LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CS6004NI Application Development

Assessment Weightage & Type

30% Group Coursework

Name	London Met ID	College ID
Arbaaz Alam	22015655	NP01CP4S220108
Bimarsha Poudel	22015671	NP01CP4S220090
Prabin Thakur	22015722	NP01CP4S220125
Prashant Baral	22015729	NP01CP4S220120
Ayush Krishna Shrestha	22015662	NP01CP4S220115

Semester

2023-24 Autumn

Assignment Due Date: 10th May 2024

Assignment Submission Date: 10th May 2024

Word Count (Where Required):

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Acknowledgements

We sincerely thank our academic tutor, Mr. Bishal GC, for his persistent support and assistance during our coursework. We have benefited much from his extensive experience, unwavering commitment to our education, first-rate learning materials, and insightful guidance on the module topics over this journey. His insightful comments and criticism have motivated us to broaden our knowledge and expertise. His enthusiasm and support have inspired us to always strive for greatness and put our all into whatever we do. We are truly appreciative of your support, since it has been crucial in helping us meet our academic goals.

We also like to thank everyone who has helped with our coursework, both directly and indirectly.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Bislerium PVT. LTD. is a technology company aiming to enhance its social media platform with a custom blogging web application. A technological business called Bislerium PVT. LTD. wants to improve its social media platform by adding a unique web application for blogging. The application will provide tools for platform management, interaction, and content production, targeting bloggers, administrators, and surfers. Our development team is tasked with fulfilling the specified objectives, integrating enterprise-level frameworks to ensure scalability and security, with an emphasis on user experience and functionality. This report details our efforts to satisfy Bislerium's goals and provide a customized solution that will improve their online visibility.

2. User Manual to run the program.

2.1. Register User and Login

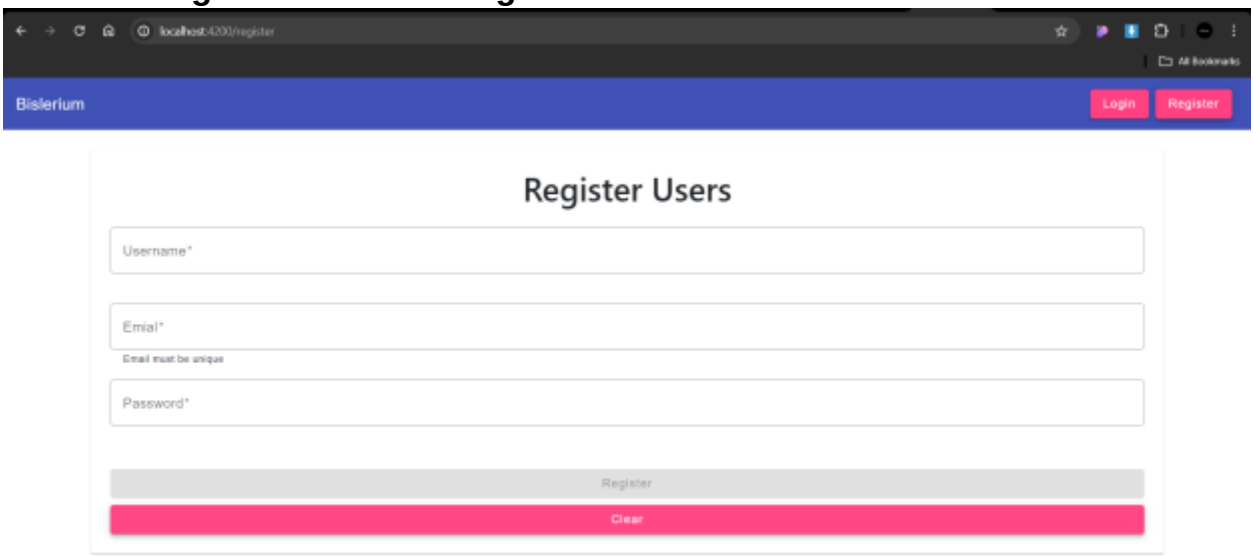


Figure 1: Register User page

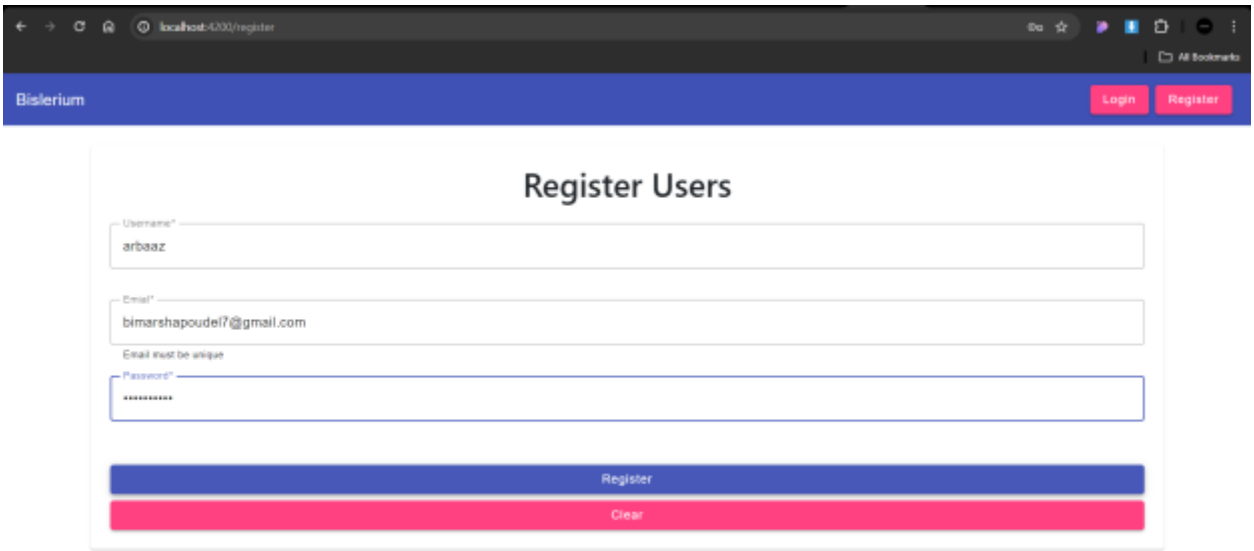


Figure 2: Add details to register user page

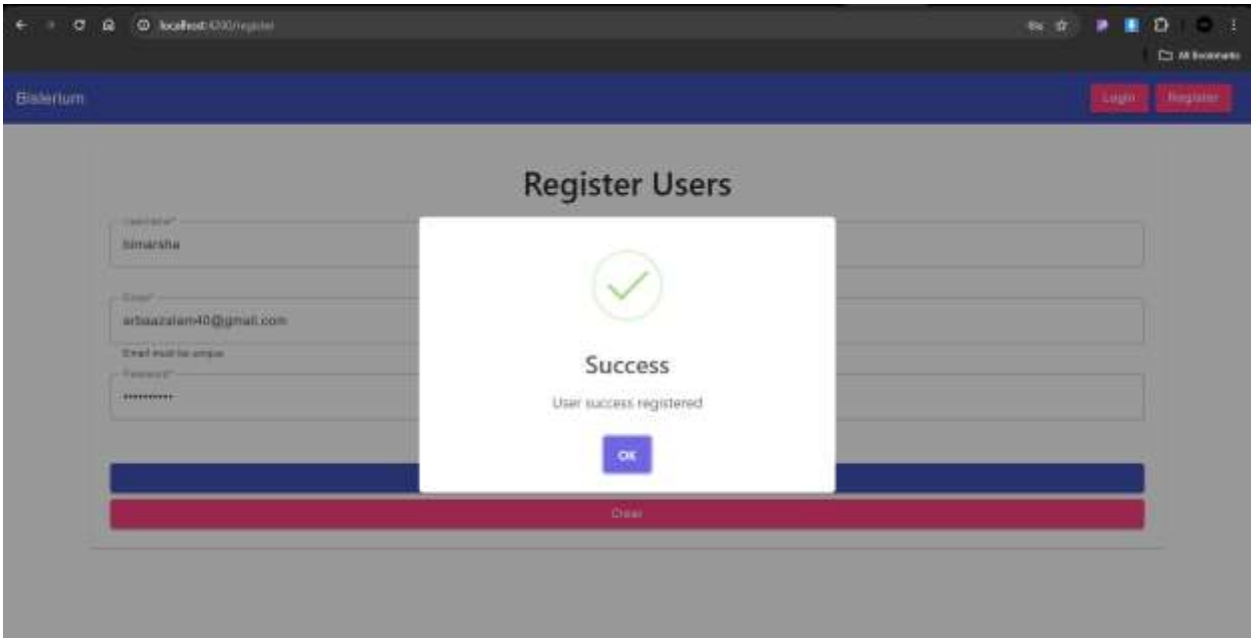


Figure 3: User registered successfully

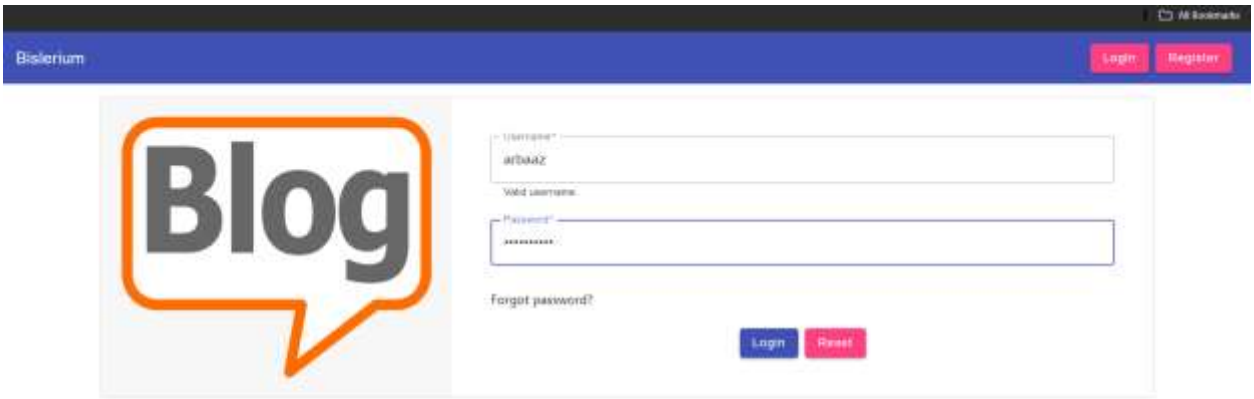


Figure 4: User login page

At first, the user is registered using a username, email address and password.

2.2. Post Blogs

Figure 5: Add Blog page

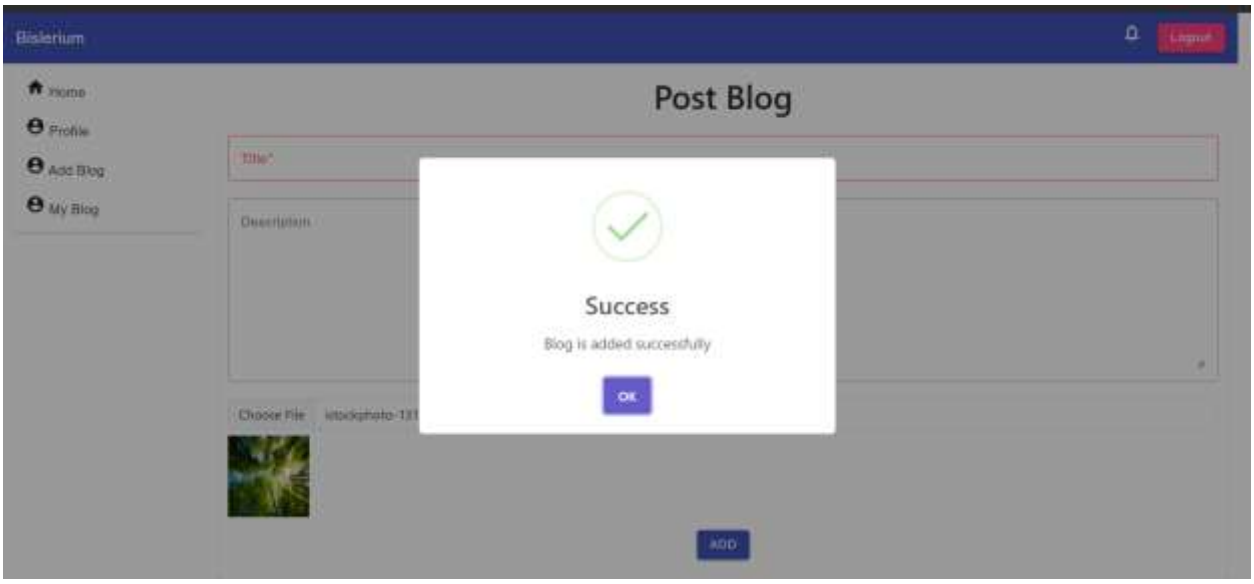


Figure 6: Blog added successfully

Figure 7:Blog added successfully

This is the add blog page. Here, user can insert the title, description, and an image for the blog. After clicking add, the blog is added successfully. This added blog can be seen in “My blog” page as well as the home page as shown below.

