Names: Wu, Peiqi; Zhou, Yuzhen

Date: 5/19/2022.

Course: Cloud Computing

**Course Project**

**Team Members:** Wu, Peiqi; Zhou, Yuzhen

**Answers to Question**

[**Understanding Eureka Peer to Peer Communication**](https://github.com/Netflix/eureka/wiki/Understanding-Eureka-Peer-to-Peer-Communication)

**Total in points (100 points total): \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Professor’s Comments:**

0 GitHub Link

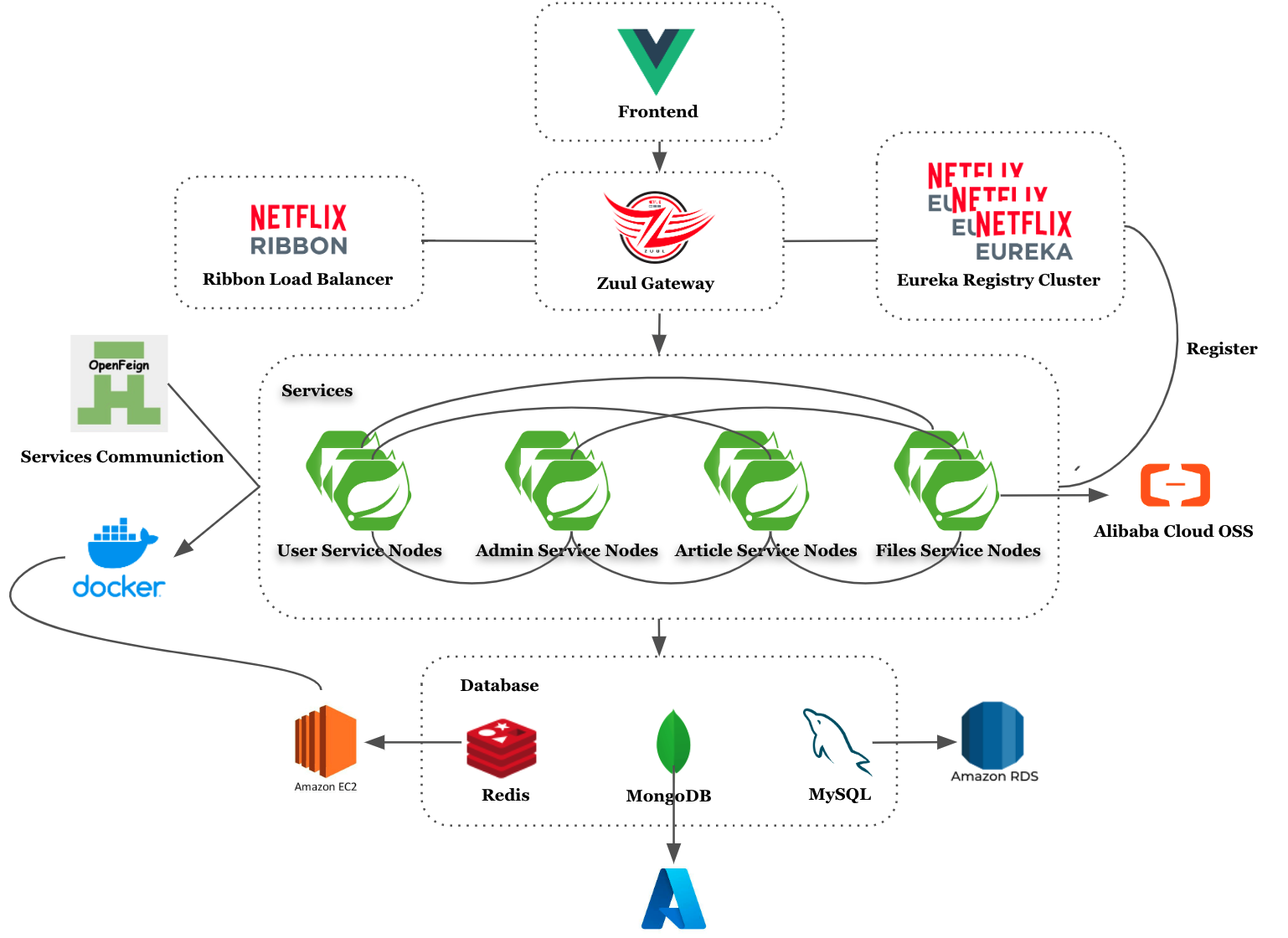
Front End: <https://github.com/ElioZhou/knowledge-planet>

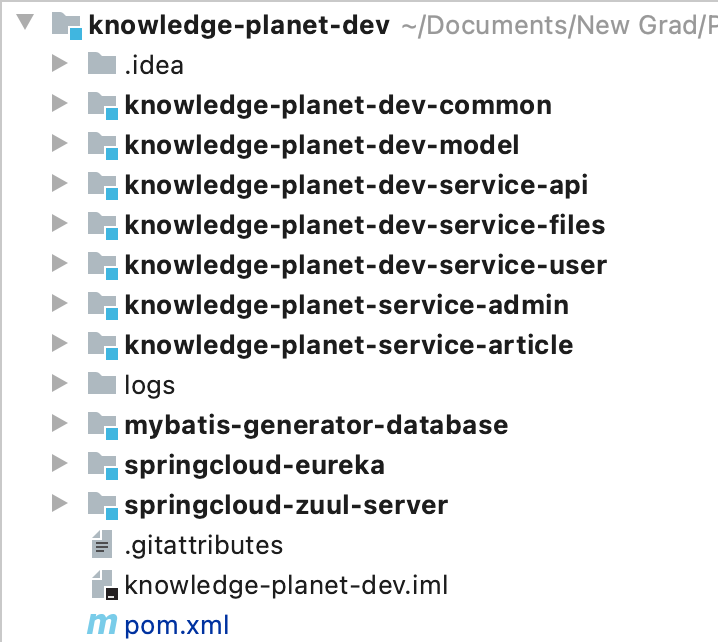
Back End: <https://github.com/ElioZhou/knowledge-planet-dev>

1 Introduction

We are building a blog web application called “Knowledge Planet” using a P2P framework. All the services can interact with each other in a Peer-to-Peer manner. Users are allowed to post blogs, follow other users, be followed, and update personal info. Admins are allowed to manage users, friend links, and review articles. The portal will show all the articles published, users can select different categories to browse the articles.

