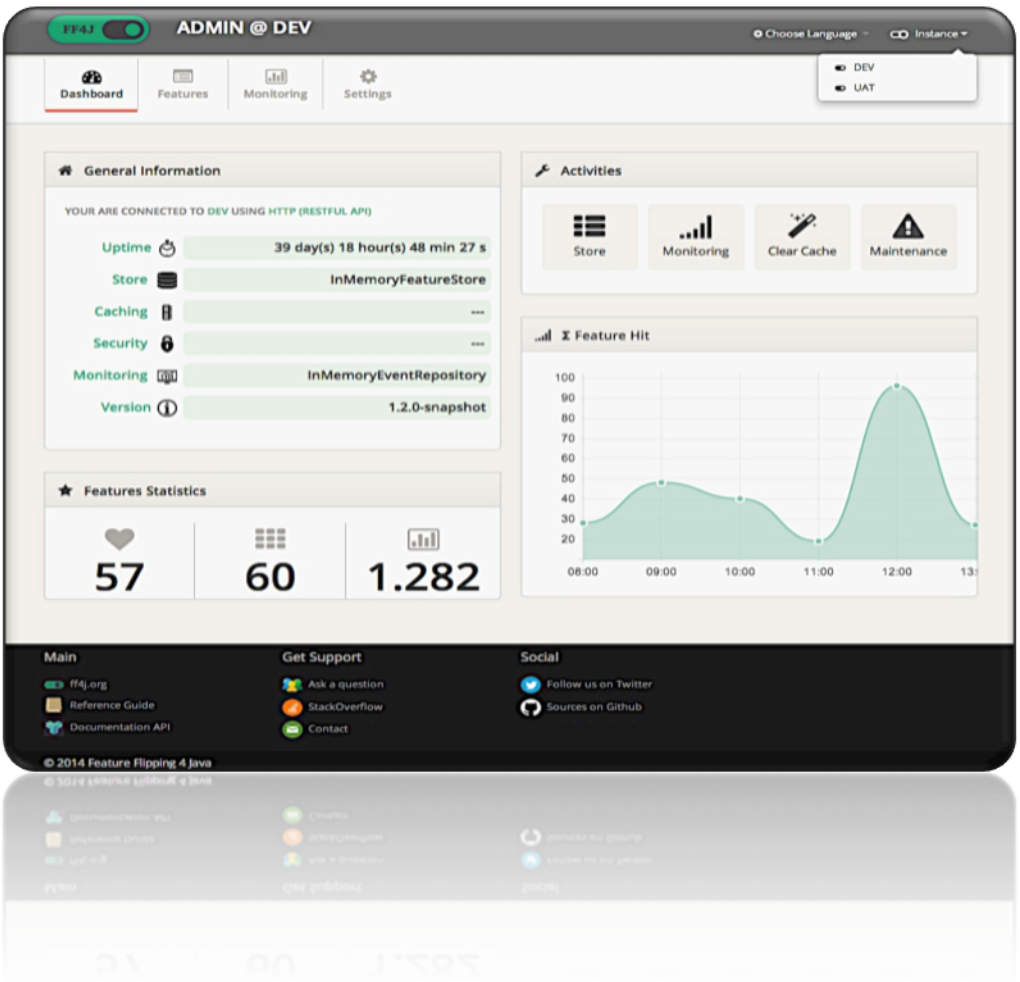
REFERENCE DOCUMENTATION



VERSION 1.2.0

Cédrick Lunven : @clunven

Ff4j Reference Guide

v1.3

2015

Cedrick Lunven cedrick.lunven@gmail.com

2015

Table of Contents

Copyright

Online version published by Cedrick LUNVEN

1. Feature Toggle

1.1 Introduction

The principe of *Feature Toggle* is to enable or disable feature through configuration, eventually at runtime. The condition statement to toggle can be a simple flag (boolean) but also a more elaborate test with a set of rules. It's also called *feature flipping*, *feature flags*, or even *feature bits*. Toggle features at runtime is mandatory to change behaviour of the application without restarting.

FF4J is an implementation of the principle for the Java Plateform. It stands as *Feature Flipping for Java*.

Note

As ff4j provides a restFul WebAPI, any application could work and check features through HTTP. It's not limited to the Java platform

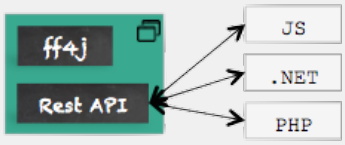


Figure 1.1.

1.2 References

1.2.1 Martin Fowler

Martin Fowler, an architect working at Thoughtworks, has written on his professional blog in 2010, a introduction to the concept. The full article can be consulted here. He is known in the community as one father of the continuous integration.

He defines feature toggle as : *The basic idea is to have a configuration file that defines a bunch of toggles for various features you have pending. The running application then uses these toggles in order to decide whether or not to show the new feature.*

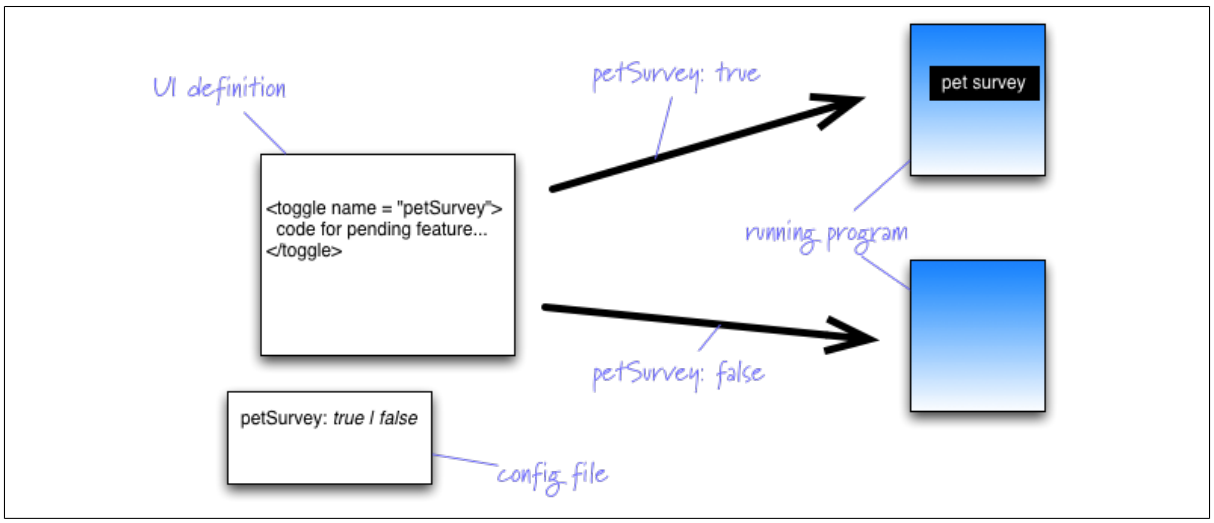


Figure 1.2. Martin Fowler reference article

1.2.2 Blogs and articles

The feature toggle has been mostly promoted by the continuous delivery. Webgiants such as GAFA (Google, Amazon, Facebook, Apple) or Etsy have post quite a lot of information of their realisations.

Table 1.1. Blog references

Title	Description
Presentation of concept on InfoQ	http://www.infoq.com/presentations/Feature-Bits
Presentation on 99Design	http://99designs.com/tech-blog/blog/2012/03/01/feature-flipping/
About Etsy	This article is available on codeascraft
About Flickr	http://code.flickr.com/blog/2009/12/02/flipping-out/
Octo Technology	introduced the concepts in their: french article but also in their book.

Figure 1.3. WebGiants practices by Octo (fr)

1.3 Use Cases

1.3.1 Continuous Delivery

1.3.1.1 Definition

As suggested by its name, the purpose of the continuous delivery set of practices is to release softwares as often as required. The delivery process is obviously automatic and triggered on demand, eventually after each developper commit. It allows 'non-event' releases : no more prepared or anticipated, but performed anytime needed. For instance, Amazon pushes code into production in average, every 11.6 seconds.

1.3.1.2 Feature Branching vs Feature Toggle

To be able to develop several features in the same time yet be compliant with the short-time development cycle there are 2 possibilities.

Feature Branching

The first solution is to create branches in the source control system for each new feature. The release is performed from sources hosted on trunk : features under development are ignored and won't be part of the build. When a feature is 'ready', the related branch is merged to the trunk. This operation may become very complex. Indeed, if several releases have be made since the initial 'fork' of the current branch, source codes can be potentially very different. This leads to a large number of conflicts to deals with.

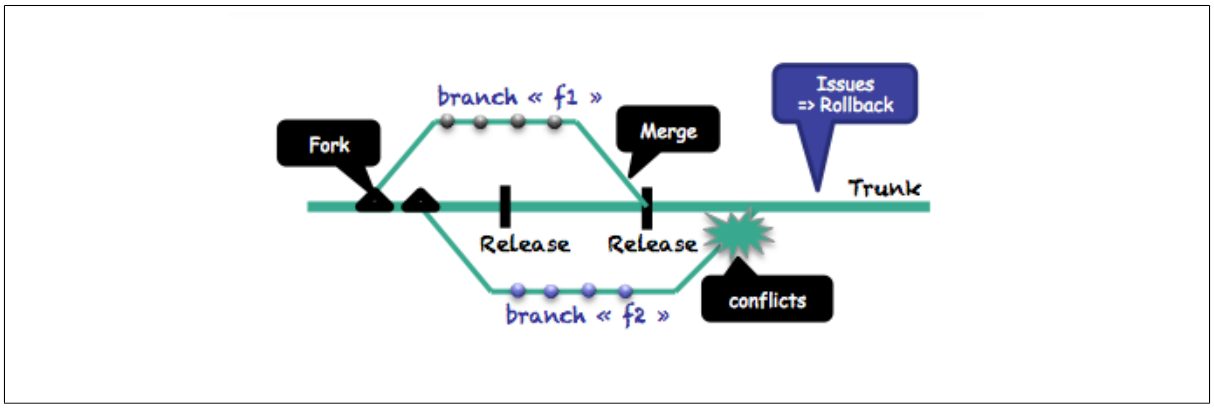


Figure 1.4. Feature Branching

Trunk-based developement

The second solution is, on the contrary, to keep on developing in the trunk. The direct consequence is that incomplete or non-working (yet compiling) code will be embedded in a release and pushed into production. To avoid any incidents the relevant source code is wrapped in a always-false condition. The value of the condition is defined through configuration. As soon as the code is ready, the condition is set to true to start executing.

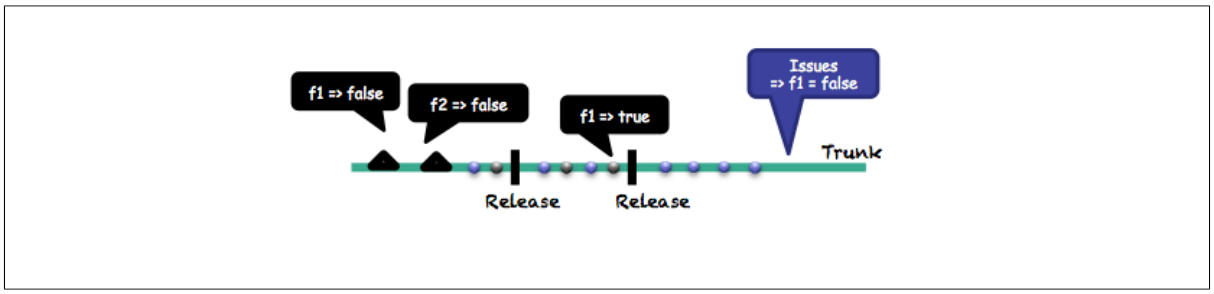


Figure 1.5. Trunk-based developement

1.3.1.3 Zero Downtime deployment

Blue/Green Deployments

BlueGreen deployment is a term to describe the old way to perform "hot deploy" in high availability architectures. The application is deployed on several nodes of a cluster. During deployment, each node is stopped, the new version of the product is released and the node is restarted. One common problem is a lack of consistency between nodes. Is the release required to change the datamodel for instance, the hot deploy is no more possible. Feature Toggle can be an answer. All servers will be updated in ther same way but without enabling the the modifications. When the environnement is ready all nodes of the cluster can activate the new feature as the same time.

