Giteway 1.0

Technical presentation

Table of content

1. Architecture

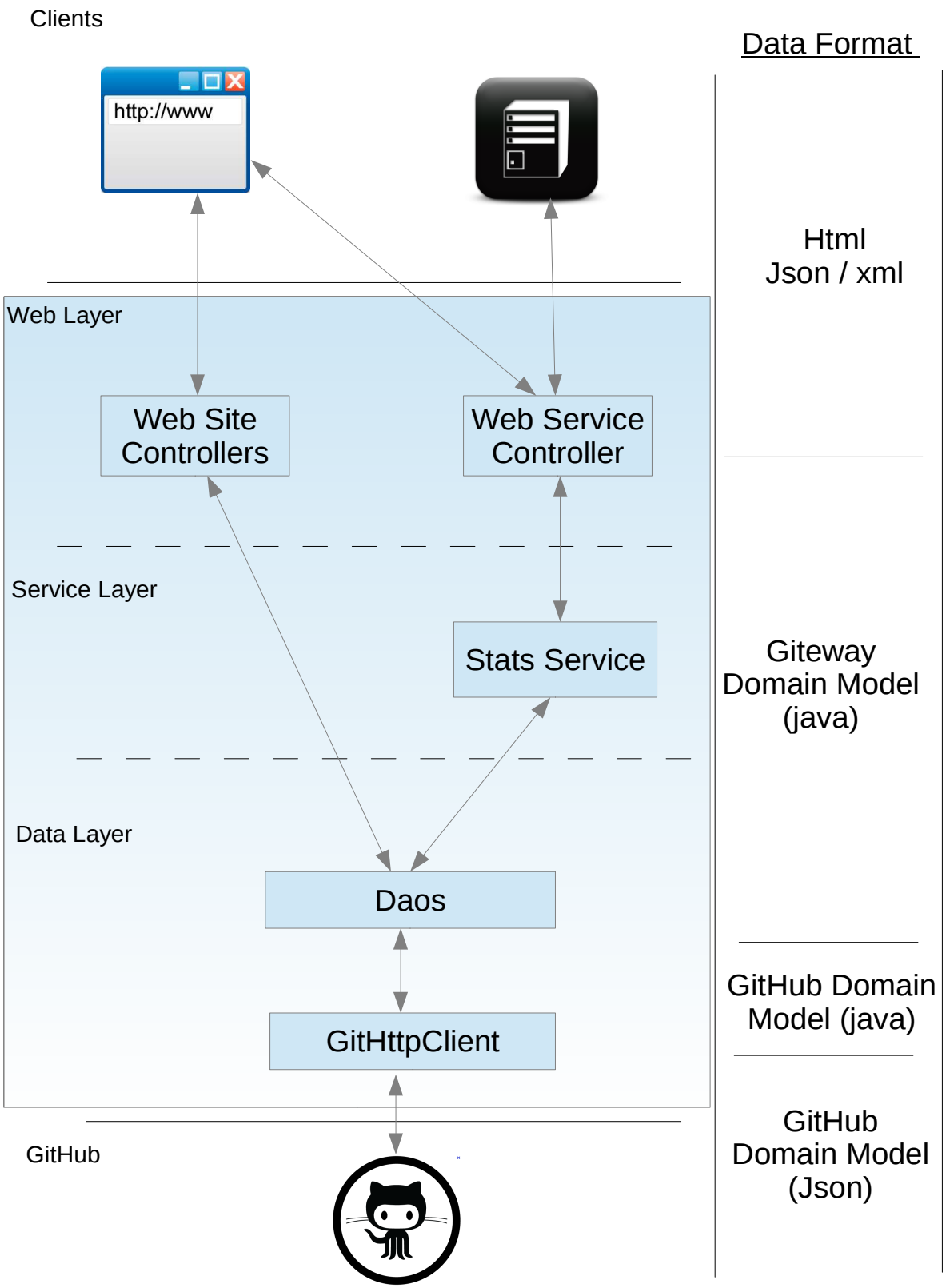


Figure 1 - Global architecture

2. Deployment

Source code : *git://github.com/Couettos/giteway.git*

Deployed application: http://giteway.cloudfoundry.com

If you wish to deploy giteway in your local environment, please check the deployment instructions in the README file at the root of the project.