2 Design Diagram





The services were developed within Spring Framework. It’s a Java Maven Project.

We implemented the P2P network leveraging Netflix OSS Eureka as a service finder. All the services would register themselves into the Eureka Registry Cluster. If one service needs to send a request to another service in the network, the service will query the Eureka Registry Cluster first to get the information of the service it’s requesting and then do peer-to-peer communication. We also integrated Netflix Ribbon as the client-side load balancer.

Feign is a declarative HTTP client developed by Netflix. Feign aims at simplifying HTTP API clients. Simply put, the developer needs only to declare and annotate an interface while the actual implementation is provisioned at runtime. Feign is used for service communication.

We used Netflix Zuul as a network gateway. All the requests from clients would query Zuul Gateway first to get the location of the target service. It would query Eureka servers to get all the service locations and work as a load balancer, distributing the request in a rolling manner.

All the important data are stored in the MySQL database on AWS RDS. We used Redis as a cache for the MySQL database to improve the query efficiency and QPS. Other less important but frequently requested data are stored on MongoDB which is located on Azure cause MongoDB has the ability to handle a lot of parallel requests.

We employ Alibaba Cloud OSS as the file storage. It has the ability to auto-review the images of files uploaded.

All the services were packaged into .jar files and wrapped into docker images. We run containers of the images on AWS EC2 Servers.

3 Service Discovery

A service registry is a phone book for services. Each service registers itself with the service registry and tells the registry where it lives (*host, port, node name*) and perhaps other service-specific metadata - things that other services can use to make informed decisions about it.

There are several popular options for service registries. Netflix built and then open-sourced their own service registry, Eureka.

From the the Spring Cloud project page: “Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boilerplate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer’s own laptop, bare metal data centers, and managed platforms such as Cloud Foundry.”

It’s easy to build eureka servers under spring cloud framework.

1. Configuration

*############################################################*

*#*

*# eureka configuration*

*#*

*############################################################*

**eureka**:

**instance**:

**hostname**: eureka-cluster-${**server.port**}

**other-node-port2**: ${**p2**:7002}

**other-node-port3**: ${**p3**:7003}

**client**:

**service-url**:

**defaultZone**: http://eureka-cluster-${**eureka.other-node-port2**}:${**eureka.other-node-port2**}/eureka/,

http://eureka-cluster-${eureka.other-node-port3}:${eureka.other-node-port3}/eureka/

**server**:

**enable-self-preservation**: **true**

**eviction-interval-timer-in-ms**: 5000

1. Enable Eureka Server

Use annotation @EnableEurekaServer on the main class of Eureka Server project.

1. Register services

Use annotation @EnableEurekaClient on the main class of Eureka client project(services). Eureka Client will send heartbeats periodically to eureka cluster to confirm its existence.

*############################################################*

*#*

*# eureka client config*

*#*

*############################################################*

**eureka**:

**client**:

**register-with-eureka**: **true**

**fetch-registry**: **true**

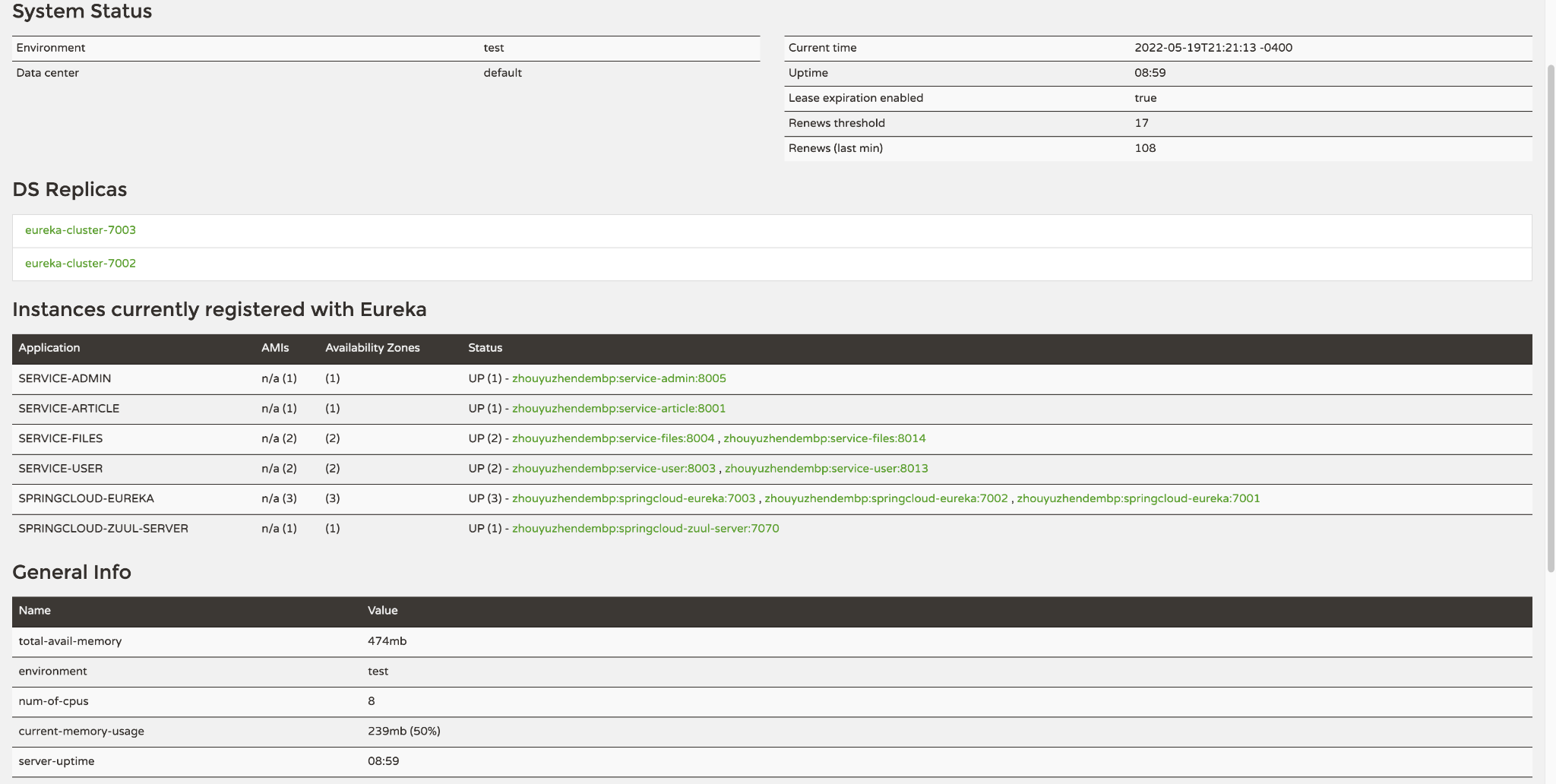
**service-url**:

**defaultZone**: http://eureka-cluster-7001:7001/eureka/,http://eureka-cluster-7002:7002/eureka/,http://eureka-cluster-7003:7003/eureka/

**instance**:

**lease-renewal-interval-in-seconds**: 3

**lease-expiration-duration-in-seconds**: 5



4 User Service

├── pom.xml

├── src

│ ├── main

│ │ ├── java

│ │ │ └── art

│ │ │ └── knowledgeplanet

│ │ │ └── user

│ │ │ ├── Application.java

│ │ │ ├── controller

│ │ │ │ ├── AppUserMngController.java

│ │ │ │ ├── HelloController.java

│ │ │ │ ├── MyFansController.java

│ │ │ │ ├── PassportController.java

│ │ │ │ └── UserController.java

│ │ │ ├── mapper

│ │ │ │ ├── AppUserMapper.java

│ │ │ │ ├── AppUserMapperCustom.java

│ │ │ │ └── FansMapper.java

│ │ │ └── service

│ │ │ ├── AppUserMngService.java

│ │ │ ├── MyFanService.java

│ │ │ ├── UserService.java

│ │ │ └── impl

│ │ │ ├── AppUserMngServiceImpl.java

│ │ │ ├── MyFanServiceImpl.java

│ │ │ └── UserServiceImpl.java

│ │ └── resources

│ │ ├── application-dev.yml

│ │ ├── application-prod.yml

│ │ ├── application.yml

│ │ ├── logback-spring.xml

│ │ └── mapper

│ │ ├── AppUserMapper.xml

│ │ ├── AppUserMapperCustom.xml

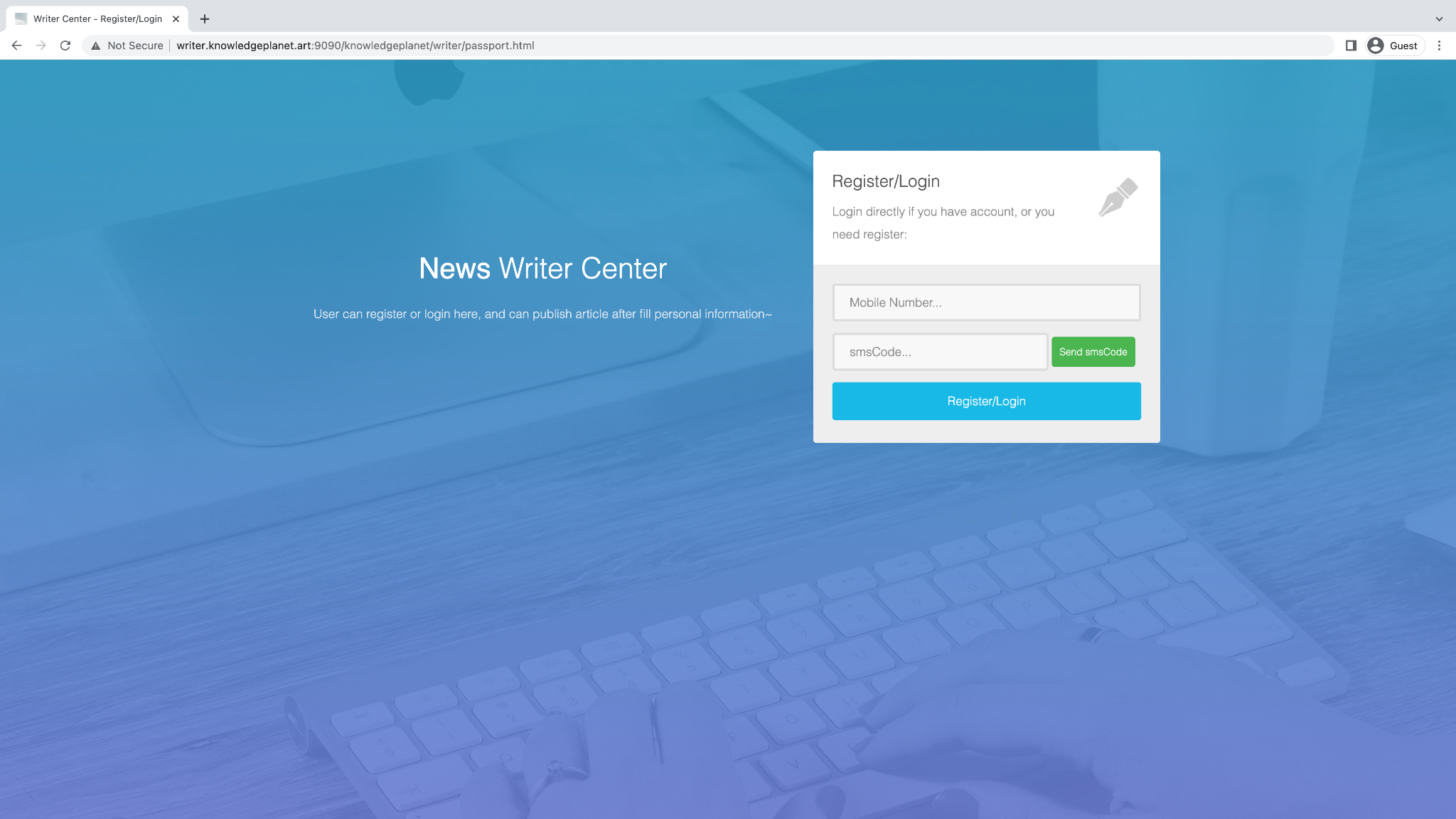
│ │ └── FansMapper.xml

│ └── test

│ └── java

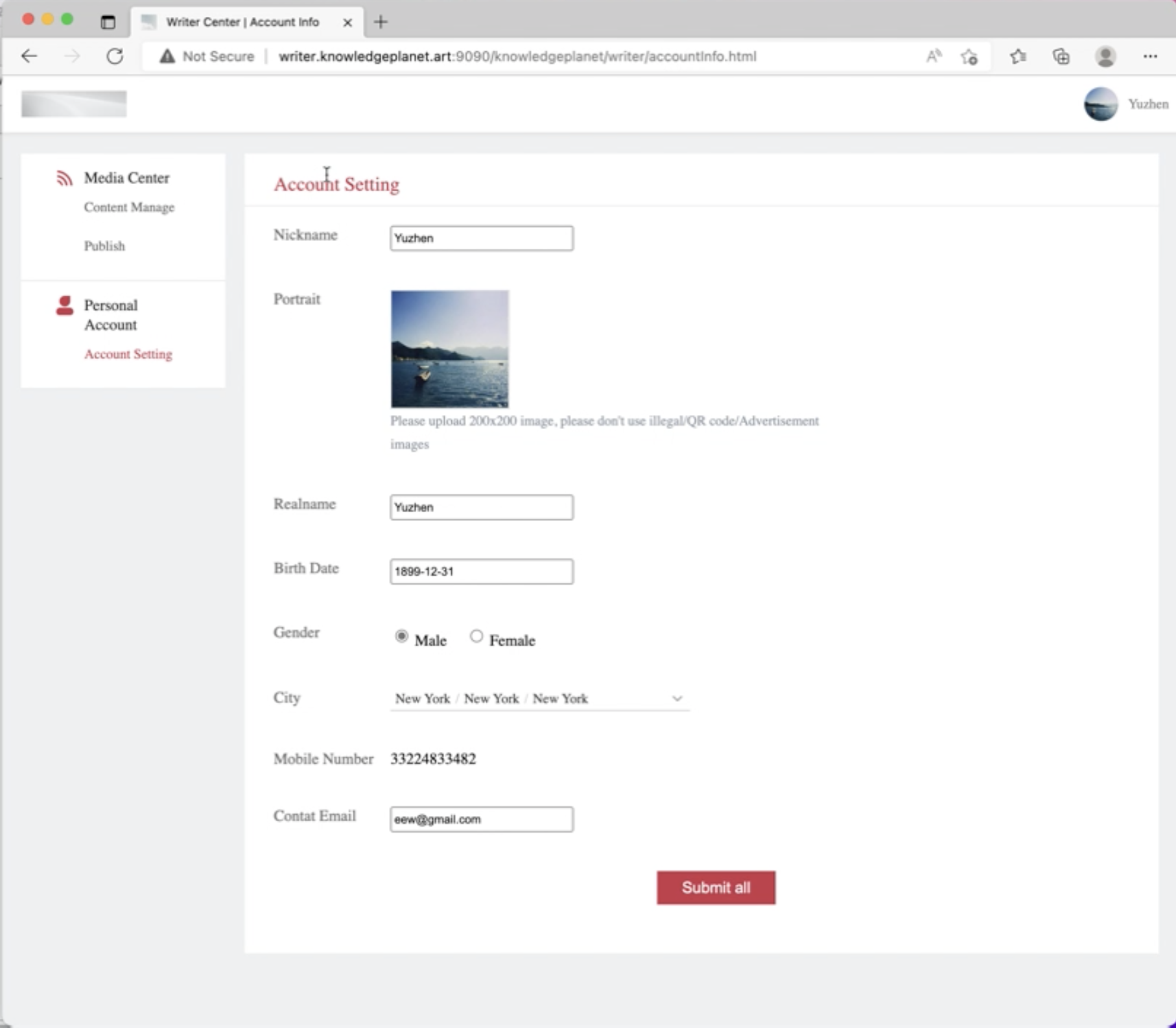
1. Login Page

We integrated Alibaba Cloud SMS service to send sms code. The generated sms code would be stored on Redis. We also store the IP address from the request. Set an interceptor to check if the user from this specific IP address sent multiple requests within 60 seconds.

