Dharmsinh Desai University, Nadiad

Faculty of Technology, Department of Computer Engineering

B.Tech. CE Semester – V

Subject: (CE-515) Advanced Technologies

Project Title: Personal Blog System

By:

Sahil Bhuva (ID: 18CEUOF008 Roll No: CE017)

Guided By:

Prof. Prashant M. Jadav

Prof. Ankit P. Vaishnav

Dharmsinh Desai University, Nadiad

Faculty of Technology, Department of Computer Engineering

CERTIFICATE

This is to certify that Advanced Technologies project entitled “Personal Blog System” is the bonafide report of work carried out by **Sahil Bhuva (ID: 18CEUOF008)** Of Department of Computer Engineering, Semester V, academic year 2020-21, under our supervision and guidance.

Guide	Guide	HOD
Prof. Prashant Jadav	**Prof. Ankit Vaishnav**	**Dr. C. K. Bhensdadia**
Assistant Professor of	Assistant Professor of	Head of the Department
Department of Computer	Department of Computer	Department of Computer
Engineering, Dharmsinh Desai University, Nadiad.	Engineering, Dharmsinh Desai University, Nadiad.	Engineering, Dharmsinh Desai University, Nadiad.

Table of Contents

1. Abstract

A Personal Blog website is a site dedicated to the blogging. The site focuses on posts and comments. The purpose of the site is that every company either it is product based or service based needs some platform like blogging to make their announcements, updates and future goals. By the means of this blogging service, features discussed above are easily maintained with high efficiency. One such good example of this service is https://blog.google.

2. Introduction

2.1 Brief Introduction

Personal Blog Site provides services to add, remove and updating of the post with access control. The site saves the time of author by giving the lots of feature on hand. Personal blog site has also feature of commenting. By the means of commenting, company can get reviews, problems and feature request by users.

2.2 Tools/Technologies used

Technologies:

HTML 5

CSS 3

Bootstrap 4

Angular 10

Typescript

Node JS

Express JS

MongoDB

Tools:

Visual Studio Code

Postman

WebStrom

Platforms:

Google Chrome

Mozilla Firefox

3. Software specification requirement

3.1 Functional Requirement

3.1.1 Show Posts

output: List of posts with the link to respective post page.

description: To list down all the post on the home page of the site**.**

precondition: There is no need to log in to as post must be available to all.

3.1.2 Manage User

- Login

input: Username and Password.

output: Success or Failure.

description: User can use this page to login to the system by entering personal credentials (username and password) and will be redirected to a home page based on role to use the features of the website.

process: Check the username and matching password in the database. If found in the database then redirect the user to his/her home page.

precondition: User must be registered on the website.

postcondition: User is logged in to the website and a new session is associated with the client connection.

- Registration

input: User needs to fill up the registration form.

output: Success or Failure based on availability.

process: If data are valid, make changes to database otherwise handle appropriately.

postcondition: User is logged in to the website and a new session is associated with the client connection.

- Comment

input: User needs to fill up text for the comment.

output: Success or Failure.

process: If data are valid, make changes to database otherwise handle appropriately.

precondition: User must be logged in before commenting.

postcondition: An alert is displayed based on success or failure. If success, comment will be displayed.

3.1.3 Manage Administrator

description: This are the feature for the administrator.

precondition: User must be logged in and must have admin role (except login feature).

- Login

  input: Username and Password.

  output: Success or Failure.

  description: User can use this page to login to the system by entering personal credentials (username and password) and will be redirected to a home page based on role to use the features of the website.

  process: Check the username and matching password in the database. If found in the database then redirect the user to his/her home page.

  precondition: User must be registered on the website and must have role of admin.

  postcondition: User is logged in to the website and a new session is associated with the client connection.

- Add Comment

  input: User needs to fill up text for the comment.

  output: Success or Failure.

  process: If data are valid, make changes to database otherwise handle appropriately.

  postcondition: An alert is displayed based on success or failure. If success, comment will be displayed.

- Delete Comment

  input: User click.

  output: Success or Failure.

  process: If validation is successful, make changes to database otherwise handle appropriately.

  postcondition: An alert is displayed based on success or failure. If success, comment will be removed.

- Add Post

input: User needs to fill up the from containing info like title, image if any, content etc.

output: Success or Failure.

description: This feature is used to add new post by the administrator.

process: If validation is successful, make changes to database otherwise handle appropriately.

postcondition: An alert is displayed based on success or failure. If success, post will be added.

- Update Post

input: User needs to fill up the from containing info like title, image if any, content etc.

output: Success or Failure.

description: This feature is used to add new post by the administrator.

process: If validation is successful, make changes to database otherwise handle appropriately.

postcondition: An alert is displayed based on success or failure. If success, post will be updated.

- Delete Post

input: User click.

output: Success or Failure.

process: If validation is successful, make changes to database otherwise handle appropriately.

postcondition: An alert is displayed based on success or failure. If success, post will be removed.

- Add another contributor

input: User needs to fill up info like name, email, password etc.

output: Success or Failure.

process: If validation is successful, make changes to database otherwise handle appropriately.

postcondition: An alert is displayed based on success or failure. If success, user will be added.

3.1.4 Logout

input: userclick.

output: Success or Failure.

process: Session is destroyed and user is redirected to index page.

precondition: User must be logged in the website.

3.2 Non-functional requirements

3.2.1 Performance Requirements

Performance should not be an issue because of all the data queries involves small pieces of data. Changing screen will require very little computation and thus will occur very quickly. Server updates could only take few seconds as long as device can maintain steady signal. Blog will also up for 24x7 except the maintenance.

3.2.2 Safety Requirements

Only the project admin will be having access to the database at the back-end. So, the admin will have rights for modifications as well as direct updates to the database.

3.2.3 Security Requirements

The server allows only authorized contributor the direct access to the internal structures. So, no naïve users or unauthorized person can't harm the project.

