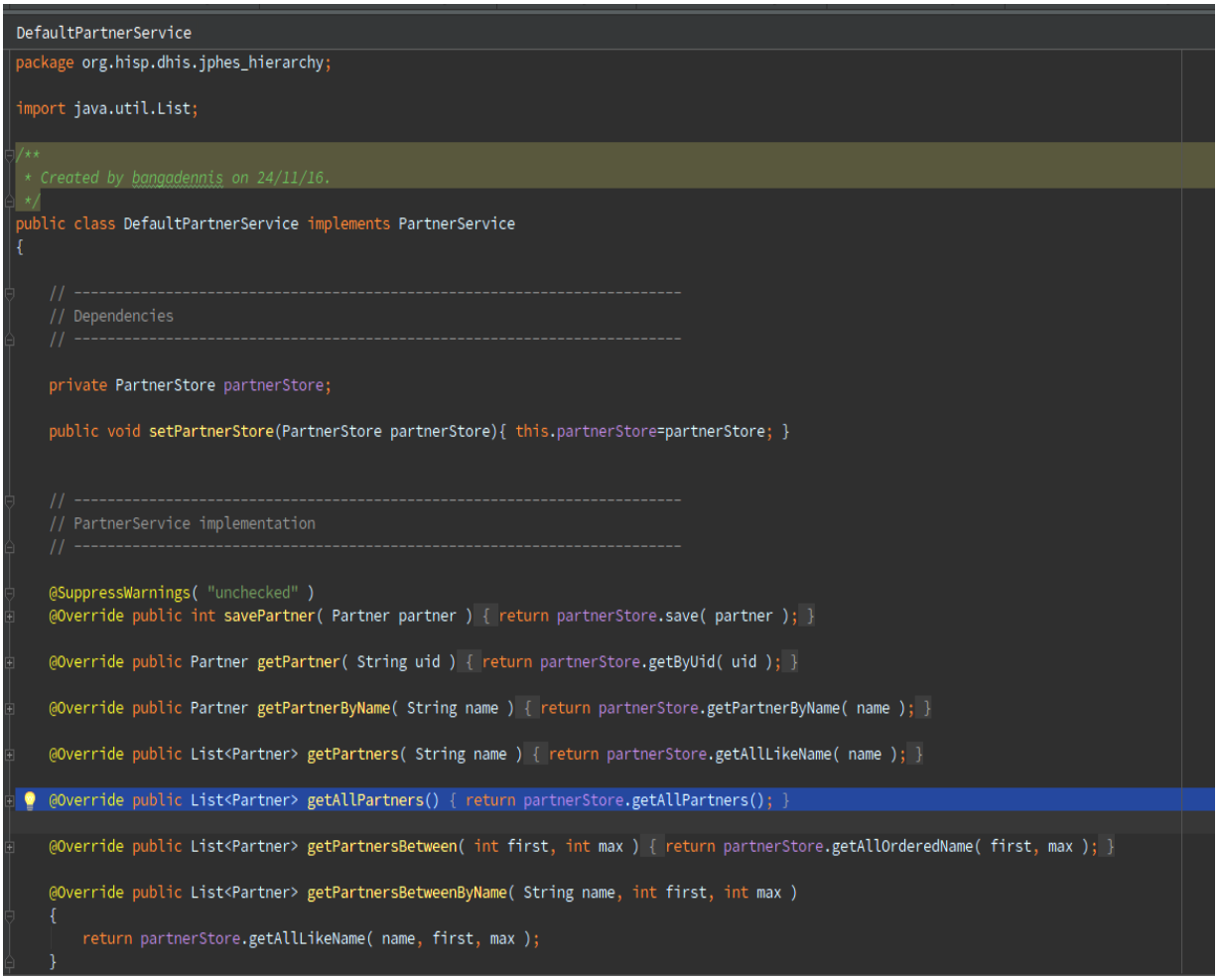


Fig. DefaultPartnerService Class that implements the PartnerService interface.

