ANINTERNSHIPREPORTON

JAVA & FSD PROGRAMMING

AReportSubmittedto

BlackbuckEngineersPvt.Ltd

Submittedby

STUDENT NAME (BBID)

B.Madhan Kishore (BBPBV051)
R.Vinay Kumar (BBPBV052)
V.Sathya Sai Danush Narayana (BBPBV053)

Blackbuck Engineers Pvt.

LtdRoadno36,JubileeHills,Hyderabad

CERTIFICATE

TABLEOF CONTENTS

ACKNOWLEDGEMENT

We would like to acknowledge all those without whom this project would not have beensuccessful. Firstly, we would like to thank our Internet and Java & FSD Programmer Mr.(RamaKrishna) who guided us throughout the project and gave his immense support. hemade us understand how to complete this project and without his presence, this project wouldnot have been complete. We also got to know a lot about parallelization and its benefits. Thisproject has been a source to learn and bring our theoretical knowledge to the real-life world.Once again, thanks to everyoneformaking this project successful.

Place :

Date :

ABSTRACT

In an era dominated by digital content consumption, creating an efficient and user-friendly blog platform is essential for individuals and organizations seeking to share their thoughts and ideas with the world. This abstract introduces a Spring project aimed at developing a robust and flexible blog platform that leverages the power of the Spring Framework.

Our Spring Blog Platform project addresses the need for a reliable content management system that offers features such as user authentication, content creation, categorization, and interactive user engagement. Leveraging the Spring ecosystem, including Spring Boot, Spring Security, and Spring Data, our platform provides a solid foundation for building a high-quality, scalable, and secure blog application.

1. INTRODUCTION

1.1. SYSTEMOVERVIEW

The Blog Platform Spring Project is a modern web application built using the Spring Framework, aimed at providing bloggers and content creators with a feature-rich, scalable, and secure platform for publishing and managing their blogs. This system overview outlines the key components and functionalities of the project.

1.2. OBJECTIVE

The primary objective of the Blog Platform Spring Project is to design, develop, and deploy a robust and feature-rich blogging platform using the Spring Framework. This project aims to meet the following key objectives:

Content creation and management

User engagement and interaction

User authentication and authorisation l

1.3. PROBLEMIDENTIFICATION

Thesearesomeoftheproblems facedbyapersonwhowants tostart astart-up!

- Complex content management
- Security vulnerabilities
- Inefficient content organisation
- Limited user engagement
- Scalability challenges
- Complex deployment
- Data privacy concerns
- Ineffective analytics
- Community buildings.

1.4. APPLICATIONS

Thiswebapplication canbeusedbyanycompany to investinnewventures

- Content management application
- User profile application
- User registration and authentication application.

1.5. LITERATURE SURVEY

w literature on the Spring Framework and its components, including Spring Boot, Spring Security, and Spring Data, to understand their capabilities and advantages in web application development.

Explore best practices for building scalable and secure web applications using Spring.

Investigate how Spring integrates with front-end technologies like Angular, React, or Vue.js for building modern web interfaces.

User Authentication and Authorization:

Study authentication and authorization mechanisms, especially those implemented in Spring Security.

Examine research on secure user authentication and authorization practices in web applications, including token-based authentication and OAuth.

literature on the Spring Framework and its components, including Spring Boot, Spring Security, and Spring Data, to understand their capabilities and advantages in web application development.

Explore best practices for building scalable and secure web applications using Spring.

Investigate how Spring integrates with front-end technologies like Angular, React, or Vue.js for building modern web interfaces.

User Authentication and Authorization:

Study authentication and authorization mechanisms, especially those implemented in Spring Security.

Examine research on secure user authentication and authorization practices in web applications, including token-based authentication and OAuth.

1.6. LIMITATIONS

- Learning curve
- Resource intensive
- Complexity
- Database scalability

2. SYSTEMANALYSIS

2.1. EXISTINGSYSTEM

Identify the Current Blogging System (if any):

Determine if there is an existing blogging system in place.
Evaluate its features, functionality, and limitations.
2. User Feedback and Usage Patterns:

Gather user feedback and assess how the current system is used.
Identify pain points, user preferences, and areas for improvement.
3. Data Migration and Compatibility:

Determine if there is existing data (e.g., blog posts, user profiles) that needs to be migrated to t
he new platform.
Analyze data formats and compatibility issues between the old and new systems.
4. Integration Points:

Identify any integrations or dependencies on external services or systems (e.g., authentication pro
viders, content delivery networks) in the current setup.
Evaluate the impact of these integrations on the new platform.