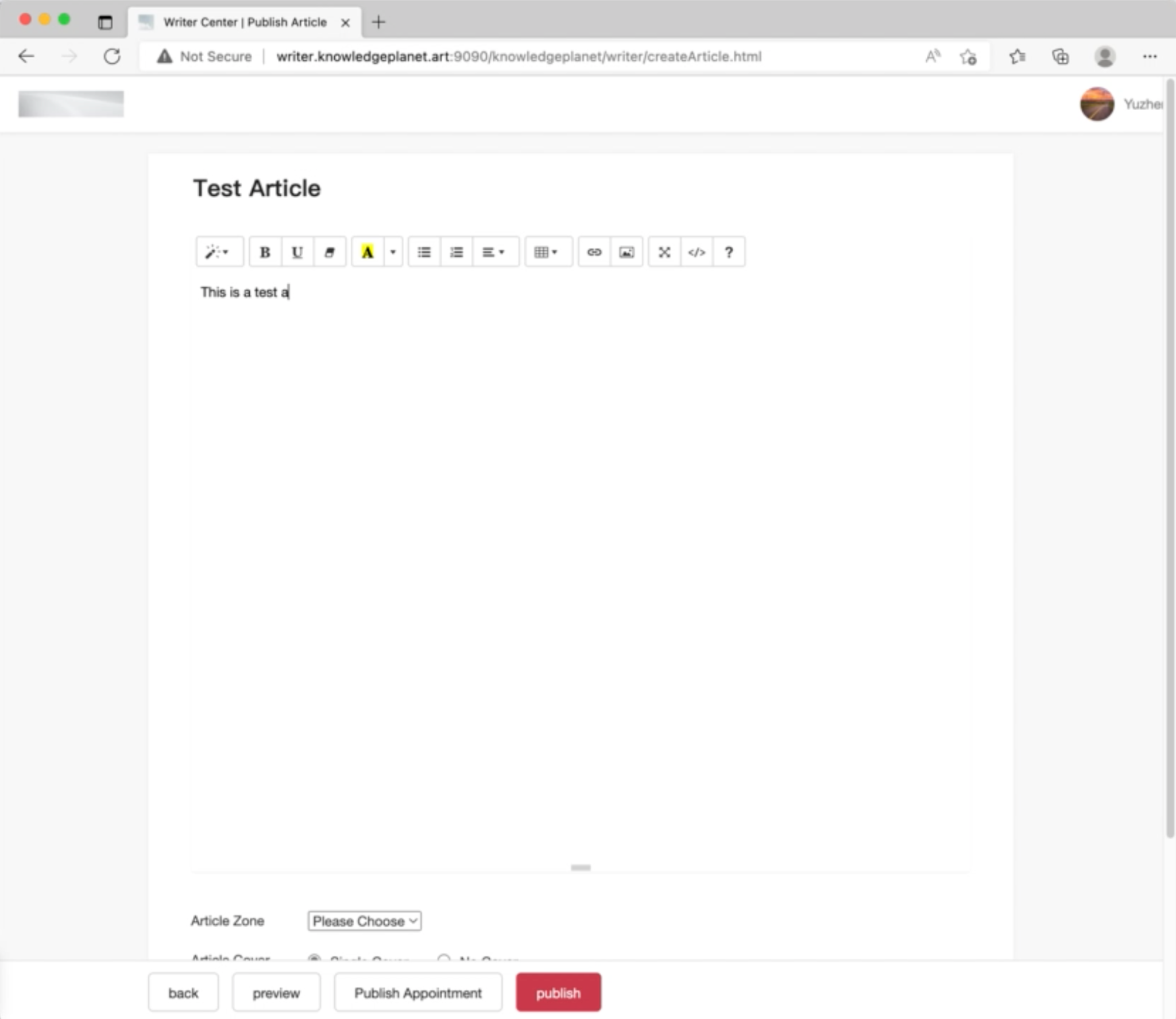


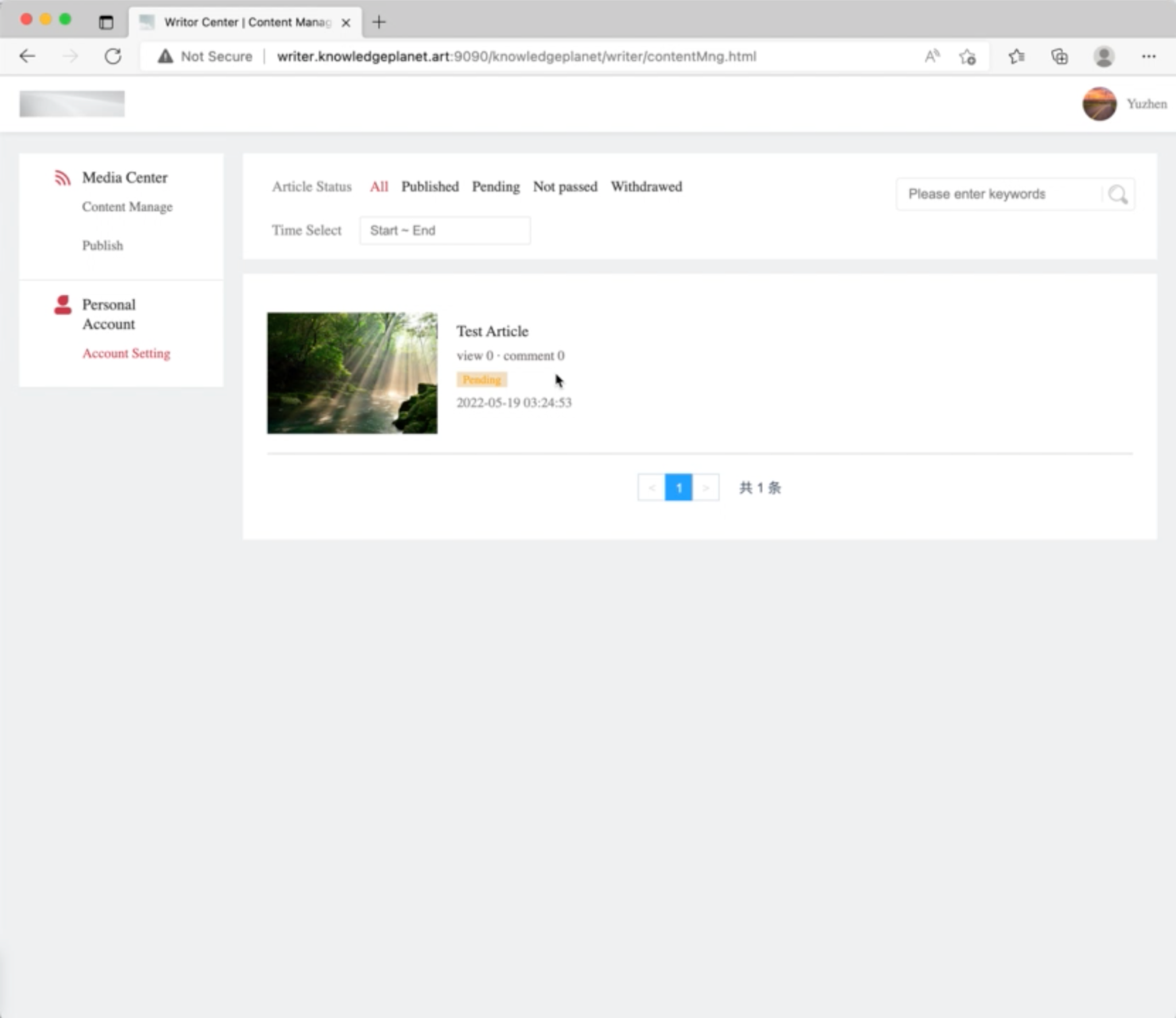
1. Update Info

We use Mybatis to manipulate the MySQL database which is implemented using AWS Relational Database Services.



1. Publish an article





5 Admin Service

.