HibernatePartnerStore Class

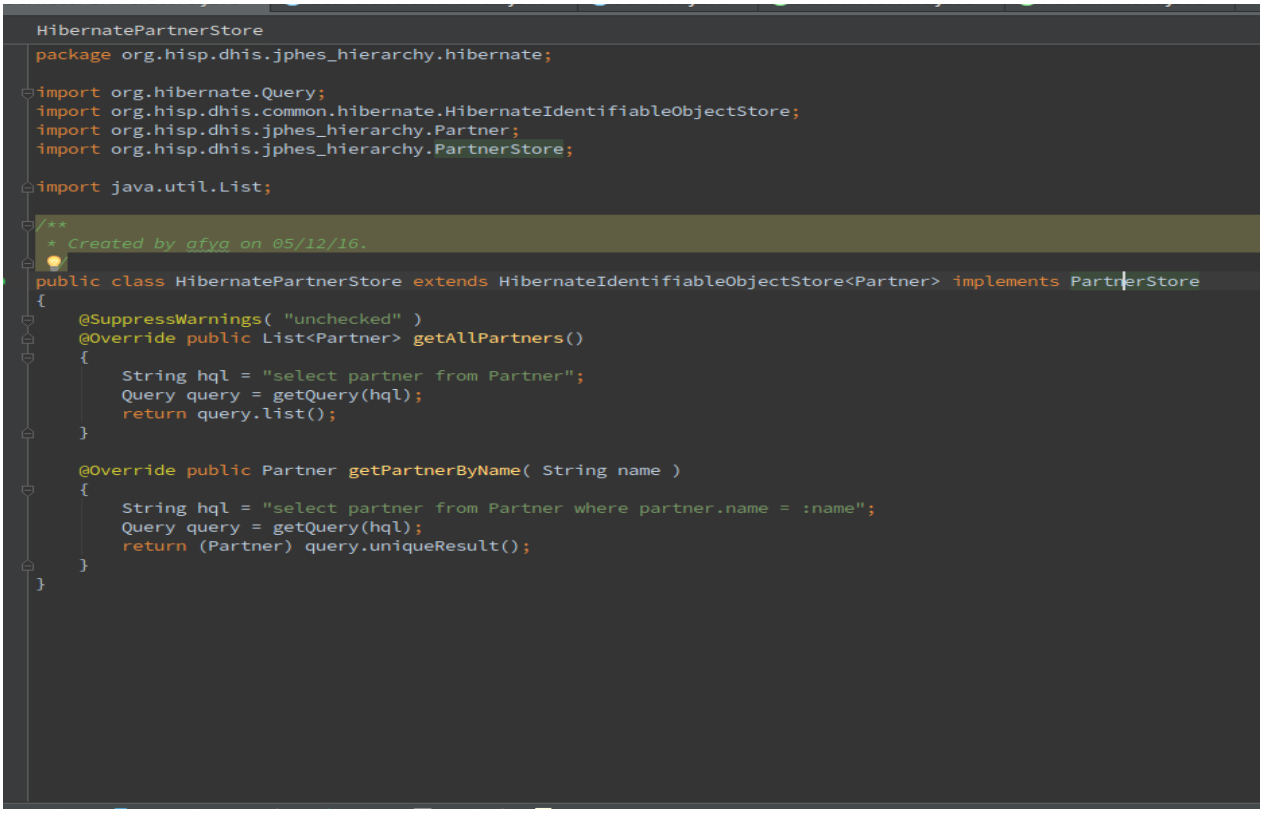


Fig. HibernatePartnerStore class extends the HibernateIdentifiableObjectStore<T> class and implements the PartnerStore interface.

Configuring the Service beans for the PartnerService and PartnerStore.

Go to the (dhis-services/dhis-service-core/src/main/resources/META-INF/dhis/bean.xml) file, and add the following bean configuration

```
<!--Custom Partner Services-->
<bean id="org.hisp.dhis.jphes_hierarchy.PartnerService"
class="org.hisp.dhis.jphes_hierarchy.DefaultPartnerService">
 <property name="partnerStore" ref="org.hisp.dhis.jphes_hierarchy.PartnerStore" />
</bean>

<!--Custom Partner Store Object-->
<bean id="org.hisp.dhis.jphes_hierarchy.PartnerStore"
class="org.hisp.dhis.jphes_hierarchy.hibernate.HibernatePartnerStore">
  <property name="clazz" value="org.hisp.dhis.jphes_hierarchy.Partner" />
  <property name="sessionFactory" ref="sessionFactory" />
  <property name="cacheable" value="true" />
</bean>
<!--Partner-->
```