2.2. PROPOSEDSYSTEM

Feature Set: Enumerate the features to be included in the new platform, including user authentication, content creation, commenting, social sharing, and search functionality.

User Roles and Permissions: Define user roles (e.g., administrators, authors, readers) and specify their respective permissions and access levels within the platform.

Database Design: Outline the proposed database schema, emphasizing how blog posts, comments, user profiles, and other data will be structured and related.

System Architecture: Describe the system architecture, including the use of Spring Boot and relevant architectural patterns (e.g., MVC, RESTful).

Frontend Design: Present the envisioned user interface design, including wireframes or mockups, layout, navigation, and user experience (UX) considerations.

Security Measures: Specify the security measures to be implemented, such as user authentication, authorization, data encryption, and protection against common web vulnerabilities.

Performance Optimization: Detail strategies for optimizing the platform's performance, including techniques to enhance responsiveness and reduce latency.

Deployment Strategy: Define the deployment strategy, including hosting options (e.g., cloud-based, on-premises) and server configurations.

Testing Plan: Outline the testing plan, including unit testing, integration testing, and user acceptance testing, along with the tools and frameworks to be used.

3. REQUIREMENTSPECIFICATION

3.1. HARDWAREREQUIREMENTS

- Server
- Database server
- Loadbalance
- Backup and storage

3.2. SOFTWAREREQUIREMENTS

- Windows 10
- Eclipse
- Postman
- Maven dependencies

3.3. COMPONENTSUSED

- Eclipse
- Postman
- GoogleChrome
- Windows 10

4. ARCHITECTUREDESIGNSPECIFICATION

4.1. SYSTEMARCHITECTURE

1. User Interface (UI):

The UI layer is the frontend of the blog platform, responsible for presenting content to users.
It can be built using modern frontend frameworks and technologies like React, Angular, or Vue.js.
The UI communicates with the backend through RESTful APIs.
2. Load Balancer:

A load balancer distributes incoming web traffic across multiple web servers to ensure scalability and high availability.
It can handle SSL termination and distribute requests to backend servers based on various load balancing algorithms.
3. Web Servers (Spring Boot Application):

Spring Boot applications serve as the core of the blog platform.
Multiple instances of the Spring Boot application run behind the load balancer to handle incoming requests.
Spring Boot's embedded web server (e.g., Tomcat) is commonly used.

4.2. DETAILEDDESIGN

1. User Interface (UI):

Frontend Framework: Utilize a modern JavaScript framework (e.g., React, Angular, Vue.js) to create a responsive and dynamic user interface.
User Experience (UX): Implement a clean and intuitive design, incorporating best practices for accessibility and mobile responsiveness.
Web Components: Create reusable UI components for consistency and maintainability.
Communication: Use AJAX or fetch API to communicate with the backend via RESTful APIs.
2. Load Balancer:

Load Balancing Algorithms: Employ load balancing algorithms such as round-robin or least connections to distribute incoming requests evenly.
SSL Termination: Offload SSL termination at the load balancer for improved security and performance.
3. Web Servers (Spring Boot Application):

Spring Boot Modules: Organize the Spring Boot application into modular components based on functionality (e.g., user management, content management).
Containerization: Use Docker for containerization to ensure consistent deployment across different environments.
Deployment: Deploy the Spring Boot application instances on virtual machines or container orchestration platforms like Kubernetes.

4.3. DATABASEDESIGN

1. Users Table:

Store information about registered users.
Fields may include:
User ID (Primary Key)
Username
Email
Password (hashed and salted)
User Role (e.g., admin, author, reader)
Registration Date
Profile Picture (URL or reference to a stored image)
2. Blog Posts Table:

Store data related to individual blog posts.
Fields may include:
Post ID (Primary Key)
Title
Content (HTML or Markdown)
Author ID (Foreign Key to Users Table)
Publication Date
Last Modification Date
Status (e.g., draft, published)
Tags or Categories
3. Comments Table:

Store comments made on blog posts.
Fields may include:
Comment ID (Primary Key)
Post ID (Foreign Key to Blog Posts Table)
Author ID (Foreign Key to Users Table)
Comment Text
Comment Date
Parent Comment ID (for threaded comments)
Status (e.g., approved, pending)