├── pom.xml

├── src

│ ├── main

│ │ ├── java

│ │ │ └── art

│ │ │ └── knowledgeplanet

│ │ │ └── admin

│ │ │ ├── Application.java

│ │ │ ├── controller

│ │ │ │ ├── AdminMngController.java

│ │ │ │ ├── CategoryMngController.java

│ │ │ │ ├── FriendLinkController.java

│ │ │ │ ├── HelloController.java

│ │ │ │ └── PWDTest.java

│ │ │ ├── mapper

│ │ │ │ ├── AdminUserMapper.java

│ │ │ │ └── CategoryMapper.java

│ │ │ ├── repository

│ │ │ │ └── FriendLinkRepository.java

│ │ │ └── service

│ │ │ ├── AdminUserService.java

│ │ │ ├── CategoryService.java

│ │ │ ├── FriendLinkService.java

│ │ │ └── impl

│ │ │ ├── AdminUserServiceImpl.java

│ │ │ ├── CategoryServiceImpl.java

│ │ │ └── FriendLinkServiceImpl.java

│ │ └── resources

│ │ ├── application-dev.yml

│ │ ├── application-prod.yml

│ │ ├── application.yml

│ │ ├── logback-spring.xml

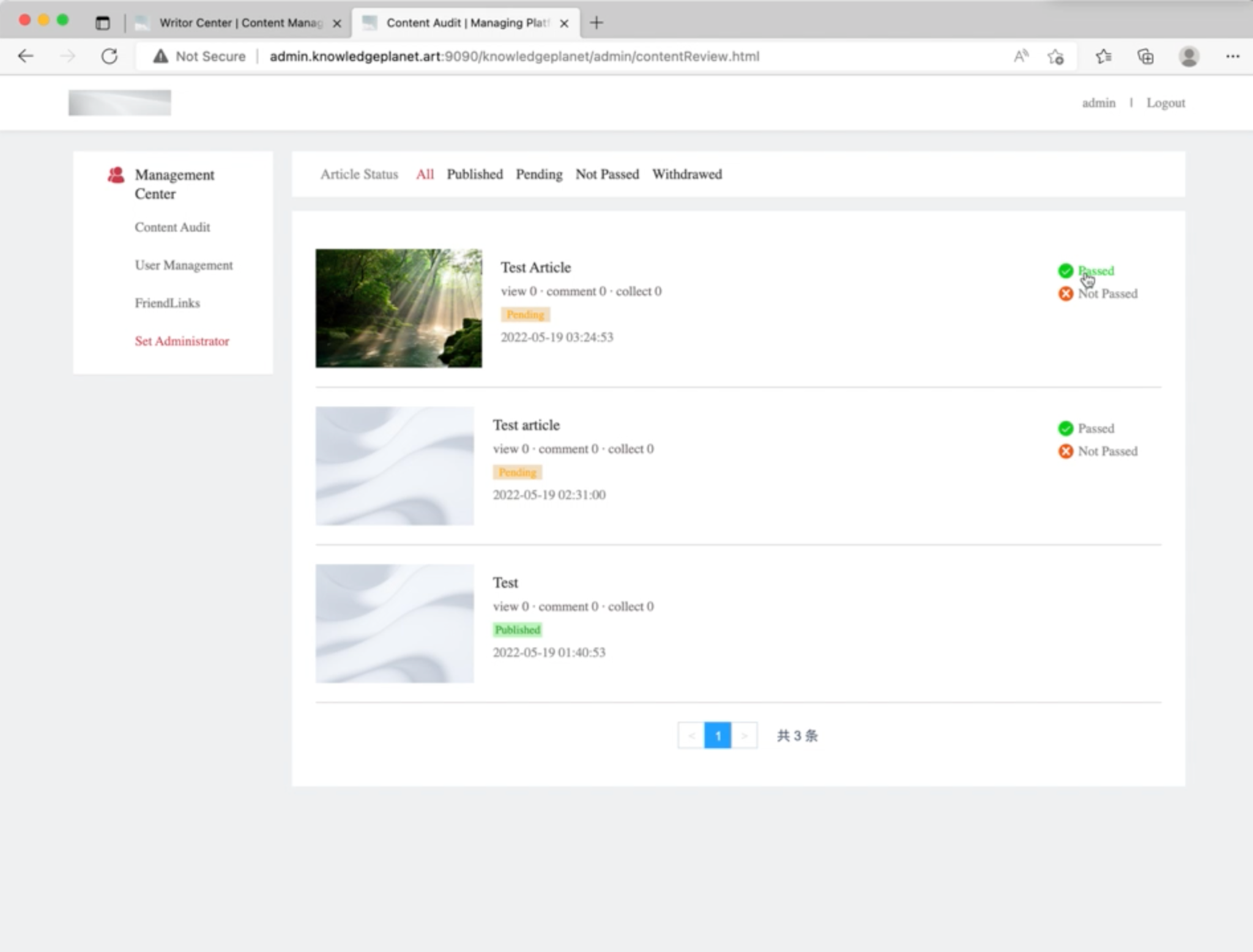
│ │ └── mapper

│ │ └── AdminUserMapper.xml

│ └── test

│ └── java