3. Technology stack

Spring 3.1: *Spring-core, Spring-webmvc, Spring-oxm, Spring-context, Spring-aop*

AOP: *Spring-AspectJ*

Logging: *SLF4J – Log4j*

Serializers/Deserializers: *Jackson, Castor*

Rest client: *Apache Commons-http*

Utilities: *LambdaJ-Hamcrest*

Caching: *Spring-EHCache*

Testing: *Junit, Mockito, Spring-test*

Front: *JSP, JSTL, JQuery,* JQueryUI

Build: Maven-Sonar

4. Model

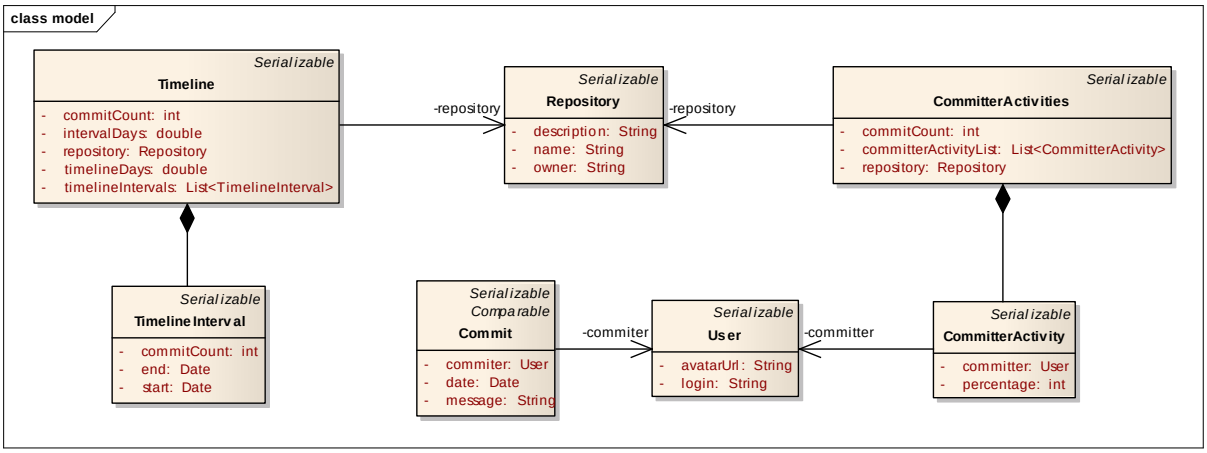


Figure 2 - The model

The model is loosely coupled from GitHub domain model. This architectural choice has been motivated by the inconsistency of GitHub DM. For example, the repositories returned by the search by keyword and by the search by user are different.

GitHub Objects are then considered as Value Objects or DTOs.

Advantages:

- Clearer architecture and object responsibilities
- Reduce the complexity of GitHub Domain Model
- Isolates the impacts of attribute modification on both DM.

Drawbacks:

- Glue code
- Many Transfer Object instantiations per request
- More code to write if a new attribute has to be mapped

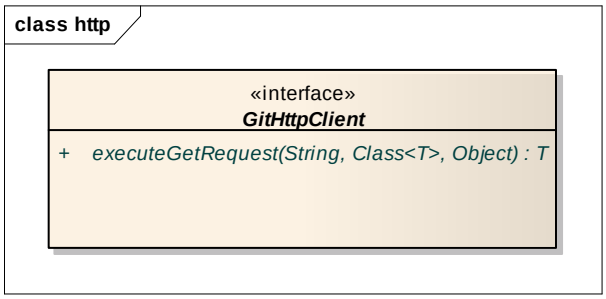
5. Data Access Layer

The data access layer requests data from GitHub API. It is an abstraction layer over HTTP and GitHub DM.