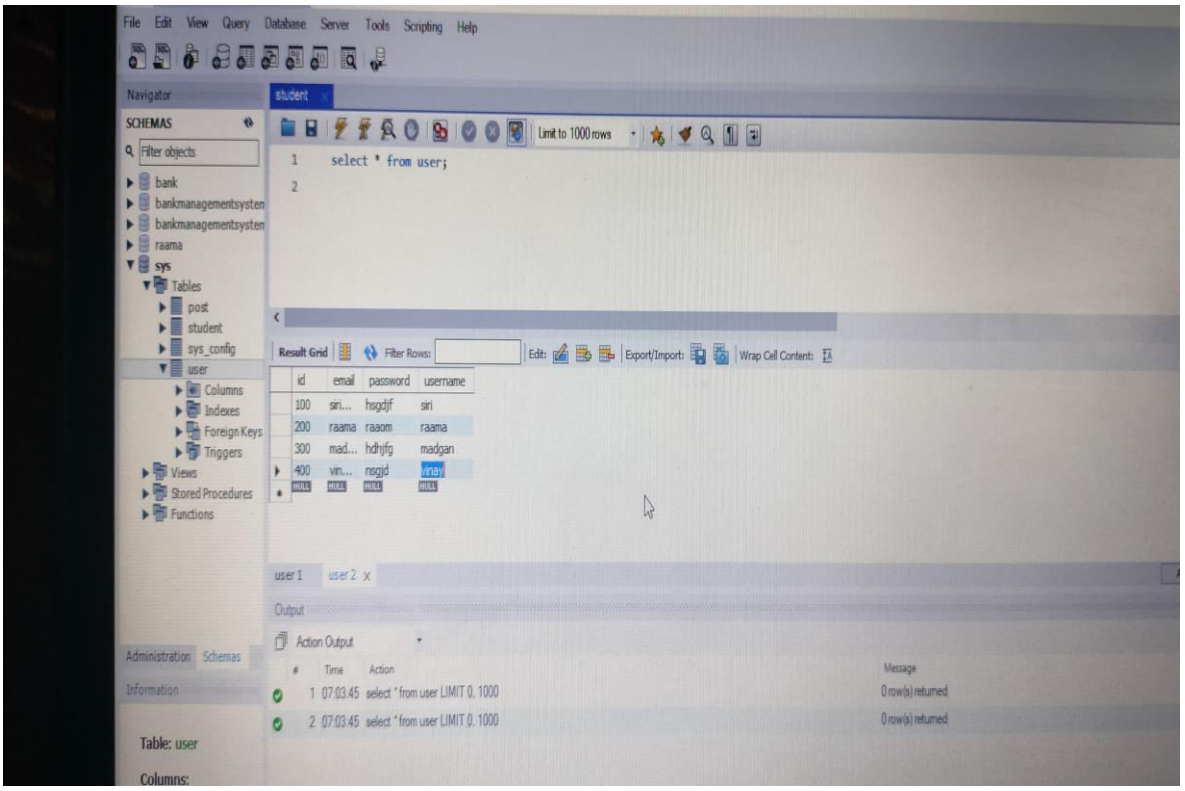


Fig1.User table