2.3. Home Page on basis of recency, popularity, all blogs with pagination

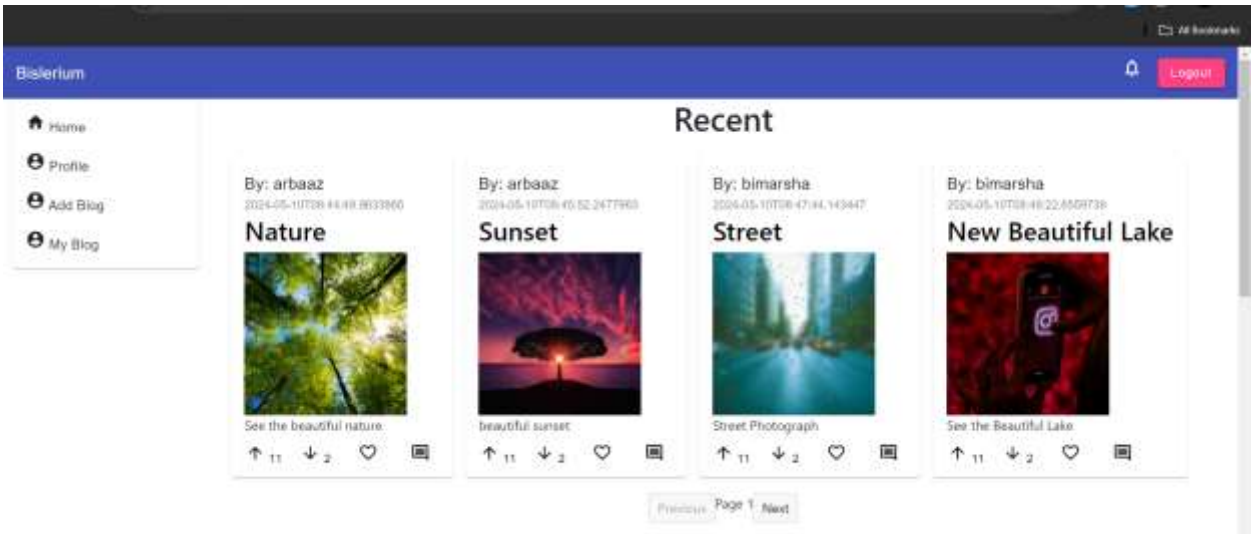


Figure 8: All blogs on basis of recency

Figure 9: All blogs

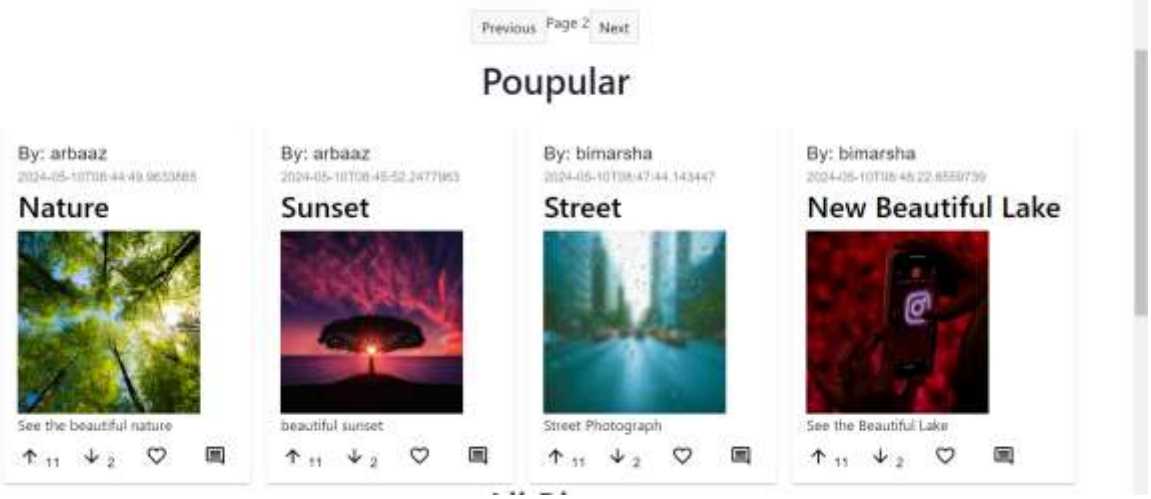


Figure 10: All blogs on basis of popularity

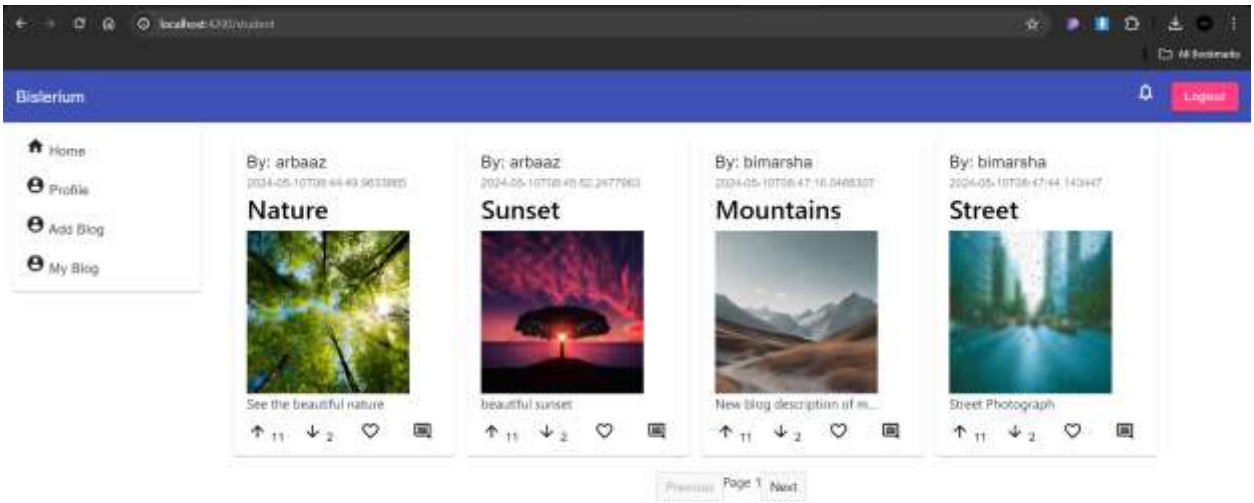


Figure 11: Home page with all blogs including pagination

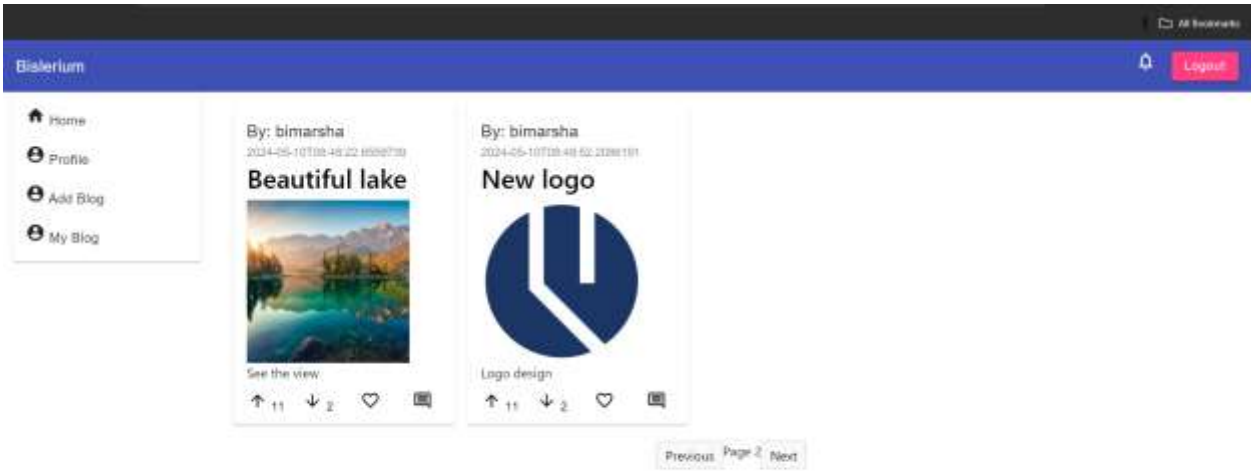


Figure 12: Page changed

2.4. Upvote and Downvote

Each blog has upvote, and downvote option.

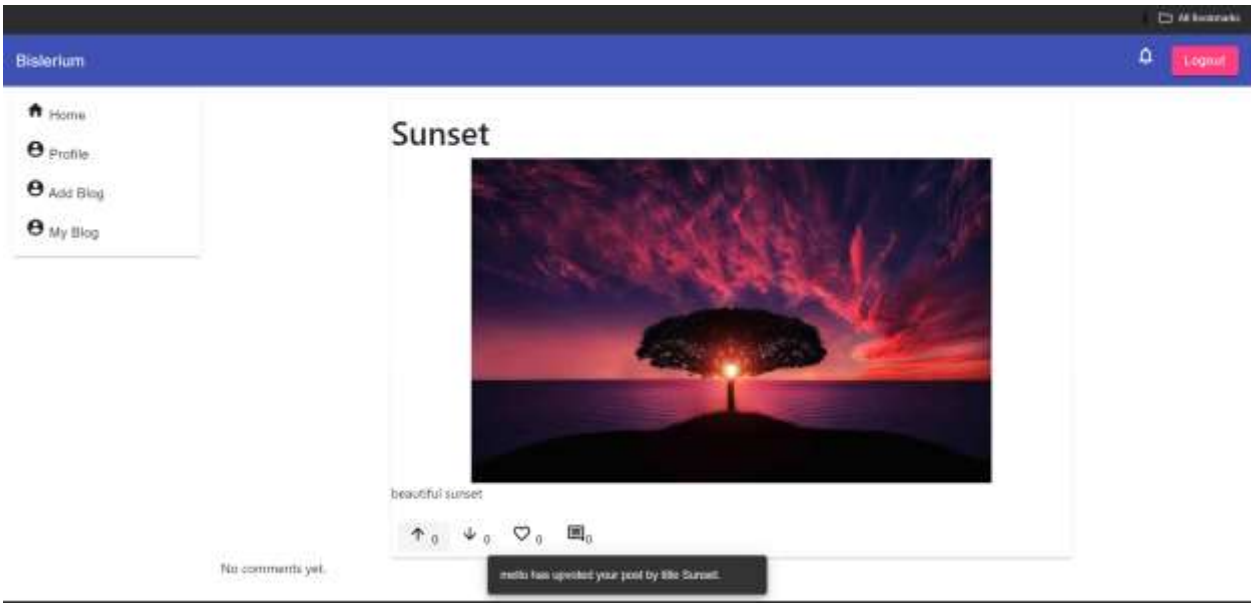


Figure 13: Upvoted notification

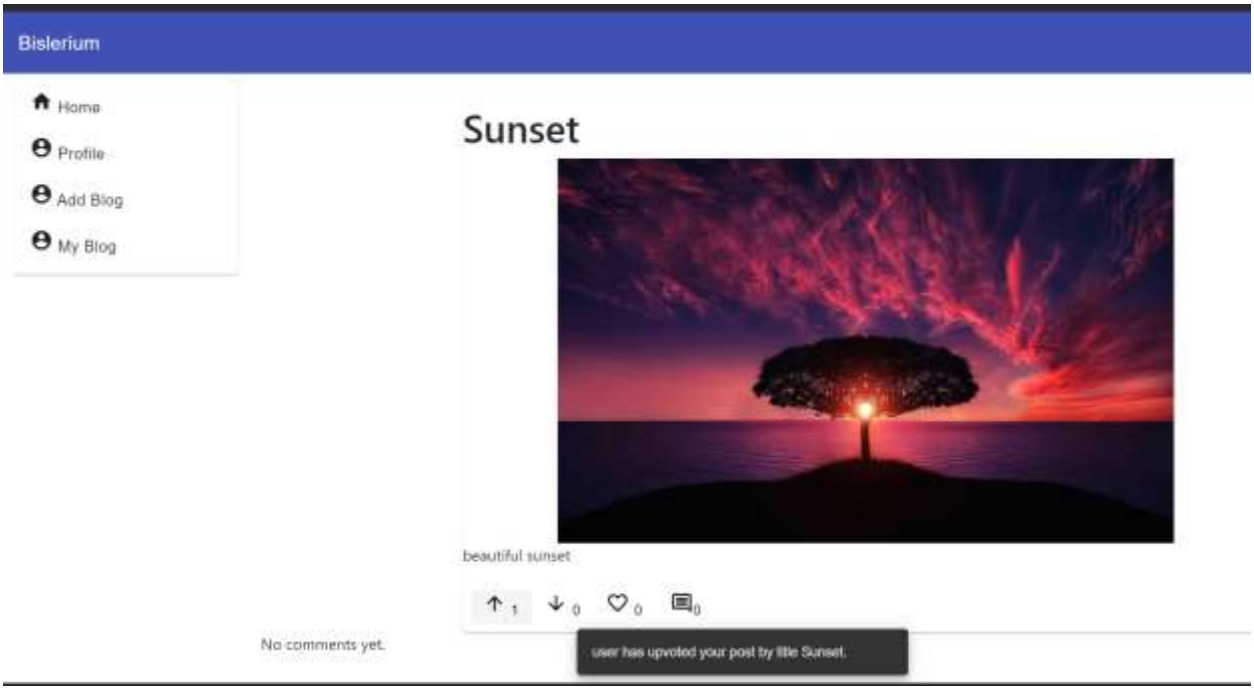


Figure 14:Upvote count increased

Here, using SingleR, the upvote notification is shown. Then the upvote count is increased.

Figure 15:Before downvote

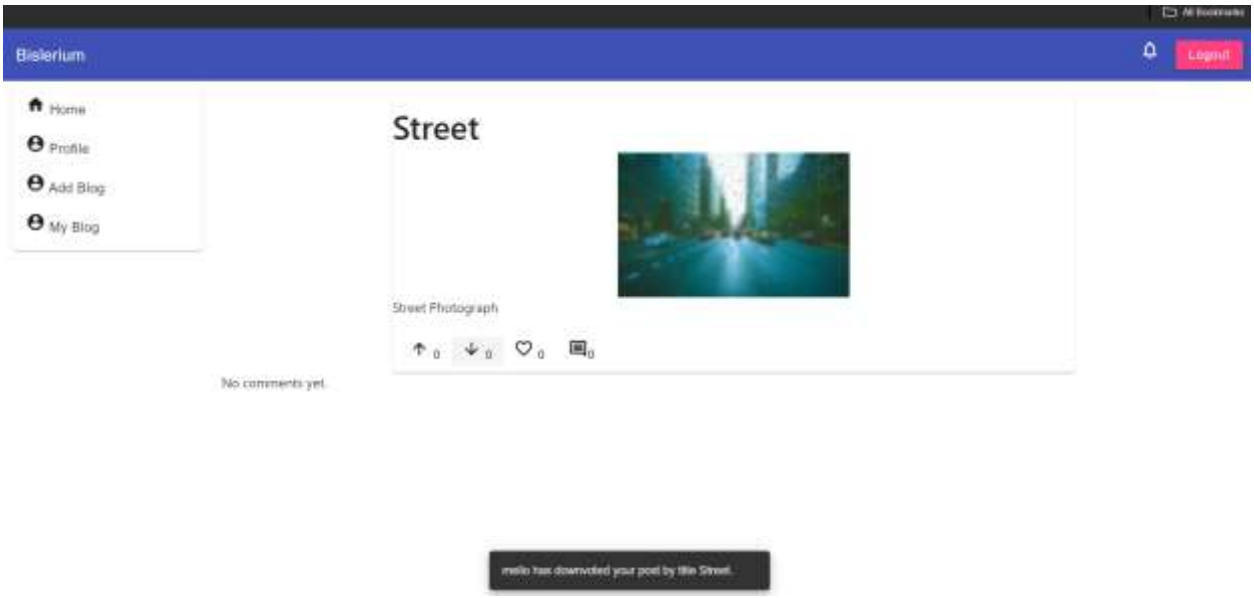


Figure 16: Downvoted notification

Figure 17: Downvote count increased

Similar to upvote, the user also gets notifications for the downvote. After that, the downvote count also increases.

2.5. Comment and reply

Users are able to comment and reply to other comments on blogs as well. Moreover, the comments and replies can be upvoted, and downvoted. The comment can also be deleted. All the activities done throughout is shown in notifications. This is done using SignalR.

