

**台州学院**

《Java高级编程》

实验报告

**实验项目:** 实验7 web开发技术应用项目实践

**学号姓名:** 2X计奇2X00000000无名氏

**专 业：**计算机科学与技术

**班 级：**23计奇班

**实验时间：**第15次课 三/四/五

电子与信息工程学院

一、实验目的及要求：

1） 综合应用web开发技术，设计并实现一个web应用项目;

2） 用文字描述画出自己项目的项目名称和主要功能；

3） 画出系统功能流程图；

4） 实现

二、实验内容：

1.用文字描述画出自己综合设计项目的：

（1）项目名称：dashstar blog

（2）主要功能：

后端：

1）用户的登录和注册：可以注册或登录账号；

2）文章管理：可以对文章进行添加、修改、删除功能，并查看所发布的文章；

3）评论管理：可以对任意文章进行评论；

前端：

1. 显示登录的用户；
2. 文章列表；
3. 文章评论区；

（3）附加功能（选做）

1） 可删除评论；

2） 隐藏用户的密码哈希；

3） 文章列表分页；

4） 分类系统；

5） 将源代码上传至 GitHub

2.画出系统功能流程图，描述自己综合设计项目的主要功能；



三、实验代码及运行结果（包括可能有的截图）：

实验主要代码：

**model/User.java**

package dev.e23.dashstar.model;

import jakarta.persistence.\*;

import lombok.Data;

import java.io.Serializable;

@Data // 表示这个类需要被 Lombok 处理，Lombok 会自动生成 getter、setter、toString、equals、hashCode 等方法

@Entity // 表示这是一个实体类，会被 Hibernate 管理

@Table(name = "users") // 表示这个实体类对应的数据库表名是 users

public class User implements Serializable {

@Id // 表示这个字段是主键

@GeneratedValue(strategy = GenerationType.IDENTITY) // 表示主键生成策略是自增

private Integer id;

@Column(name = "username", unique = true) // 表示这个字段对应的数据库表中的列名是 username，并且这个字段是唯一的

private String username;

@Column(name = "nickname") // 表示这个字段对应的数据库表中的列名是 nickname

private String nickname;

@Column(name = "password") // 表示这个字段对应的数据库表中的列名是 password

private String password;

@Column(name = "role") // 表示这个字段对应的数据库表中的列名是 role

private String role = "user"; // 表示这个字段的默认值是 user

}

**model/Article.java**

package dev.e23.dashstar.model;

import com.fasterxml.jackson.annotation.JsonProperty;

import jakarta.persistence.\*;

import lombok.Data;

import java.io.Serializable;

@Data

@Entity

@Table(name = "articles") // 表示这个实体类对应的数据库表名是 articles

public class Article implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column(name = "title")

private String title;

@Column(name = "content")

private String content;

@ManyToOne(fetch = FetchType.EAGER) // 表示这个字段是多对一的关系，即一篇文章只能属于一个用户

@JoinColumn(name = "author\_id") // 表示这个字段对应的数据库表中的列名是 author\_id

private User author; // 表示这个字段对应的实体类是 User，实际上存储进数据库的是 User 的 id

@Column(name = "author\_id", insertable = false, updatable = false)

@JsonProperty("author\_id") // 表示这个字段在 JSON 序列化时，字段名是 author\_id（不然就会是 authorId）

private Integer authorId;

@Column(name = "created\_at")

@JsonProperty("created\_at") // 表示这个字段在 JSON 序列化时，字段名是 created\_at（不然就会是 createdAt）

private Long createdAt;

}

**model/Comment.java**

package dev.e23.dashstar.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonProperty;

import jakarta.persistence.\*;

import lombok.Data;

import java.io.Serializable;

@Data

@Entity

@Table(name = "comments") // 表示这个实体类对应的数据库表名是 comments

public class Comment implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column(name = "content")

private String content;

@ManyToOne // 表示这个字段是多对一的关系，即一个评论只能属于一篇文章

@JoinColumn(name = "article\_id") // 表示这个字段对应的数据库表中的列名是 article\_id

@JsonIgnore // 查找的评论并不需要文章的详细信息，所以我们可以忽略

private Article article; // 表示这个字段对应的实体类是 Article，实际上存储进数据库的是 Article 的 id

@Column(name = "article\_id", insertable = false, updatable = false)

@JsonProperty("article\_id")

private Integer articleId;

@ManyToOne(fetch = FetchType.EAGER) // 表示这个字段是多对一的关系，即一个评论只能属于一个用户

@JoinColumn(name = "user\_id") // 表示这个字段对应的数据库表中的列名是 user\_id

private User user; // 表示这个字段对应的实体类是 User，实际上存储进数据库的是 User 的 id

@Column(name = "user\_id", insertable = false, updatable = false)