4. Design

4.1 XML, DTD, XSD, XSLT

4.1.1 XML

```
<posts>
  <post _id="1">
    <title>Post 1</title>
    <author>Sahil Bhuva</author>
    <date>31/10/2020</date>
    <imagePath>assets/img/desk.jpg</imagePath>
    <content>This is Post</content>
    <comments>
      <comment _id="1">
        <author>ABC</author>
        <content>XYZ</content>
        <date>31/10/2020</date>
      </comment>
    </comments>
  </post>
</posts>
-------------------------------------------------------------------------
<users>
  <user _id="1">
    <name>Sahil Bhuva</name>
    <email>bhuvasahil@gmail.com</email>
    <role>admin</role>
  </user>
</users>
```

4.1.2 DTD

```
<?xml encoding="UTF-8"?>
  <!ELEMENT posts (post)?>
  <!ELEMENT post (title,author,date,imagePath,content,comments)>
  <!ATTLIST post _id CDATA #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT imagePath (#PCDATA)>
  <!ELEMENT comments (comment)?>
  <!ELEMENT comment (author,content,date)>
  <!ATTLIST comment _id CDATA #REQUIRED>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT date (#PCDATA)>
  <!ELEMENT content (#PCDATA)>
-------------------------------------------------------------------
<?xml encoding="UTF-8"?>
  <!ELEMENT users (user)>
  <!ELEMENT user (name,email,role)>
  <!ATTLIST user _id CDATA #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
  <!ELEMENT role (#PCDATA)>
```

4.1.3 XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="posts">
    <xs:complexType>
```

```xml
      <xs:sequence>
        <xs:element ref="post" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="post">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="author"/>
        <xs:element ref="date"/>
        <xs:element ref="imagePath"/>
        <xs:element ref="content"/>
        <xs:element ref="comments"/>
      </xs:sequence>
      <xs:attribute name="_id" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="title" type="xs:string"/>
  <xs:element name="imagePath" type="xs:string"/>
  <xs:element name="comments">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="comment" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="comment">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="author"/>
        <xs:element ref="content"/>
        <xs:element ref="date"/>
      </xs:sequence>
      <xs:attribute name="_id" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="author" type="xs:string"/>
  <xs:element name="date" type="xs:date"/>
  <xs:element name="content" type="xs:string"/>
</xs:schema>
-------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="users">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="user"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="user">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="email">
          <xs:simpleType>
            <xs:restriction base="xs:string">
```

```
              <xs:pattern value="[a-z0-9._%+-]+@[a-z0-9.-]+.[a-
z]{2,3}$" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element ref="role"/>
      </xs:sequence>
      <xs:attribute name="_id" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="email" type="xs:string"/>
  <xs:element name="role">
    <xs:complexType>
      <xs:choice>
        <xs:element name="admin" type="xs:string"/>
        <xs:element name="role" type="xs:string"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.1.4 XSLT

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
        <body>
            <h2>Posts</h2>
            <table border="1">
            <tr>
                <th>Title</th>
                <th>Author</th>
                <th>Date</th>
                <th>ImagePath</th>
                <th>Content</th>
                <th>Comments</th>
            </tr>
            <xsl:for-each select="posts/post">
                <tr>
                    <td><xsl:value-of select="title"/></td>
                    <td><xsl:value-of select="author"/></td>
                    <td><xsl:value-of select="date"/></td>
                    <td><xsl:value-of select="imagePath"/></td>
                    <td><xsl:value-of select="content"/></td>
                    <td>
                    <table>
                        <xsl:for-each select="comments/comment">
                        <tr>
                            <td><xsl:value-of select="author"/></td>
                            <td><xsl:value-of select="content"/></td>
                            <td><xsl:value-of select="date"/></td>
                        </tr>
                        </xsl:for-each>
                    </table>
                    </td>
                </tr>
            </xsl:for-each>
            </table>
```

```
      </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
-------------------------------------------------------------------
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Users</h2>
        <table border="1">
          <tr>
            <th>Name</th>
            <th>Email</th>
            <th>Role</th>
          </tr>
          <xsl:for-each select="users/user">
            <tr>
              <td><xsl:value-of select="name"/></td>
              <td><xsl:value-of select="email"/></td>
              <td><xsl:value-of select="role"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