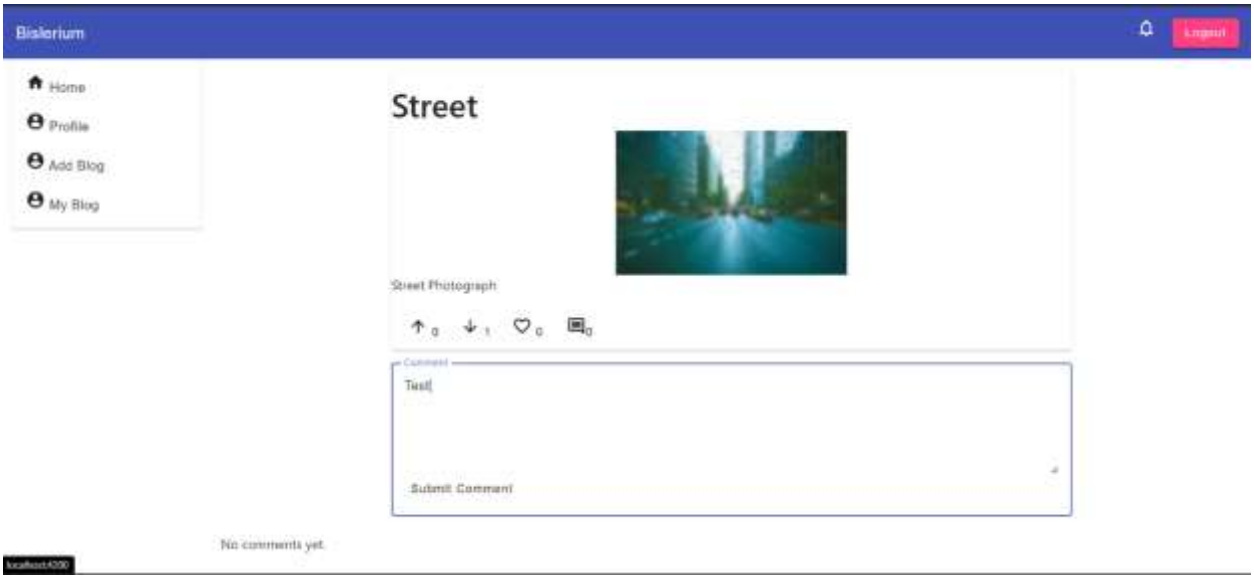


Figure 18: Filling out the comment form

Figure 19: Comment posted

The comment is posted and visible on the blog.

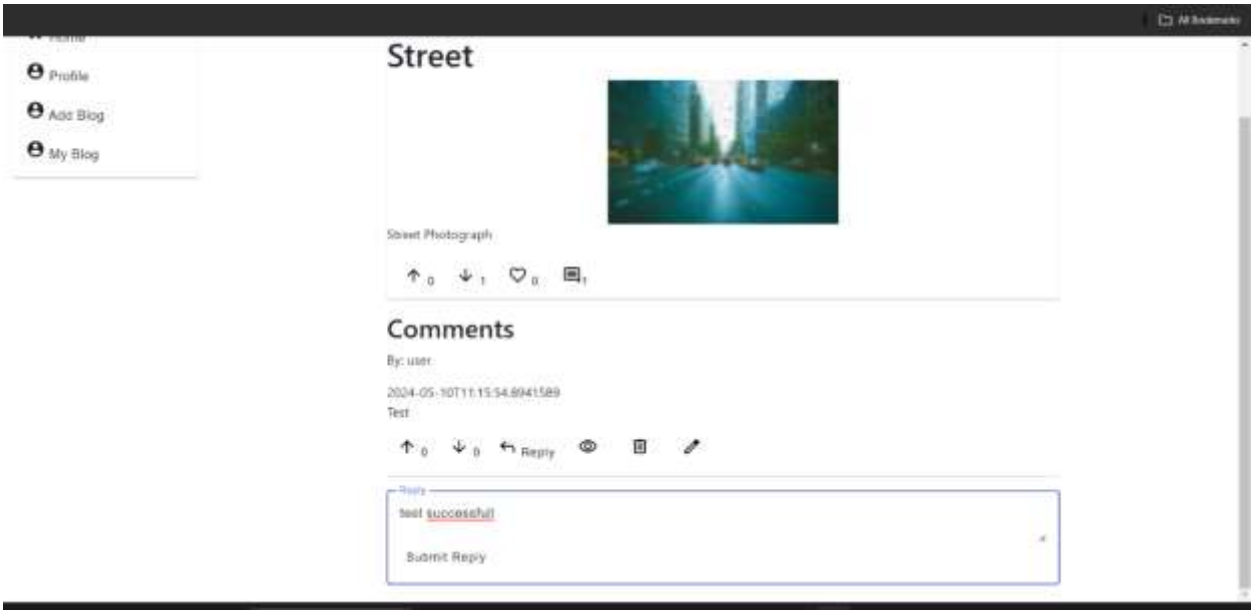


Figure 20: Reply to the comment is filled

Figure 21: Comment reply is posted

Here, each of the comment can also be replied to. When a reply is added, it is posted on the blog below the comment it is replied to.

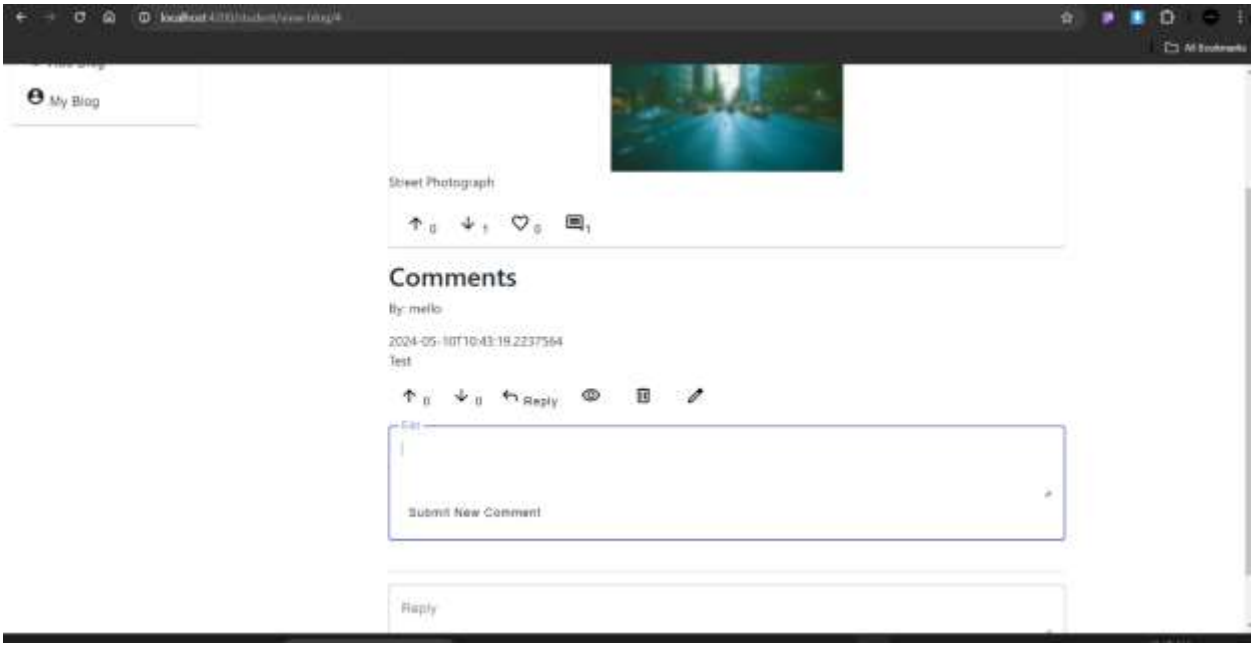


Figure 22: Editing the "test" comment

The comment can also be edited.

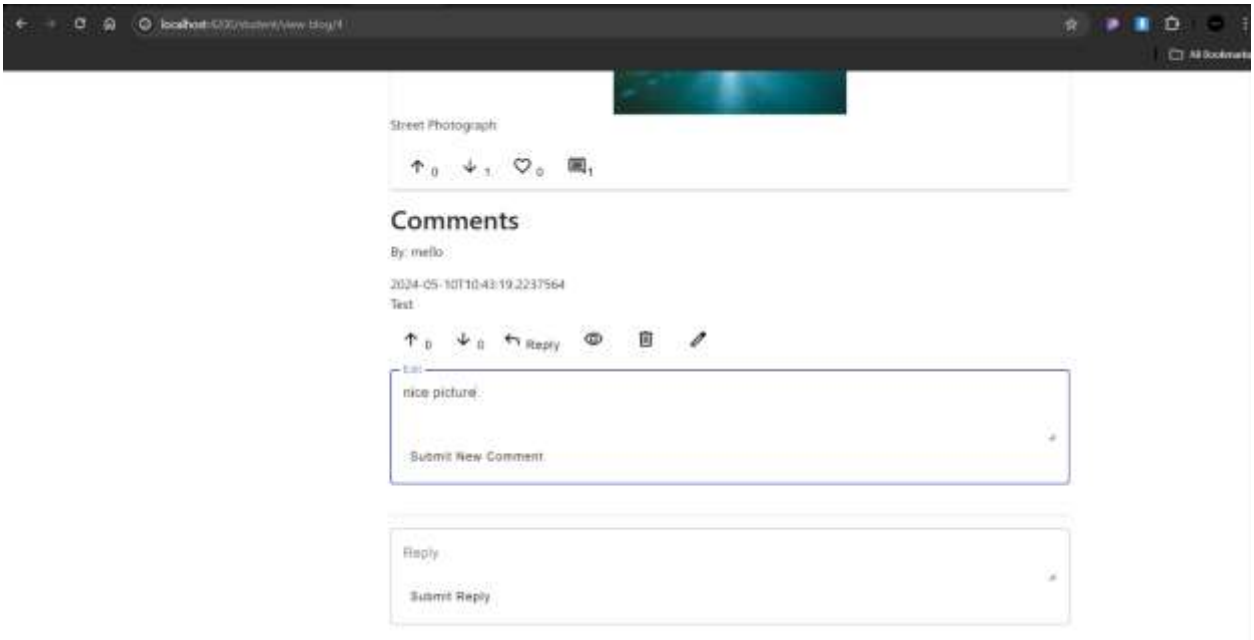


Figure 23:Changing content of comment

The new comment is passed.

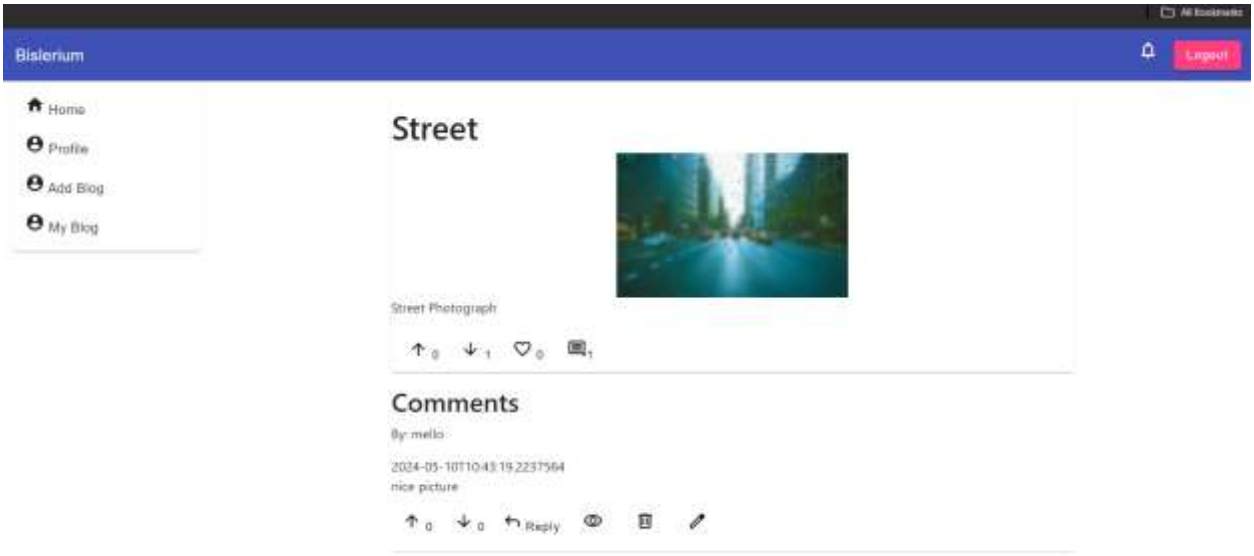


Figure 24: Comment edited successfully

The comment is edited.

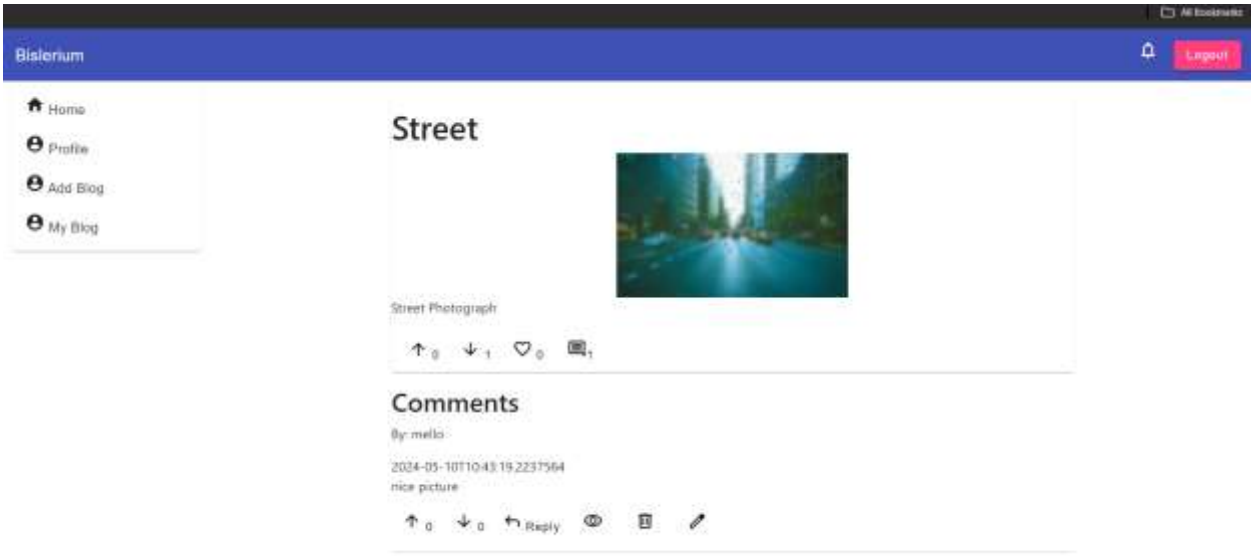


Figure 25: Before upvoting the comment

The picture above shows the comment before upvoting.

Figure 26: Comment after upvoting

The comment has been upvoted and the upvote count is increased.

Also, see that there is 0 downvotes on this comment.

Figure 27: Downvoted comment

After clicking the downvote option, the downvote count is increased.