Hibernate Mapping

Create a package/directory partner/hibernate under the (dhis-services/dhis-service-core/src/main/resources/org/hisp/dhis), then create Partner.hbm.xml file then add the following properties.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"
 [<!ENTITY identifiableProperties SYSTEM
"classpath://org/hisp/dhis/common/identifiableProperties.hbm">]
 >
<hibernate-mapping>
 <class name="org.hisp.dhis.jphes_hierarchy.Partner" table="partner">
   <cache usage="read-write" />
   <id name="id" column="partnerid">
     <generator class="native" />
   </id>
   <property name="uid" column="uid" unique="true" length="11" />
   <property name="code" column="code" not-null="false" unique="false" length="230" />
   <property name="created" type="timestamp"/>
   <property name="lastUpdated" type="timestamp"/>
   <property name="name" column="name" not-null="true" length="230" />
   <property name="displayName" column="displayName" not-null="false" length="230" />
 </class>
</hibernate-mapping>
```

Web Module

The web module is based on Struts2 web framework. Create a dhis-web-partner module, and add it on the (/dhis-2/dhis-web) module.

dhis-web module structure.

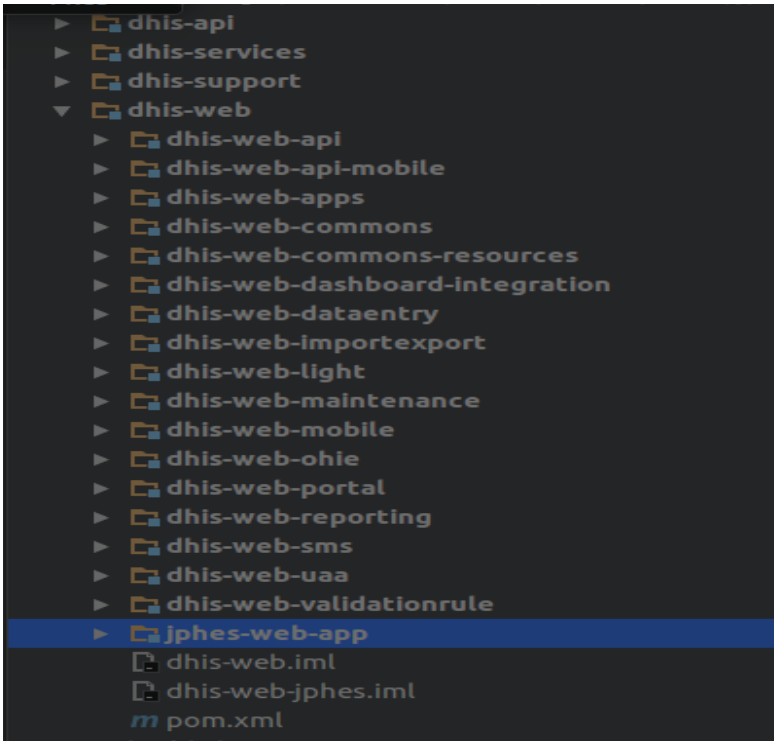


Fig. dhis-web module structure

DHIS Web App structure

- src/main/java
- src/main/resources
- src/main/webapp
- pom.xml

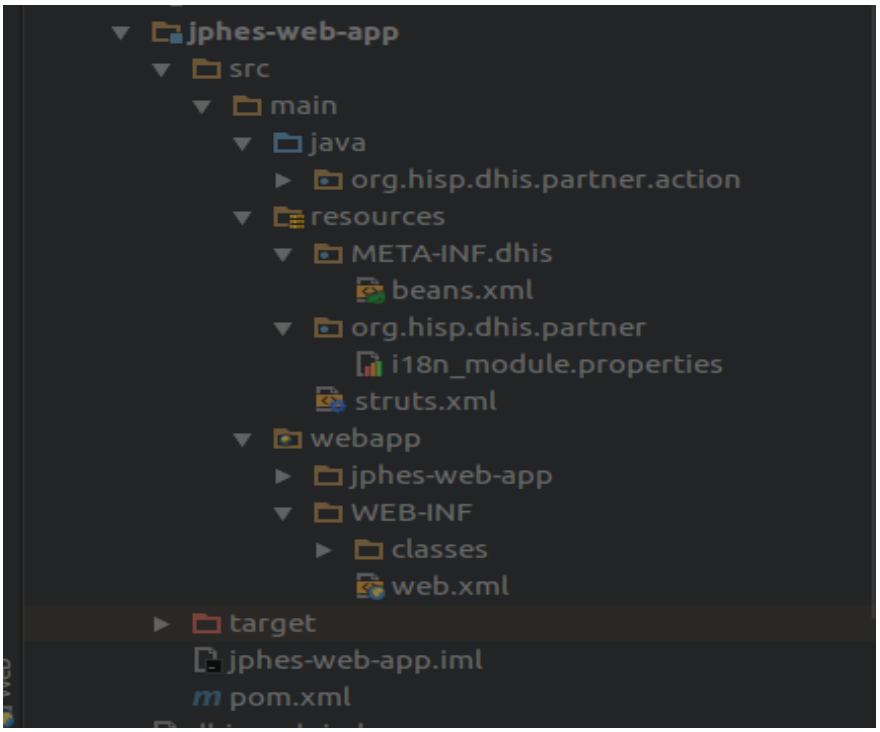


Fig. dhis-web app module structure

dhis-web-partner pom.xml file configurations.