4.2 DFD Diagram

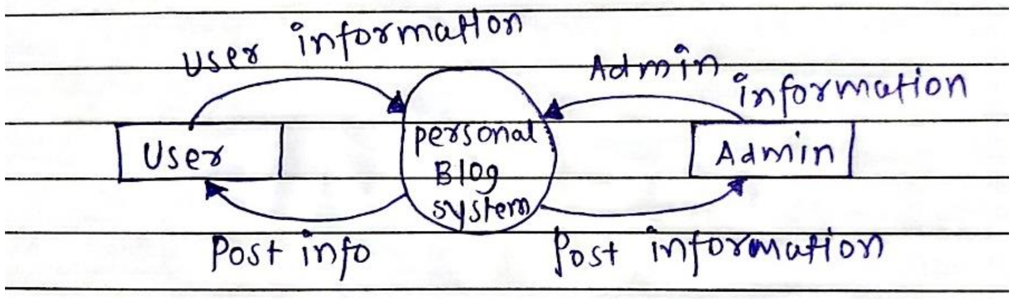


Figure 4.2.1 Context 0 Diagram

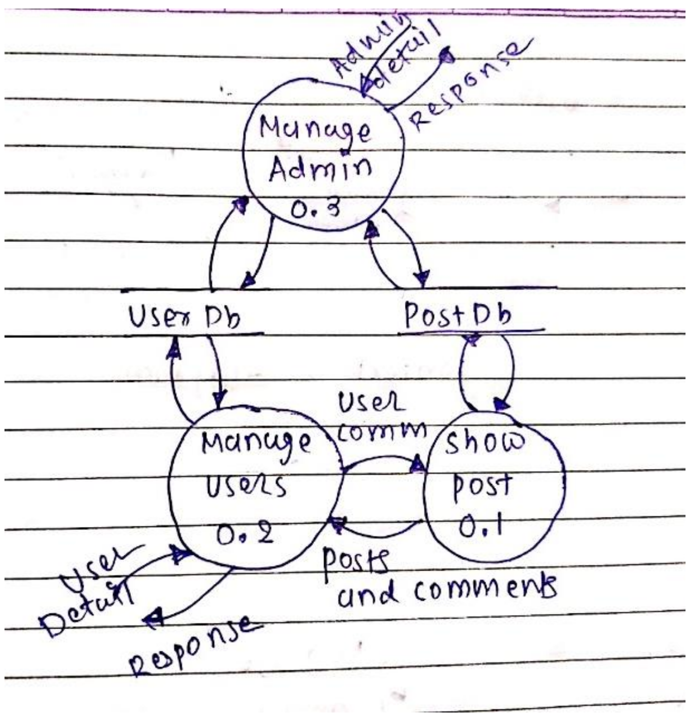


Figure 4.2.2 Level 1 Diagram

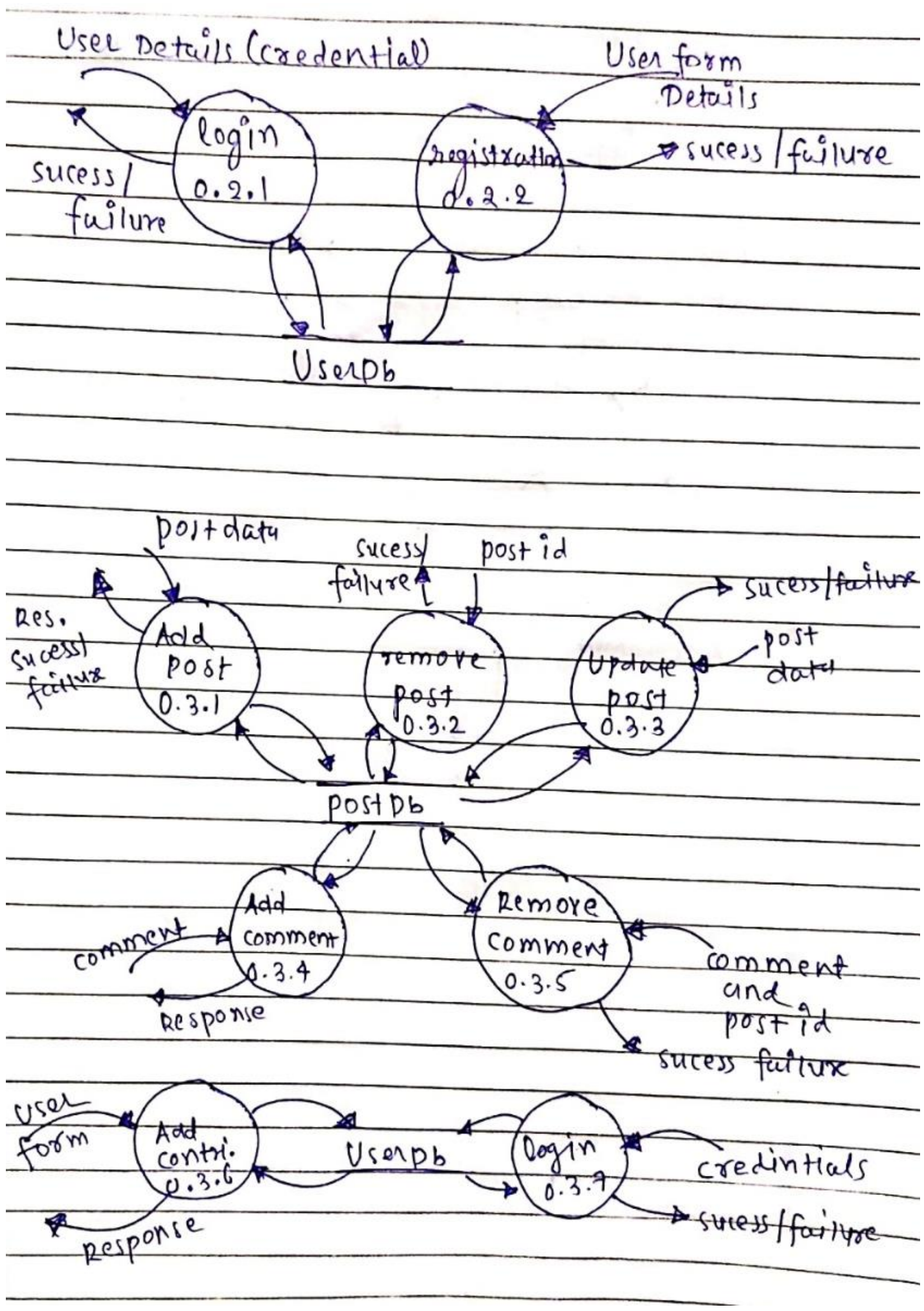
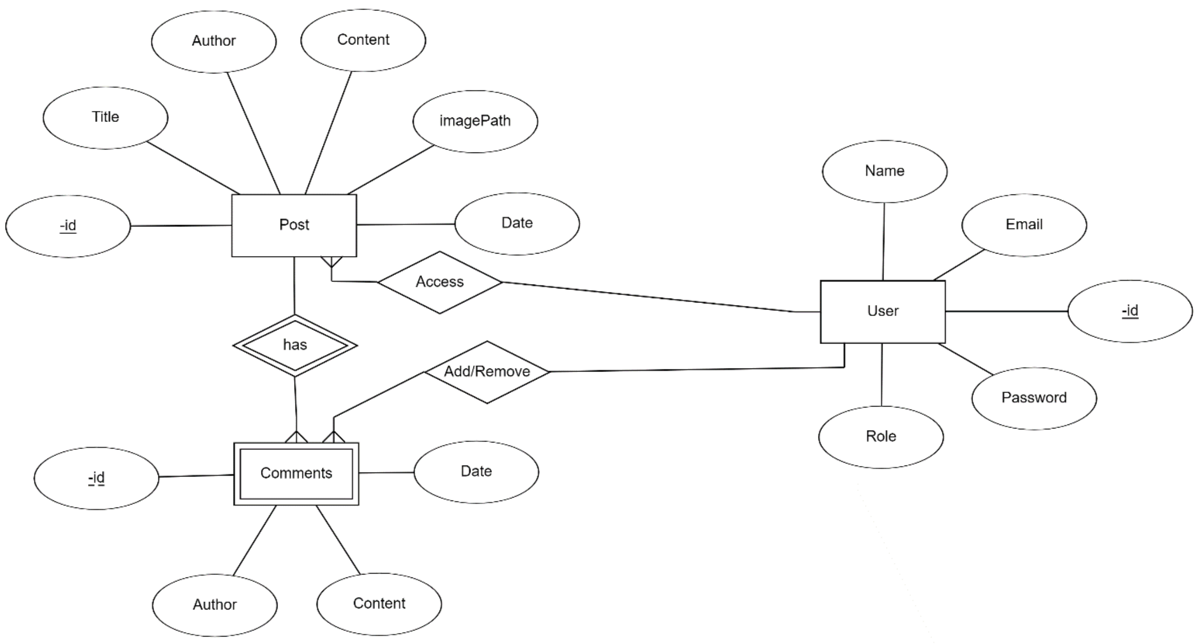