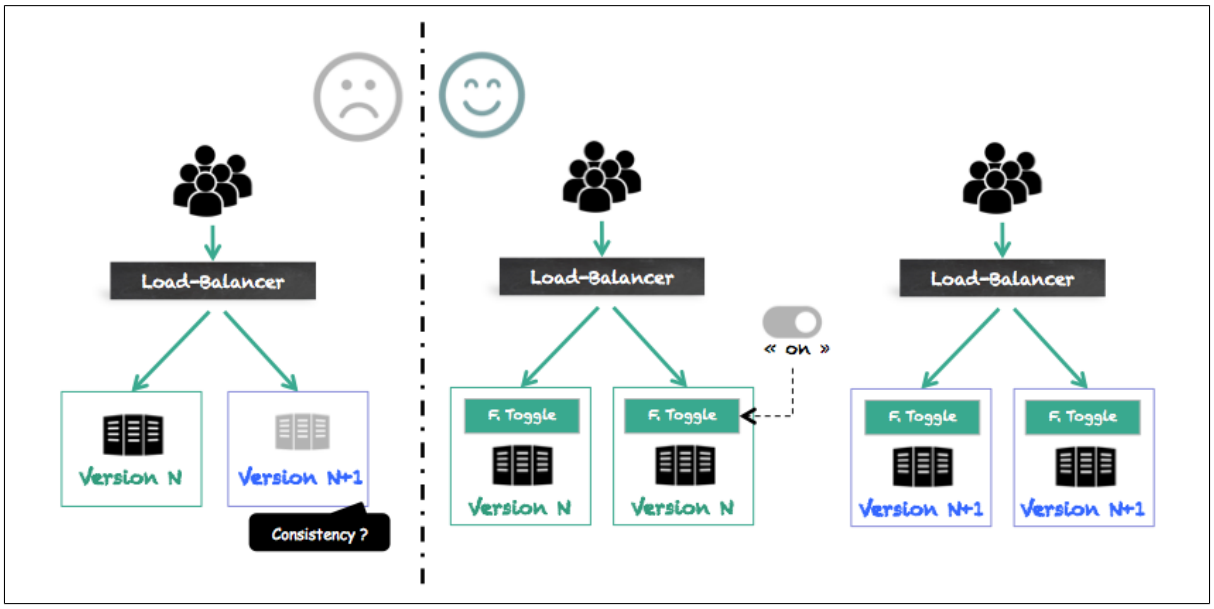


Figure 1.6. Blue/Green deployment

Canary Release

The concept of canary release is to enable a feature for a subset of the users population. Thoses users may have a particular role ()like "beta-tester", or may be located in some place (like pilot), or like Amazon feature can be first propose to employees and then to customers. It's an excellent way to get feedback from users but also real metrics on production environment. The feature toggle system wrapped any feature definition which can be enabled at runtime through configuration console.

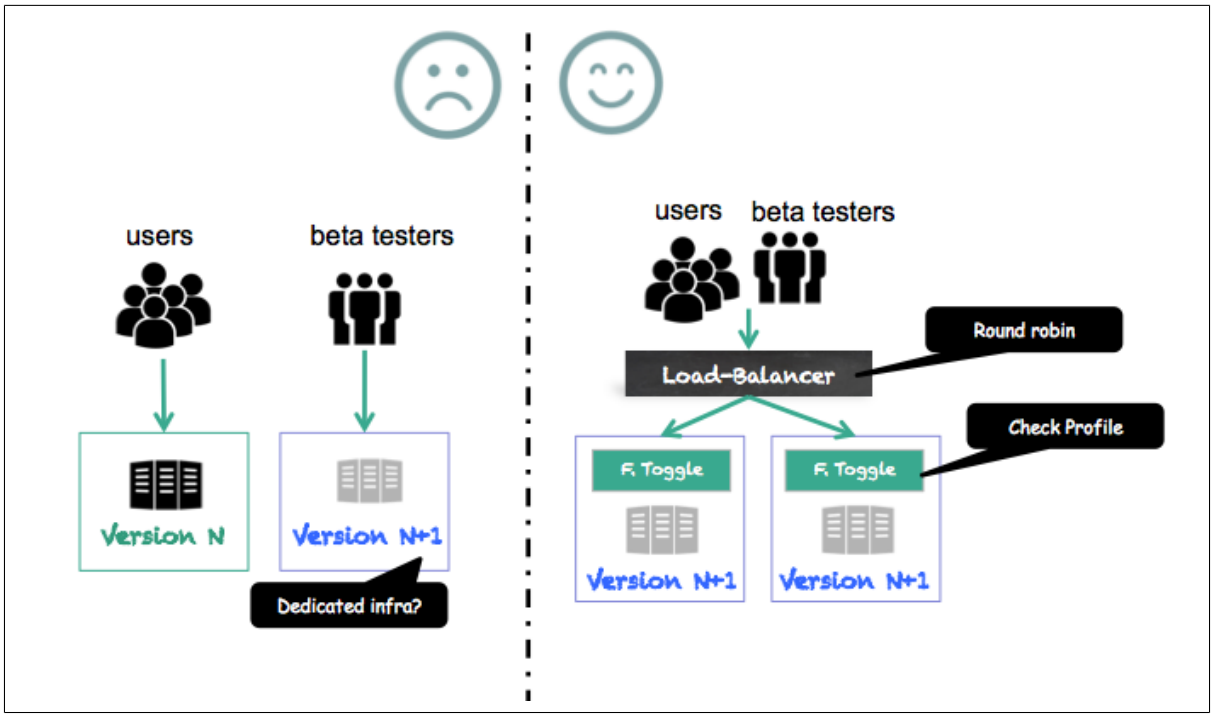


Figure 1.7. Canary Release

DarkLaunch

The concept of Dark launch is to enable features progressively. For instance, only a fixed rate of incoming requests will use the new version of the product. The main advantage is to measure the

impact of evolutions for a limited flow of requests and then anticipate any load, performance or capacity problems. As detailed later, ff4j provides a "PonderationStrategy"" to implement exactly this use case.

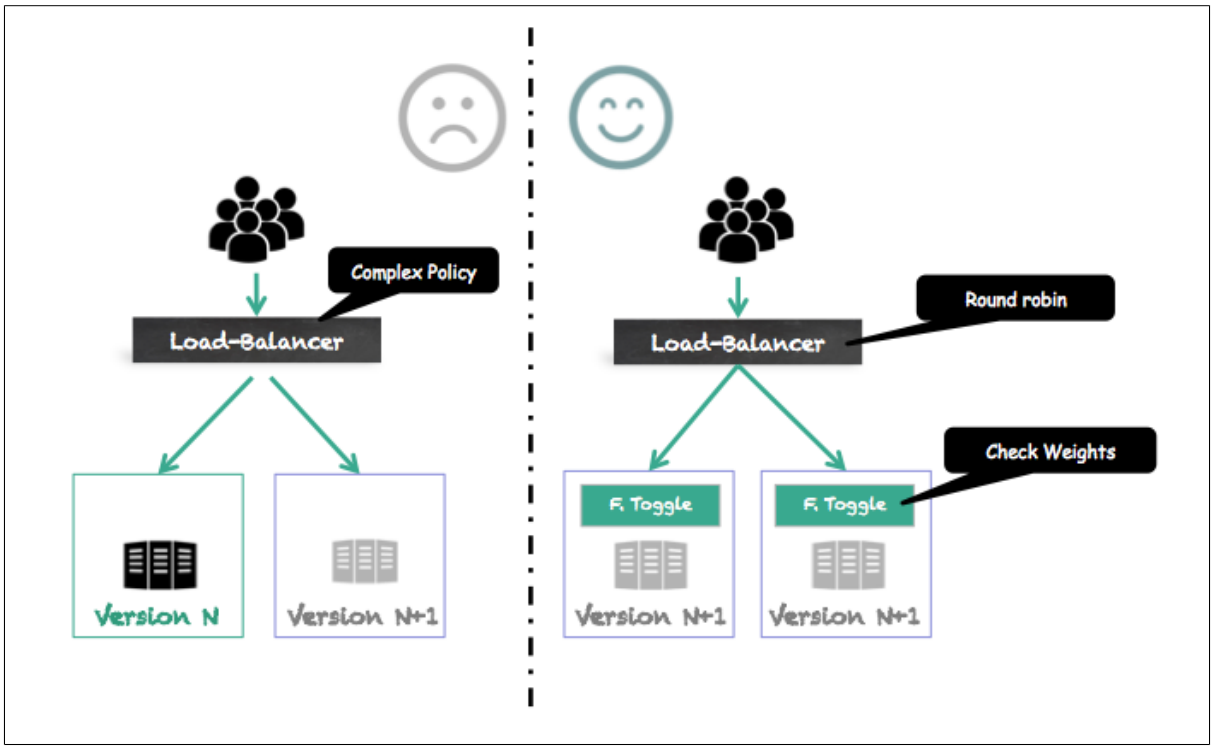


Figure 1.8. Dark Launch

1.3.1.4 Thin Client deployment

This clever use case should be considered when the release process is painful or the application is installed on third party devices. It has been used by Facebook for their iOS application. To publish an application on the appstore the process is always the same. Developpers package and submit the new version to apple teams which MANUALLY validate it. It can take up to 48H. Once the application is published, any single user must have to download the application - again - from Apple servers.

The basic idea of *Thin Client deployment* is to never publish new versions of the application in stores, but, instead, at startup, check current installed application against a web service and download new content if required.

Feature toggle should be part of the process. When the feature is ready, it's activated in the server. Target clients (not all if you do no want to) at startup of their applications will get the updates.

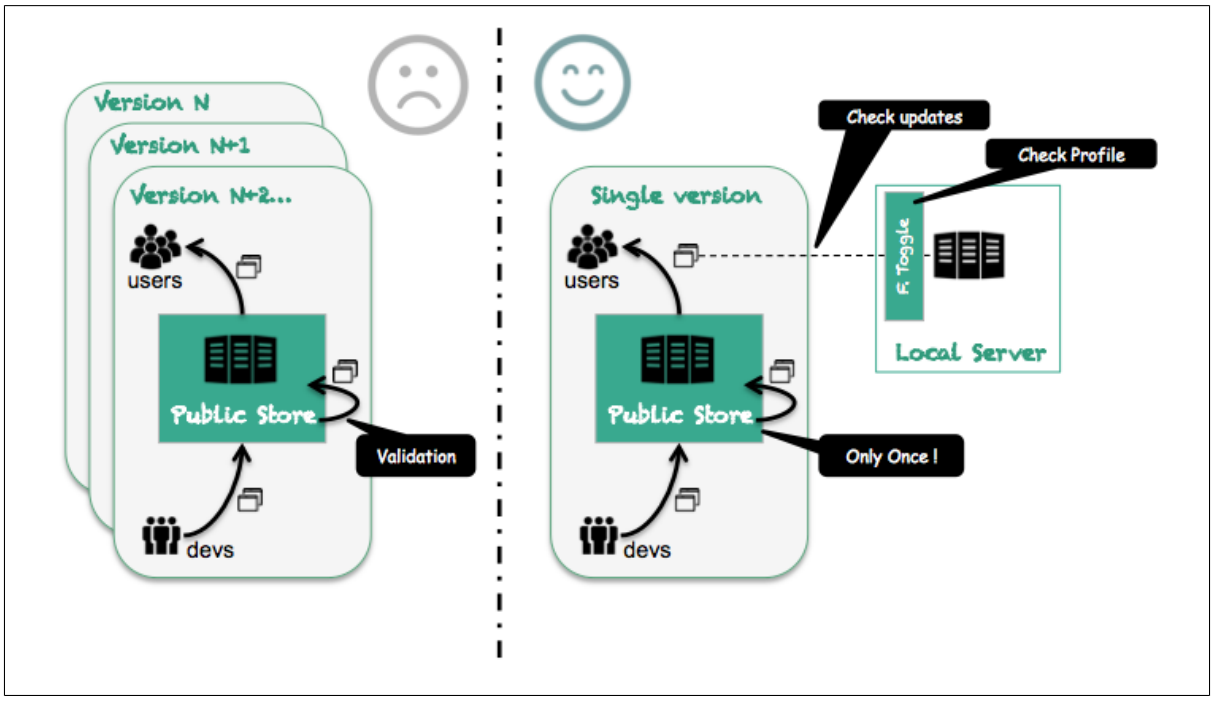


Figure 1.9. Thin-client deployment

1.3.2 Ops and Infrastructure

1.3.2.1 Graceful Degradation

The graceful degradation is a capability of a system to disable non-core functionnalities to free ressources for more important ones. As an example, on heavy load of an e-commerce website, you can priorize the requests of customers which already have something in their cart, they are more likely to buy something. When a request hit the landing page, and feature toggle strategy evaluates that it's not an important one, the user could be redirect to a dedicated page telling him to try later.