@JsonProperty("user\_id") // 表示这个字段在 JSON 序列化时，字段名是 user\_id（不然就会是 userId）

private Integer userId;

@Column(name = "created\_at")

@JsonProperty("created\_at") // 表示这个字段在 JSON 序列化时，字段名是 created\_at（不然就会是 createdAt）

private Long createdAt;

}

**repository/UserRepository.java**

package dev.e23.dashstar.repository;

import dev.e23.dashstar.model.User;

import dev.e23.dashstar.util.HibernateUtil;

import jakarta.enterprise.context.ApplicationScoped;

import jakarta.persistence.EntityManager;

import jakarta.persistence.PersistenceException;

import java.util.List;

@ApplicationScoped // 声明这是一个应用范围的 Bean，需要的地方可以使用 @Inject 注入

public class UserRepository {

public List<User> findAll() throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

List<User> users = null;

try {

em.getTransaction().begin(); // 开始事务

users = em

.createQuery("SELECT u FROM User u", User.class)

.getResultList(); // 查询所有用户

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("", e);

} finally {

em.close(); // 关闭 EntityManager

}

return users;

}

public User findByID(Integer id) throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

User user = null;

try {

em.getTransaction().begin(); // 开始事务

user = em

.createQuery("SELECT u FROM User u WHERE u.id = :id", User.class)

.setParameter("id", id)

.getSingleResult(); // 查询指定 ID 的用户

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("user\_not\_found", e);

} finally {

em.close(); // 关闭 EntityManager

}

return user;

}

public User findByUsername(String username) throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

User user = null;

try {

em.getTransaction().begin(); // 开始事务

user = em

.createQuery("SELECT u FROM User u WHERE u.username = :username", User.class)

.setParameter("username", username)

.getSingleResult(); // 查询指定 username 的用户

em.getTransaction().commit();

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("user\_not\_found", e);

} finally {

em.close(); // 关闭 EntityManager

}

return user;

}

public void create(User user) throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

try {

em.getTransaction().begin(); // 开始事务

em.persist(user); // 保存用户

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("user\_already\_exists", e);

} finally {

em.close(); // 关闭 EntityManager

}

}

}

**repository/ArticleRepository.java**

package dev.e23.dashstar.repository;

import dev.e23.dashstar.model.Article;

import dev.e23.dashstar.util.HibernateUtil;

import jakarta.enterprise.context.ApplicationScoped;

import jakarta.persistence.EntityManager;

import jakarta.persistence.PersistenceException;

import java.util.List;

@ApplicationScoped // 声明这是一个应用范围的 Bean，需要的地方可以使用 @Inject 注入

public class ArticleRepository {

public List<Article> findAll() throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

List<Article> articles = null;

try {

em.getTransaction().begin(); // 开始事务

articles = em

.createQuery("SELECT a FROM Article a", Article.class)

.getResultList(); // 查询所有文章

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("articles\_not\_found", e);

} finally {

em.close(); // 关闭 EntityManager

}

return articles;

}

public Article findByID(Integer id) throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

Article article = null;

try {

em.getTransaction().begin(); // 开始事务

article = em

.createQuery("SELECT a FROM Article a WHERE a.id = :id", Article.class)

.setParameter("id", id)

.getSingleResult(); // 查询指定 ID 的文章

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("article\_not\_found", e);

} finally {

em.close(); // 关闭 EntityManager

}

return article;

}

public void create(Article article) throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

try {

em.getTransaction().begin(); // 开始事务

em.persist(article); // 保存文章

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("", e);

} finally {

em.close(); // 关闭 EntityManager

}

}

public void update(Article article) throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

try {

em.getTransaction().begin(); // 开始事务

Article existingArticle = em.find(Article.class, article.getId()); // 查找已经存在的 article

try {

HibernateUtil.copyNonNullProperties(article, existingArticle); // 手动合并需要更改的字段

} catch (IllegalAccessException e) {

throw new RuntimeException(e);

}

em.merge(existingArticle); // 更新文章

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("", e);

} finally {

em.close(); // 关闭 EntityManager

}

}

}

**repository/CommentRepository.java**

package dev.e23.dashstar.repository;

import dev.e23.dashstar.model.Comment;

import dev.e23.dashstar.util.HibernateUtil;

import jakarta.enterprise.context.ApplicationScoped;

import jakarta.persistence.EntityManager;

import jakarta.persistence.PersistenceException;

import java.util.List;

@ApplicationScoped // 声明这是一个应用范围的 Bean，需要的地方可以使用 @Inject 注入

public class CommentRepository {

public List<Comment> findByArticleId(Integer id) throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