Figure 4.2.3 Level 2 Diagram

4.3 ER Diagram



4.4 Data Dictionary

1.User								
Sr. No.	**Field Name**	**Data Type**	**Width**	**Required**	**Unique**	**PK/FK**	**Referenced table**	**Description**
1.	_id	Varchar	10	Yes	Yes	PK		Auto increment key generated by database itself
2.	Email	Varchar	10	Yes	Yes			Unique id defined by user to login
3.	Password	Varchar	128	Yes	No			
4.	Name	Varchar	50	Yes	No			
5.	Role	Varchar	30	Yes	No			

2. Post								
Sr. No.	**Field Name**	**Data Type**	**Width**	**Required**	**Unique**	**PK/FK**	**Referenced table**	**Description**
1.	_id	Varchar	10	Yes	Yes	PK	Work as strong entity for Comment table	Auto increment key generated by database itself
2.	Author	Varchar	10	Yes	No			
3.	Date	DateTime	-	Yes	No			
4.	Content	Varchar	500	Yes	No			
5.	imagePath	Varchar	30	Yes	No			
6.	Title	Varchar	50	Yes	No			

3. Comment								
Sr. No.	**Field Name**	**Data Type**	**Width**	**Required**	**Unique**	**PK/FK**	**Referenced table**	**Description**
1.	_id	Varchar	10	Yes	No		Composite key with _id field od of Post table.	Auto increment key generated by database itself
2.	Author	Varchar	10	Yes	No			
3.	Date	DateTime	-	Yes	No			
4.	Content	Varchar	500	Yes	No			

5. Implementation Details

5.1 Module

- ➢ Login Module

  This module takes users credentials and verifies it with registered users, if user is not registered then error message is shown else if they match then login user.

- ➢ Registration Module

  This module is used to store user’s data to the database and enables the user to login to the system. All the field in this module is required.

- ➢ Client Post Module

  This module is used to get the posts and comments associated with each post. This module is basically providing basic feature of the site for showing the post and comment to users visiting the site.

- ➢ Admin Post Module

  This module is extension of client post module. This module is accessed only by admin users. This is the module which is used to provide feature of post creation, updating, deletion etc.

5.2 Function Prototypes

List given below are the function prototype for the major functionality. Prototype are in the reference to JavaScript/ TypeScript Language.

- Post Modules

```
public getPost(): Observable<IPost[]>
public getPostById(id): Observable<IPost>
public addPost(data): Observable<IPost>
public addComment(data): Observable<IPost>
public deletePost(id): Observable<IPost>
public deleteComment(postid, commentid): Observable<IPost>
public updatePost(data): Observable<any>
public uploadImage(data): Observable<any>
```

- Accounts Modules

```
public login(data): Observable<ILogin>
public create(data): Observable<ILogin>
public delete(id): Observable<ILogin>
```

- Post Backend

```
//GET: localhost:4000/post/get
postRoute.route('/get').get((req, res) => {})

//GET: localhost:4000/post/get/:id
postRoute.route('/get/:id').get((req, res) => {})
```

```
//POST: localhost:4000/post/add/post
postRoute.route('/add/post').post((req, res) => {})

//POST: localhost:4000/post/add/comment
postRoute.route('/add/comment').post((req, res) => {})

//DELETE: localhost:4000/post/delete/:id
postRoute.route('/delete/:id').delete((req, res) => {})

//DELETE:
localhost:4000/post/delete/comment/:postid/:commentid
postRoute.route('/delete/comment/:postid/:commentid').delet
e((req, res) => {});

//PUT: localhost:4000/post/update/:id
postRoute.route('/update/:id').put((req, res) => {})
```

- Login Backend

```
//POST: localhost:4000/account/login
accountRoute.route('/login').post((req, res) => {});

//POST: localhost:4000/account/create
accountRoute.route('/create').post((req, res) => {});

//DELETE: localhost:4000/account/delete/:id
accountRoute.route('/delete/:id').delete((req, res) => {})
```

6. Testing

Testing for backend is done using Postman and for frontend manual testing is done.