Another example is the set up of quota. Imagine you would like to create a chatroom with 20 people. Once the quota is reached, the new users will be put in waiting room.

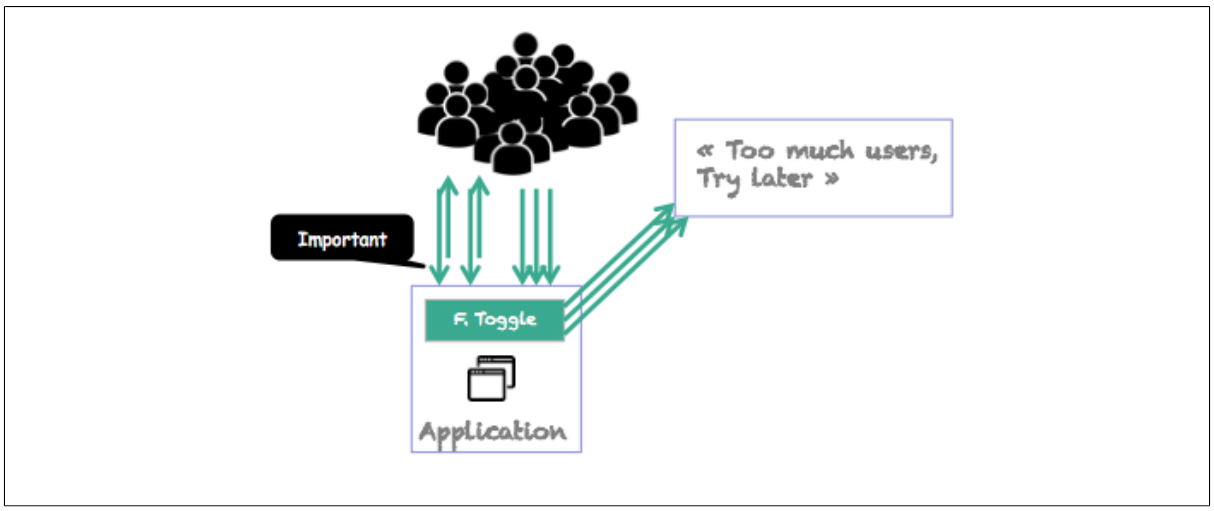


Figure 1.10. Graceful Degradation

1.3.2.2 Service Catalog

A feature can be distributed among several applications. For those uses cases the features storage is a unique repository and all applications work as clients. Once the administrator toggle off a feature, a whole part of IS could shut down. This pattern can be used to handle a collection of services.

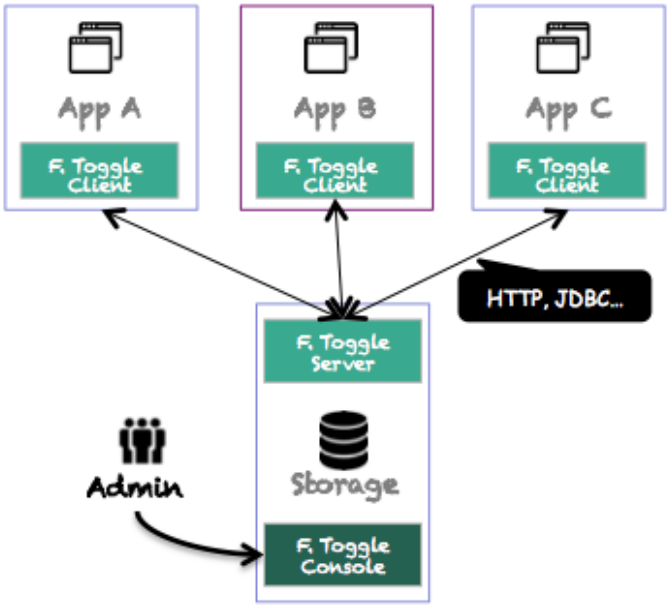


Figure 1.11. Service Catalog

1.3.3 Business Toggle

1.3.3.1 Business Rules

The flipping of features can be driven by a set of high level rules with a decision table or decision tree. For thoses cases you should implement your own logic and not only rely on a single flag. We use the term business rules but also for ff4j `Flipping Strategy`. The behaviour is described in the following flowchart :

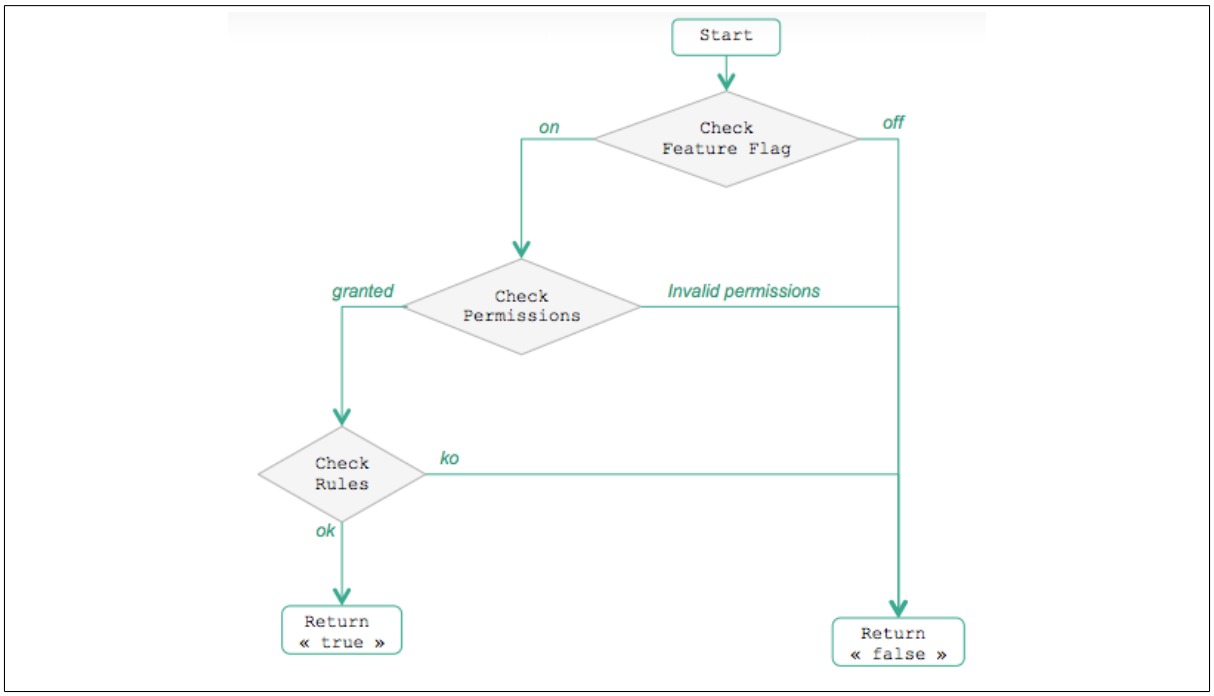


Figure 1.12. Business rules

1.3.3.2 A/B Testing

This term is well defined in wikipedia with : "*In marketing and business intelligence, A/B testing is jargon for a randomized experiment with two variants, A and B, which are the control and treatment in the controlled experiment. It is a form of statistical hypothesis testing with two variants leading to the technical term, Two-sample hypothesis testing, used in the field of statistics. Other terms used for this method include bucket tests and split testing but these terms have a wider applicability to more than two variants. In online settings, such as web design (especially user experience design), the goal is to identify changes to web pages that increase or maximize an outcome of interest (e.g., click-through rate for a banner advertisement). Formally the current web page is associated with the null hypothesis.*"

The feature toggle pattern allows to choose between one variant or another based on a custom strategy. The target is to measure the transformation rate to estimate its business added value. Monitoring capabilities are mandatory. FF4J saved any access to the feature out-of-the-box. Those statistics are available any time in the `EventRepository`, check monitoring part for more information.

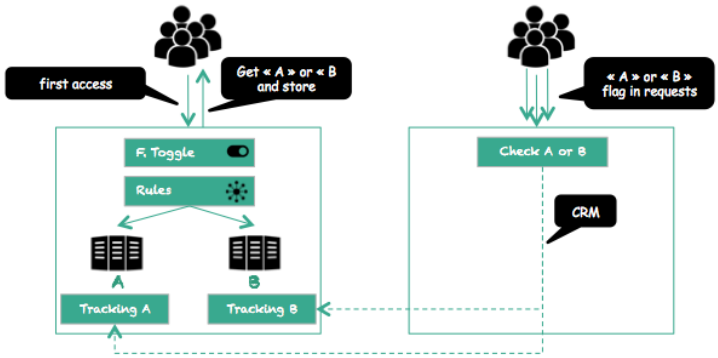


Figure 1.13. A/B testing

2. Getting Started

Note

All source code and working samples are available HERE for download and testing.

2.1 About Feature

A feature represents any service, treatment, or functionality. It is identified by a unique reference name (or uid) within the application runtime. In a feature toggle environnement each feature has a status or a state which indicate if it's enabled or not (ie: disabled). Toggling or flipping is the action to change the state of the feature.

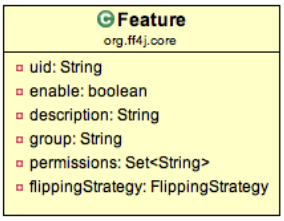


Figure 2.1. Feature UML Diagram

2.2 Introducing FeatureStore

The featureStore is the persistent unit to store the features with their attributes and status. It proposes a set of CRUD operations to work with features but also groups of feature or permissions on features. As detailed in further cahpters, different implementations will persist the data in different location such as relational database (rdbms, jdbc) , NoSQl databases (mongodb, redis), InMemory and even others. to

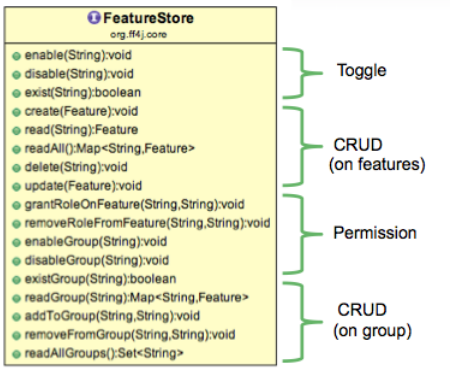


Figure 2.2. Feature Store

2.3 Ff4j core class

It's the **single class** to be used in your code. It's wrapped any other components of the framework, `FeatureStore` included).