List<Comment> comments = null;

try {

em.getTransaction().begin(); // 开始事务

comments = em

.createQuery("SELECT c FROM Comment c WHERE c.articleId = :articleId", Comment.class)

.setParameter("articleId", id)

.getResultList(); // 查询指定文章的评论

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("", e);

} finally {

em.close(); // 关闭 EntityManager

}

return comments;

}

public void create(Comment comment) throws PersistenceException {

EntityManager em = HibernateUtil.getEntityManager();

try {

em.getTransaction().begin(); // 开始事务

em.persist(comment); // 保存评论

em.getTransaction().commit(); // 提交事务

} catch (PersistenceException e) {

em.getTransaction().rollback(); // 当出了异常时回滚事务

throw new RuntimeException("", e);

} finally {

em.close(); // 关闭 EntityManager

}

}

}

**handler/UserHandler.java**

package dev.e23.dashstar.handler;

import dev.e23.dashstar.model.User;

import dev.e23.dashstar.model.request.UserLoginRequest;

import dev.e23.dashstar.repository.UserRepository;

import dev.e23.dashstar.security.Secured;

import dev.e23.dashstar.util.BCryptUtil;

import dev.e23.dashstar.util.JwtUtil;

import jakarta.inject.Inject;

import jakarta.ws.rs.\*;

import jakarta.ws.rs.core.MediaType;

import jakarta.ws.rs.core.Response;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@Path("/users") // /api/users/\*

public class UserHandler {

@Inject // 要求注入一个 UserRepository 实例（依赖于 UserRepository 中的 @ApplicationScoped 注解）

private UserRepository userRepository;

@GET // 需要使用 HTTP GET 方法访问

@Path("/") // /api/users

@Secured({"user", "admin"}) // 限制只有真正的 user 和 admin 才可访问这个接口

@Produces(MediaType.APPLICATION\_JSON) // 返回 JSON 格式的数据

public Response getAllUsers() {

List<User> users = userRepository.findAll();

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.OK);

res.put("data", users);

return Response.status(Response.Status.OK).entity(res).build();

}

@GET // 需要使用 HTTP GET 方法访问

@Path("/{id}") // /api/users/{id}

@Produces(MediaType.APPLICATION\_JSON)

public User getUserById(@PathParam("id") int id /\* 从路由的 {id} 中获得 id \*/) {

return userRepository.findByID(id);

}

@POST // 需要使用 HTTP POST 方法访问

@Path("/login") // /api/users/login

@Produces(MediaType.APPLICATION\_JSON) // 返回 JSON 格式的数据

@Consumes(MediaType.APPLICATION\_JSON) // 接收 JSON 格式的数据

public Response login(UserLoginRequest request /\* 接收的 JSON 实际上需要是一个 UserLoginRequest 的模型 \*/) {

/\*

\* 这里补充说明一下上面使用到的 UserLoginRequest，我们管这种在业务层、控制层使用到的模型叫 DTO，数据传输对象，他们通常不会被用来存储进数据库。

\* 但是他们所提供的某些字段，会与其相关的模型有直接联系，比如我们这里的 UserLoginRequest 就是需要它的 username 和 password 字段，用于与数据库交互以及进行业务层校验。

\*/

User user = userRepository.findByUsername(request.getUsername()); // 根据用户名查找用户

if (user == null) { // 用户不存在

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.BAD\_REQUEST);

return Response.status(Response.Status.BAD\_REQUEST).entity(res).build();

}

if (!BCryptUtil.checkPassword(request.getPassword(), user.getPassword())) { // 密码错误

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.BAD\_REQUEST);

res.put("msg", "wrong");

return Response.status(Response.Status.BAD\_REQUEST).entity(res).build();

}

String token = JwtUtil.generateToken(user.getId()); // 生成 JWT，并且在 JWT 内部存储用户的 ID

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.OK);

res.put("token", token);

res.put("data", user);

return Response.status(Response.Status.OK).entity(res).build();

}

@POST // 需要使用 HTTP POST 方法访问

@Path("/register") // /api/users/register

@Produces(MediaType.APPLICATION\_JSON) // 返回 JSON 格式的数据

@Consumes(MediaType.APPLICATION\_JSON) // 接收 JSON 格式的数据

public Response register(User user /\* 接收的 JSON 实际上需要是一个 User 模型 \*/) {

user.setRole("user"); // 默认赋予 user 权限

List<User> users = userRepository.findAll();

if (users.isEmpty()) {

user.setRole("admin"); // 当数据库中没有用户时，第一个注册的用户就是管理员

}

user.setPassword(BCryptUtil.hashPassword(user.getPassword())); // 将提供的密码经过 BCrypt 加密后存储

userRepository.create(user);

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.CREATED);

return Response.status(Response.Status.CREATED).entity(res).build();