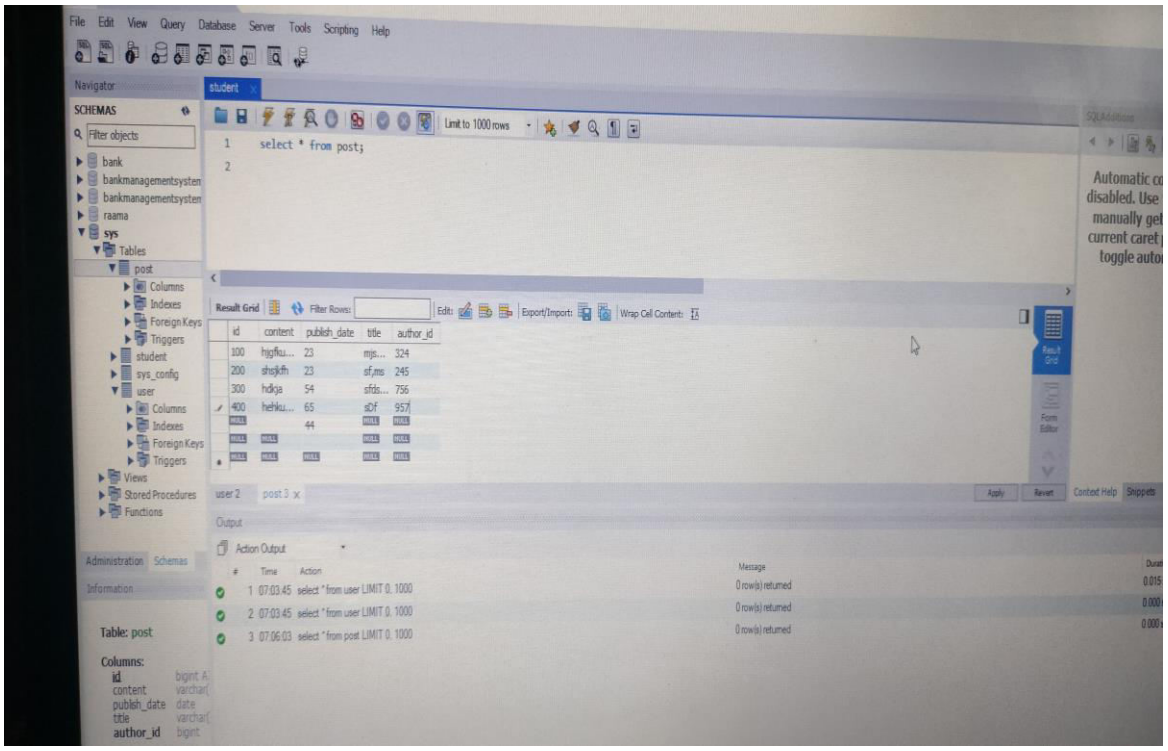


Fig2.Post table

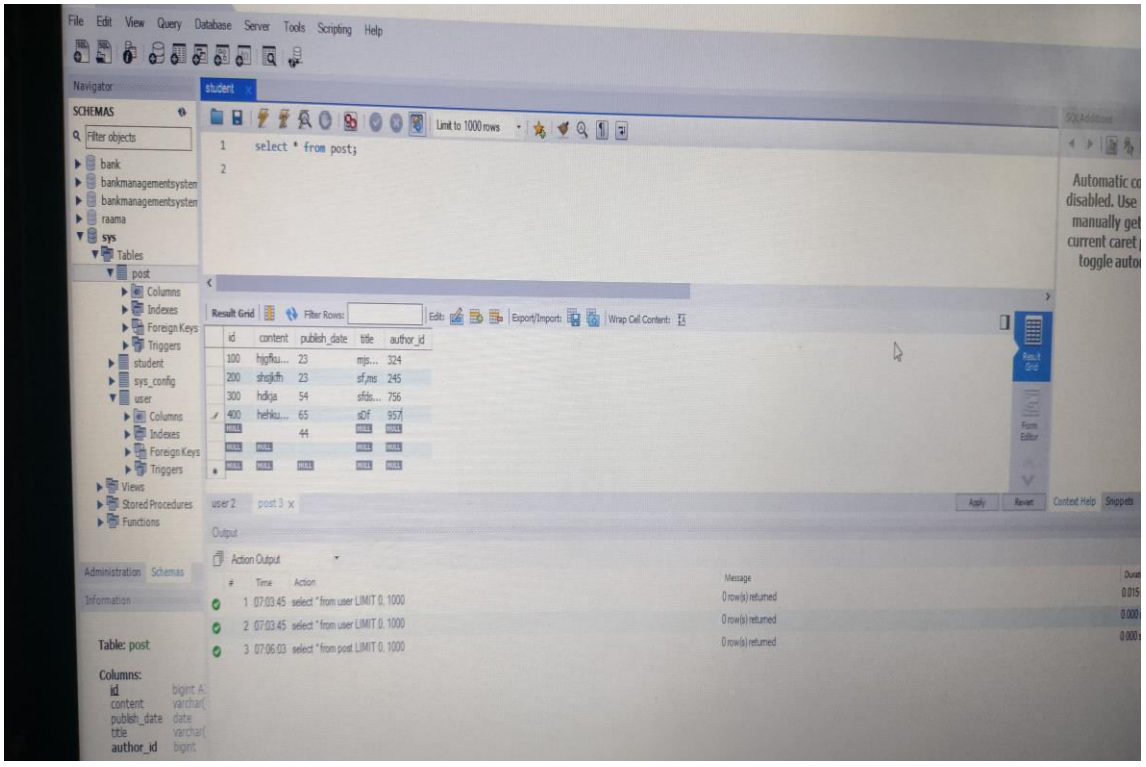