Figure 2.3. Feature Store

2.4 First samples

In this part we guide you to create a working example from scratch

- Create a empty maven project

```
mvn archetype:create -Dpackaging=jar -Dversion=1.0 -DartifactId=ff4j-simple -DgroupId=org.ff4j.sample
```

- Declare this dependency into your `pom.xml` file/

```
<dependency>
 <groupId>org.ff4j</groupId>
  <artifactId>ff4j-core</artifactId>
  <version>1.2.0</version>
</dependency>
```

- Create the following `ff4j.xml` file in 'src/test/resources' folder (create it does not exist)

```
<?xml version="1.0" encoding="UTF-8" ?>
<features>
 <feature uid="sayHello"   enable="true" description="my first feature" />
 <feature uid="sayGoodBye" enable="false" />
</features>
```

- Write the following Junit test : (you may have to update junit version in your pom file)

```
package org.ff4j.sample;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.fail;

import org.ff4j.FF4j;
import org.junit.Test;

public class HelloWorldTest {

    @Test
    public void myFirstFF4JTest() {

        FF4j ff4j = new FF4j("ff4j.xml");
        assertEquals(2, ff4j.getFeatures().size());
        assertTrue(ff4j.exist("sayHello"));
        assertTrue(ff4j.check("sayHello"));

        // Test value at runtime
```

```
        if (ff4j.check("sayHello")) {
            // Feature ok !
            System.out.println("Hello World !");
        } else {
            fail();
        }
    }
}
```

Features are loaded from xml configuration file (ff4j.xml) and registered in a store (default is in-memory).

If a feature does not exist, the method `check(..)` will raise a FeatureNotFoundException but you can change this behaviour by setting the autoCreate flag as true. If feature is not found the method will return false.

- Update your unit test with this second method illustrating `autoCreate`

```
@Test
  public void autoCreateFeatureEnableTest() {

    // Default : store = inMemory, load features from ff4j.xml file
    FF4j ff4j = new FF4j("ff4j.xml");

    try {
        ff4j.check("autoCreatedFeature");
        fail(); // error is Expected here
    } catch(FeatureNotFoundException fnfe) {
        System.out.println("Standard behaviour");
    }

    // Change default behavior
    ff4j.autoCreate(true);

    if (!ff4j.check("autoCreatedFeature")) {
       System.out.println("Not available but code won't failed, feature created");
       assertTrue(ff4j.exist("autoCreatedFeature"));
       assertFalse(ff4j.check("autoCreatedFeature"));
    } else {
      fail();
    }
  }
```

Features can be created programmatically (for testing purposes for instance).

- Update your unit test with this third method illustrating dynamic creation of features

  Remember : Once implementing a Feature flipping pattern, services must be tested WITH and WITHOUT features enabled

```
@Test
    public void createFeatureDynamically() {

        // Initialize with empty store
        FF4j ff4j = new FF4j();

        // Dynamically register new features
        ff4j.create("f1").enable("f1");

        // Testing
        assertTrue(ff4j.exist("f1"));
        assertTrue(ff4j.check("f1"));
    }
```

As describe before the core sequence diagram to `check` the status of a feature is the following :

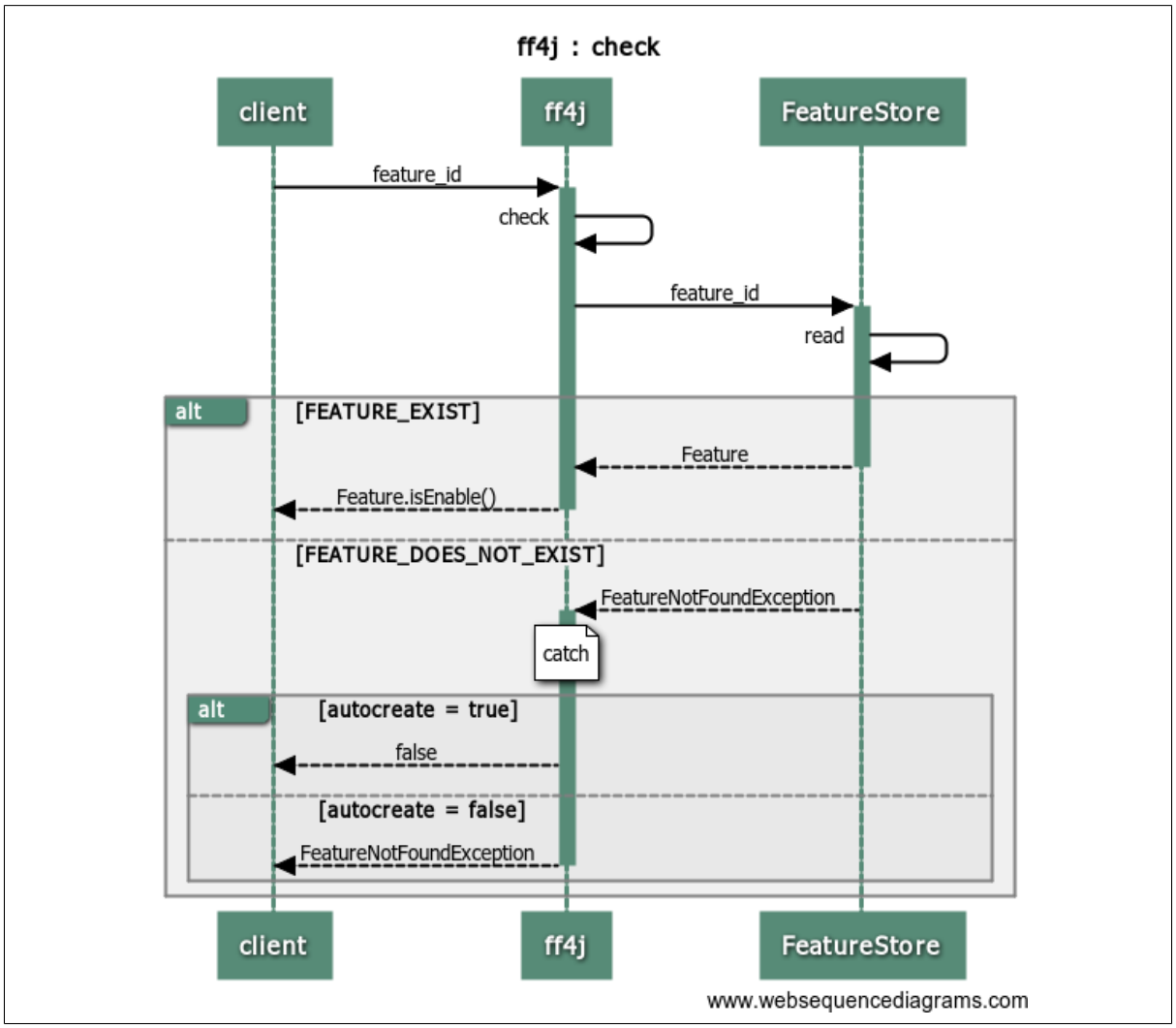


Figure 2.4. Sequence Diagram Core

2.5 Integration with Spring

The `ff4j` component can be easily defined as a Spring Bean.

- Add Spring dependencies to your project

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>4.0.5.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-context</artifactId>
   <version>4.0.5.RELEASE</version>
</dependency></programlisting>
```

- Add the following `applicationContext.xml` file to your `src/test/resources`

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
```

```
  <bean id="ff4j" class="org.ff4j.FF4j" >
    <property name="store" ref="ff4j.store.inmemory" />
  </bean>

  <bean id="ff4j.store.inmemory" class="org.ff4j.store.InMemoryFeatureStore" >
    <property name="location" value="ff4j.xml" />
  </bean>

</beans>
```

- The features are registered within in-memory store.Write the following spring-oriented test

```
        package org.ff4j.sample;

import static org.junit.Assert.fail;

import org.ff4j.FF4j;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath:*applicationContext.xml"})
public class CoreSpringTest {

    @Autowired
    private FF4j ff4j;

    @Test
    public void testWithSpring() {
        // Test value at runtime
        if (ff4j.check("sayHello")) {
            // Feature ok !
            System.out.println("Hello World !");
        } else {
            fail();
        }
    }
}
```

3. Core Concepts

3.1 Feature Groups

Features can be gathered as group. It is then possible to toggle the whole group. This capability can be useful for instance, if you want to group all the "user stories" of sprint in the same release.

- Let's create a new XML file `ff4j-group.xml` to illustrate

```
<?xml version="1.0" encoding="UTF-8" ?>
<features>

 <!-- Sample Feature Group -->
 <feature-group name="release-2.3">
   <feature uid="users-story1" enable="false" />
   <feature uid="users-story2" enable="false" />
 </feature-group>

 <feature uid="featA" enable="true"  />
 <feature uid="featB" enable="false" />

</features>
```

- Here is a sample utilisation of groups.

```
@Test
public void myGroupTest() {

   FF4j ff4j = new FF4j("ff4j-groups.xml");

   // Check features loaded
   assertEquals(4, ff4j.getFeatures().size());
   assertTrue(ff4j.exist("featA"));
   assertTrue(ff4j.exist("users-story1"));
   assertTrue(ff4j.getStore().existGroup("release-2.3"));
   System.out.println("Features loaded OK");

   // Given
   assertFalse(ff4j.check("users-story1"));
   assertFalse(ff4j.check("users-story2"));

   // When
   ff4j.enableGroup("release-2.3");

   // Then
   assertTrue(ff4j.check("users-story1"));
   assertTrue(ff4j.check("users-story2"));

}
```

- You can also access to all operation dynamically through the `FeatureStore`

```
@Test
public void workWithGroupTest() {

   // Given
   FF4j ff4j = new FF4j("ff4j-groups.xml");
   assertTrue(ff4j.exist("featA"));

   // When
   ff4j.getStore().addToGroup("featA", "new-group");

   // Then
   assertTrue(ff4j.getStore().existGroup("new-group"));
   assertTrue(ff4j.getStore().readAllGroups().contains("new-group"));
```

```
   Map<String, Feature> myGroup = ff4j.getStore().readGroup("new-group");
   assertTrue(myGroup.containsKey("featA"));

   // A feature can be in a single group
   // Here changing => deleting the last element of a group => deleting the group
   ff4j.getStore().addToGroup("featA", "group2");
   assertFalse(ff4j.getStore().existGroup("new-group"));
}
```

3.2 Aspect Oriented Programming (AOP)

3.2.1 Overview

From the beginning of this guide, we use intrusive tests statements within source code to perform flipping like in :

```
if (ff4j.check("featA")) {
  // new code
} else {
  // legacy
}
```

This approach is quite intrusive into source code. You can nested different feature toggles at you may consider to clean often your code and remove obsolete features. A good alternative is to rely on Dependency Injection, also called Inversion of control (ioc) to choose the correct implementation of the service at runtime.

Ff4j provide the `@Flip` annotation to perform flipping on methods using AOP proxies. At runtime, the target service is proxified by the ff4j component which choose an implementation instead of another using feature status (enable/disable). It leverage on Spring AOP Framework.