7. Screenshots

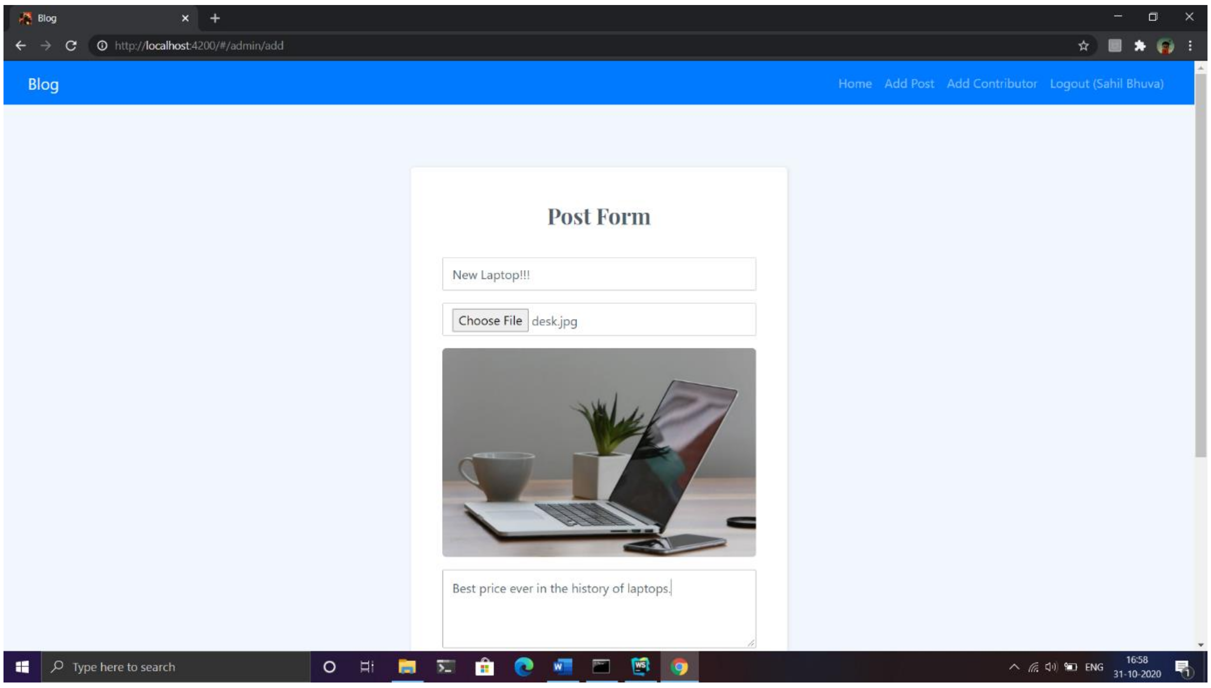


Figure 6.1 Admin Add Post Feature

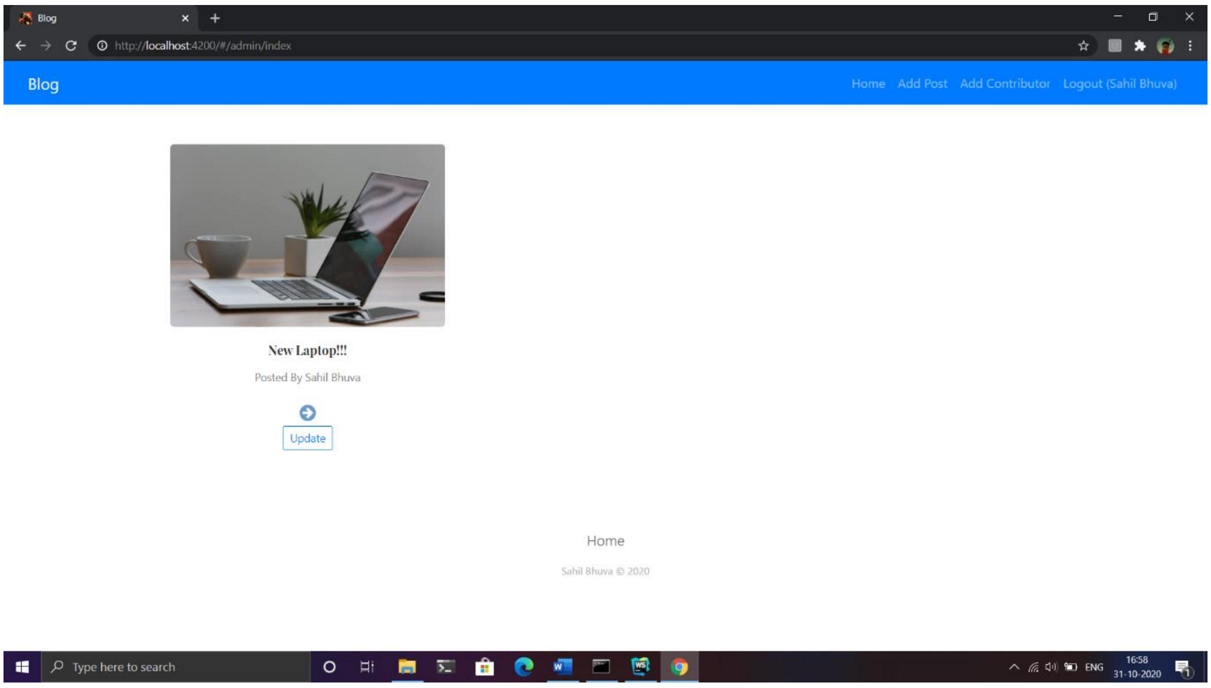


Figure 6.2 Admin index page with update button