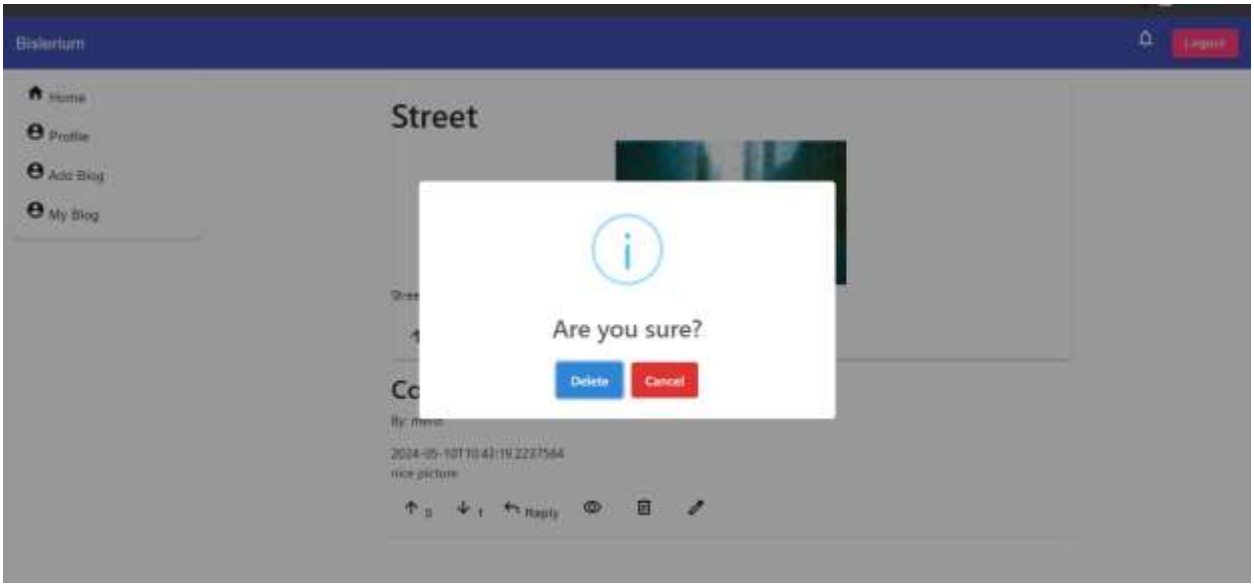


Figure 28: Confirmation to delete comment

The delete button is clicked on the comment.

Figure 29: Comment deleted

After confirmation, the comment is deleted from the blog as shown above.

Figure 30: SignalR notification

All the activities on the blog with comments, upvotes, downvotes are shown to the user in the notification portion. This is done using SignalR.

2.6. My blog

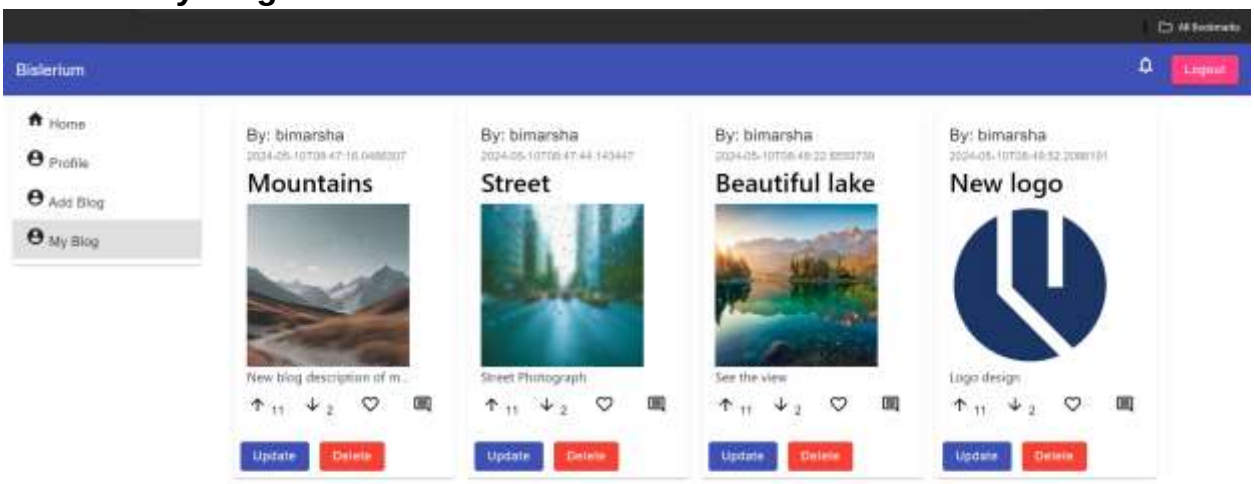


Figure 31: My blogs page with each blog update and delete option

This is how the “My blog” page looks like.

2.6.1. Delete blog

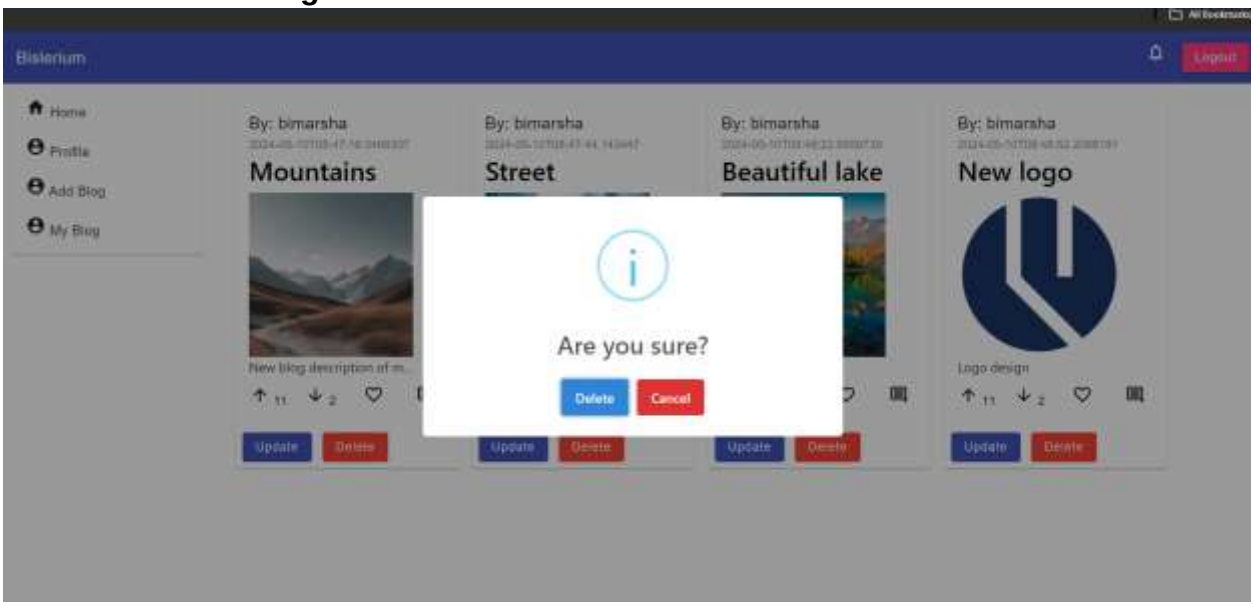


Figure 32: Delete Blog

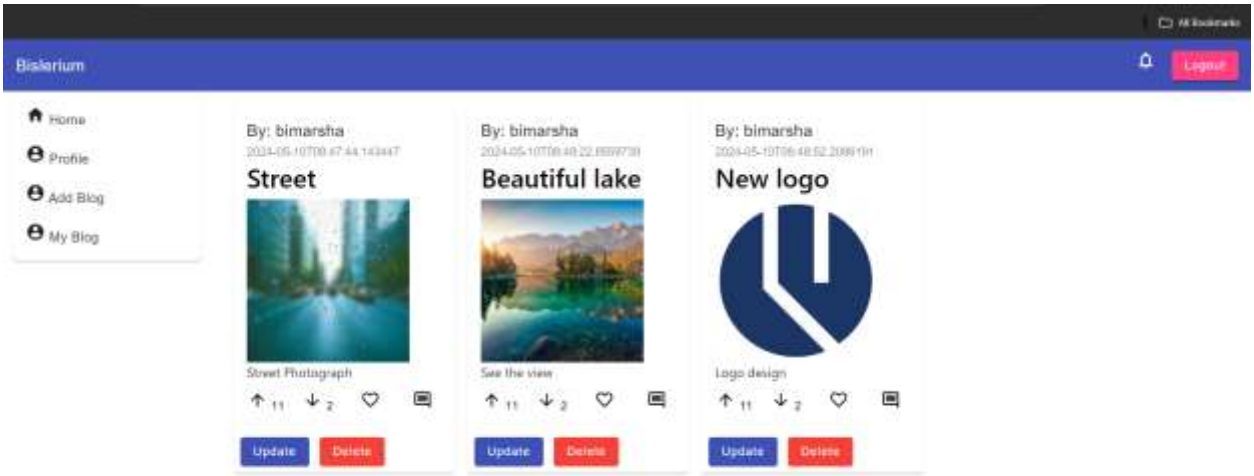


Figure 33: Blog deleted

As shown in pictures above, user can delete their own blogs using the delete button in my blogs page.

2.6.2. Update

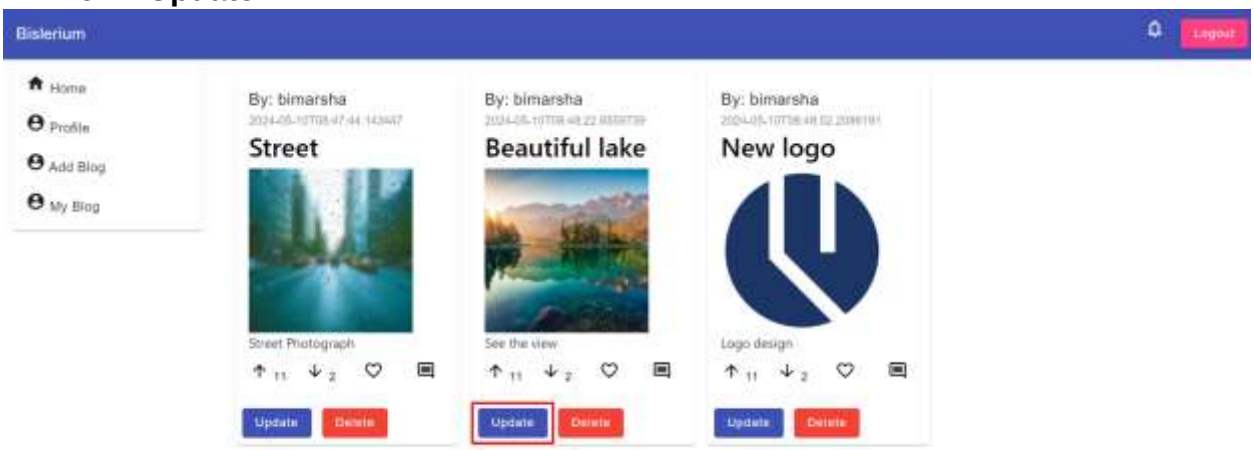
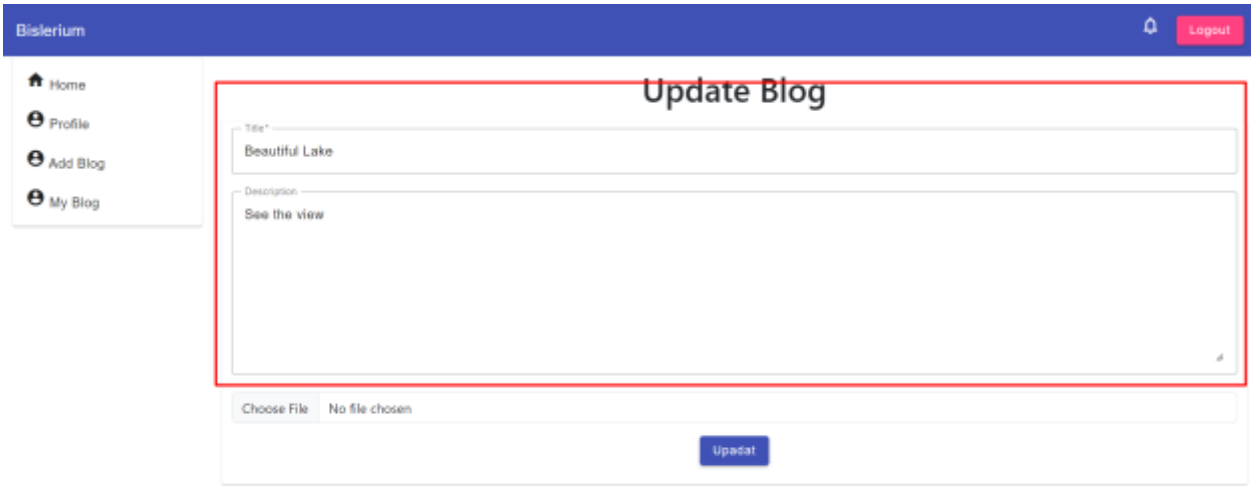


Figure 34: Update blog



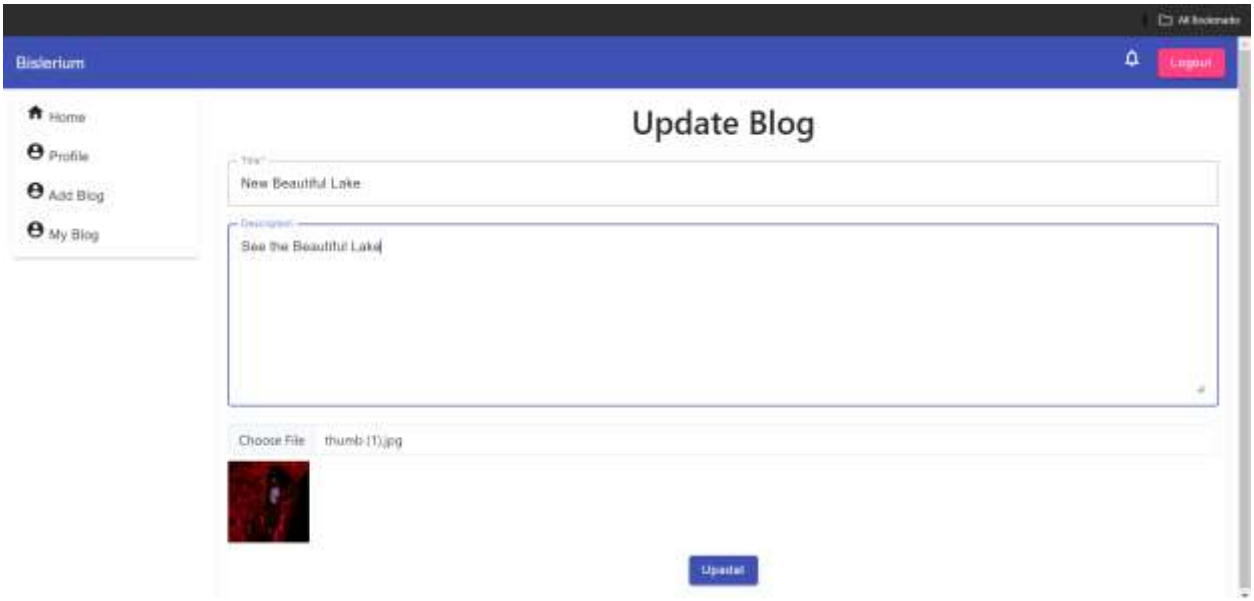


Figure 35: Update blog form

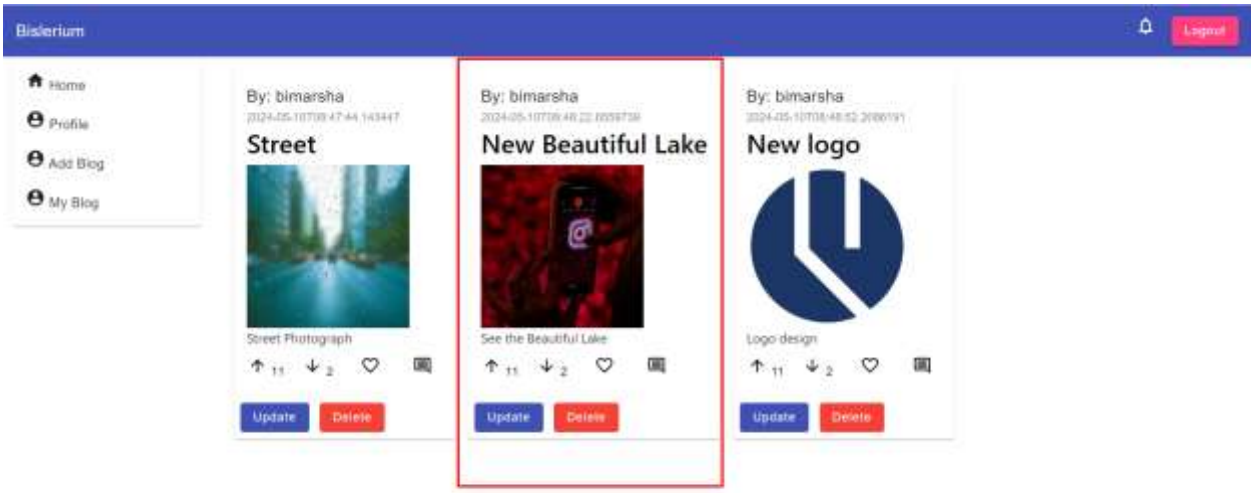


Figure 36: Blog Updated Successfully

Similar to delete button, user can also update each of their blogs using the update button. After clicking on the update button, a form is displayed which allows users to update their blogs.