3.2.2 Illustrate with example

In the following chapter, we modify the project created in getting started to illustrate flipping through aop

- Add the dependency to `ff4j-aop` in your project

```
<dependency>
 <groupId>org.ff4j</groupId>
  <artifactId>ff4j-aop</artifactId>
  <version>1.2.0</version>
</dependency>
```

- Define a sample interface with the annotation :

```
public interface GreetingService {

   @Flip(name="language-french", alterBean="greeting.french")
   String sayHello(String name);

}
```

- Define a first implementation, to tell hello in english

```
@Component("greeting.english")
public class GreetingServiceEnglishImpl implements GreetingService {
    public String sayHello(String name) {
      return "Hello " + name;
    }
}
```

- Define a second implementation, to tell hello in french

```
@Component("greeting.french")
public class GreetingServiceFrenchImpl implements GreetingService {
  public String sayHello(String name) {
    return "Bonjour " + name;
  }
}
```

- The AOP capability leverage on Spring Framework. To enable the Autoproxy, please ensure that the package `org.ff4j.aop` is scanned by spring at startup. The `applicationContext-aop.xml` should look like :

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package="org.ff4j.aop, org.ff4j.sample"/>

  <bean id="ff4j" class="org.ff4j.FF4j" >
    <property name="store" ref="ff4j.store.inmemory" />
  </bean>

  <bean id="ff4j.store.inmemory" class="org.ff4j.store.InMemoryFeatureStore" >
    <property name="location" value="ff4j-aop.xml" />
  </bean>

</beans>
```

- Create a dedicated ff4j.xml file with the feature name `language-french` let's say `ff4j-demo-aop.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<features>
 <feature uid="language-french" enable="false" />
</features>
```

- Demonstrate how does it work through a test :

```
import junit.framework.Assert;

import org.ff4j.FF4j;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:*applicationContext-aop.xml")
public class FeatureFlippingThoughAopTest {

    @Autowired
    private FF4j ff4j;

    @Autowired
    @Qualifier("greeting.english")
    private GreetingService greeting;

    @Test
    public void testAOP() {
        Assert.assertTrue(greeting.sayHello("CLU").startsWith("Hello"));
        ff4j.enable("language-french");
```

```
        Assert.assertTrue(greeting.sayHello("CLU").startsWith("Bonjour"));
    }

}
```

3.3 Permissions and security

3.3.1 Overview

You may have to enable a feature only for a subset of your users. They are belong to a dedicated group or get a dedicated profile. With the `Canary Realease` pattern for instance, the feature could be activated only for beta-tester.

ff4j does not provideany users/groups definition system but, instead, leverage on existing one like Spring Security or Apache Chiro. A set of permissions is defined for each feature but the permissions must already exists in the external security provider. Permissions will be checked if, and only if, the feature is enabled.

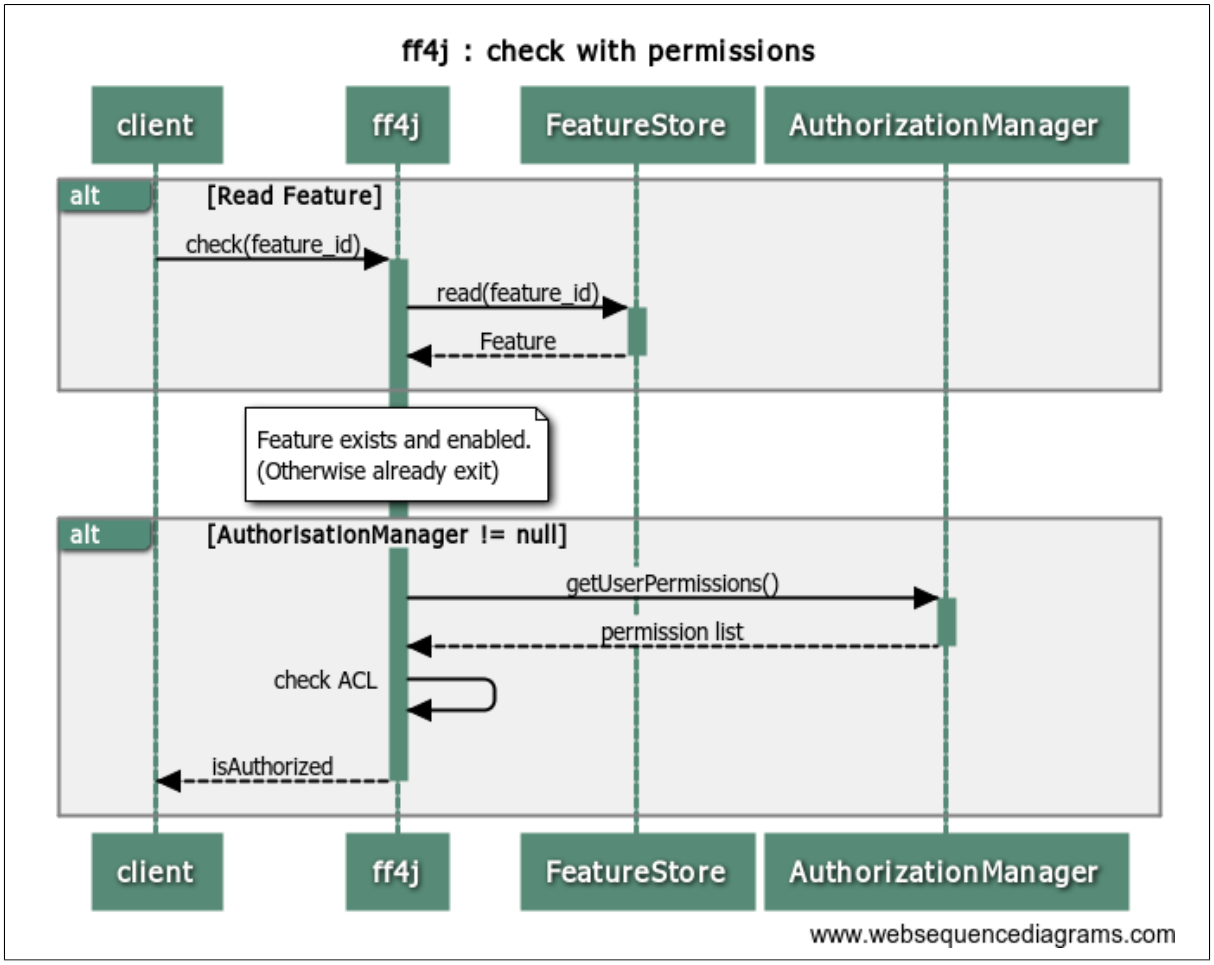


Figure 3.1. AuthorizationManager sequence diagram

3.3.2 AuthorizationManager

This is the class where ff4j evaluates users permissions against granted list at feature level. An implementation is available out-of-the-box to work with `Spring security` framework. There are 2 methods to implements. The first one is retrieving current user profiles (to be tested against features ACL), and the second will return a union of all permissions avaiable within the sysem. It's used in administration console to display permissions avec an editable list.

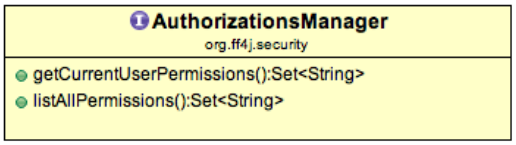


Figure 3.2. AuthorizationManager UML Diagram

3.3.3 Illustrate through sample code

In this sample we will create a custom implementatino of `AuthorizationManager` which keep the list of permissions in Memory.

- There is no new extra required dependency to implement the `AuthorizationManager` is in the `ff4j-core.jar` file. Here is a sample implementation.

```
public class CustomAuthorizationManager implements AuthorizationsManager {

  public static ThreadLocal<String> currentUserThreadLocal = new ThreadLocal<String>();

  private static final Map<String, Set<String>> permissions = new HashMap<String, Set<String>>();

  static {
    permissions.put("userA", new HashSet<String>(Arrays.asList("user", "admin", "beta")));
    permissions.put("userB", new HashSet<String>(Arrays.asList("user")));
    permissions.put("userC", new HashSet<String>(Arrays.asList("user", "beta")));
  }

  /** {@inheritDoc} */
  @Override
  public Set<String> getCurrentUserPermissions() {
    String currentUser = currentUserThreadLocal.get();
    return permissions.containsKey(currentUser) ? permissions.get(currentUser) : new
 HashSet<String>();
  }

  /** {@inheritDoc} */
  @Override
  public Set<String> listAllPermissions() {
    Set<String> allPermissions = new HashSet<String>();
    for (Set<String> subPersmission : permissions.values()) {
      allPermissions.addAll(subPersmission);
    }
    return allPermissions;
  }

}
```

- Create a `ff4j.xml` file with dedicated roles. A user will be able to use the `sayHello` feature it's enabled and if he has the permission `admin`. In the same way a user can use `sayGoodBye` if, and only if, he has the beta OR the user permission.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration>
<features>

 <feature uid="sayHello" description="my first feature" enable="true">
  <security>
   <role name="admin" />
  </security>

 </feature>

 <feature uid="sayGoodBye" description="null" enable="true">
  <security>
   <role name="beta" />
```

```
   <role name="user" />
  </security>
 </feature>

</features>
```

- Here is the unit test to illustrate :

```
@Test
public void sampleSecurityTest() {

  // Create FF4J
  FF4j ff4j = new FF4j("ff4j-security.xml");
  // Add the Authorization Manager Filter
  AuthorizationsManager authManager = new CustomAuthorizationManager();
  ff4j.setAuthorizationsManager(authManager);

  // Given : Feature exist and enable
  assertTrue(ff4j.exist("sayHello"));
  assertTrue(ff4j.getFeature("sayHello").isEnable());

  // Unknow user does not have any permission => check is false
  CustomAuthorizationManager.currentUserThreadLocal.set("unknown-user");
  System.out.println(authManager.getCurrentUserPermissions());
  assertFalse(ff4j.check("sayHello"));

  // userB exist bit he has not role Admin
  CustomAuthorizationManager.currentUserThreadLocal.set("userB");
  System.out.println(authManager.getCurrentUserPermissions());
  assertFalse(ff4j.check("sayHello"));

  // userA is admin
  CustomAuthorizationManager.currentUserThreadLocal.set("userA");
  System.out.println(authManager.getCurrentUserPermissions());
  assertTrue(ff4j.check("sayHello"));
}
```

3.3.4 Working with Spring Security

Even if creating a custom `AuthorizationManager` is possible, you may want to use a well defined security framework such as Spring Security. The support of the framework is provided out-of-the-box

- Add the following dependency to your `pom.xml` file.

```
<dependency>
 <groupId>org.ff4j</groupId>
  <artifactId>ff4j-aop</artifactId>
  <version>1.2.0</version>
</dependency>
```

- Define a spring security `UserDetails` implementation with the following `applicationContext-security.xml` file.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <bean id="ff4j" class="org.ff4j.FF4j" >
    <property name="store"      ref="ff4j.store.inmemory" />
    <property name="authorizationsManager" ref="ff4j.authorizationManager.spring" />
  </bean>

  <bean id="ff4j.store.inmemory" class="org.ff4j.store.InMemoryFeatureStore" >
```

```
    <property name="location" value="ff4j-security.xml" />
  </bean>

  <bean id="ff4j.authorizationManager.spring" class="org.ff4j.security.SpringSecurityAuthorisationManager"
 >
  </bean>

</beans>
```

- The `ff4j-security.xml` file has not changed from last sample

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration>
<features>

 <feature uid="sayHello" description="my first feature" enable="true">
  <security>
   <role name="admin" />
  </security>

 </feature>

 <feature uid="sayGoodBye" description="null" enable="true">
  <security>
   <role name="beta" />
   <role name="user" />
  </security>
 </feature>

</features>
```

- Create the following test. It instanciates a spring security context and authenticate a 'userA' with the permission 'beta'.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath:*applicationContext-security.xml"})
public class SampleSpringSecurityTest {

    @Autowired
    private FF4j ff4j;

    /** Security context. */
    private SecurityContext securityCtx;

    @Before
    public void setUp() throws Exception {
        securityCtx = SecurityContextHolder.getContext();

        // UserA got the roles : beta, user, admin
        List<GrantedAuthority> listOfRoles = new ArrayList<GrantedAuthority>();
        listOfRoles.add(new SimpleGrantedAuthority("beta"));
        User userA = new User("userA", "passwdA", listOfRoles);

        // Creadentials for UserA
        String userName = userA.getUsername();
        String passwd = userA.getPassword();
        UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(userName,
 passwd, listOfRoles);
        token.setDetails(userA);

        // Create a security context with
        SecurityContext context = new SecurityContextImpl();
        context.setAuthentication(token);
        SecurityContextHolder.setContext(context);
    }

    @Test
    public void testIsAuthenticatedAndAuthorized() {
```

```
        // Given userA is authenticated in Spring
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        Assert.assertTrue(auth.isAuthenticated());

        // UserA has not expected role 'admin'
        assertTrue(ff4j.exist("sayHello"));
        assertTrue(ff4j.getFeature("sayHello").isEnable());
        assertTrue(ff4j.getFeature("sayHello").getPermissions().contains("admin"));
        assertFalse(ff4j.check("sayHello"));

        // UserA has expected role 'beta'
        assertTrue(ff4j.exist("sayGoodBye"));
        assertTrue(ff4j.getFeature("sayGoodBye").isEnable());
        assertTrue(ff4j.getFeature("sayGoodBye").getPermissions().contains("beta"));
        assertTrue(ff4j.check("sayGoodBye"));
    }

    @After
    public void tearDown() {
        SecurityContextHolder.setContext(securityCtx);
    }
```

Note

The spring security context has been here created in the test, in web applications, the security Context within the HTTP thread with a `ThreadLocal`.