}

}

**handler/ArticleHandler.java**

package dev.e23.dashstar.handler;

import dev.e23.dashstar.model.Article;

import dev.e23.dashstar.model.Comment;

import dev.e23.dashstar.model.User;

import dev.e23.dashstar.repository.ArticleRepository;

import dev.e23.dashstar.repository.CommentRepository;

import dev.e23.dashstar.repository.UserRepository;

import dev.e23.dashstar.security.Secured;

import jakarta.inject.Inject;

import jakarta.ws.rs.\*;

import jakarta.ws.rs.core.Context;

import jakarta.ws.rs.core.MediaType;

import jakarta.ws.rs.core.Response;

import jakarta.ws.rs.core.SecurityContext;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@Path("/articles") // /api/articles/\*

public class ArticleHandler {

@Inject // 要求注入一个 ArticleRepository 实例

private ArticleRepository articleRepository;

@Inject // 要求注入一个 CommentRepository 实例

private CommentRepository commentRepository;

@Inject // 要求注入一个 UserRepository 实例

private UserRepository userRepository;

@GET // 指定 HTTP GET 请求

@Path("/") // /api/articles

@Produces(MediaType.APPLICATION\_JSON) // 指定返回的数据类型是 JSON

public Response getAllArticles() {

List<Article> articles = articleRepository.findAll();

Map<String, Object> res= new HashMap<>();

res.put("code", Response.Status.OK);

res.put("data", articles);

return Response.status(Response.Status.OK).entity(res).build();

}

@GET // 指定 HTTP GET 请求

@Path("/{id}") // /api/articles/{id}

@Produces(MediaType.APPLICATION\_JSON) // 返回 JSON 格式的数据

public Response getArticleById(@PathParam("id") Integer id) {

Article article = articleRepository.findByID(id);

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.OK);

res.put("data", article);

return Response.status(Response.Status.OK).entity(res).build();

}

@GET // 指定 HTTP GET 请求

@Path("/{id}/comments") // /api/articles/{id}/comments

@Produces(MediaType.APPLICATION\_JSON) // 返回 JSON 格式的数据

public Response getComments(@PathParam("id") Integer id) {

List<Comment> comments = commentRepository.findByArticleId(id);

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.OK);

res.put("data", comments);

return Response.status(Response.Status.OK).entity(res).build();

}

@POST // 指定 HTTP POST 请求

@Path("/") // /api/articles

@Secured({"admin"}) // 限制只有真正的 admin 才可访问这个接口，毕竟只有管理员才能在自己的博客上发文章

@Consumes(MediaType.APPLICATION\_JSON) // 接收 JSON 格式的数据

@Produces(MediaType.APPLICATION\_JSON) // 返回 JSON 格式的数据

public Response createArticle(Article article /\* 接收的 JSON 实际上需要是一个 Article 的模型 \*/, @Context SecurityContext securityContext /\* 从请求上下文中获得在 AuthenticationFilter 中的 SecurityContext \*/) {

User author = userRepository.findByID(Integer.valueOf(securityContext.getUserPrincipal().getName())); // 从 SecurityContext 中获得当前登录的用户

article.setAuthor(author); // 设置文章的作者

article.setCreatedAt(System.currentTimeMillis() / 1000); // 设置文章的创建时间

articleRepository.create(article); // 创建文章

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.OK);

return Response.status(Response.Status.OK).entity(res).build();

}

@PUT // 指定 HTTP PUT 请求

@Path("/") // /api/articles

@Secured({"admin"}) // 限制只有真正的 admin 才可访问这个接口，毕竟只有管理员才能在自己的博客上更新文章

@Consumes(MediaType.APPLICATION\_JSON) // 接收 JSON 格式的数据

@Produces(MediaType.APPLICATION\_JSON) // 返回 JSON 格式的数据

public Response updateArticle(Article article /\* 接收的 JSON 实际上需要是一个 Article 的模型 \*/, @Context SecurityContext securityContext /\* 从请求上下文中获得在 AuthenticationFilter 中的 SecurityContext \*/) {

article.setAuthorId(Integer.valueOf(securityContext.getUserPrincipal().getName())); // 设置文章的作者

articleRepository.update(article); // 更新文章

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.OK);

return Response.status(Response.Status.OK).entity(res).build();

}

}

**handler/CommentHandler.java**

package dev.e23.dashstar.handler;

import dev.e23.dashstar.model.Article;

import dev.e23.dashstar.model.Comment;

import dev.e23.dashstar.model.User;

import dev.e23.dashstar.repository.ArticleRepository;

import dev.e23.dashstar.repository.CommentRepository;

import dev.e23.dashstar.repository.UserRepository;

import dev.e23.dashstar.security.Secured;

import jakarta.inject.Inject;

import jakarta.ws.rs.\*;

import jakarta.ws.rs.core.Context;

import jakarta.ws.rs.core.MediaType;

import jakarta.ws.rs.core.Response;

import jakarta.ws.rs.core.SecurityContext;

import java.util.HashMap;

import java.util.Map;