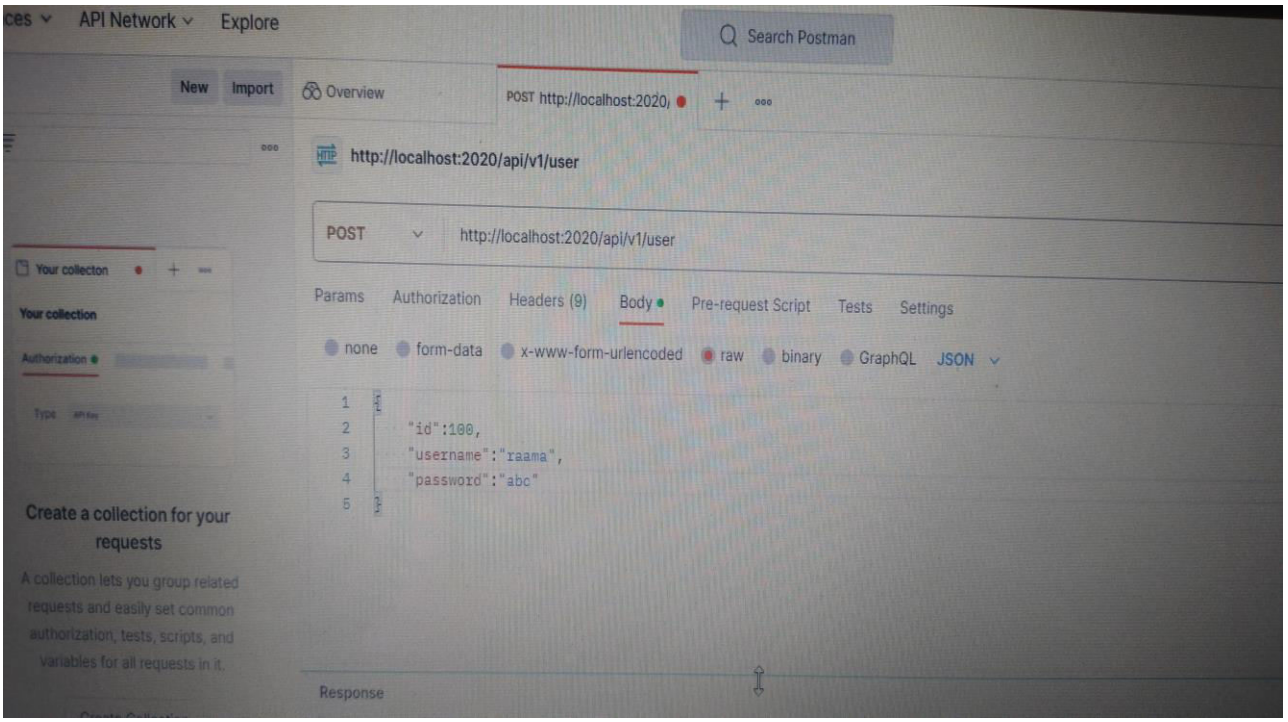
