CrossSearch - Design document - Java Architect (with NoSql and SOLR)

By Tiago Vinicius

Requirements

The main goal is to build a system where users, with distinct roles, are capable of publishing and searching news articles.

Therefore, it was built an application called CrossSearch to address these requirements.

Functional requirements

Publisher role:

- Publish and manage a list of articles

User role:

- Search articles

Non-Functional requirements

- Must be coded in Java and related frameworks
- MySQL as relational database to store articles and users credentials
- Use Security authentication/authorization framework
- Apache Solr as enterprise search engine

Data model

CrossSearch model consists in three entities:

Users: stores user name, password, and enabled, that represents if an account is valid.

User_roles: contains a user name, a role and refers a user.

Articles: contains an id, which identifies uniquely an article, title, content and author. Category and tags could be searchable fields in an indexed document later in Solr. Image field represents location path where nginx will serve a image. All those fields will consist a document that will be indexed in Solr via DIH.

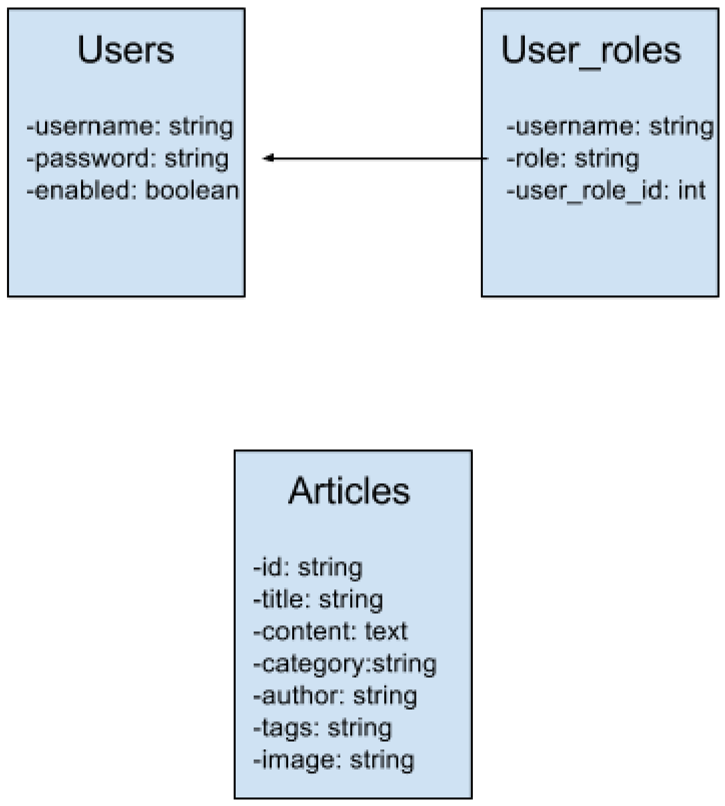

Figure 1: CrossSearch Data model

Architecture

This is a high level description of solution architecture. Users begin interaction with CrossSearch portal. Publisher users must be logged to create articles. User credentials and articles data are stored in MySQL.

After that, a Solr DIH routine is started and articles are indexed in Solr collection. Then regular users can perform a search. Articles snippet results will appear right below search box. Click in snippet will redirect to full article page.

For snippets and full articles, images served by Nginx are presented to users.