@Path("/comments") // /api/comments/\*

public class CommentHandler {

@Inject // 要求注入一个 CommentRepository 实例

private CommentRepository commentRepository;

@Inject // 要求注入一个 UserRepository 实例

private UserRepository userRepository;

@Inject // 要求注入一个 ArticleRepository 实例

private ArticleRepository articleRepository;

@POST // 指定 HTTP POST 请求

@Path("/") // /api/comments

@Secured({"user", "admin"}) // 限制只有真正的 user 和 admin 才可访问这个接口，只有经过认证的用户才能发评论，不然乱套了

@Consumes(MediaType.APPLICATION\_JSON) // 指定请求体的媒体类型为 JSON

@Produces(MediaType.APPLICATION\_JSON) // 接收 JSON 格式的数据

public Response createComment(Comment comment /\* 接收的 JSON 实际上需要是一个 Comment 的模型 \*/, @Context SecurityContext securityContext /\* 从请求上下文中获得在 AuthenticationFilter 中的 SecurityContext \*/) {

User user = userRepository.findByID(Integer.valueOf(securityContext.getUserPrincipal().getName())); // 从 SecurityContext 中获得当前登录的用户

comment.setUser(user); // 设置评论的用户

Article article = articleRepository.findByID(comment.getArticleId()); // 根据评论中的文章 ID 获得文章

comment.setArticle(article); // 设置评论的文章

comment.setCreatedAt(System.currentTimeMillis() / 1000); // 设置评论的创建时间（Unix 时间戳）

commentRepository.create(comment); // 创建评论

Map<String, Object> res = new HashMap<>();

res.put("code", Response.Status.OK);

return Response.status(Response.Status.OK).entity(res).build();

}

}

**listener/HibernateListener.java**

package dev.e23.dashstar.listener;

import dev.e23.dashstar.util.HibernateUtil;

import jakarta.servlet.ServletContextEvent;

import jakarta.servlet.ServletContextListener;

import jakarta.servlet.annotation.WebListener;

import jakarta.persistence.PersistenceException;

@WebListener

public class HibernateListener implements ServletContextListener {

@Override

public void contextInitialized(ServletContextEvent sce) {

try {

HibernateUtil.init(); // 创建 EntityManagerFactory，即使得 Hibernate 可以管理实体类并创建数据库连接。若失败，则关闭服务器。

} catch (PersistenceException ex) {

System.err.println("Database connection failed. Server will shut down.");

shutdownServer();

}

}

@Override

public void contextDestroyed(ServletContextEvent sce) {

if (HibernateUtil.getEntityManagerFactory() != null && HibernateUtil.getEntityManagerFactory().isOpen()) {

HibernateUtil.getEntityManagerFactory().close();