The data access tier provides two mechanisms:

1st: Request Github over HTTP, map it to DTOs

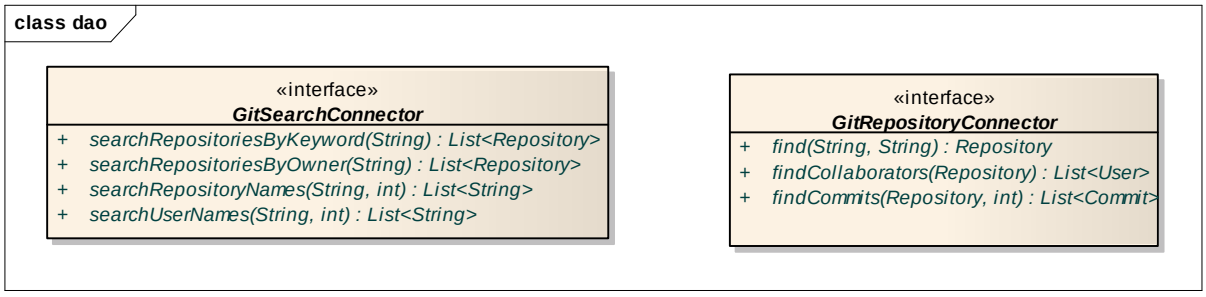
The data is accessible over HTTP in a json format. The interface GitHttpClient requests the data and converts it into a set of mapped class representing the GitHub DM (DTOs).



Note : GitHttpClient encapsulates a DefaultHttpClient thread safe singleton object which is instantiated with a PoolingClientConnectionManager. This approach enables DefaultHttpClient to reuse connections from the pool. Between two requests, the connections are kept alive. This is of great interest as the SSL handshake increases the latency.

2nd: Map GitHub DM to Giteway DM

The DAOs receive DTOs from GitHttpClient and convert these into Giteway domain model.

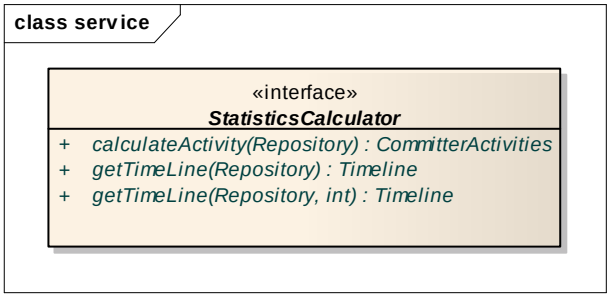


6. Service Layer

The service layer encapsulates the business logic of Giteway. The component calculates statistics. The data is accessed through the data access layer.

The API:

- Defines the timeline
- Calculates the user's collaboration

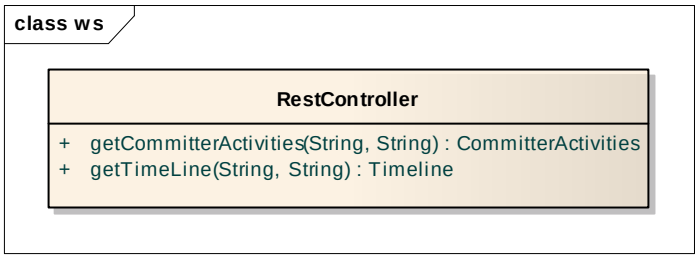


7. Presentation Layer

The presentation tier is divided in two parts: a restful WS and the website and.

a. Restful web service

Giteway exposes a restful WS which provides statistics on GitHub repositories.



Services:

1. The timeline data at : *<app>/restful/repos /<owner>/<repositoryName>/timeline*
2. The committers' activity at : *<app>/restful /repos/<owner>/<repositoryName>/activity*

The data can be returned in two different formats depending on the "Accept" request header value:

1. Json (Accept: application/json)
2. Xml (Accept: application/xml)

```
$ curl.exe -H "Accept: application/xml" http://localhost:8090/giteway/restful/repos/SpringSource/spring-framework/timeline
<?xml version="1.0" encoding="UTF-8"?>
<timeline>
    <repository>
        <name>spring-framework</name>
        <name>SpringSource</name>
    </repository>
    <commitCount>100</commitCount>
    <timelineDays>12.849120381944445</timelineDays>
    <intervalDays>0.6424560300925926</intervalDays>
    <timelineInterval>
        <start>2012-11-25T21:08:06.000+01:00</start>
        <end>2012-11-26T12:33:14.200+01:00</end>
        <commitCount>19</commitCount>
    </timelineInterval>
    <timelineInterval>
        <start>2012-11-26T12:33:14.201+01:00</start>
        <end>2012-11-27T03:58:22.401+01:00</end>
        <commitCount>16</commitCount>
    </timelineInterval>
    <timelineInterval>
        <start>2012-11-27T03:58:22.402+01:00</start>
        <end>2012-11-27T19:23:30.602+01:00</end>
        <commitCount>5</commitCount>
    </timelineInterval>
```

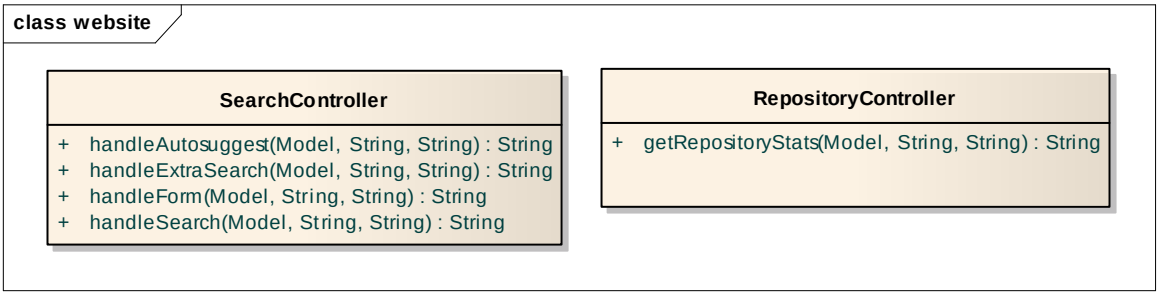
Figure 3 - Timeline xml

```
$ curl.exe -H "Accept: application/json" http://localhost:8090/giteway/restful/repos/SpringSource/spring-framework/timeline
{
  "repository" : {
    "name" : "spring-framework",
    "owner" : "SpringSource",
    "description" : null
  },
  "timelineIntervals" : [ {
    "start" : 1353874086000,
    "end" : 1353929594200,
    "commitCount" : 19
  }, {
    "start" : 1353929594201,
    "end" : 1353985102401,
    "commitCount" : 16
  }, {
    "start" : 1353985102402,
    "end" : 1354040610602,
    "commitCount" : 5
  }, {
    "start" : 1354040610603,
    "end" : 1354096118803,
    "commitCount" : 1
  }, {
    "start" : 1354096118804,
    "end" : 1354151627004,
    "commitCount" : 3
  }, {
    "start" : 1354207135206,
    "end" : 1354262643406,
    "commitCount" : 14
  }, {
```


Figure 4 – Committers' activity json

b. The Website

Two Spring webmvc controllers handles the two views. The search view, and the stats view.



i. The search view

The search view provides an interface for users to search for a repository.

A user can search repositories by keyword or by owner.

Autosuggest is available for both search types.