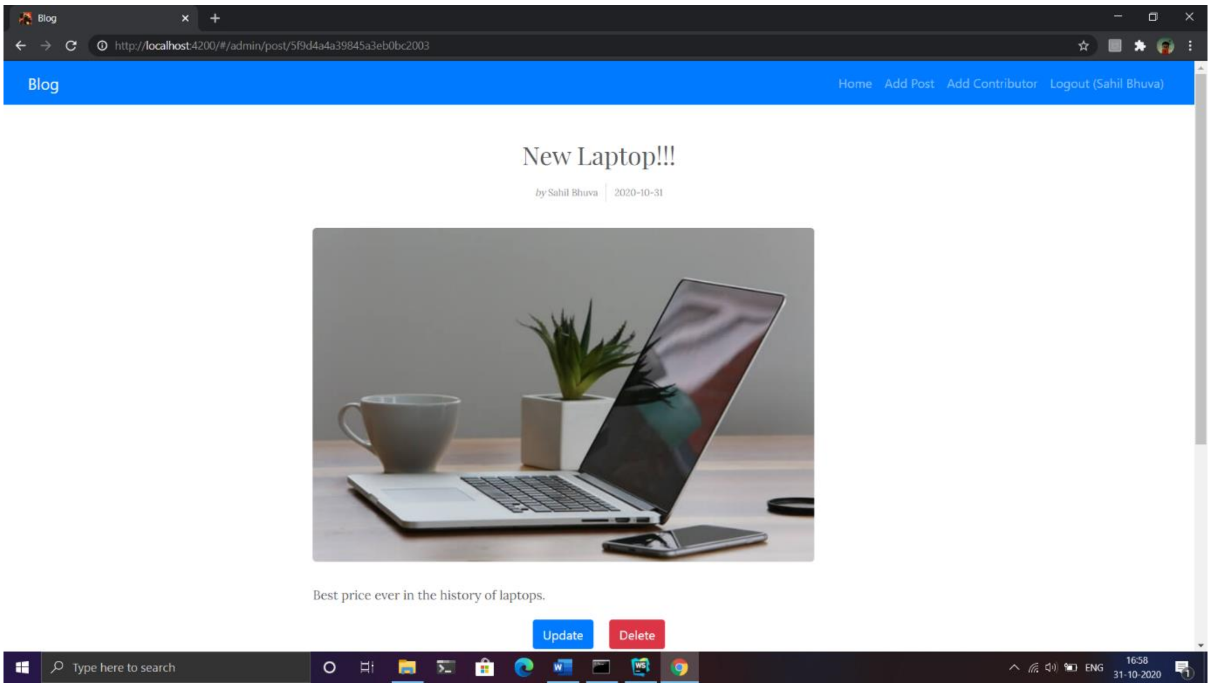


Figure 6.3 Post specific page with content and update delete button

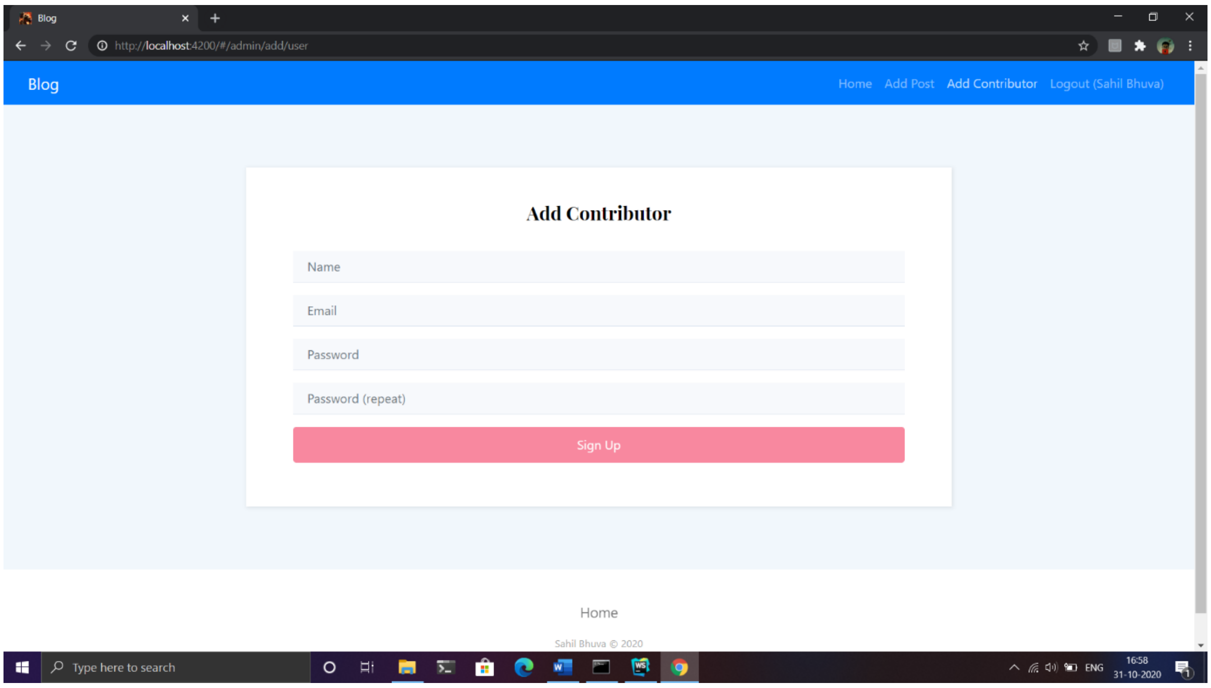


Figure 6.1 To add another contributor