The admin service is in charge of interacting with many databases, such as user databases and admin databases. We use Mybatis to simplify the CRUD operation.



Admins do the page reviews, basically, modifying a column in the database showing the status of the article.

Admins also manage the friend links which are demonstrated on the portal page.

6 Files Service

.

├── pom.xml

├── src

│ ├── main

│ │ ├── java

│ │ │ └── art

│ │ │ └── knowledgeplanet

│ │ │ └── files

│ │ │ ├── Application.java

│ │ │ ├── controller

│ │ │ │ ├── FileUploaderController.java

│ │ │ │ └── HelloController.java

│ │ │ ├── resource

│ │ │ │ └── FileResource.java

│ │ │ └── service

│ │ │ ├── UploaderService.java

│ │ │ └── impl

│ │ │ └── UploaderServiceImpl.java

│ │ └── resources

│ │ ├── application-dev.yml

│ │ ├── application-prod.yml

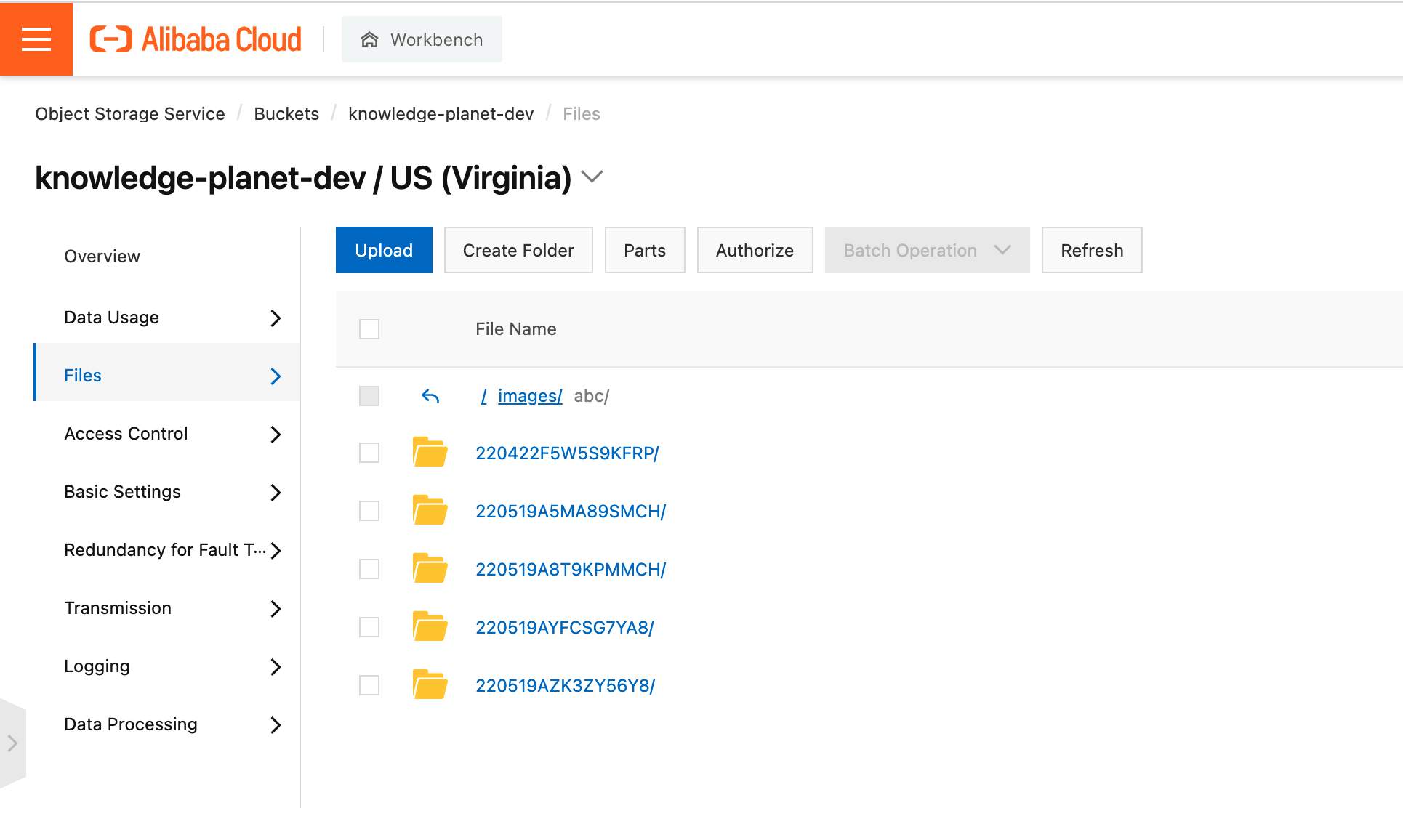
│ │ ├── application.yml

│ │ ├── file-dev.properties

│ │ ├── file-prod.properties

│ │ └── logback-spring.xml

We use Alibaba Cloud OSS to store the files.