2.6.3. Files Exceeding 3MB

Figure 37: Adding blog file normally

Figure 38: The image file exceeds 3MB error

Here, an error is shown to the user when the file size exceeds 3MB.

2.7. Update profile

Figure 39: Update User profile

Figure 40: Update profile form to make changes

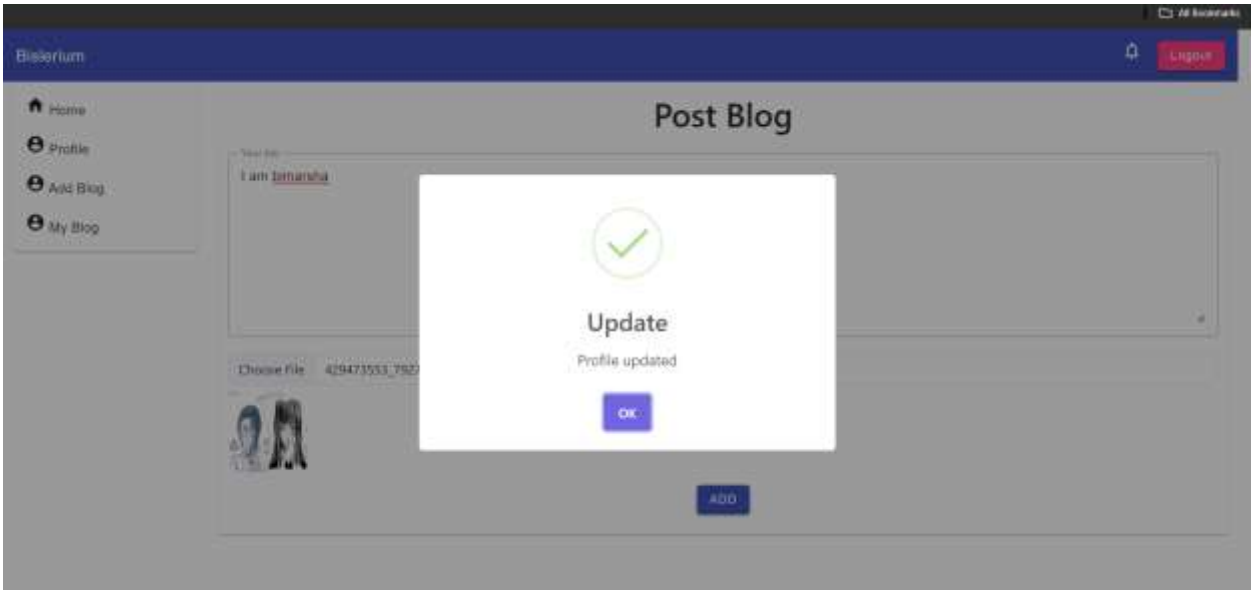


Figure 41: Profile updated successfully

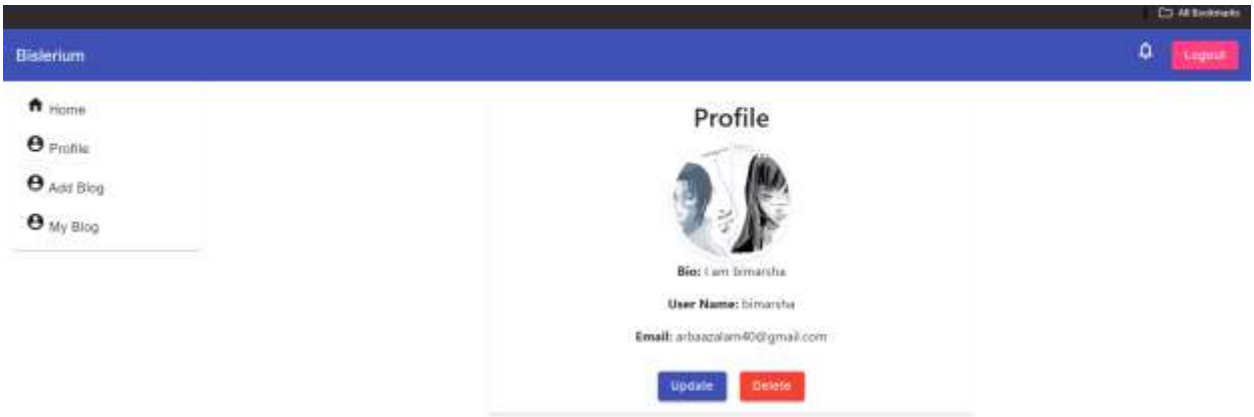


Figure 42: Changes in profile saved

Users can also update their profile from the profile page which allows users to update their bio, and their profile picture.

2.8. Delete profile

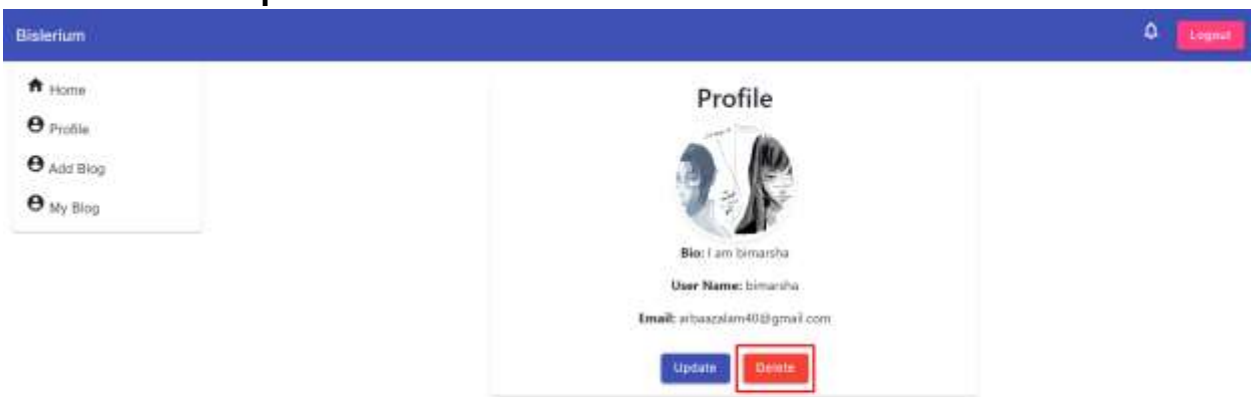


Figure 43: Delete User profile

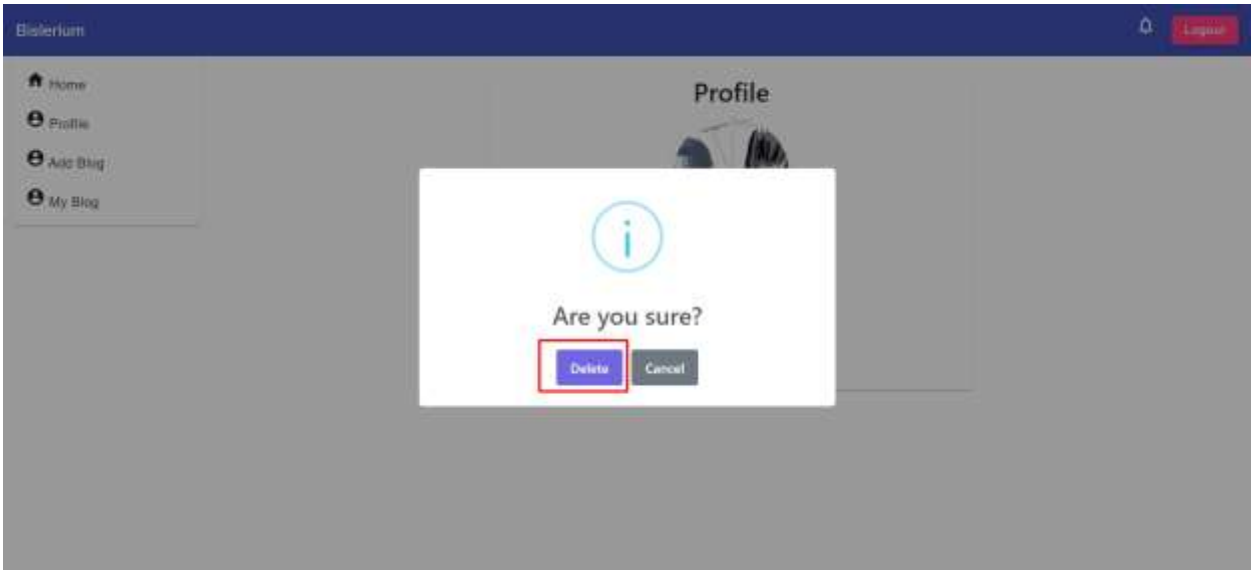


Figure 44: Confirmation message

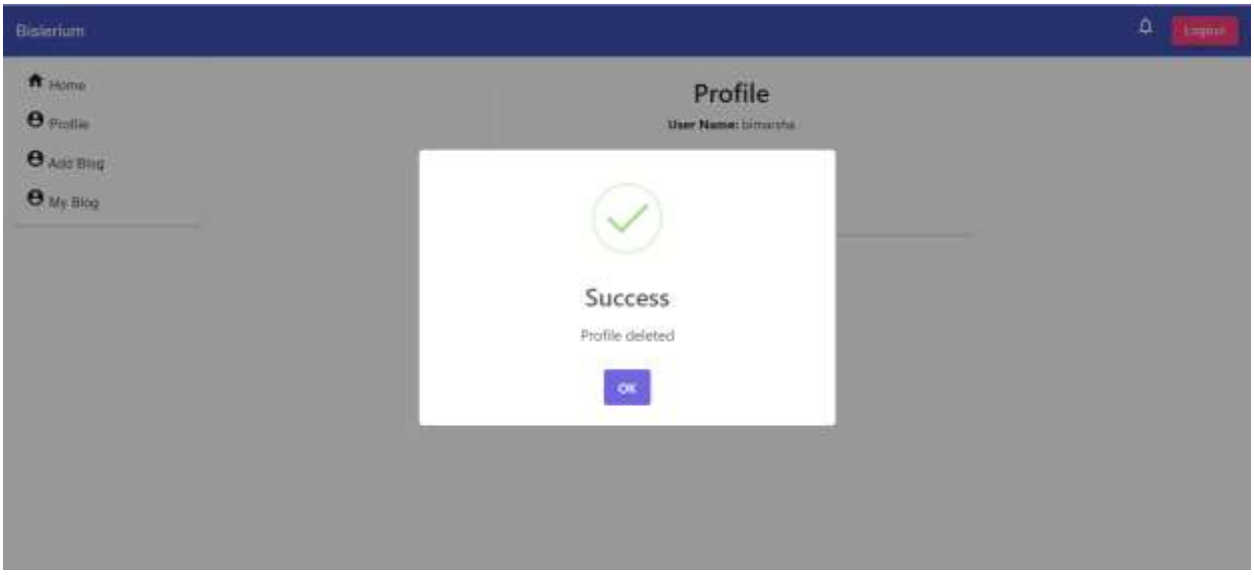


Figure 45: Profile deleted successfully message

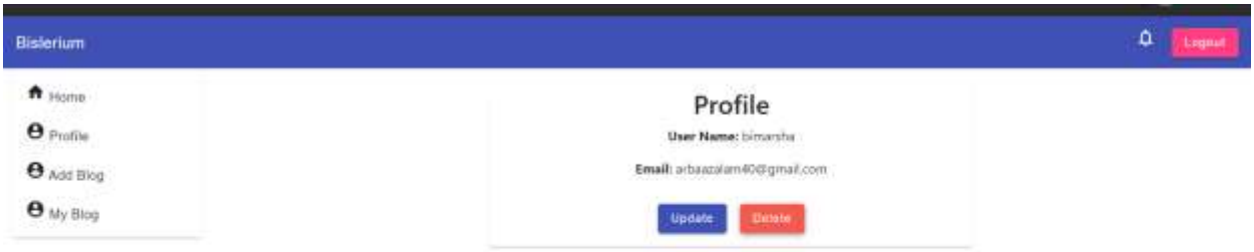


Figure 46: All profile contents are deleted

The delete profile button can delete all the details of the profile as shown in the pictures above.

2.9. Logout

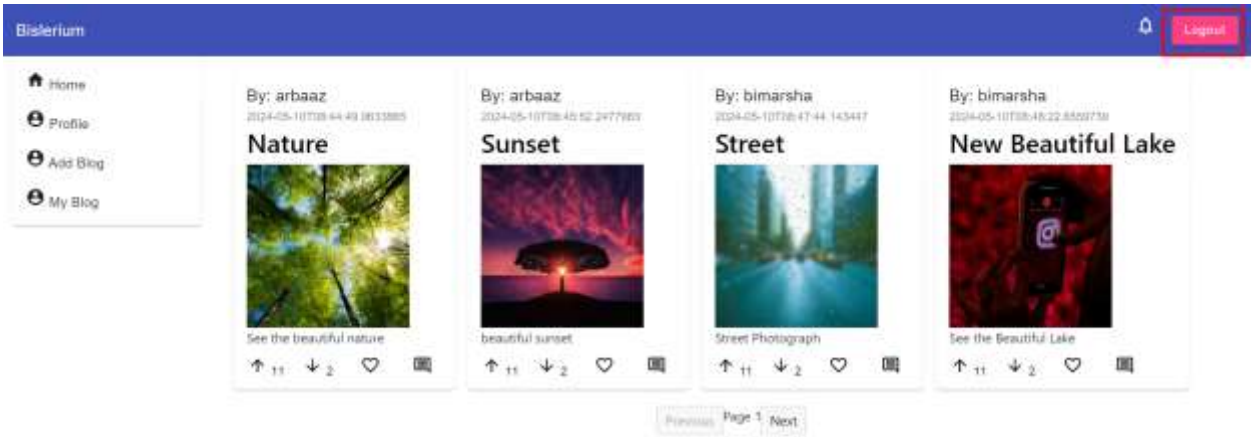


Figure 47:Logout Button

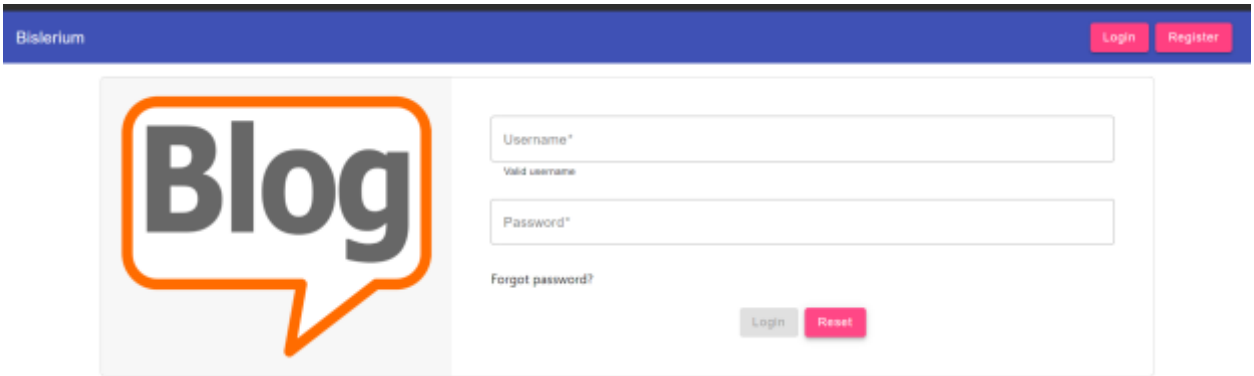


Figure 48: Logged out successfully

The logout button logs the user out and the display is back to login page.