3.4 Flipping Strategy

3.4.1 Overview

As introduced in the first chapter, the behavior of a feature can be enslaved with your custom implementation and rules. With `ff4j`, once the feature is enabled AND current authenticated user is granted, a test is performed to evaluate the value of `FlippingStrategy`.

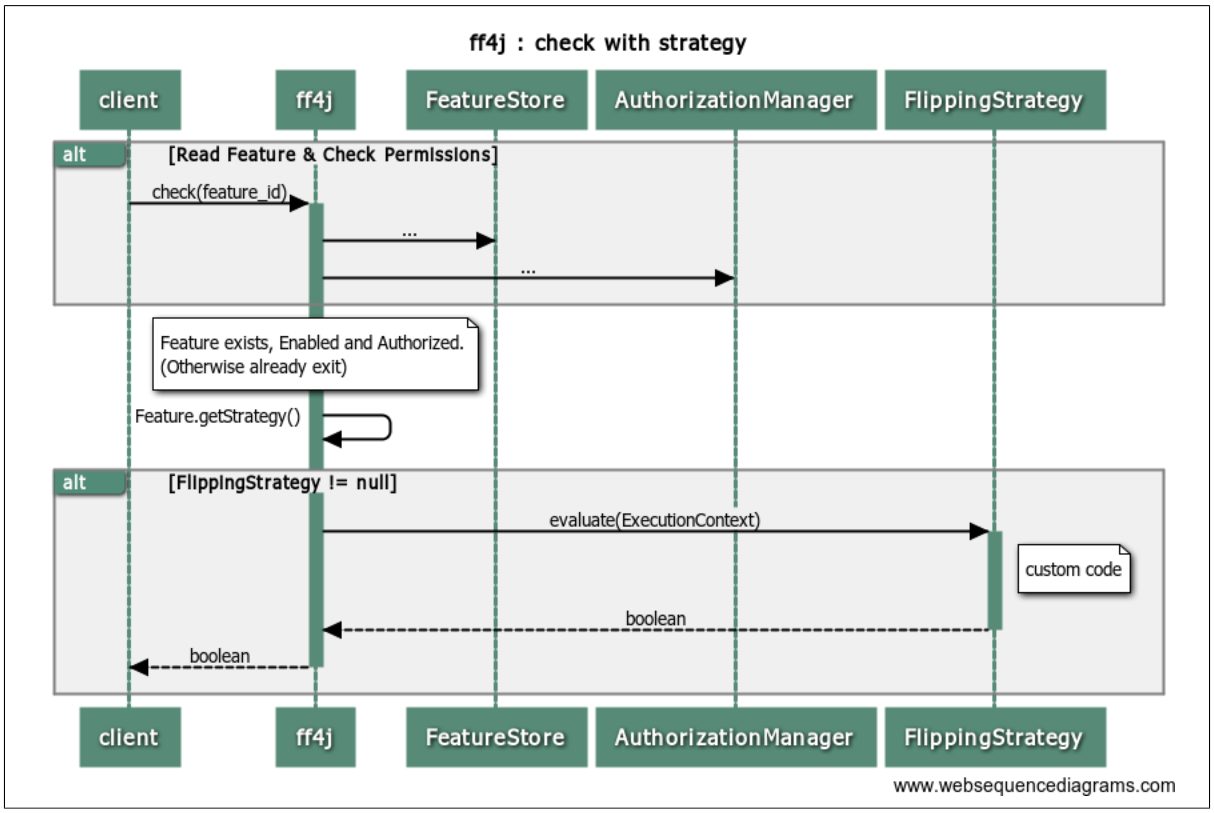


Figure 3.3. FlippingStrategy UML Diagram

The class is set up with a map of "initial parameters" and the `init(..)` method must be implemented. The getter of those parameters must also be implemented (for serialization purposes) and obviously the test is performed within `evaluate(...)` method.

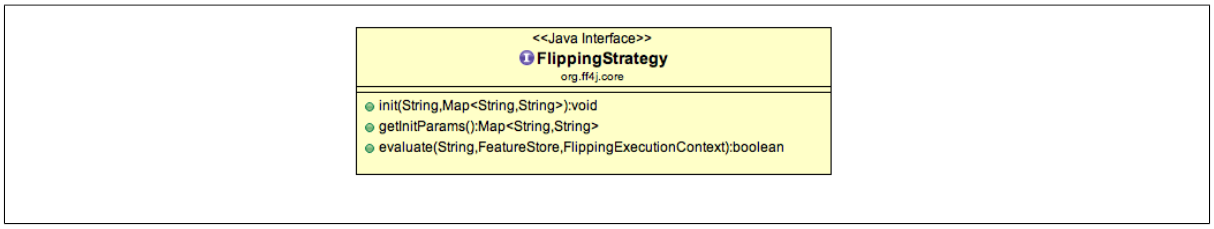


Figure 3.4. FlippingStrategy UML Diagram

The `evaluate()` method expects a `FlippingExecutionContext` which hold parameters as key/value pairs and provides to you the feature name and a reference to the feature store.

3.4.2 Illustrate custom strategies (1)

There is a bunch of strategies provided out-of-the-box but to understand the concept we propose to create our own. In this sample we will toggle feature if, and only if the request is made during office time let's say 09:00 to 18:00.

- There is no new extra required dependency to implement the `FlippingStrategy` is in the `ff4j-core.jar` file. Create the following class strategy class. Note that it inherit from `AbstractFlipStrategy`, it's not mandatory but provide a bunch of helpers.

```
public class OfficeHoursFlippingStrategy extends AbstractFlipStrategy {

    /** Start Hour. */
    private int start = 0;
```

```
    /** Hend Hour. */
    private int end = 0;

    /** {@inheritDoc} */
    @Override
    public void init(String featureName, Map<String, String> initValue) {
        super.init(featureName, initValue);
        assertRequiredParameter("startDate");
        assertRequiredParameter("endDate");
        start = new Integer(initValue.get("startDate"));
        end = new Integer(initValue.get("endDate"));
    }

    /** {@inheritDoc} */
    @Override
    public boolean evaluate(String fName, FeatureStore fStore, FlippingExecutionContext ctx) {
        int currentHour = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
        return (currentHour >= start && currentHour < end);
    }

}
```

- Create a `ff4j-strategy-1.xml` with a feature reference our new strategy :

```
<?xml version="1.0" encoding="UTF-8" ?>
<features>

 <feature uid="sayHello" enable="true" description="some desc">
  <flipstrategy class="org.ff4j.sample.strategy.OfficeHoursFlippingStrategy" >
   <param name="startDate">9</param>
   <param name="endDate">18</param>
  </flipstrategy>
 </feature>

</features>
```

- And the test to illustrate the behavior create the following unit test :

```
public class OfficeHoursFlippingStrategyTest  {

  // Initialization of target 'ff4j'
  private final FF4j ff4j = new FF4j("ff4j-strategy-1.xml");

  @Test
  public void testCustomStrategy() throws Exception {
    // Given
    assertTrue(ff4j.exist("sayHello"));
    FlippingStrategy fs = ff4j.getFeature("sayHello").getFlippingStrategy();
    assertTrue(fs.getClass() == OfficeHoursFlippingStrategy.class);
    assertEquals("9", fs.getInitParams().get("startDate"));

    // When
    int hour = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
    boolean isNowOfficeTime = (hour > 9) && (hour < 18);

    // Then
    assertEquals(isNowOfficeTime, ff4j.check("sayHello"));
  }
}
```

3.4.3 Illustrate custom strategies (2)

This second sample will illustrate the `FlippingExecutionContext` behaviour. We create a strategy to enable a feature only for a subset of geographical regions.

- Create a strategy, initialized with the granted regions and expected user region's within execution context :

```
public class RegionFlippingStrategy extends AbstractFlipStrategy {

  /** initial parameter. */
  private static final String INIT_PARAMNAME_REGIONS = "grantedRegions";

  /** current user attribute */
  public static final String PARAMNAME_USER_REGION = "region";

  /** Initial Granted Regions. */
  private final Set<String> setOfGrantedRegions = new HashSet<String>();

  /** {@inheritDoc} */
  @Override
  public void init(String featureName, Map<String, String> initValue) {
    super.init(featureName, initValue);
    assertRequiredParameter(INIT_PARAMNAME_REGIONS);
    String[] arrayOfRegions = initValue.get(INIT_PARAMNAME_REGIONS).split(",");
    setOfGrantedRegions.addAll(Arrays.asList(arrayOfRegions));
  }

  /** {@inheritDoc} */
  @Override
  public boolean evaluate(String fName, FeatureStore fStore, FlippingExecutionContext ctx) {
    // true means required here
    String userRegion = ctx.getString(PARAMNAME_USER_REGION, true);
    return setOfGrantedRegions.contains(userRegion);
  }
}
```

- Create a xml file with a feature using the strategy :

```
<?xml version="1.0" encoding="UTF-8" ?>
<features>
  <feature uid="notForEurop" enable="true" >
    <flipstrategy class="org.ff4j.sample.strategy.RegionFlippingStrategy" >
      <param name="grantedRegions">ASIA,AMER</param>
    </flipstrategy>
  </feature>
</features>
```

- Create the unit test :

```
public class RegionFlippingStrategyTest {

  // ff4j
  private final FF4j ff4j = new FF4j("ff4j-strategy-2.xml");

  // sample execution context
  private final FlippingExecutionContext fex = new FlippingExecutionContext();

  @Test
  public void testRegionStrategy() throws Exception {
    // Given
    assertTrue(ff4j.exist("notForEurop"));
    FlippingStrategy fs = ff4j.getFeature("notForEurop").getFlippingStrategy();
    assertTrue(fs.getClass() == RegionFlippingStrategy.class);
    assertEquals("ASIA,AMER", fs.getInitParams().get("grantedRegions"));

    // When
    fex.addValue(RegionFlippingStrategy.PARAMNAME_USER_REGION, "AMER");
    // Then
    assertTrue(ff4j.check("notForEurop", fex));

    // When
    fex.addValue(RegionFlippingStrategy.PARAMNAME_USER_REGION, "EUROP");
    // Then
    assertFalse(ff4j.check("notForEurop", fex));
```

```
  }
}
```

3.4.4 Overriding Strategy

Sometimes, even if a feature has a defined strategy, you would like to override it for a single invocation. The `FF4J` class provides another `check()` method which take a flipping strategy as second parameter. The strategy will overrides the existing one.

- Here is a sample unit test to illustrate the behavior :

```
public class OverridingStrategyTest {
  // ff4j
  private final FF4j ff4j = new FF4j("ff4j-strategy-1.xml");

  @Test
  public void testBehaviourOfOverriding() {
    assertTrue(ff4j.exist("sayHello"));

    // Behaviour of the strategy
    FlippingStrategy fs = ff4j.getFeature("sayHello").getFlippingStrategy();
    assertTrue(fs.getClass() == OfficeHoursFlippingStrategy.class);
    assertEquals("9", fs.getInitParams().get("startDate"));
    int hour = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
    boolean isNowOfficeTime = (hour > 9) & (hour < 18);
    assertEquals(isNowOfficeTime, ff4j.check("sayHello"));

    // New Strategy : ReleaseDate with date in the past ==> Always true
    FlippingStrategy newStrategy = new ReleaseDateFlipStrategy(new Date(System.currentTimeMillis() -
 100000));
    assertTrue(ff4j.checkOveridingStrategy("sayHello", newStrategy, null));
  }
}
```

3.4.5 Available Strategies

As previously detailed, a `FlippingStrategy` implements a custom logic to perform flipping. The ff4j framework provides a set of strategies out-of-the-box.