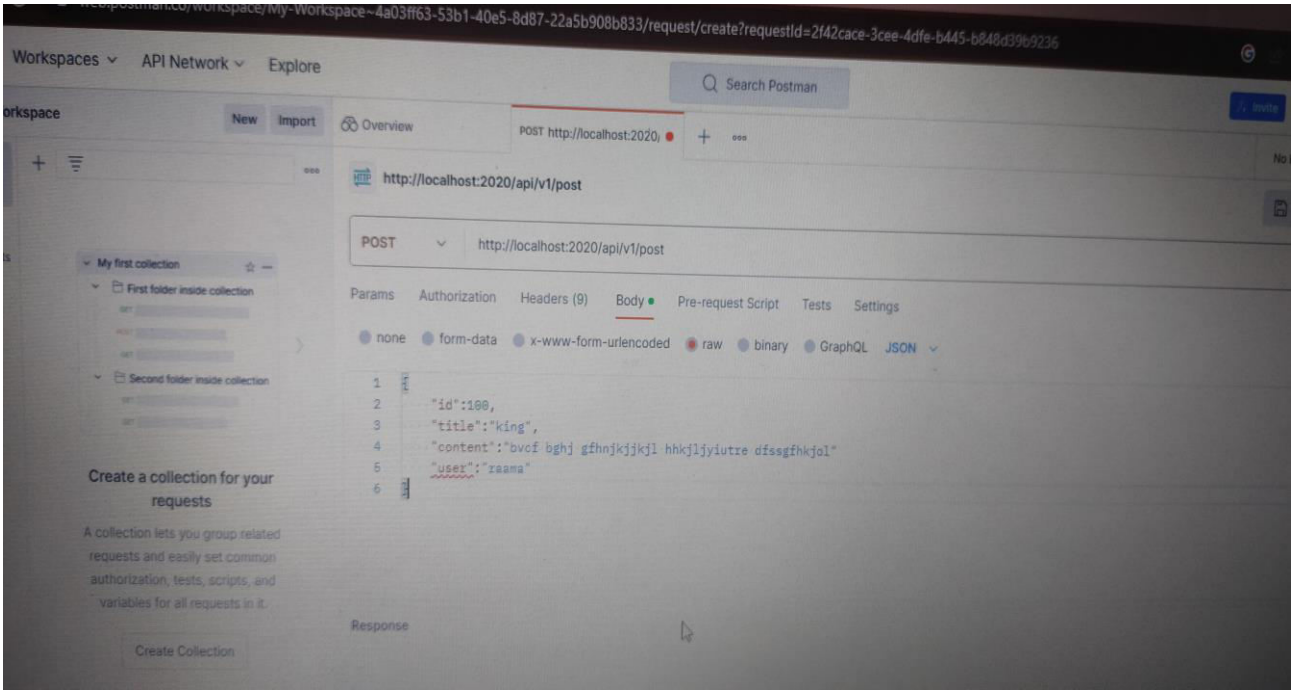
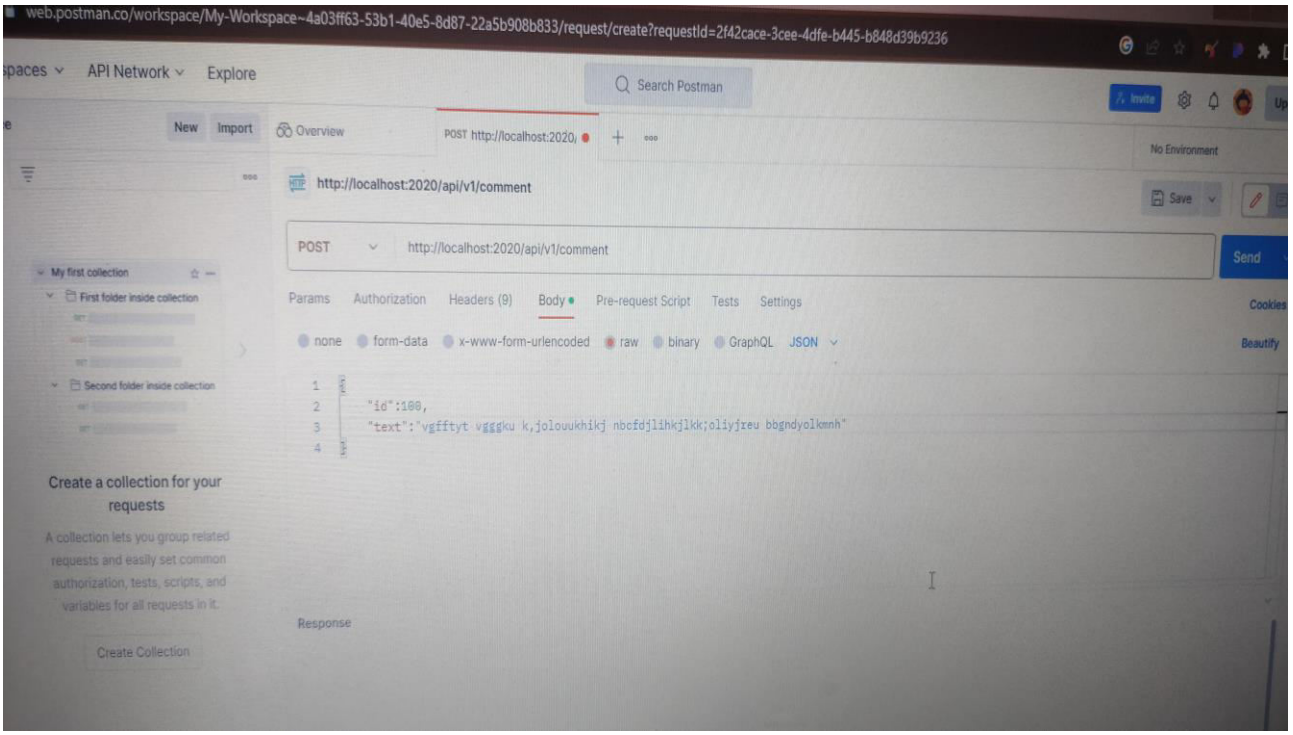