2.10. Home without login

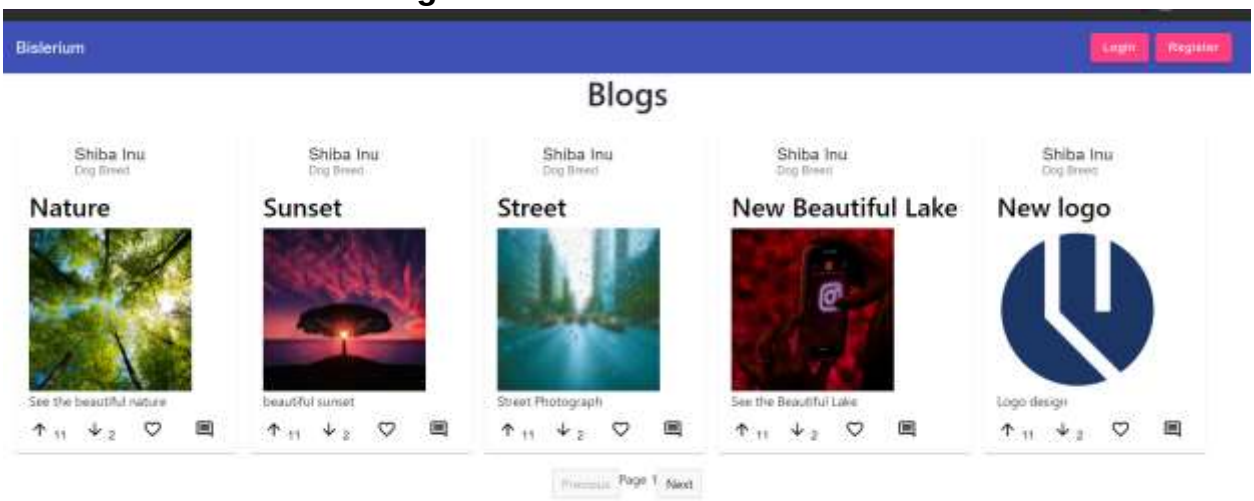


Figure 49: Home page without logging in

Figure 50: Home page without logging in (2)

Users can also access the home page without having an account or logging in. The users can only see the blogs without actually logging in.

2.11. Forget Password

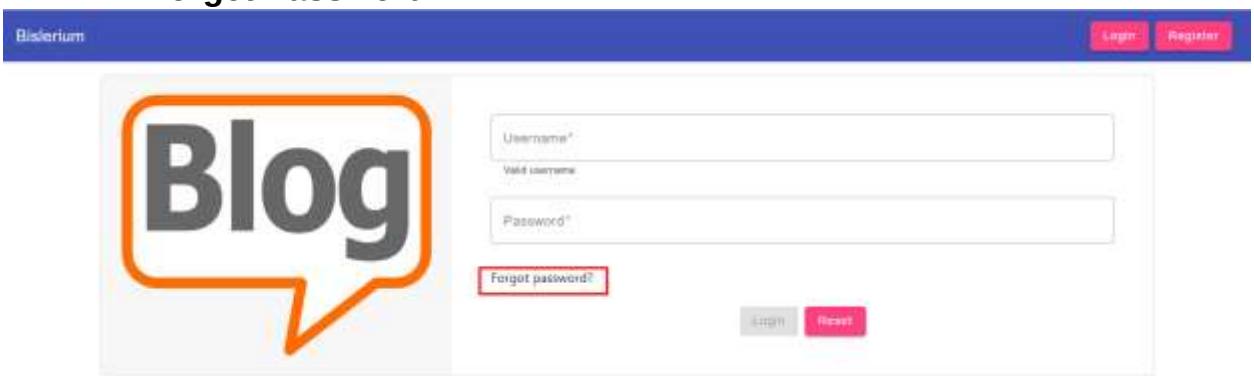


Figure 51:Forgot Password

Figure 52: Enter user email address

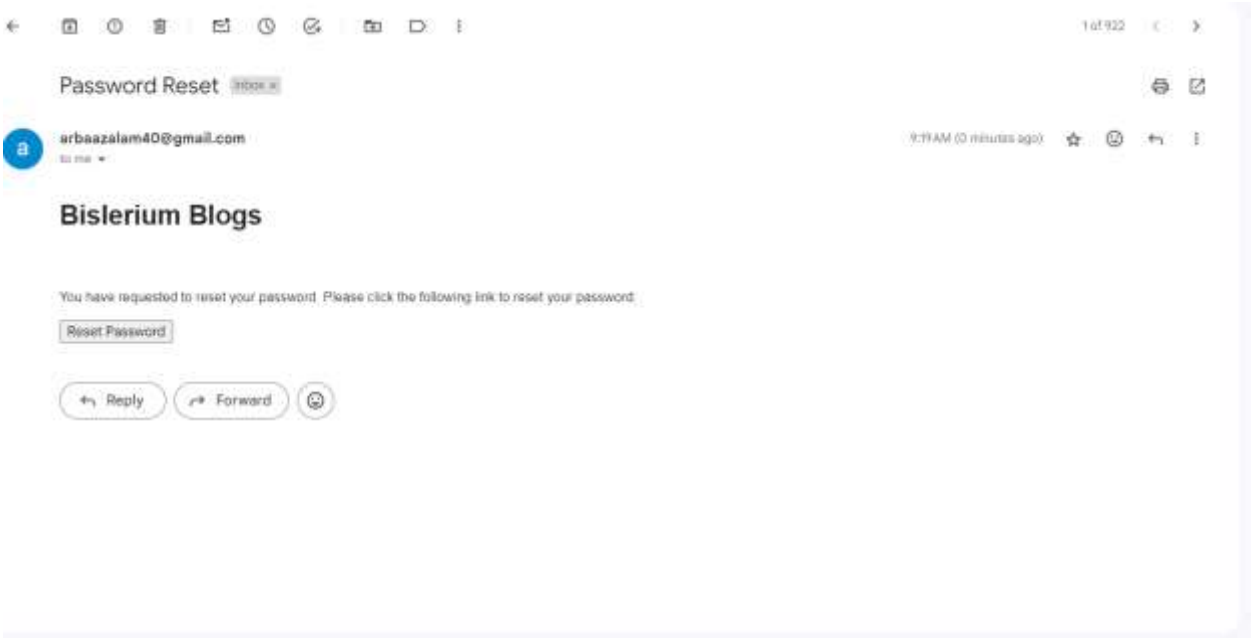


Figure 53: Reset Password email sent to user email address

Figure 54: Enter new password and reset password

There is also a forget password in the login page. After clicking on this button, the user is asked to enter their email address. After that, an email to reset the password is sent to the user email address. After following the link in the email, it is directed to a page where the user can enter new password and reset the password.

2.12. Admin Panel

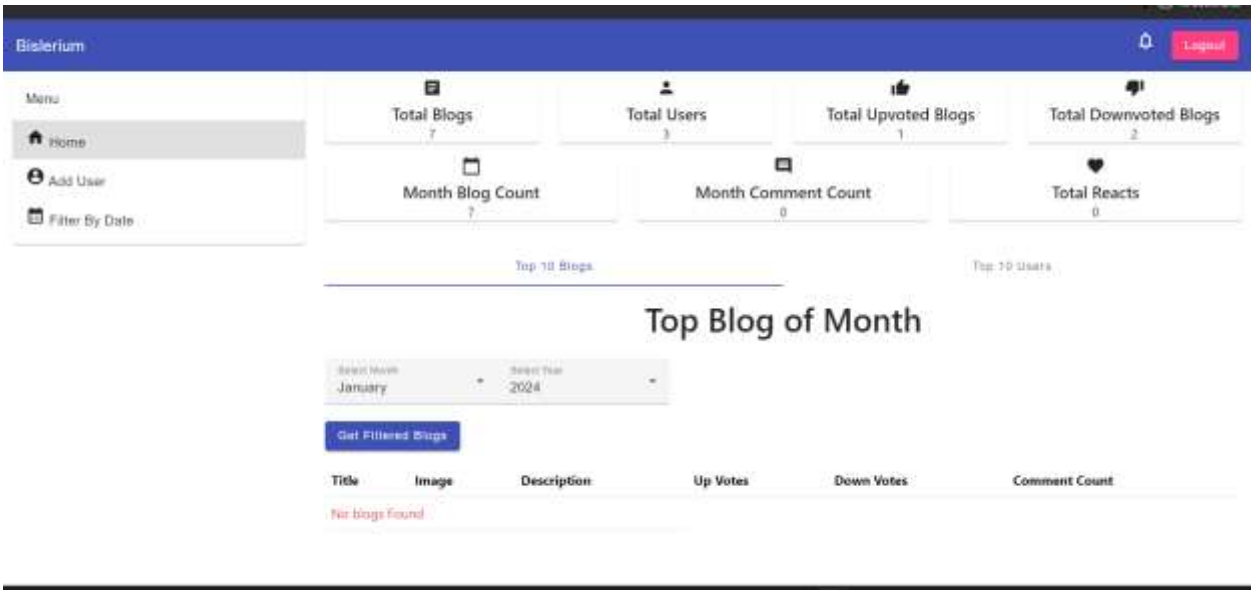


Figure 55: Admin Dashboard

This is the admin panel dashboard.