```
dhis-web-partner/pom.xml
--------------------------------------------------------------------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <parent>
   <groupId>org.hisp.dhis</groupId>
   <artifactId>dhis-web</artifactId>
   <version>2.21</version>
 </parent>

 <artifactId>jphes-web-app</artifactId>
 <packaging>war</packaging>
 <name>Partner Web App</name>

 <build>
   <finalName>dhis-web-partner</finalName>
 </build>

 <description>This is the JPHES web portal to perform partner attribution.</description>
---------------------------------------------------------------------------------------------------------------------
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Add the artifact id for the dhis-web-partner module on the dhis-web pom.xml file as follows, under the modules attribute.

```
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
dhis-web/pom.xml
--------------------------------------------------------------------------------------------------------------------
 <parent>
   <groupId>org.hisp.dhis</groupId>
   <artifactId>dhis</artifactId>
   <version>2.21</version>
 </parent>
 <artifactId>dhis-web</artifactId>
 <packaging>pom</packaging>
 <name>DHIS Web Modules Project</name>
 <description>This project is a web based GUI for the DHIS 2 system.</description>
```

```
<modules>
  <module>dhis-web-commons</module>
  <module>dhis-web-commons-resources</module>
  <module>dhis-web-api-mobile</module>
  <module>dhis-web-api</module>
  <module>dhis-web-ohie</module>
  <module>dhis-web-uaa</module>
  <module>dhis-web-maintenance</module>
  <module>dhis-web-dataentry</module>
  <module>dhis-web-importexport</module>
  <module>dhis-web-validationrule</module>
  <module>dhis-web-reporting</module>
  <module>dhis-web-dashboard-integration</module>
  <module>dhis-web-light</module>
  <module>dhis-web-mobile</module>
  <module>dhis-web-sms</module>
  <module>dhis-web-apps</module>
  <module>dhis-web-partner</module>
  <module>dhis-web-portal</module>
</modules>
-----------------------------------------------------------------------------------------------------------
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Configuring the dhis-web-portal

Add your webapp as a dependency in the dhis-web/dhis-web-portal module POM file as follows:

dhis-web-portal/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <parent>
   <groupId>org.hisp.dhis</groupId>
   <artifactId>dhis-web</artifactId>
   <version>2.21</version>
 </parent>

 <artifactId>dhis-web-portal</artifactId>
 <packaging>war</packaging>
 <name>DHIS Web Portal</name>

 <build>
   <finalName>dhis</finalName>
 </build>
```

```
...
 <dependency>
   <groupId>org.hisp.dhis</groupId>
   <artifactId>dhis-web-partner</artifactId>
   <version>${project.version}</version>
   <type>war</type>
 </dependency>
</dependencies>
.............
```

Then start developing the web module packages. Refer to (https://github.com/uonafya/jphes/tree/dennis-dev) jphes-web-app module.

References

1. DHIS2 Website - https://dhis2.org