3.4.5.1 Expression Language

The idea behind this strategy is to evaluate a boolean expression by combining several feature with moore algebra. AND, OR, NOT and brackets are available. This way you can define features depending of status of other.

- Create a XML file, we want to check that feature 'D' is flipped if : `(A AND B ) OR NOT(C) OR NOT(B)`. The expression is wrapped in a CDATA block. The charater for the operand AND is `&`, for OR it's `|` and for NOT it's `!`

```
<?xml version="1.0" encoding="UTF-8" ?>
<features>
  <feature uid="A" enable="true"  />
  <feature uid="B" enable="false" />
  <feature uid="C" enable="false" />
  <feature uid="D" enable="true">
    <flipstrategy class="org.ff4j.strategy.el.ExpressionFlipStrategy">
      <param name="expression"><![CDATA[A & B | !C | !B]]></param>
    </flipstrategy>
  </feature>
  <feature uid="E" enable="true">
    <flipstrategy class="org.ff4j.strategy.el.ExpressionFlipStrategy">
      <param name="expression"><![CDATA[A & B]]></param>
```

```
    </flipstrategy>
  </feature>
  <feature uid="F" enable="true">
    <flipstrategy class="org.ff4j.strategy.el.ExpressionFlipStrategy">
      <param name="expression"><![CDATA[A | B]]></param>
    </flipstrategy>
  </feature>
</features>
```

- The behaviour is detailed in the following unit test

```
public class ExpressionStrategyTest {

  // ff4j
  private final FF4j ff4j = new FF4j("ff4j-strategy-expression.xml");

  @Test
   public void testExpressions() {

     // Given
     assertTrue(ff4j.exist("A"));
     assertTrue(ff4j.exist("B"));
     assertTrue(ff4j.exist("C"));
     ff4j.enable("D");
     ff4j.enable("E");
     ff4j.enable("F");

     // When A=FALSE, B=TRUE, C=TRUE
     assertFalse(ff4j.check("A"));
     assertTrue(ff4j.check("B"));
     assertTrue(ff4j.check("C"));

     // THEN
     // E = A AND B = FALSE AND TRUE = FALSE
     assertFalse(ff4j.check("E"));
     // F = A OR R = FALSE OR TRUE = TRUE
     assertTrue(ff4j.check("F"));
     // D = (A AND B) OR NOT(B) OR NOT(C) = (false & true) or false or false
     assertFalse(ff4j.check("D"));

     // When enabling A
     ff4j.enable("A");

     // THEN
     // E = A AND B = TRUE AND TRUE = TRUE
     assertTrue(ff4j.check("E"));
     // F = A AND B = TRUE OR TRUE = TRUE
     assertTrue(ff4j.check("F"));
     // D = (A AND B) OR NOT(B) OR NOT(C) = (true & true) or false or false
     assertTrue(ff4j.check("D"));
  }
}
```

3.4.5.2 ReleaseDate

The purpose of this strategy is the made a feature available from a fixed date (like a releaseDate). Before the defined date, the feature is always false and after it's true. The format to set up the date is YYYY-MM-dd-HH:mm

- Here a sample XML file :

```
<?xml version="1.0" encoding="UTF-8" ?>
<features>
  <feature uid="PAST" enable="true" description="Always true as in the past">
    <flipstrategy class="org.ff4j.strategy.ReleaseDateFlipStrategy" >
      <param name="releaseDate" value="2013-07-14-14:00" />
    </flipstrategy>
  </feature>
```

```
  <feature uid="FUTURE" enable="true" description="Always false as in the (far) future">
    <flipstrategy class="org.ff4j.strategy.ReleaseDateFlipStrategy" >
      <param name="releaseDate" value="3013-07-14-14:00" />
    </flipstrategy>
  </feature>
</features>
```

- And the related unit test :

```
public class ReleaseDateFlipStrategyTest  {

  // initialize ff4j
  FF4j ff4j = new FF4j("ff4j-strategy-releasedate.xml");

  @Test
  public void testReleaseDateStrategy() throws ParseException {

    // Given
    assertTrue(ff4j.exist("PAST"));
    Feature fPast = ff4j.getFeature("PAST");
    ReleaseDateFlipStrategy rdsPast = (ReleaseDateFlipStrategy) fPast.getFlippingStrategy();
    assertTrue(new Date().after(rdsPast.getReleaseDate()));
    // Then
    assertTrue(ff4j.check("PAST"));

    // Given
    assertTrue(ff4j.exist("FUTURE"));
    Feature fFuture = ff4j.getFeature("FUTURE");
    ReleaseDateFlipStrategy rdsFuture = (ReleaseDateFlipStrategy) fFuture.getFlippingStrategy();
    Assert.assertTrue(new Date().before(rdsFuture.getReleaseDate()));
    // Then
    assertFalse(ff4j.check("FUTURE"));
  }
}
```

3.4.5.3 ClientList Strategy

The purpose of this strategy is to enable a feature for a limited list of clients. Each client must present its 'hostname' in the context. If the hostname is in the white list, it's ok. The attribute to set up is `clientHostName`. The values are separated by a comma, there are no spaces between values.

- Here a sample XML file :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration>
<features>
  <feature uid="pingCluster" enable="true" description="limit client hosts">
    <flipstrategy class="org.ff4j.strategy.ClientFilterStrategy" >
      <param name="grantedClients">127.0.0.1,srvprd01,srvprd02</param>
    </flipstrategy>
  </feature>
</features>
```

- And the related unit test :

```
public class ClientListStrategyTest {

  // initialize ff4j
  FF4j ff4j = new FF4j("ff4j-strategy-clientfilter.xml");

  @Test
  public void testClientFilter() {
    // Given
    assertTrue(ff4j.exist("pingCluster"));
    assertTrue(ff4j.getFeature("pingCluster").isEnable());
```

```
    // When no host provided, Then error
    try {
      assertFalse(ff4j.check("pingCluster"));
      fail(); // error as parameter not present in execution context
    } catch (IllegalArgumentException iae) {
      assertTrue(iae.getMessage().contains(ClientFilterStrategy.CLIENT_HOSTNAME));
    }

    // When invalid host provided, Then unavailable
    FlippingExecutionContext fex = new FlippingExecutionContext();
    fex.addValue(ClientFilterStrategy.CLIENT_HOSTNAME, "invalid");
    assertFalse(ff4j.check("pingCluster", fex));

    // When correct hostname... OK
    fex.addValue(ClientFilterStrategy.CLIENT_HOSTNAME, "srvprd01");
    assertTrue(ff4j.check("pingCluster", fex));
  }
}
```

3.4.5.4 ServerFilterList Strategy

The purpose of this strategy is to enable a feature for a limited list of servers. The feature will be available only if the hostname of hosting server is in the white list. The attribute to set up is `serverHostName` but it'not required. If not provided ff4j will ask the JVM for the current hostname (through the `InetAddress`) The values are separated by a comma, there are no spaces between values.

- Here a sample XML file :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration>
<features>
  <feature uid="onlyOnPRODServers" enable="true" description="some desccheck hostname">
    <flipstrategy class="org.ff4j.strategy.ServerFilterStrategy" >
      <param name="grantedServers">srvprd01,srvprd02,srvprd03</param>
    </flipstrategy>
  </feature>
 </features>
```

- And the related unit test :

```
public class ServerListStrategyTest {

  // initialize ff4j
  FF4j ff4j = new FF4j("ff4j-strategy-serverfilter.xml");

  @Test
  public void testServerFilter() throws UnknownHostException {
    // Given
    assertTrue(ff4j.exist("onlyOnPRODServers"));
    assertTrue(ff4j.getFeature("onlyOnPRODServers").isEnable());

    // When invalid host provided, Then unavailable
    FlippingExecutionContext fex = new FlippingExecutionContext();
    fex.addValue(ServerFilterStrategy.SERVER_HOSTNAME, "invalid");
    assertFalse(ff4j.check("onlyOnPRODServers", fex));

    // When correct hostname... OK
    fex.addValue(ServerFilterStrategy.SERVER_HOSTNAME, "srvprd01");
    assertTrue(ff4j.check("onlyOnPRODServers", fex));

    // When no host provided, Then try to identified by itself but not SECURE
    System.out.println("Trying..." + InetAddress.getLocalHost().getHostName() + " against white
 list");
    // my laptop hostname is not in the whitelist
    assertFalse(ff4j.check("onlyOnPRODServers"));
  }
}
```

3.4.5.5 Ponderation

The purpose of this strategy is to enable a feature for a percentage of requests. It could be useful in Dark Launch zero downtime deployment pattern for instance. It expected a parameter `weight` but if not provided is set up to its default value 0.5

- Here a sample XML file :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration>
<features>

  <!-- Ponderation to 0 -->
  <feature uid="pond_0"  enable="true" description="some desc">
    <flipstrategy class="org.ff4j.strategy.PonderationStrategy" >
      <param name="weight" value="0" />
    </flipstrategy>
  </feature>

  <!-- Ponderation to 1 -->
  <feature uid="pond_1" enable="true" description="some desc" >
    <flipstrategy class="org.ff4j.strategy.PonderationStrategy" >
      <param name="weight" value="1" />
    </flipstrategy>
  </feature>

  <feature uid="pond_06"  enable="true" description="some desc">
    <flipstrategy class="org.ff4j.strategy.PonderationStrategy" >
      <param name="weight" value="0.6" />
    </flipstrategy>
  </feature>

  <feature uid="pondDefault" enable="true" description="some desc">
    <flipstrategy class="org.ff4j.strategy.PonderationStrategy" />
  </feature>

</features>
```

- And the related unit test :

```
public class PonderationFlippingStrategyTest {

  // initialize ff4j
  FF4j ff4j = new FF4j("ff4j-strategy-ponderation.xml");

  @Test
  public void testPonderation() {

    // Given : weight = 0
    assertTrue(ff4j.exist("pond_0"));
    // Then => always false
    assertFalse(ff4j.check("pond_0"));

    // Given : weight = 100%
    assertTrue(ff4j.exist("pond_1"));
    // Then => Always true
    assertTrue(ff4j.check("pond_1"));

    // Given : weight = 60%
    assertTrue(ff4j.exist("pond_06"));
    // When : Try 1 million times
    double success = 0.0;
    for (int i = 0; i < 1000000; i++) {
       if (ff4j.check("pond_06")) {
         success++;
       }
    }
    // Then, percentage ok with great precision
```