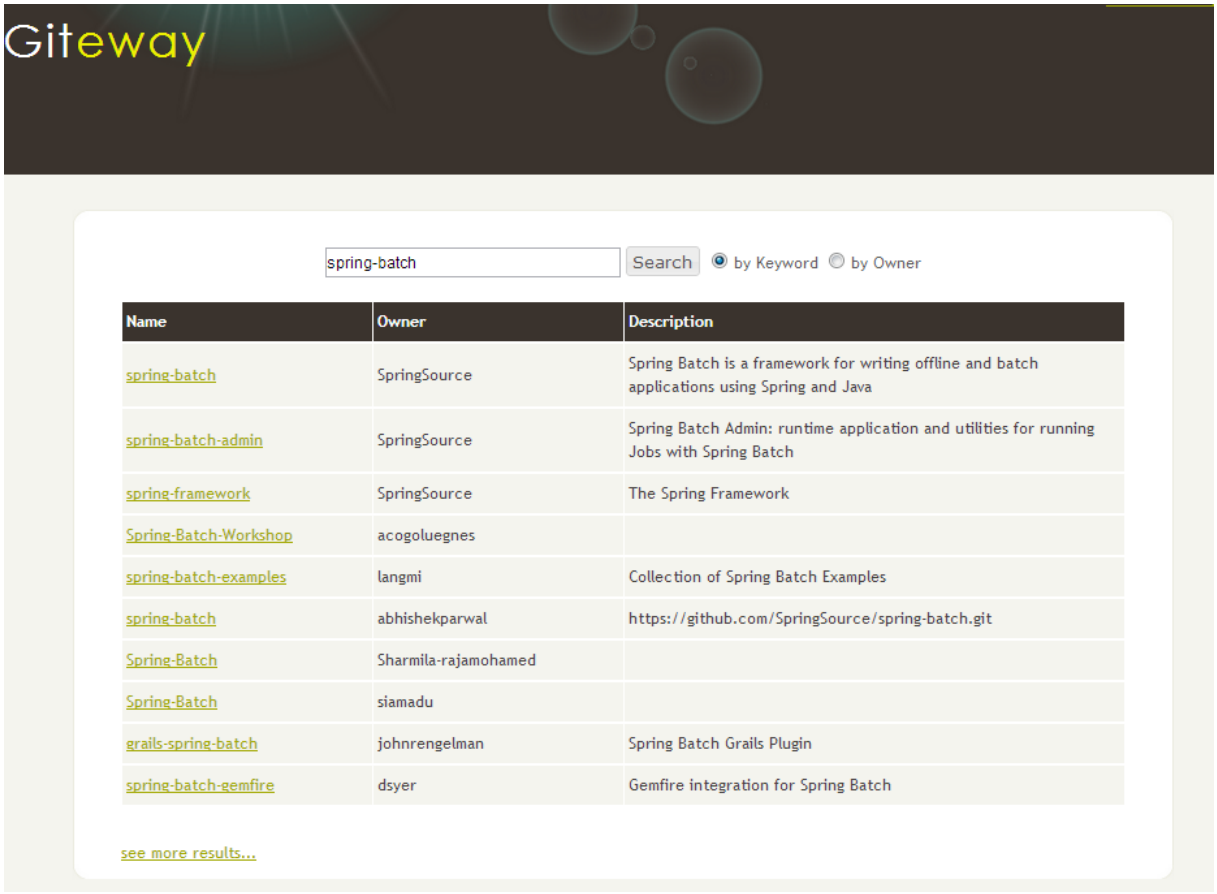


Figure 5 - The search view (by keyword)

ii. The statistics view

The statistics view displays statistics and collaborators.

The statistics loading mechanism is based on Ajax and Giteway restful WS. This approach enables an elegant, flexible and extendable mechanism.