}

}

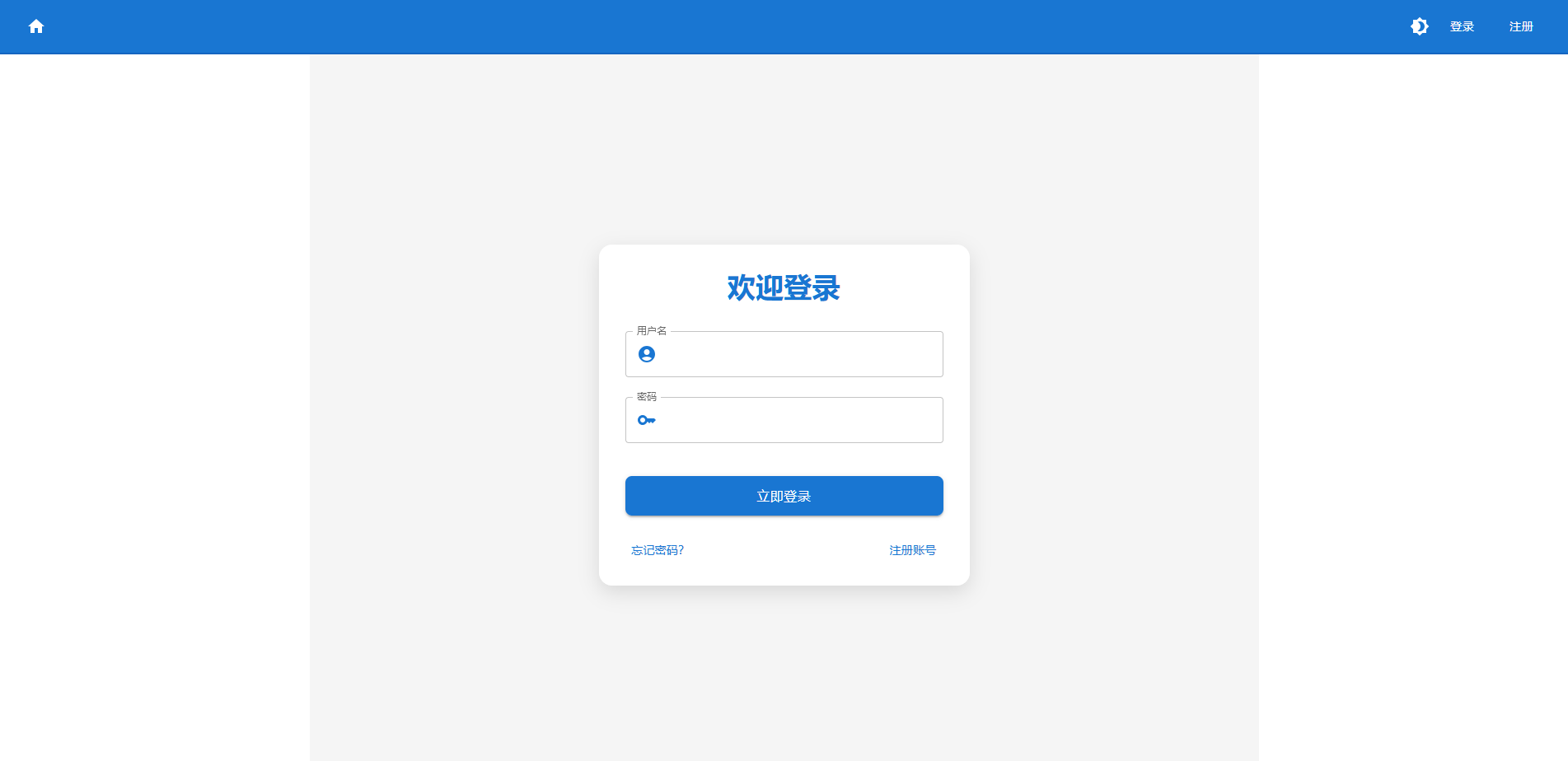
private void shutdownServer() {

System.exit(1);