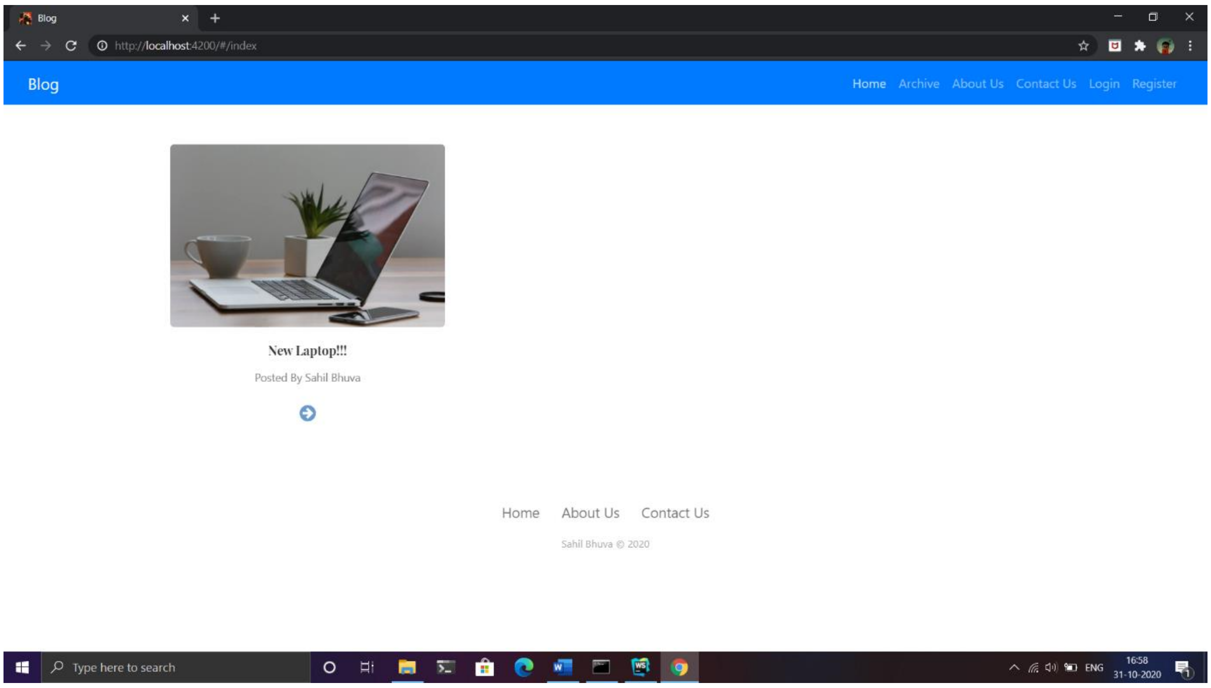


Figure 6.2 Home page to the user who is not logged in

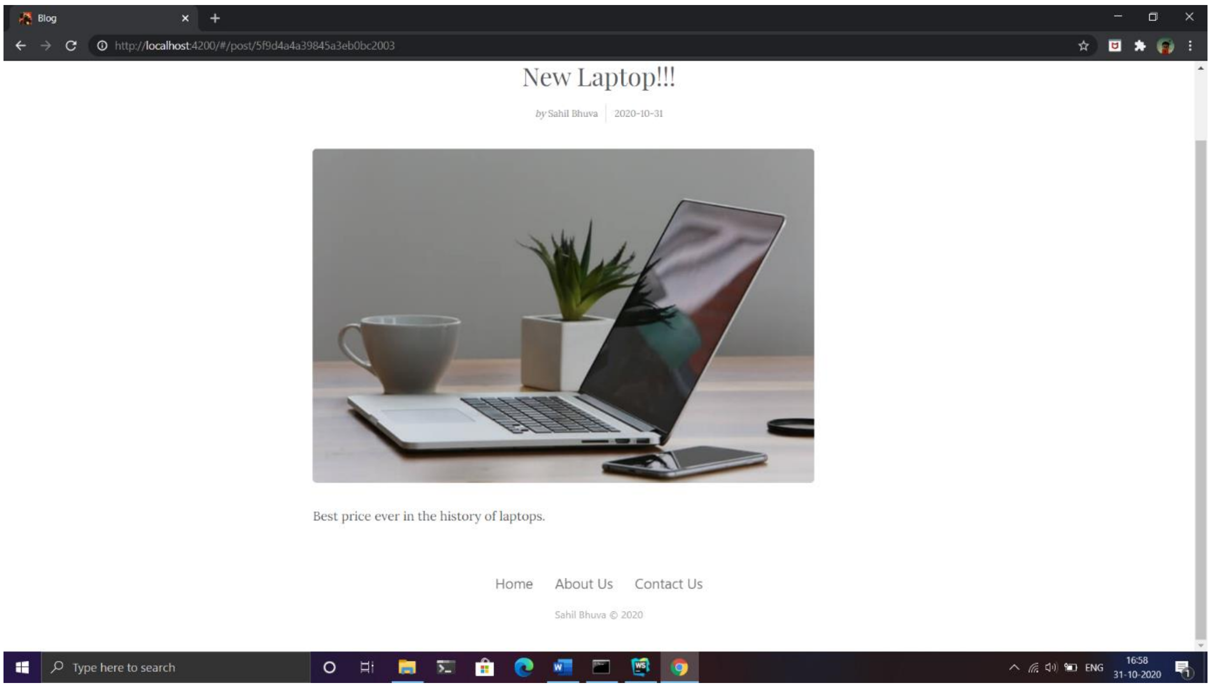
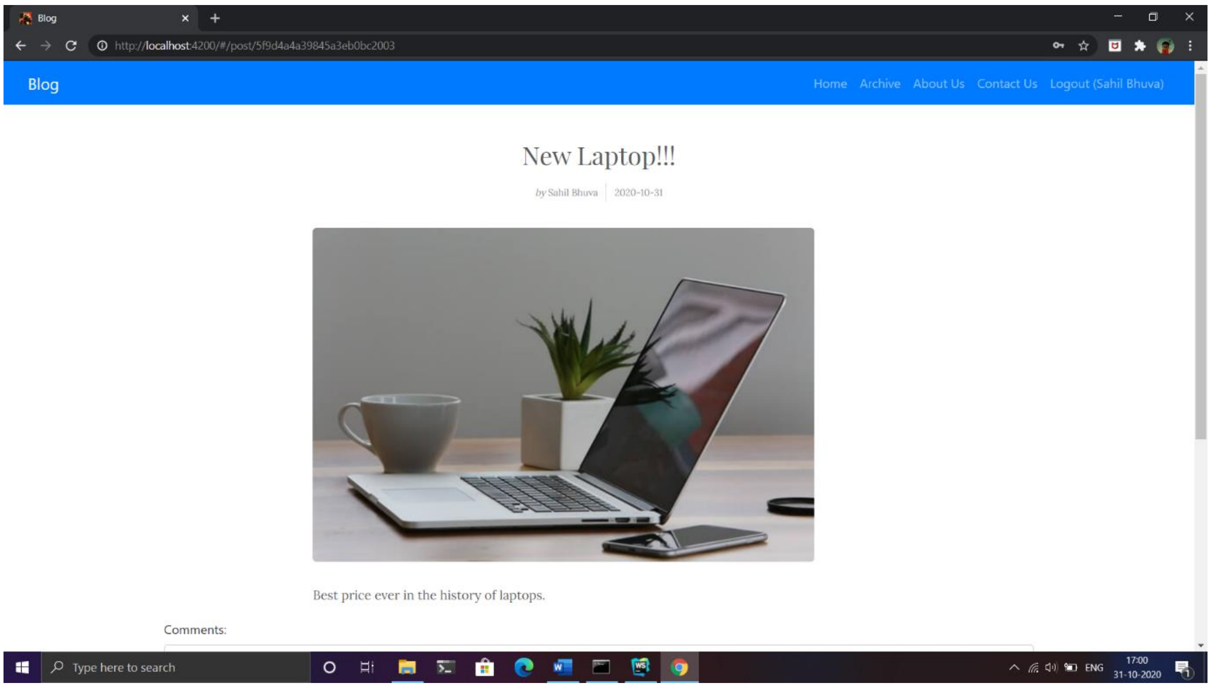