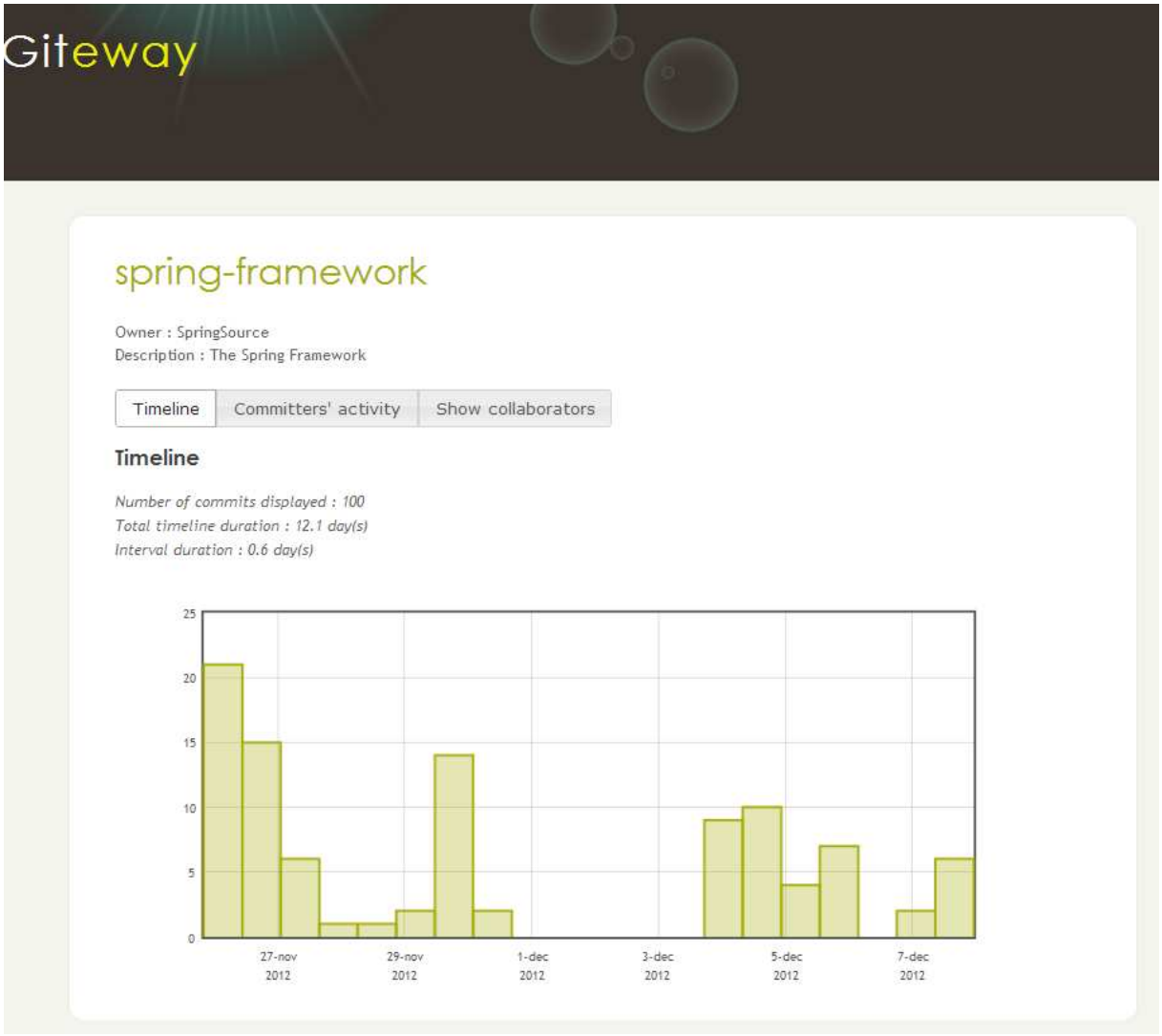


Figure 6 - The statistic view (timeline)

Note: The Website has been designed with a restful approach. Every page is bookmarkable as URLs define which information is displayed. No session or history is held on the server.

8. Caching

Giteway accesses Github data remotely. Because the communication can be slow, a cache has been set up on the DAOs method returns. Spring provides an easy, non-intrusive way of handling caching without any coding. The spring-ehcache approach has been chosen.

9. AOP

AOP handle the logging with an around advice weaved on all methods of the application except the front methods. Spring-AspectJ approach has been used.

10. Testing - Check-Style - Code coverage

Every tier of giteway is unit tested.

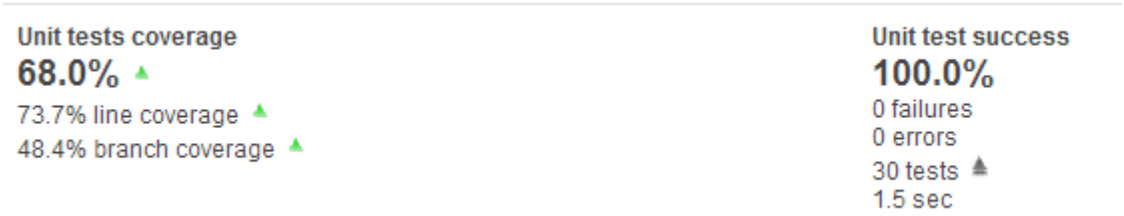


Figure 7 - Unit test analyse

The components have been isolated from each other by mocking the dependencies with mockito. Spring-test has been used for the tooling but no integration tests are performed.

The application code has been analyzed with Sonar.

Figure 8 - CheckStyle