Alibaba OSS also has the ability to do file reviews such as finding images related to sex or violence.

7 Article Service

.

├── pom.xml

├── src

│ ├── main

│ │ ├── java

│ │ │ └── art

│ │ │ └── knowledgeplanet

│ │ │ └── article

│ │ │ ├── Application.java

│ │ │ ├── controller

│ │ │ │ ├── ArticleController.java

│ │ │ │ ├── ArticlePortalController.java

│ │ │ │ ├── CommentController.java

│ │ │ │ ├── FreemarkerController.java

│ │ │ │ └── HelloController.java

│ │ │ ├── mapper

│ │ │ │ ├── ArticleMapper.java

│ │ │ │ ├── ArticleMapperCustom.java

│ │ │ │ ├── CommentsMapper.java

│ │ │ │ └── CommentsMapperCustom.java

│ │ │ ├── service

│ │ │ │ ├── ArticlePortalService.java

│ │ │ │ ├── ArticleService.java

│ │ │ │ ├── CommentPortalService.java

│ │ │ │ └── impl

│ │ │ │ ├── ArticlePortalServiceImpl.java

│ │ │ │ ├── ArticleServiceImpl.java

│ │ │ │ └── CommentPortalServiceImpl.java

│ │ │ └── task

│ │ │ └── TaskPublishArticles.java

│ │ └── resources

│ │ ├── application-dev.yml

│ │ ├── application-prod.yml

│ │ ├── application.yml

│ │ ├── mapper

│ │ │ ├── ArticleMapper.xml

│ │ │ ├── ArticleMapperCustom.xml

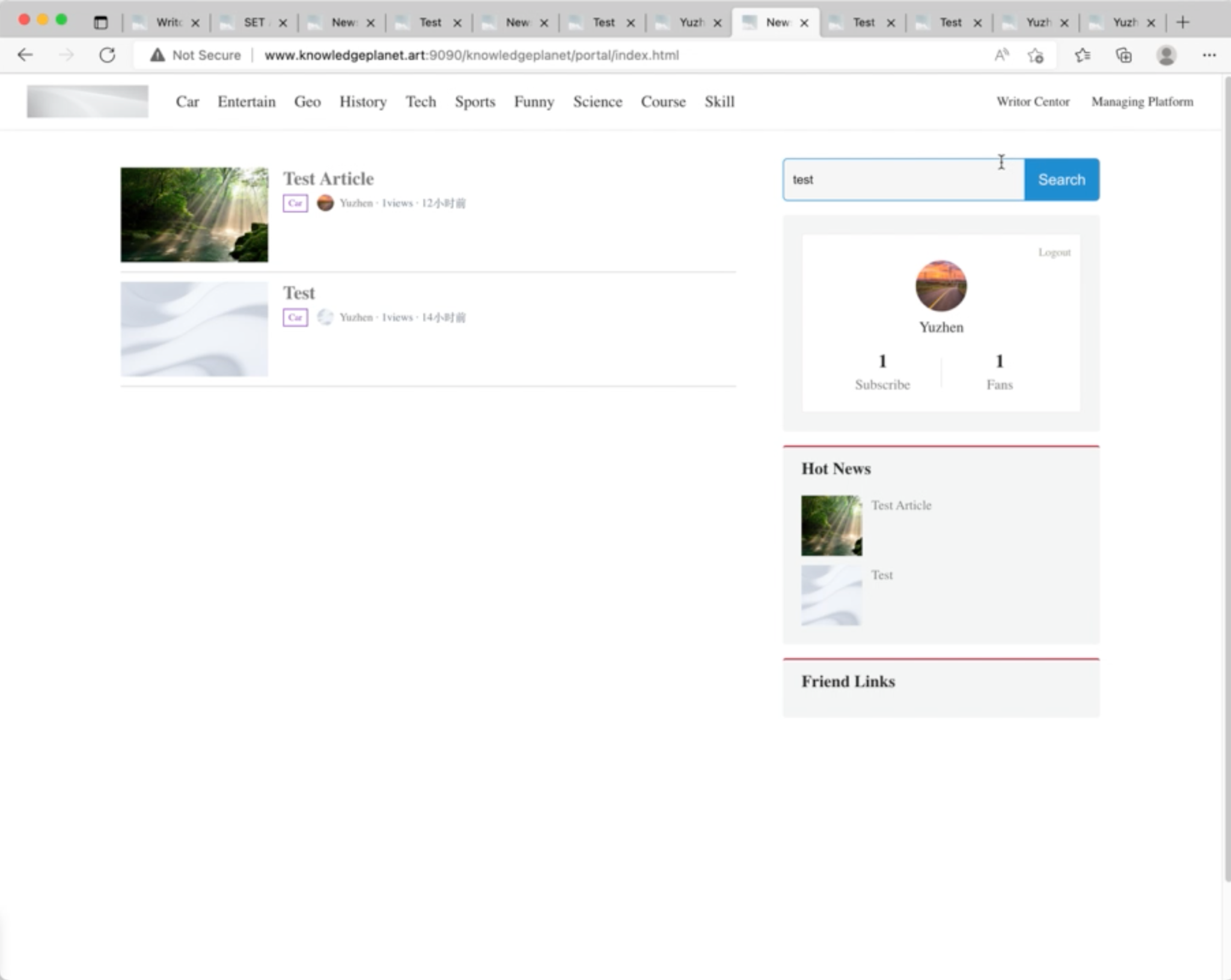
│ │ │ ├── CommentsMapper.xml

│ │ │ └── CommentsMapperCustom.xml

│ │ └── templates

│ │ ├── detail.ftl

│ │ └── stu.ftl



We still use spring framework to implement. We use controllers to organize all the services and do the request routing.

The article service is responsible for the business related to the articles. Such as page views, page generations.