Figure 6.3 Post page for the user who is not logged in


Blog
http://localhost:4200/#/post/5f9d4a4a39845a3eb0bc2003
Blog
Home Archive About Us Contact Us Logout (Sahil Bhuva)
Type here to search
ENG
17:00
31-10-2020

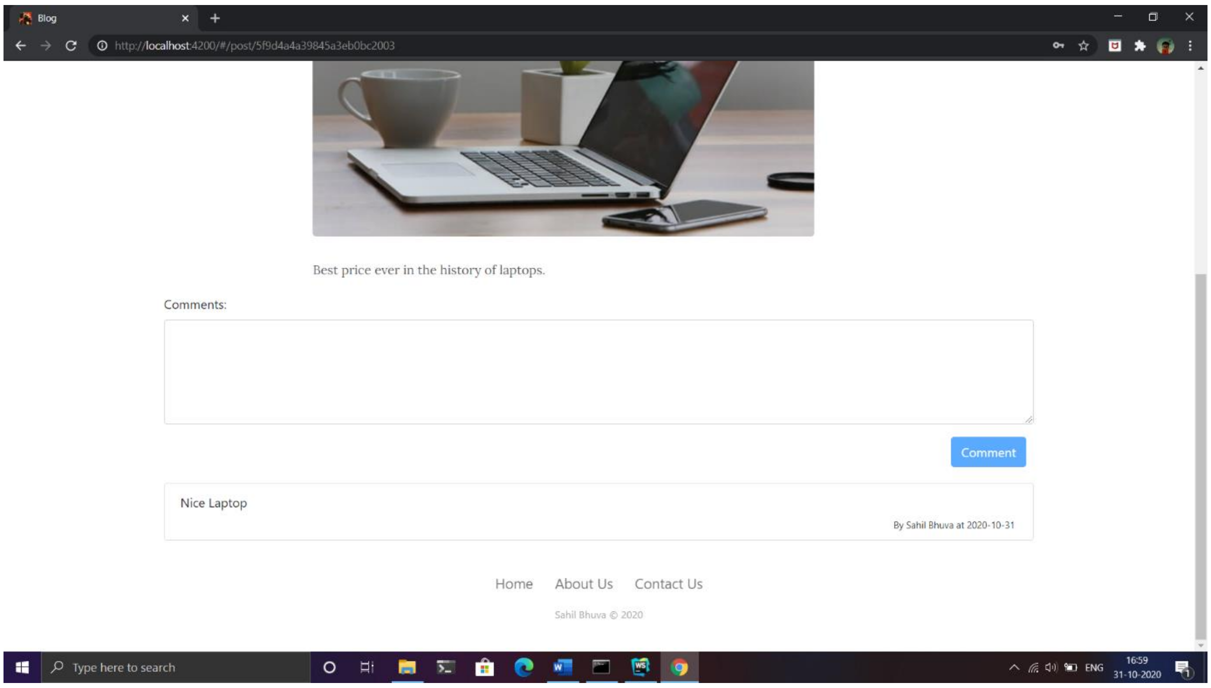
New Laptop!!!

by Sahil Bhuva | 2020-10-31

Best price ever in the history of laptops.

Comments:

Figure 6.4 User with client role



Best price ever in the history of laptops.

Comments:

Nice Laptop

By Sahil Bhuva at 2020-10-31

Sahil Bhuva © 2020

Figure 6.5 Once logged in comment feature is enabled

These are major functionalities but not the least.

8. Conclusion

The functionalities implemented in system after understanding all the system modules according to the requirements. Functionalities That are successfully implemented in the system are:

- Creation of the post
- Deletion of the post
- Updating of the post
- Login
- Registration
- Adding Comment
- Deleting Comment
- Adding another admin user
- Session management with guards to manage unauthorized access
- Uploading image of the post with multer

9. Limitation and Future expansion

- Limitation
    - The password hashing mechanism is not used.
    - Deletion and updating of the post only by author are not implemented.
    - Updation of comment is not possible at the moment.
- Future expansion
    - New backend APIs can be added.
    - More attractive UI.
    - More access control over the posts and comments.

10. Bibliography

The site developed from idea of https://blog.google. During the development of the site I have taken references from these sites:

- https://www.google.co.in
- http://www.wikipedia.org
- http://www.getbootstrap.com
- http://www.w3schools.com
- http://www.tutorialspoint.com
- http://www.angular.io
- http://www.stackoverflow.com