Figure 56: Filter top month admin side

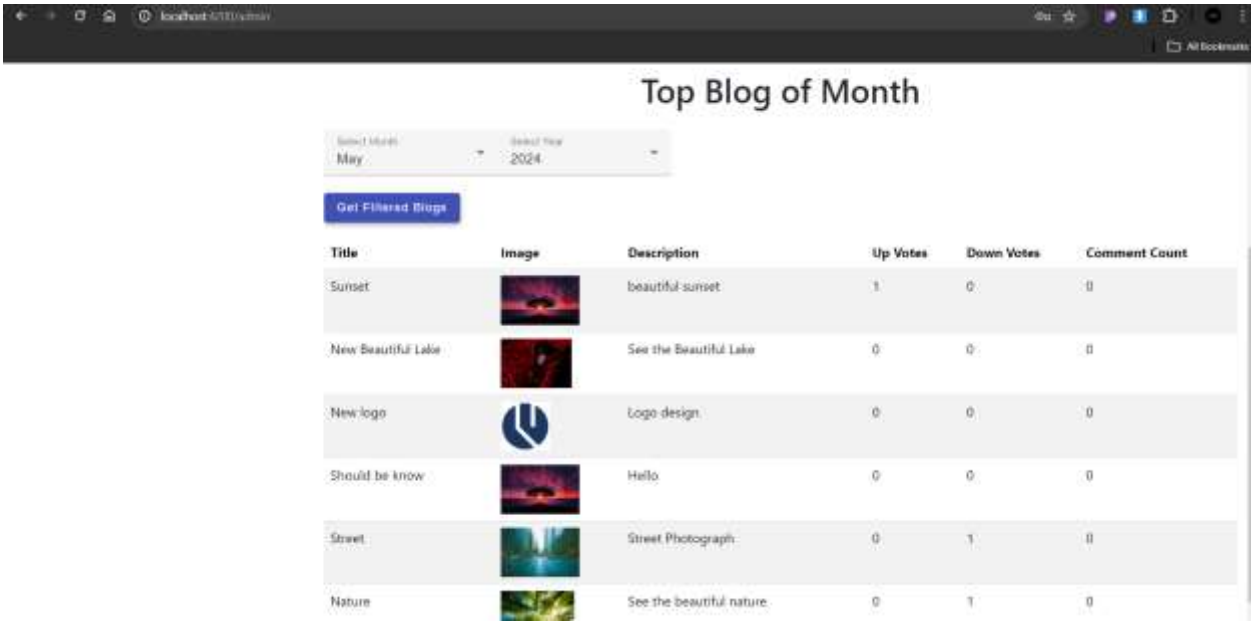


Figure 57: Value of Month that has values admin side

Figure 58: Top user admin side

Figure 59: All blogs for One month

2.12.1. Register Admin

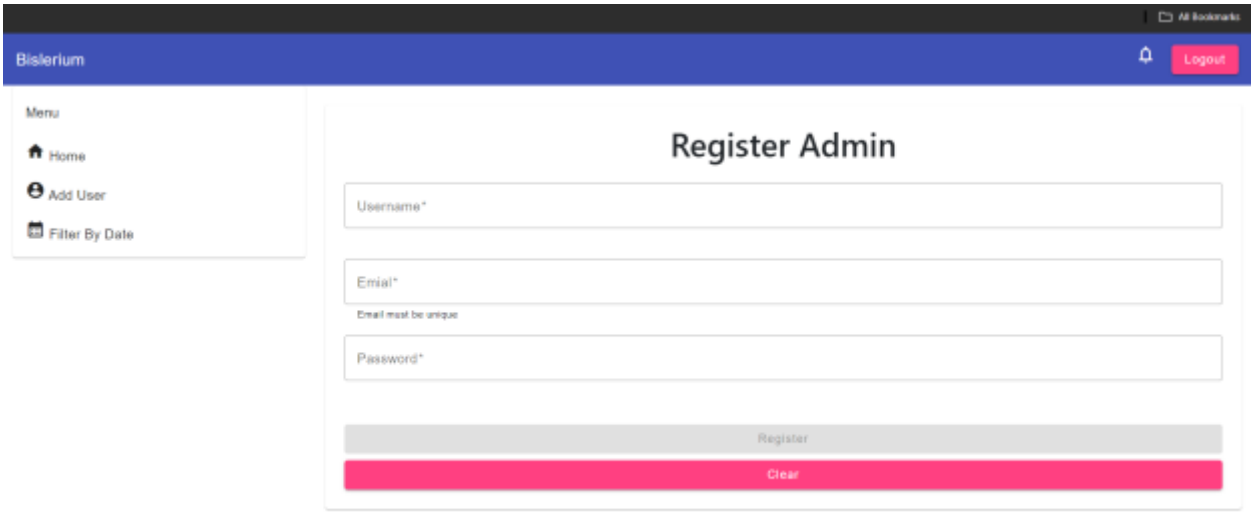


Figure 60: Register Admin form

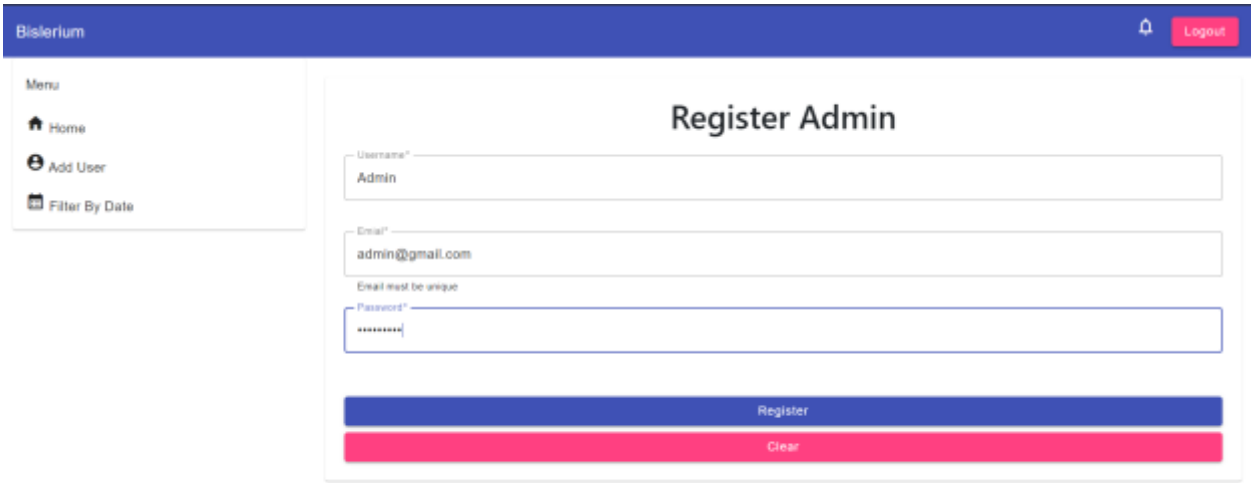


Figure 61: Add new admin details

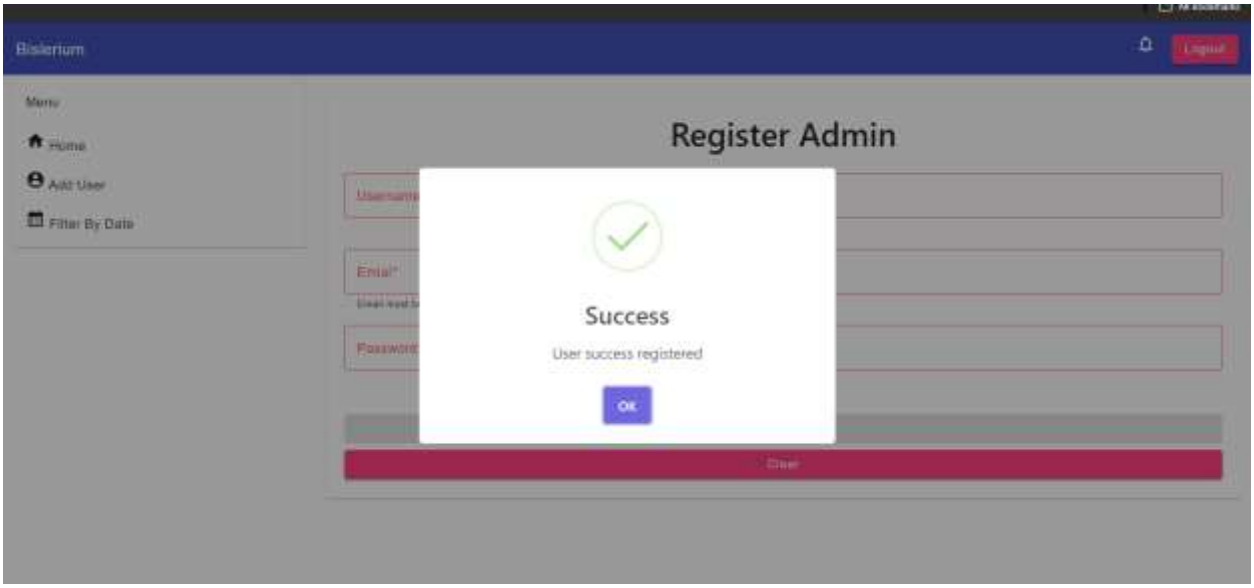


Figure 62: New admin added successfully by admin

Here, the admin is allowed to add another admin. Admin is fill out the register admin form and give admin access to another account.

3. Description of logical solution to functionalities

3.1. AuthenticationController

The AuthenticationController within the application handles user authentication, registration, profile management, and password reset functionalities.

The GetUserProfile function retrieves user profile information by querying the database based on the provided user ID.

When a new user registers using the RegisterUser or RegisterAdmin functions, a new IdentityUser is created with the provided registration data. If the user role does not exist, these functions also create the corresponding role ("user" or "admin") and assign it to the user.

The Login function authenticates user login attempts by verifying the provided credentials and returns an informative response in case of authentication failure. Upon successful authentication, it returns user details along with their role.

The ForgotPassword function initiates the password reset process by generating a reset token and sending a password reset email to the user.

The ResetPassword function resets the user's password using the provided token and new password.

Additionally, the ChangeProfile function allows users to update their profile information, including bio and image, while the DeleteProfile function enables users to delete their profile from the system. These functions collectively provide robust user authentication and profile management capabilities within the application.

3.2. blogsController

The BlogsController in the application serves as the endpoint for managing blog-related functionalities. It interacts with the DataContext to handle data operations and utilizes various services such as IHubContext, SignInManager, UserManager, and IWebHostEnvironment for additional functionalities. The controller provides several HTTP endpoints for different operations.

The GetDashboardData function retrieves aggregated statistics about the total number of blogs, users, upvotes, downvotes, reactions, comments, and monthly blog and comment counts. It returns these statistics as JSON.

The TopTenBlog and UsersWithMostBlogs functions fetch the top ten popular blogs and users with the most blogs, respectively, based on certain criteria like upvotes, downvotes, and comment counts.

The Getblog, GetRecentsblog, and GetsblogbyPopularity functions retrieve blogs based on pagination, sorting by creation date, and popularity, respectively. They return paginated lists of blogs with detailed information such as title, image, description, creation date, user name, comments, and profile image.

The GetBlogByUserId function retrieves blogs by a specific user ID, while the GetUpvoteDownVote function retrieves the count of upvotes and downvotes for a particular blog.

The controller also includes endpoints for creating, updating, and deleting blogs, along with functionalities for handling blog reactions (upvotes/downvotes), toggling reactions, and counting reactions for a blog.

Overall, the BlogsController provides comprehensive functionalities for managing blogs and their associated data within the application.

3.3. CommentsController

The CommentsController orchestrates various operations related to managing comments within the application. It provides endpoints for retrieving comments, updating existing ones, creating new comments, and deleting them.

For updating comments (PutComment), it verifies the existence of the comment, updates its content and timestamp, and saves the changes, handling concurrency exceptions gracefully.

Creating a new comment (PostComment) involves processing the incoming request, creating a new comment entity, persisting it to the database, and sending notifications to relevant users via SignalR.

Additionally, the controller allows users to upvote and downvote comments (Upvote and Downvote), maintaining a count of votes for each comment. Finally, it supports deleting comments (DeleteReply), ensuring the deletion of the specified comment from the database. Throughout these functions, error handling and data consistency are prioritized to maintain the integrity of the comment system.

3.4. NotifyController

The NotifyController is responsible for managing notifications for users. It provides an endpoint for retrieving notification data based on a user's ID. When a client sends a request to retrieve notifications (GetNotificationData), the

controller queries the database for notifications associated with the specified user ID. It then extracts unique blog IDs from these notifications and retrieves the corresponding blog data. Afterward, it filters the notifications based on the associated blogs to ensure that only relevant notifications are returned to the user. Finally, it responds with the filtered notifications if any are found, or returns a "Not Found" status message if no notifications are found for the given user ID. Throughout this process, error handling and efficient querying techniques are employed to optimize the retrieval of notification data.

3.5. RepliesController

The RepliesController manages various operations concerning replies within the system.

The PostReply([FromForm] ReplyRequest reply) function facilitates the creation of new replies, enabling users to contribute to ongoing discussions by adding their perspectives.

Lastly, the ReplyExists(int id) function offers a utility for verifying the existence of a reply based on its ID, facilitating robust error handling and ensuring accurate query results. Together, these functions form a cohesive framework for managing replies effectively within the application, supporting seamless interaction and maintenance of user-generated content.

4. Software Architecture

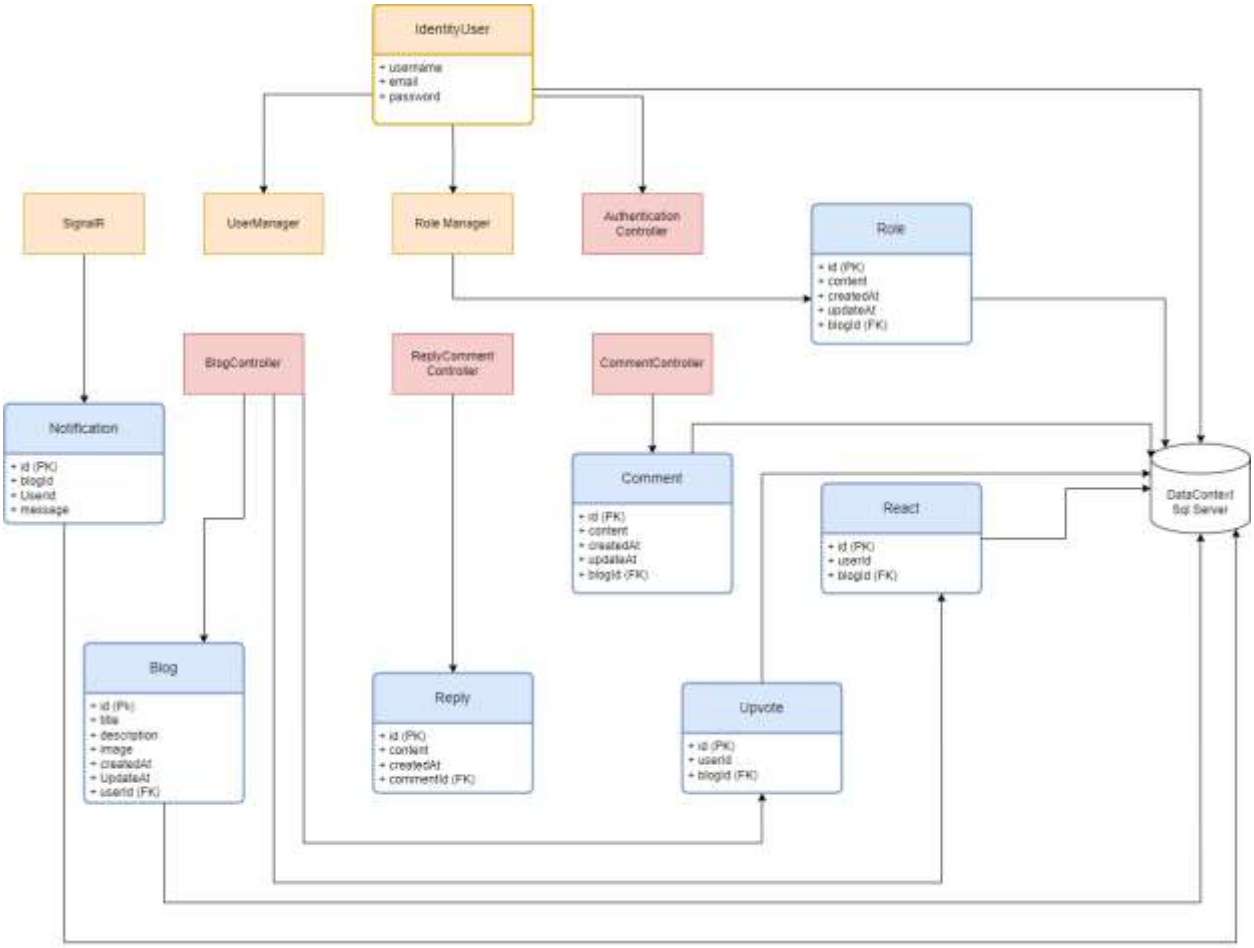


Figure 63: System Architecture

Here, the yellow boxes are classes taken from external sources. The red boxes are the controllers, and the blue boxes are database models.

We have used REST API for the development of this application.

Representational State Transfer (REST) is an architectural style that establishes a set of guidelines for developing web services. REST API is a straightforward, flexible method of accessing online services without the need for processing. (Geeksforgeeks, 2023)

REST was implemented by creating APIs using .NET and then it was implemented through Angular.

5. Details of the classes' properties and methods.

5.1. Authentication Controller Class

a. Properties Decription

Table 1: AuthenticationController Classes' Properties Description

Properties	Description
_context: DataContext	A reference to the DataContext class, allowing access to the database context for querying and manipulating data related to user profiles and authentication.
_webHostEnvironment: IWebHostEnvironment	A reference to the IWebHostEnvironment interface, providing information about the web hosting environment, such as root path and content root path.
_userManager: UserManager<IdentityUser>	A reference to the UserManager<IdentityUser> class, responsible for managing user accounts, including creation, deletion, and retrieval.
_signInManager: SignInManager<IdentityUser>	A reference to the SignInManager<IdentityUser> class, handling user sign-in and sign-out operations.
_configuration: IConfiguration	A reference to the IConfiguration interface, allowing access to configuration settings defined in appsettings.json or other configuration sources.

b. Methods Description

Table 2: AuthenticationController Classes' Methods description

Methods	Description
_GetUserProfile(string id)	Retrieves the profile information of a user specified by the provided user ID. This method queries the database to fetch user details and associated profile data.
RegisterUser(Register model)	Registers a new user based on the provided registration model. This method creates a new IdentityUser instance with the specified email, username, and password, and adds it to the user database. If successful, the user is assigned the "user" role.
RegisterAdmin(Register model)	Similar to RegisterUser, this method registers a new user with admin privileges. If successful, the user is assigned the "admin" role.
Login(Login model)	Authenticates a user based on the provided login credentials. If successful, the user is signed in and their user details along with role information are returned.
Logout()	Signs out the currently authenticated user.
ForgotPassword(string email)	Initiates the process of resetting a user's password by sending a password reset email to the specified email address.

ResetPassword(string token, string newPassword, string email)	Resets the password of a user specified by their email address using a password reset token and sets it to the provided new password.
ChangeProfile(ChangeProfileRequest request, IFormFile file)	Allows a user to update their profile information, including biography and profile image. If the user already has a profile, it updates the existing profile; otherwise, it creates a new profile.