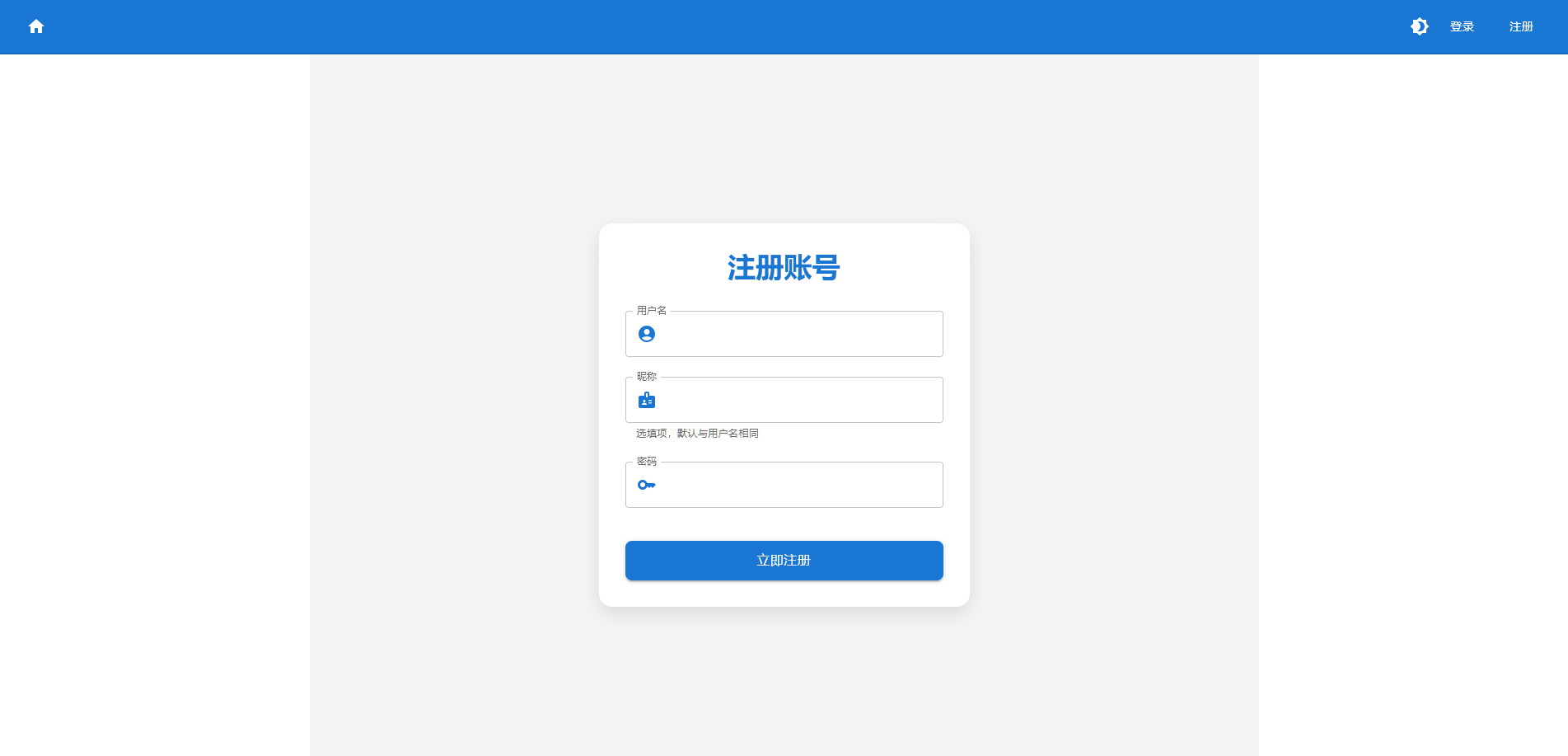
}

}

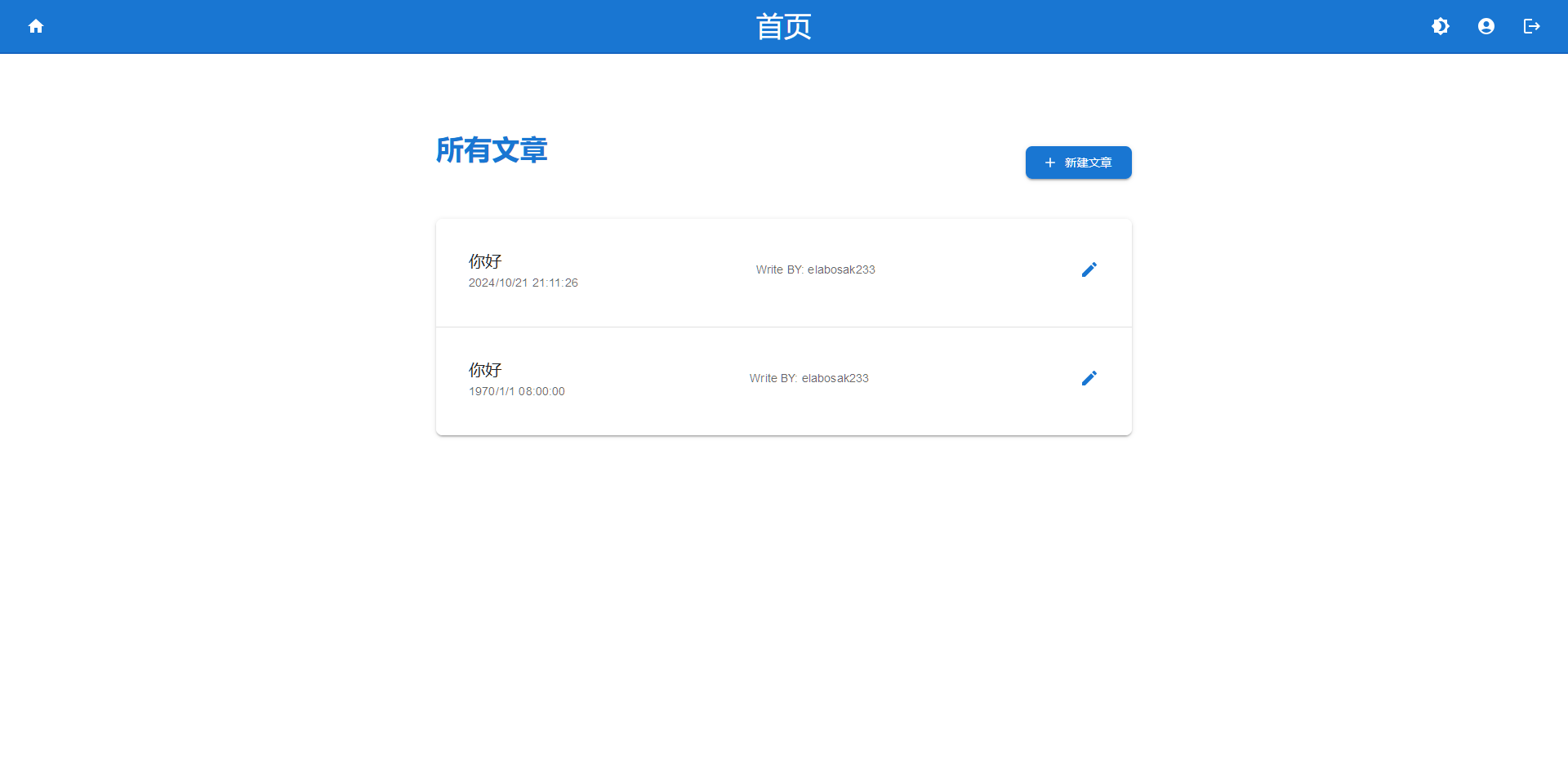
系统运行结果截图：