```
    double resultPercent = success / 1000000;
    assertTrue(resultPercent < (0.6 + 0.001));
    assertTrue(resultPercent > (0.6 - 0.001));
  }
}
```

3.5 Feature Stores

3.5.1 Introduction

As already introduced in getting started the `FeatureStore` is the persistent unit whare are saved the features with their attributes and status. It proposes a set of CRUD operations to work with features but also, groups of features, and, permissions on features.

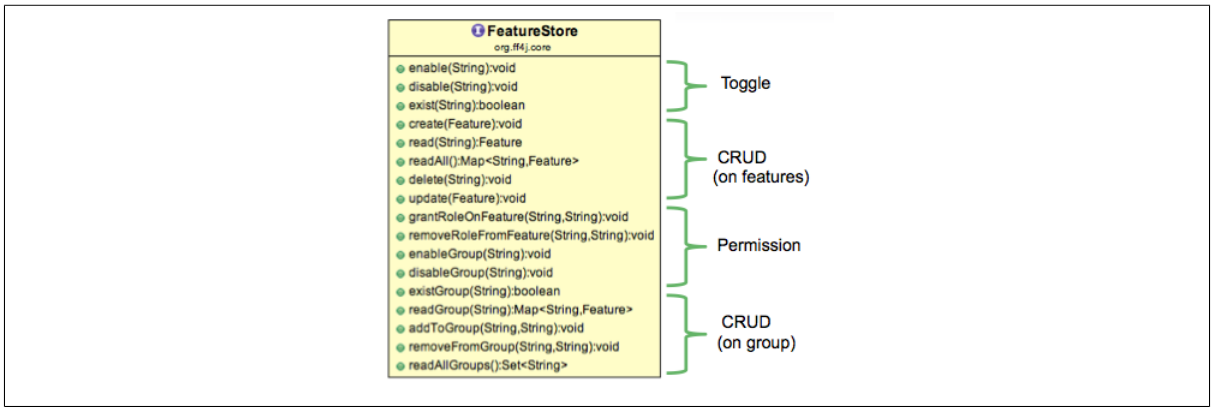


Figure 3.5. Feature Store

The `FeatureStore` is only an interface and several implementations are available for different kind of storage middleware. All examples in this guide have been written using the simplest yet less powerful store the default InMemeoryStore relying on XML, with other stores any modifications perform on features are kept.

3.5.1.1 Objectives

There are multiple purposed of the store :

- It allows to enable or disable features AT RUNTIME.
- It allows to PERSIST status of features even when restarting applications (except for InMemoryStore/ XML stores).
- It allows to synchronized status of features between DIFFERENT APPLICATIONS (except for InMemoryStore/XML stores)

3.5.1.2 Architecture Patterns

An "embedded" or "in memory" `FeatureStore` has 2 main drawbacks. The modifications performed at runtime are lost when restarting, you cannot be consistent in a cluster with several nodes. Yet it's the most fast. You must consider your production environment requirements to make good choices.

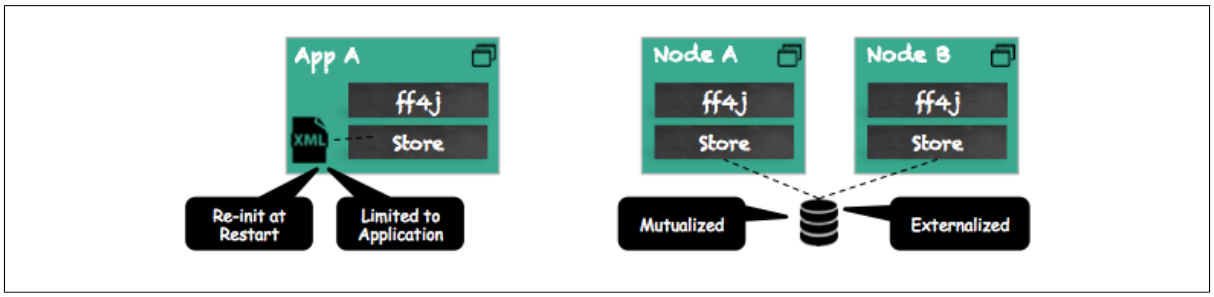


Figure 3.6. Embedded vs Externalized Store

3.5.2 InMemoryFeatureStore

The implementation of this `FeatureStore` parses an XML document (file or stream) at startup and store the features as a ConcurrentHashMap in memory. It's the default implementation of FF4J. It does not required any external librairies and is available in the module `ff4j-core`. The XML schema of the file (or XSD) can be find there. http://ff4j.org/schema/ff4j.xsd

You can declared the schema in the XML file with the following schema :

```
<?xml version="1.0" encoding="UTF-8" ?>
<features xmlns="http://www.ff4j.org/schema/ff4j"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ff4j.org/schema/ff4j http://ff4j.org/schema/ff4j.xsd">
  <!-- Here your declarations -- >
</features>
```

There are a lot of samples in this reference guide. Yet, let's summarize everything in a quite complete one

```
<?xml version="1.0" encoding="UTF-8" ?>
<features xmlns="http://www.ff4j.org/schema/ff4j" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.ff4j.org/schema/ff4j http://ff4j.org/schema/ff4j.xsd">

  <!-- Simplest -->
  <feature uid="A" enable="true" />

  <!-- Add description -->
  <feature uid="B" description="Expect to say good bye..." enable="false" />

  <!-- Security stuff -->
  <feature uid="C" enable="false">
    <security>
      <role name="USER"  />
   <role name="ADMIN" />
 </security>
  </feature>

  <!-- Some strategies and a group -->
  <feature-group name="strategies">

    <feature uid="S1" enable="true">
   <flipstrategy class="org.ff4j.strategy.el.ExpressionFlipStrategy">
        <param name="expression" value="A | B" />
      </flipstrategy>
    </feature>

    <feature uid="S2" enable="true">
      <flipstrategy class="org.ff4j.strategy.ReleaseDateFlipStrategy">
        <param name="releaseDate" value="2013-07-14-14:00" />
      </flipstrategy>
    </feature>

    <feature uid="S3" description="null" enable="true">
      <flipstrategy class="org.ff4j.strategy.PonderationStrategy">
```

```
        <param name="weight" value="0.5" />
      </flipstrategy>
    </feature>

    <feature uid="S4" description="z" enable="true">
      <flipstrategy class="org.ff4j.strategy.ClientFilterStrategy">
        <param name="grantedClients" value="c1,c2" />
      </flipstrategy>
    </feature>

    <feature uid="S5" description="null" enable="true">
      <flipstrategy class="org.ff4j.strategy.ServerFilterStrategy">
        <param name="grantedServers" value="s1,s2" />
      </flipstrategy>
    </feature>

  </feature-group>
</features>
```

The XML format can be generated whatever the `FeatureStore` configured. It should be use to export configuration from an environment and insert into another. As detailed further, this operation is available in the web console.

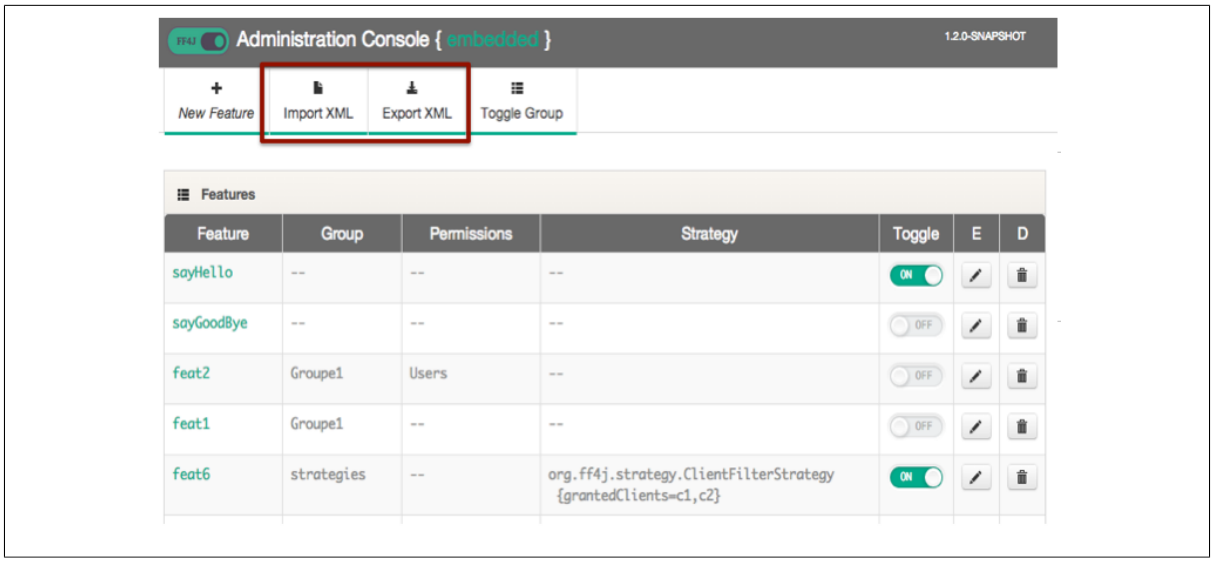


Figure 3.7. Export and Imports actions in web console

If you would like to do it in your own code, here is the way to do it.

```
public class ImportExportXmlTest {

    @Test
    public void testImport() throws FileNotFoundException {
        // Given
        FF4j ff4j = new FF4j();
        // When
        FileInputStream fis = new FileInputStream(new File("src/test/resources/ff4j.xml"));
        Map<String, Feature> mapsOfFeat = new FeatureXmlParser().parseConfigurationFile(fis);
        for (Entry<String, Feature> feature : mapsOfFeat.entrySet()) {
            if (ff4j.exist(feature.getKey())) {
                ff4j.getStore().update(feature.getValue());
            } else {
                ff4j.getStore().create(feature.getValue());
            }
        }
        // Then
        assertEquals(2, ff4j.getFeatures().size());
    }
```

```
    @Test
    public void testExport() throws IOException {
        FF4j ff4j = new FF4j("ff4j.xml");

        InputStream in = ff4j.exportFeatures();

        // Write into console
        byte[] bbuf = new byte[4096];
        int length = 0;
        while ((in != null) && (length != -1)) {
            length = in.read(bbuf);
        }
        System.out.print(new String(bbuf));
    }
}
```

3.5.3 Relational Database FeatureStore

The first store to externalize feature is to rely on a "database", a relational database to be exact. It's accessed through the Java Database Connectivity API, JDBC.

3.5.3.1 Data Model

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

```
-- Main Table to store Features
CREATE TABLE FF4J_FEATURES (
  "FEAT_UID"      VARCHAR(100),
  "ENABLE"    INTEGER NOT NULL,
  "DESCRIPTION"  VARCHAR(255),
  "STRATEGY"  VARCHAR(255),
  "EXPRESSION"     VARCHAR(255),
  "GROUPNAME"  VARCHAR(255),
  PRIMARY KEY("FEAT_UID")
);

-- Roles to store ACL, FK to main table
CREATE TABLE FF4J_ROLES (
  "FEAT_UID"     VARCHAR(50) REFERENCES FF4J_FEATURES("FEAT_UID"),
  "ROLE_NAME"    VARCHAR(50),
  PRIMARY KEY("FEAT_UID", "ROLE_NAME")
);
```

3.5.3.2 Core JDBC

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.5.3.3 Spring JDBC

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.5.4 MongoDB FeatureStore

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.5.4.1 Overview

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.5.4.2 Sample Code

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.5.5 Remote HTTP (client) FeatureStore

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.5.5.1 Overview

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.5.5.2 Sample Code

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.6 Caching

Text here

3.6.1 Architecture Concerns

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.6.2 Working with EHCache

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.6.3 Working with Redis

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.7 Monitoring

Text here

3.7.1 Overview

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.7.2 Metrics

Usage vs actions, exporter Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

3.7.3 Curves and Graphics

Usage vs actions, exporter Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4. Web

4.1 Embedded Console

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.1.1 Overview

Phylosophy, capability, architecture, limits as security Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.1.2 Declaring Servlet

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.1.3 User Guide

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.2 Taglib Library

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.2.1 Introducing Taglib

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.2.2 Available Tags

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.3 RestFul API

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.3.1 Introduction

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.3.2 State Diagram

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.3.3 API BluePrint

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.3.4 Security

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.3.5 Sample Clients

HttpClient, CURL, Javascript.... Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.4 WebConsole Full Stack

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.4.1 Introduction

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.4.2 Configuration

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

4.4.3 User Guide

Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

5. Advanced Concepts

5.1 JMX Support

Definition, UML Diagram, description attributes, ff4j definition, status

5.1.1 Overview

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.

5.1.2 Sample Code

Status, crud, storage,Sed egestas molestie elit. Mauris urna mi, scelerisque vitae, ultrices vel, euismod vel, eros. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Pellentesque dictum felis a nisi.