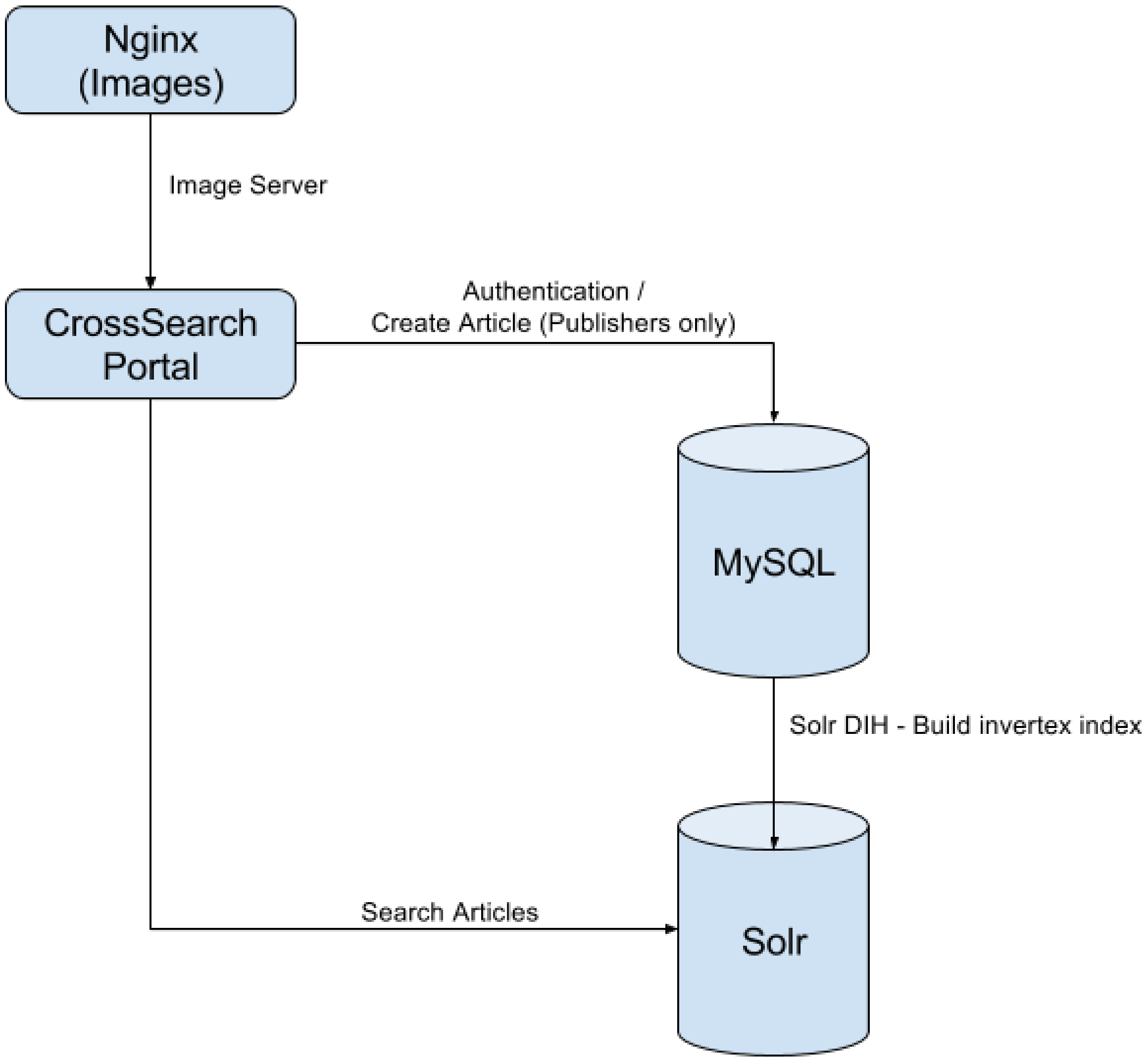
Nginx (Images)
Image Server
CrossSearch Portal
Authentication / Create Article (Publishers only)
MySQL
Solr DIH - Build invertex index
Search Articles
Solr

Figure 2: High level architecture

Flow diagrams

- Blue: Login use case flow
- Yellow: Search use case flow
- Green: Create article use flow