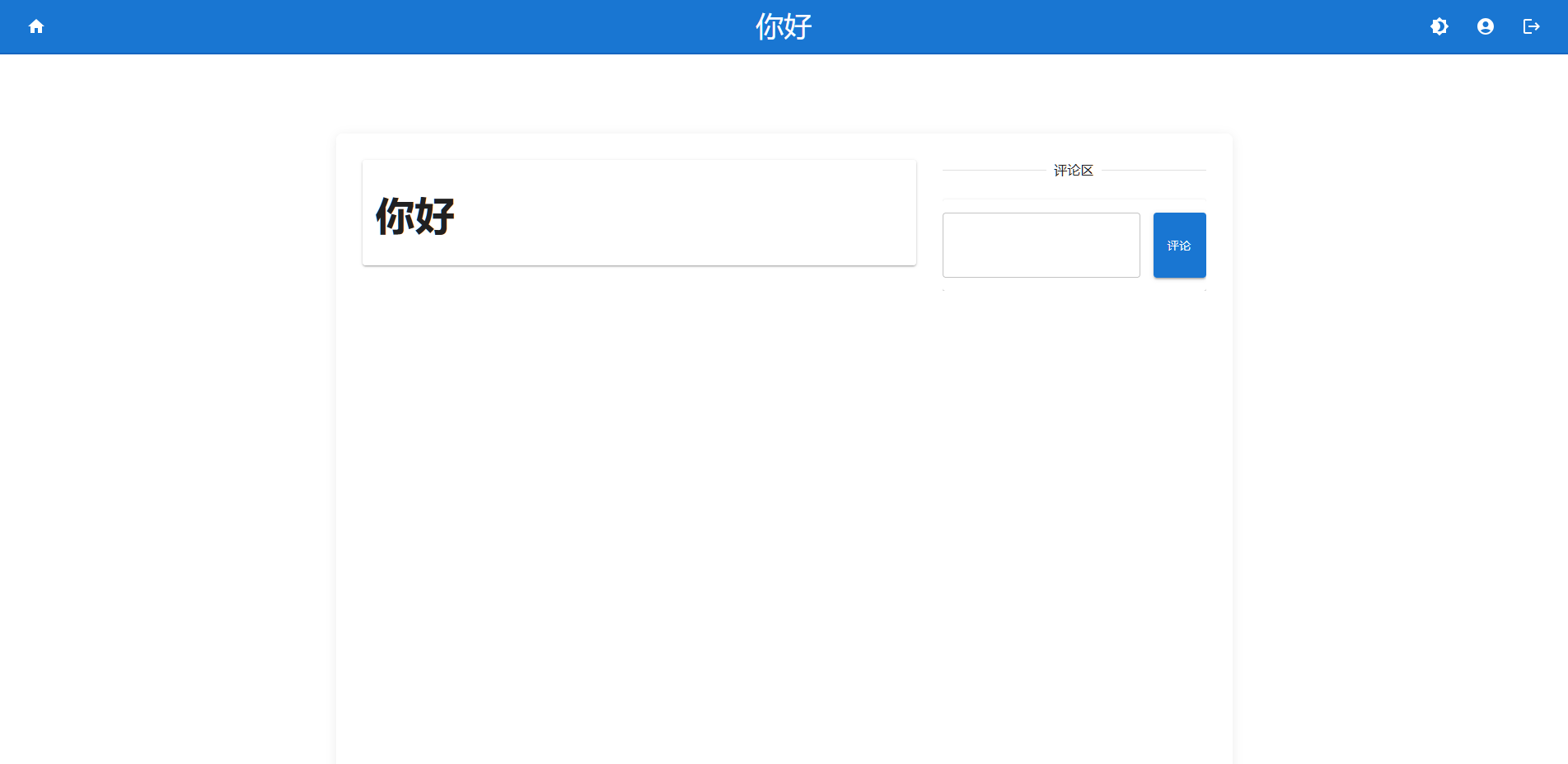
(图1 用户登录界面)



(图2 用户注册界面)



(图3 首页文章列表页面)



(图4文章页面)



(图5文章编辑页面)