# Architectural Overview

# tech_stack.png

## Front-End Layer

### Technologies

* Adobe Flex v3.4.0
* Swiz v0.6.2
* Apache Tiles v2.2.1
* Spring MVC v3.0
* SWFObject v2.2
* JQuery v1.3.6

### Overview

The front-end is driven by SpringMVC controllers that delegate to an Apache Tiles view resolver. The tile responsible for loading the flash application leverages swfobject. The JSP page passes the host, port and context root to the flash application so it will know where to make its AMF calls. The flash application is created with Adobe Flex and leverages Swiz, which is a dependency injection framework for AS3. We have standardized on Jquery as our javascript library of choice.

## Back-end Layer

### Technologies

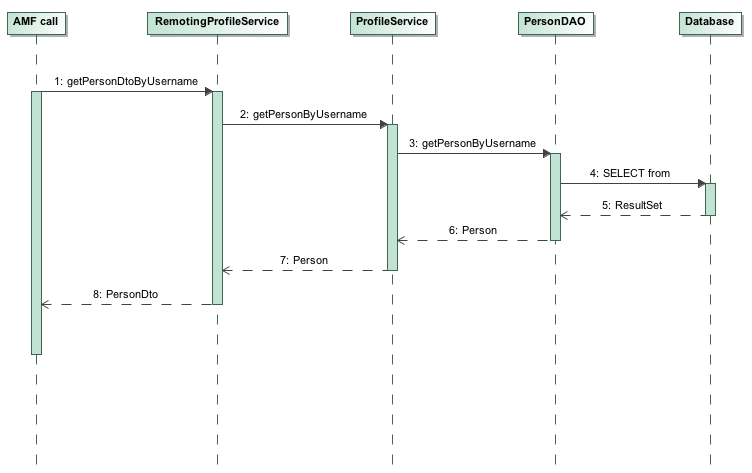
* Spring v3.0.0
* Spring Security v2.0.5
* Spring Flex v1.0.0
* Hibernate EJB v3.4.0
* Hibernate Search 3.1.1
* Hibernate Validator v3.4.0
* OSCache
* EHCache
* Dozer v5.1
* Jasypt v1.5
* Tomcat v6.0.20
* URLRewrite v3.1.0

### Overview

The back-end is a typical best practice Spring / JPA / Hibernate implementation. The interesting pieces are search and conversion mapping.

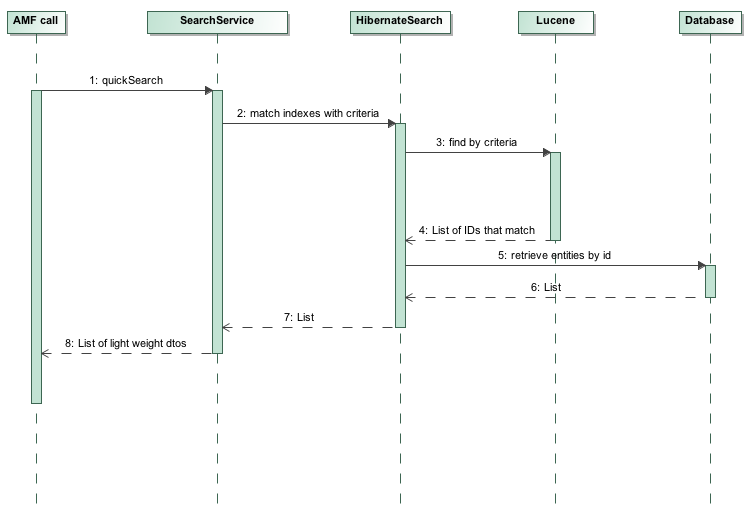
The web application is loaded using spring’s listener class and reading in applicationContext.xml and it’s how the dependency injection for the domain layer occurs. We also have a spring dispatcher servlet that starts its own context by reading in dispatcher-servlet.xml. It defines everything related to the web tier.

Apart from the Spring specific information in web.xml, we have also defined spring security, url rewriting, amf message broker and caching with OSCache there. Spring Security enforces both url based and method level security. It also secures the amf channel. More details on our usage of Spring Security can be found in security.xml. Url rewriting is a filter we use to make” SEO happy” urls. Our mappings can be found in urlrewrite.xml. The AMF message broker servlet is our gateway to remoting. OSCache takes care of caching our static content in memory.



Figure

Figure 1 shows a generic call to our call stack. All AMF calls are done to services that are capable of managing data for remote calls such as AMF and Web Services. In the case of Figure 1, the caller is trying to retrieve a PersonDto (light-weight transfer object) by calling the method getPersonDtoByUsername and passing a username as a parameter. The remoting service immediately forwards the request to ProfileService, which is the heart of all profile related data. As this call doesn’t require anything but a database call, ProfileService asks the PersonDao to retrieve it a Person entity from the database matching the specific username. When RemotingProfileService gets a hold of the entity, it converts it to a PersonDto object using the Dozer framework in order to “flatten” the entity object and returns it to the caller.



Figure

Figure 2 shows a typical call that involves searching non-structured data using Hibernate Search with Lucene under the hood. In this example, the called is trying to retrieve data from our quickSearch method that is the simplest search method we have. The SearchService calls hibernate search, which will in turn query Lucene if any of the keywords match one of our people in the system. If it does, Lucene will return ids for these people along with a relevancy for each one. Hibernate Search then goes to the database and queries the people on their primary keys. Once we have a list of Person entities, the SearchService converts this list into light-weight PersonLite objects, using Dozer, that only contain the most important information that we need to display as part of our search results.

## Persistence Layer

We are using PostgreSQL v8.4 for this application.