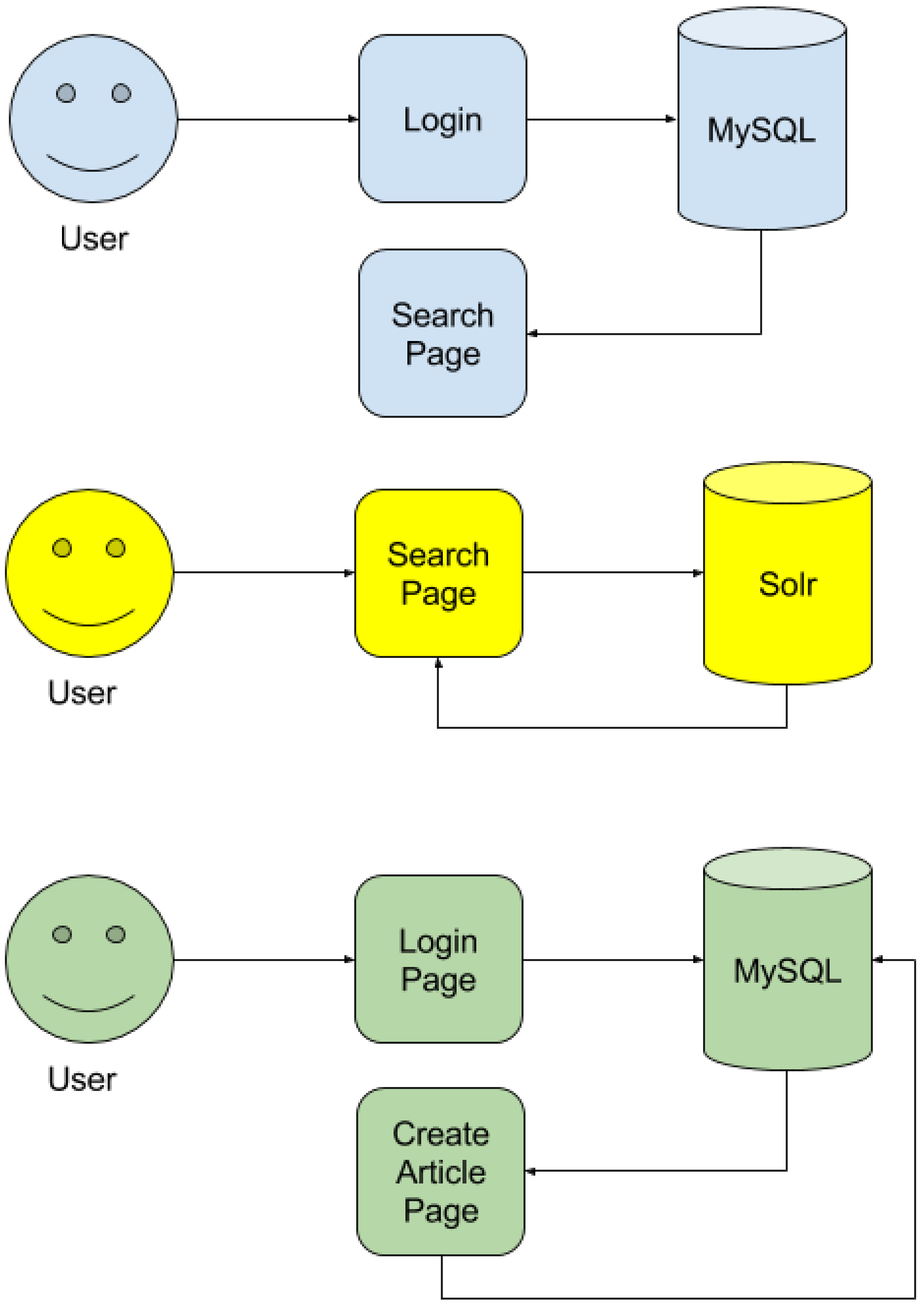

Figure 3: Flow diagrams

Technical details

Following components were adopted in solution with some explanation:

Maven: most used dependency manager. I could use Gradle here, and also Bower to manage AngularJS and Bootstrap dependencies but for simplicity, I kept Maven.

Spring boot velocity starter: easily bootstrap a Java Spring-MVC based application with Apache Velocity as template renderer

MySQL: Relational database choice. Postgres would do the same Job.

JDBI: Simple lightweight relational database client. As just Save Article and User retrieval from database use cases were implemented, I did not see advantages in using JPA.

Solr: Enterprise search engine. I configured a brand new Solr 6 version.

Spring Security: As Spring-MVC was bootstrapped in project, Spring Security was a natural choice of authentication/authorization framework

Twitter Bootstrap: like Spring boot, I have just wanted an easy way to build a frontend with css and js components.

AngularJS: Allow showing search result in Single Page Application pattern.

Nginx: could be setup to proxy the main application, but for now, just serve images stored in disk.