Fig3.Comment table

5. SYSTEM IMPLEMENTATION







6. CONCLUSION AND FUTURE ENHANCEMENTS

In this system design, we have laid out a comprehensive architecture for a Spring Boot-based blog platform. This design addresses various aspects, including user authentication, content management, scalability, and performance. The key components include a user-friendly frontend, load balancing for high availability, a Spring Boot backend, a robust database schema, and security features like user authentication and authorization. Additionally, we've incorporated optional components such as caching, asynchronous processing, and CDN integration to enhance the platform's performance and user experience.

The proposed database schema efficiently stores user profiles, blog posts, comments, and other relevant data. It follows best practices for data organization and integrity while facilitating efficient querying and retrieval.

This design is a solid foundation for developing a robust and feature-rich blog platform that can cater to a wide range of users and content creators.

7. APPENDICES

7.1.User Entity

```
package com.vits.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class User {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String username;
   private String password;
        public Long getId() {
                return id;
        }
        public void setId(Long id) {
                this.id = id;
        }
        public String getUsername() {
                return username;
        }
        public void setUsername(String username) {
                this.username = username;
        }
        public String getPassword() {
                return password;
        }
        public void setPassword(String password) {
                this.password = password;
        }
        @Override
        public String toString() {
                return "User [id=" + id + ", username=" + username + ", password=" + password + "]";
        }
        public User(Long id, String username, String password) {
                super();
                this.id = id;
                this.username = username;
                this.password = password;
        }

  public User() {
```

```
    }
}
```

7.2.Post Entity

```
package com.vits.entity;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.xml.stream.events.Comment;

@Entity
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String content;

    @ManyToOne
    private User author;

    @OneToMany(mappedBy = "post", cascade = CascadeType.ALL)
    private List<Comment> comments = new ArrayList<>();

        public Long getId() {
                return id;
        }

        public void setId(Long id) {
                this.id = id;
        }

        public String getTitle() {
                return title;
        }

        public void setTitle(String title) {
                this.title = title;
        }

        public String getContent() {
                return content;
        }

        public void setContent(String content) {
                this.content = content;
```

```
	}

	public User getAuthor() {
		return author;
	}

	public void setAuthor(User author) {
		this.author = author;
	}

	public List<Comment> getComments() {
		return comments;
	}

	public void setComments(List<Comment> comments) {
		this.comments = comments;
	}

	@Override
	public String toString() {
		return "Post [id=" + id + ", title=" + title + ", content=" + content + ", author=" + author + ",
comments="
				+ comments + "]";
	}

	public Post(Long id, String title, String content, User author, List<Comment> comments) {
		super();
		this.id = id;
		this.title = title;
		this.content = content;
		this.author = author;
		this.comments = comments;
	}

   public Post() {

   }
}
```

7.3.Comment entity

```
package com.vits.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

@Entity
public class Comment {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
```

```
    private String text;

    @ManyToOne
    private Post post;

        public Long getId() {
            return id;
        }

        public void setId(Long id) {
            this.id = id;
        }

        public String getText() {
            return text;
        }

        public void setText(String text) {
            this.text = text;
        }

        public Post getPost() {
            return post;
        }

        public void setPost(Post post) {
            this.post = post;
        }

        @Override
        public String toString() {
            return "Comment [id=" + id + ", text=" + text + ", post=" + post + "]";
        }

        public Comment(Long id, String text, Post post) {
            super();
            this.id = id;
            this.text = text;
            this.post = post;
        }

    public Comment() {

    }
}
```

Comment controller:

```
package com.vits.controller;

import java.util.List;

import javax.xml.stream.events.Comment;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.vits.Service.CommentService;

@RestController
@RequestMapping("/api/comments")
public class CommentController {
    private final CommentService commentService;

    @Autowired
    public CommentController(CommentService commentService) {
        this.commentService = commentService;
    }

    @PostMapping
    public ResponseEntity<String> createComment(@RequestBody Comment comment) {
        commentService.createComment(comment);
        return ResponseEntity.ok("Comment created successfully.");
    }

    @GetMapping("/{id}")
    public ResponseEntity<Comment> getCommentById(@PathVariable Long id) {
        Comment comment = commentService.getCommentById(id);
        return ResponseEntity.ok(comment);
    }

    @GetMapping("/post/{postId}")
    public ResponseEntity<List<Comment>> getCommentsByPostId(@PathVariable Long postId) {
        List<Comment> comments = commentService.getCommentsByPostId(postId);
        return ResponseEntity.ok(comments);
    }

    // Add more comment-related endpoints as needed
}
```