5.2. blogController class

a. Properties Description

Table 3: blogController properties description

Properties	Description
_blogNotification: IHubContext<NotificationHub, INotificationHub>	A reference to the SignalR hub context, allowing the controller to send notifications to clients.
_signInManager: SignInManager<IdentityUser>	A reference to the SignInManager<IdentityUser> class, facilitating user sign-in operations.
_userManager: UserManager<IdentityUser>	A reference to the UserManager<IdentityUser> class, responsible for managing user accounts, including creation, deletion, and retrieval.
__webHostEnvironment: IWebHostEnvironment	A reference to the IWebHostEnvironment interface, providing information about the web hosting environment, such as root path and content root path.
_context: DataContext	A reference to the DataContext class, allowing access to the database context for querying and manipulating blog-related data.

b. Methods Description

Table 4: blogController methods decription

Methods	Description
Getblog(PaginationFilter filter)	Retrieves a paginated list of blogs along with their associated user details, comments, and reactions. Allows optional filtering based on pagination parameters.
Getblog(int? month = null, int? year = null)	Retrieves a list of blogs filtered by the specified month and year, if provided. Includes details such as blog ID, title, image, description, creation date, user name, comment count, and reaction count.
GetRecentsblog()	Retrieves a list of recent blogs ordered by creation date in descending order. Includes details such as blog ID, title, image, description, creation date, user name, comment count, and reaction count.
GetsblogbyPopularity()	Retrieves a list of blogs ordered by popularity, calculated based on factors such as upvotes, downvotes, and comments. Includes details such as blog ID, title, image, description, creation date, user name, comment count, and reaction count.
GetBlogByUserId(string userId)	Retrieves a list of blogs created by the specified user. Includes details such as blog ID, title, image, description, creation

	date, user name, comment count, and reaction count.
Getblog(int id)	Retrieves details of a specific blog identified by the provided ID. Includes details such as blog ID, title, image, description, creation date, user name, and associated comments.
Putblog(int id, CreateBlogRequest model, IFormFile file)	Updates the details of a blog specified by the provided ID with the information provided in the request model. Allows updating the blog title, description, and image.
Postblog(CreateBlogRequest request, IFormFile file)	Creates a new blog based on the information provided in the request model. Allows uploading an image for the blog.
Deleteblog(int id)	Soft deletes the blog identified by the provided ID by setting the IsDeleted flag to true.
blogExists(int id)	Checks if a blog exists in the database based on the provided ID.
GetUpvoteDownVote(int blogId)	Retrieves the count of upvotes and downvotes for a specified blog.
Upvote(int blogId, string userId)	Records an upvote for the specified blog by the provided user ID and sends a notification to all clients using SignalR.
Downvote(int blogId, string userId)	Records a downvote for the specified blog by the provided user ID.

CountReactionsForBlog(int blogId)	Retrieves the count of reactions (likes) for the specified blog.
ToggleReact(string userId, int blogId)	Toggles the reaction (like) status for the specified blog by the provided user ID.

5.3. CommentsController class

a. Properties Description

Table 5: CommentsController properties description

Properties	Description
_context: DataContext	A reference to the DataContext class, providing access to the database context for querying and manipulating comment-related data.

b. Methods Description

Table 6: CommentsController methods description

Methods	Description
GetComments()	Retrieves all comments from the database
PutComment(int id, [FromForm] Comments comment)	Updates an existing comment in the database.
PostComment([FromForm] CreaateCommentRequest comment)	Creates a new comment in the database.
CommentsExists(int id)	Checks if a comment with the specified ID exists in the database.
GetUpvoteDownVote(int commentId)	Retrieves the count of upvotes and downvotes for a specific comment.
Upvote(int commentId, string userId)	Records an upvote for a specific comment by a user.
Downvote(int commentId, string userId)	Records a downvote for a specific comment by a user.

5.4. RepliesController Class

a. Properties Description

Table 7: RepliesController properties description

Properties	Description
_context	A reference to the DataContext class, providing access to the database context for querying and manipulating reply-related data.

b. Methods Description

Table 8: RepliesController methods description

Methods	Description
GetReply()	Retrieves all replies from the database.
GetComments(int id)	Retrieves a specific comment by its ID from the database.
PutReply(int id, [FromForm] Reply reply)	Updates an existing reply in the database.
PostReply([FromForm] ReplyRequest reply)	Creates a new reply in the database.
DeleteReply(int id)	Deletes a specific reply from the database.
ReplyExists(int id)	Checks if a reply with the specified ID exists in the database

5.5. Blog Class

a. Properties

Table 9: Blog class properties description

Properties	Description
Id	Type: int Description: Represents the unique identifier of the blog.
Title	Type: string Description: Represents the title of the blog.
Description	Type: string Description: Represents the description or content of the blog.
Image	Type: string Description: Represents the URL or path to the image associated with the blog.
IsDeleted	Type: bool Description: Indicates whether the blog has been marked as deleted.
CreatedAt	Type: DateTime Description: Represents the date and time when the blog was created.
Userid	Type: string Description: Represents the user ID of the author of the blog.
User	Type: IdentityUser Description: Represents the user who authored the blog. It is a navigation property to the associated user.

Comments	Type: ICollection<Comments> Description: Represents a collection of comments associated with the blog.
React	Type: ICollection<React> Description: Represents a collection of reactions (e.g., likes, dislikes) associated with the blog.
Upvote	Type: ICollection<Upvote> Description: Represents a collection of upvotes associated with the blog.

5.6. Comments class

a. Properties

Table 10: Comments class properties description

Properties	Description
Id	Type: int Description: Represents the unique identifier of the comment.
Content	Type: string Description: Represents the content or text of the comment.
CreatedAt	Type: DateTime Description: Represents the date and time when the comment was created.
Userid	Type: string Description: Represents the user ID of the author of the comment.
User	Type: IdentityUser

	Description: Represents the user who authored the comment. It is a navigation property to the associated user.
BlogId	Type: int Description: Represents the ID of the blog to which the comment belongs.
Blog	Type: Blog Description: Represents the blog to which the comment belongs. It is a navigation property to the associated blog.
Replies	Type: ICollection<Reply> Description: Represents a collection of replies associated with the comment.

5.7. Comment Upvote Class

a. Properties

Table 11: Comments upvote properties description

Properties	Description
Id	Type: int Description: Represents the unique identifier of the comment upvote.
CommentId	Type: int Description: Represents the ID of the comment that this upvote is associated with.
Comment	Type: Comments Description: Represents the comment that this upvote is associated with. It is a

	navigation property to the associated comment.
Userid	Type: string Description: Represents the user ID of the user who upvoted the comment.
User	Type: IdentityUser Description: Represents the user who upvoted the comment. It is a navigation property to the associated user.
IsUpvoted	Type: bool Description: Represents whether the upvote is an upvote (true) or a downvote (false).

5.8. Login Class

a. Properties Description

Table 12: Login class properties description

Properties	Description
Username	Type: string Description: Represents the username entered by the user for login. Required: Yes
Password	Type: string Description: Represents the password entered by the user for login. Required: Yes
RememberMe	Type: bool

	Description: Indicates whether the user wants to be remembered for future logins. Required: No

5.9. MyUser class

a. Properties

Table 13: MyUser class properties description

Properties	Description
Id	Type: string Description: Inherited from IdentityUser, represents the unique identifier for the user.
UserName	Type: string Description: Inherited from IdentityUser, represents the username of the user.
NormalizedUserName	Type: string Description: Inherited from IdentityUser, represents the normalized username of the user.
Email	Type: string Description: Inherited from IdentityUser, represents the email address of the user.
NormalizedEmail	Type: string Description: Inherited from IdentityUser, represents the normalized email address of the user.
EmailConfirmed	Type: bool

	Description: Inherited from IdentityUser, indicates whether the email address has been confirmed.
PasswordHash	Type: string Description: Inherited from IdentityUser, represents the hashed password of the user.
SecurityStamp	Type: string Description: Inherited from IdentityUser, represents a security stamp that should be regenerated when a user's security-sensitive information changes.
ConcurrencyStamp	Type: string Description: Inherited from IdentityUser, represents a value used for optimistic concurrency.
PhoneNumber	Type: string Description: Inherited from IdentityUser, represents the phone number of the user.
PhoneNumberConfirmed	Type: bool Description: Inherited from IdentityUser, indicates whether the phone number has been confirmed.
TwoFactorEnabled	Type: bool Description: Inherited from IdentityUser, indicates whether two-factor authentication is enabled for the user.
LockoutEnd	Type: DateTimeOffset?

	Description: Inherited from IdentityUser, represents the DateTimeOffset when the user's lockout period ends.
LockoutEnabled	Type: bool Description: Inherited from IdentityUser, indicates whether the user can be locked out.
AccessFailedCount	Type: int Description: Inherited from IdentityUser, represents the number of failed access attempts.
Image	Type: string Description: Represents the image URL associated with the user's profile.
Bio	Type: string Description: Represents the biography or description of the user.
Roles	Type: IEnumerable<IdentityRole> Description: Represents the roles assigned to the user.

5.10. Paginated List class

a. Properties

Table 14: Paginated List class properties description

Properties	Description
PageNumber	Type: int Description: Represents the current page number of the paginated list.
PageSize	Type: int Description: Represents the maximum number of items that can be displayed on a single page of the paginated list.
TotalPages	Type: int Description: Represents the total number of pages in the paginated list based on the total number of records and the page size.
TotalRecords	Type: int Description: Represents the total number of records/items in the entire list, not just the current page.
Data	Type: IEnumerable<T> Description: Represents the collection of items of type T that are present in the current page of the paginated list. It is initialized as an empty enumerable.

5.11. NotificationData class

a. Properties

Table 15: NotificationData class properties description

Properties	Description
BlogID	Type: int Description: Represents the ID of the blog associated with the notification.
Title	Type: string Description: Represents the title of the blog associated with the notification.
Message	Type: string Description: Represents the content or message of the notification.
UserId	Type: string Description: Represents the ID of the user to whom the notification is targeted.

5.12. PaginationFilter Class

a. Properties

Table 16: PaginationFilter class properties description

Properties	Description
PageNumber	Type: int Default Value: 1 Description: Represents the current page number of the pagination filter. Defaults to 1 if not specified.
PageSize	Type: int Default Value: 10 Description: Represents the maximum number of items that can be displayed on a single page of the paginated list. Defaults to 10 if not specified.

5.13. Profile Class

a. Properties

Table 17: Profile class properties description

Properties	Description
Id	Type: int Description: Represents the unique identifier for the profile.
Image	Type: string Nullable: Yes

	Description: Represents the URL or path to the image associated with the profile.
Bio	Type: string Nullable: Yes Description: Represents the biography or description associated with the profile.
UserId	Type: string Description: Represents the foreign key referencing the identity user associated with the profile.
User	Type: IdentityUser Description: Represents the navigation property linking to the identity user associated with the profile.

5.14. React Class

a. Properties

Table 18: React class properties description

Properties	Description
Id	Type: int Description: Represents the unique identifier for the react.
UserId	Type: string Description: Represents the foreign key referencing the identity user who reacted.
User	Type: IdentityUser Description: Represents the navigation property linking to the identity user who reacted.

BlogId	Type: int Description: Represents the foreign key referencing the blog to which the react belongs.
Blog	Type: Blog Description: Represents the navigation property linking to the blog to which the react belongs.

5.15. Register Class

a. Properties

Table 19: Register class properties description

Properties	Description
Username	Type: string Description: Represents the username of the user registering.
Email	Type: string Description: Represents the email address of the user registering. Validation: Decorated with [EmailAddress] attribute to ensure the input value is a valid email address.
Password	Type: string Description: Represents the password of the user registering. Validation: Decorated with [DataType(DataType.Password)] attribute to specify that the property represents a password.

5.16. Reply class

a. Properties

Table 20: Reply class properties description

Properties	Description
Id	Type: int Description: Represents the unique identifier of the reply.
content	Type: string Description: Represents the content of the reply.
CreatedAt	Type: DateTime? Description: Represents the date and time when the reply was created.
Userid	Type: string? Description: Represents the user ID of the user who created the reply.
User	Type: IdentityUser Description: Represents the user who created the reply.
CommentId	Type: int Description: Represents the ID of the comment to which the reply belongs.
Comment	Type: Comments Description: Represents the comment to which the reply belongs.

5.17. Upvotes class

a. Properties

Table 21: Upvotes class properties description

Properties	Description
Id	Type: int Description: Represents the unique identifier of the upvote.
BlogId	Type: int Description: Represents the ID of the blog for which the upvote is given.
Blog	Type: Blog Description: Represents the blog for which the upvote is given.
Userid	Type: string Description: Represents the user ID of the user who gave the upvote.
User	Type: IdentityUser Description: Represents the user who gave the upvote.
IsUpvoted	Type: bool Description: Indicates whether the upvote is positive (true) or negative (false).

6. Personal Experience

6.1. Arbaaz Alam

My experience working on this coursework using ASP .NET was both challenging and rewarding. My grasp of web application development was expanded by the dynamic experience of working on the Blog APIs, Single R real-time notifications, and pagination components. My ASP.NET Core abilities were honed through designing RESTful endpoints for CRUD activities, which also highlighted the significance of data security and effective request handling. User engagement was increased by integrating Single R for real-time notifications, but careful optimization was needed to guarantee dependability. By efficiently managing huge datasets, the application's speed was enhanced by the implementation of pagination components. Each aspect posed distinct difficulties, which aided in my development as a developer and strengthened my resolve to provide reliable and intuitive online apps.

6.2. Bimarsha Poudel

Using Angular to construct the frontend and developing Comment APIs for this coursework was a rewarding experience that improved my understanding of full-stack web development. My backend development abilities were sharpened by creating RESTful APIs for comment management, which highlighted the importance of data security and integrity. Working with Angular on the frontend promoted creativity in creating user experiences that are both intuitive and aesthetically pleasing by offering insights into client-side rendering and UI design. In order to complete these objectives, the team needed to communicate and coordinate well, which improved our capacity to work together in a variety of settings. My desire to create powerful online solutions that skilfully combine sophisticated front-end user interfaces with back-end functionality was further stoked by this experience.

6.3. Prabin Thakur

This project's authentication API development was a good educational experience. I was able to learn more about the details of security protocols and user authentication, which improved my knowledge of backend programming. Working together on features like user login and registration made me realize how crucial it is for frontend and backend systems to integrate seamlessly. All in all, it was a fulfilling chance to support the project's inception while putting an emphasis on user privacy and data security.

6.4. Prashant Baral

My understanding of frontend and backend development was expanded through the dynamic experience of working on Reply APIs and frontend/UI design. Developing the reply feature helped me get more insight into backend systems, while working on frontend/UI design improved my ability to make user interfaces that are easy to understand. This unique position helped me improve as a developer and gave me insightful knowledge about the software development life cycle. Moreover, working on the top 10 blogs for the Admin side was also learning experience.

6.5. Ayush Krishna Shrestha

My participation in the creation of admin panel APIs and the creation of comprehensive documentation have given me invaluable knowledge about the complexities of technical communication and system architecture. To ensure smooth system operation, building APIs to handle admin functionalities required careful planning and implementation, fostering a deeper understanding of backend systems. Concurrently, creating thorough documentation made it easier to communicate and comprehend the architecture and features of the project. Although it was hectic, the dual role improved both my technical proficiency and my ability to clearly communicate difficult ideas.

7. References

Geeksforgeeks. (2023, August 7). *Geeksforgeeks*. Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/rest-api-introduction/