Post controller:

```
package com.vits.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.vits.Service.PostService;
import com.vits.entity.Post;

@RestController
@RequestMapping("/api/posts")
public class PostController {
    private final PostService postService;

    @Autowired
    public PostController(PostService postService) {
        this.postService = postService;
    }

    @PostMapping
    public ResponseEntity<String> createPost(@RequestBody Post post) {
        postService.createPost(post);
        return ResponseEntity.ok("Post created successfully.");
    }

    @GetMapping("/{id}")
    public ResponseEntity<Post> getPostById(@PathVariable Long id) {
        Post post = postService.getPostById(id);
        return ResponseEntity.ok(post);
    }

    @GetMapping("/user/{userId}")
    public ResponseEntity<List<Post>> getPostsByUserId(@PathVariable Long userId) {
        List<Post> posts = postService.getPostsByUserId(userId);
        return ResponseEntity.ok(posts);
    }

    // Add more post-related endpoints as needed
}
```

User Controller:

```
package com.vits.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.vits.Service.UserService;
import com.vits.entity.User;
```

```
@RestController
@RequestMapping("/api/users")
public class UserController {
    private final UserService userService;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }

    @PostMapping("/register")
    public ResponseEntity<String> registerUser(@RequestBody User user) {
        userService.registerUser(user);
        return ResponseEntity.ok("User registered successfully.");
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
        User user = userService.getUserById(id);
        return ResponseEntity.ok(user);
    }

    // Add more user-related endpoints as needed
}
```

User Repository:

```
package com.vits.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.vits.entity.User;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
    User registerUser(String username);
    User getUserById(long id);
}
```

Post Repository:

```
package com.vits.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.vits.entity.Post;
```

```
@Repository
public interface PostRepository extends JpaRepository<Post, Long> {
}
```

Comment Controller:

```
package com.vits.repository;

import javax.xml.stream.events.Comment;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CommentRepository extends JpaRepository<Comment, Long> {

}
```

8. REFERENCES

8.1. LISTOFJOURNALS

- TY - BOOK, AU - Kumar, Dr. Gopaldas, PY - 2018/03/15 T1 - INDIANSTARTUPS-ISSUES,CHALLENGES,ANDOPPORTUNITIES
- TY - BOOK , AU - Salamzadeh, Aidin,AU - Kawamorita, Hiroko , PY - 2015/01/01,T1 - Startup Companies: Life Cycle and Challenges , DO - 10.13140/RG.2.1.3624.8167
- TY - BOOK, AU - Sobolev, Aleksandr, PY - 2020/12/07, SP - 612, EP - 617, T1 - Start-Ups Evaluation With The Help Of Web-Based Platforms, DO - 10.15405/epsbs.2020.12.79
- Salomon, V. (2018). Strategies of Startup Evaluation on Crowdinvesting Platforms:the Case of Switzerland. *Journal of Innovation Economics & Management*, 26, 63-88.https://doi.org/10.3917/jie.pr1.0029

- TY - JOUR, AU - Slavik, Stefan, PY - 2020/01/01, SP- 01063, T1- Businessideas in start-ups, VL - 83, DO - 10.1051/shsconf/20208301063,JO - SHS Web ofConferences.
- TY - JOUR, AU - Čalopa, M.K., AU - Horvat, J., AU - Lalić, M., PY - 2014/01/01, SP - 19, EP - 44, T1 - Analysis of financing sources for start-upcompanies,VL-19, JO-Management(Croatia)
- TY - JOUR. AU - Janaji, Siti, AU - Ibrahim, Fahmi, AU - Ismail, Kamariah, PY - 2021/06/21,SP-88, EP-92, T1-StartupsandSources of Funding, VL- 2

8.2. LISTOFWEBSITES

1. https://www.apachefriends.org/download.html
2. https://code.visualstudio.com/download
3. https://www.php.net/downloads.php
4. https://www.researchgate.net/publication/347392298_Start-Ups_Evaluation_With_The_Help_Of_Web-Based_Platforms
5. cairn.info/revue-journal-of-innovation-economics-2018-2-page-63.htm
6. https://www.researchgate.net/publication/346557861_Business_ideas_in_start-ups
7. https://www.researchgate.net/publication/287307261_Analysis_of_financing_sources_for_start-up_companies/citation/download
8. https://www.researchgate.net/publication/352559631_Startups_and_Sources_of_Funding