Offbeat Escape

Design Documentation

University of Windsor

Objective

This document serves as a design document for the application Offbeat Escape.

Author – Team 4

Recipient - Dr. Aznam Yacoub

State – Finalised

Written On – 13-04-2021

History

Date	Updates	State
05-03-2021	Design added for User Authentication, Add friends, Post configuration – Reader &Writer features.	Incomplete
18-03-2021	Design updated for previously added features and added for general feed and reading list.	
31-03-2021	Design added for Inspiration History & trending posts.	
13-04-2021	Updated	Finalised

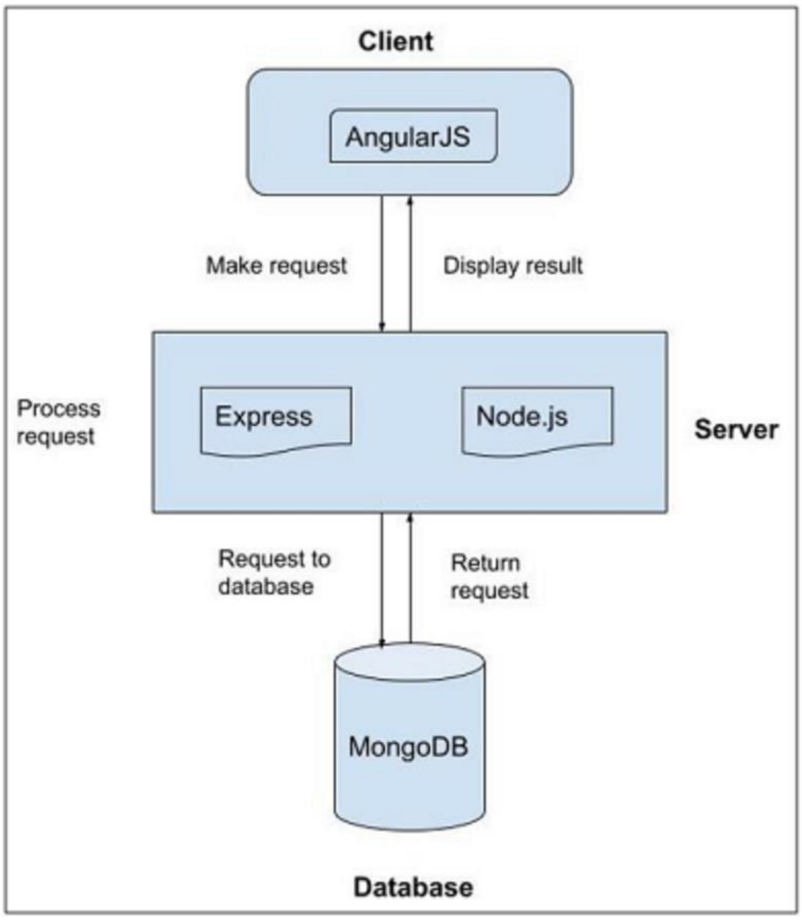
Index

Introduction

This document defines the design of the product Offbeat Escape. It explains the various components, features and system level design associated with it. The document assumes that the software requirements specification is clear and there are no ambiguities in the software requirements. Algorithms and pseudo code will possibly define all components of the system. The actual implementation may vary slightly depending upon the technologies incorporated.

Architecture

The application's front-end is developed in Angular while the application's backend is developed in Node and Express JS. Mongo DB has been used as the application's database. The following diagram explains the application's architecture.



System Design

In this section we will describe in detail various components of the application and walk through its design. The various components of the system are explained below:

Login

Present a page to user where the registered user can login using email id and password. The page should also contain social login features.

Various functions of the login component are as follows:

- onLoginClick()

  1. Capture the email id and password from login form into an object.
  2. Validate the credentials by calling login API.
  3. Use a service to make http requests.
  4. Create a method inside service to validate login which accepts user credentials and sends the credentials as request payload to the respective API.
  5. The login API connects to mongo DB and checks whether the user exists in database.
  6. Make sure to handle http requests correctly by implementing an error handler in the service file. This handler will take instance of error object and print it in developer console.
  7. Use the response from the restful API by subscribing it inside component file.
  8. Once response is successful use router object to navigate to dashboard.

- signInWithFB()

  1. Use the window object to redirect the user to facebook's authentication.
  2. Ensure new users are able to navigate to Facebook authorization portal.
  3. Save the Facebook OAuth token in the database for the logged in user.

- signinWithGoogle()

1. Use the window object to redirect the user to Google's authentication.
2. Ensure new users are able to navigate to google authorization portal.
3. Save the google OAuth token in the database for the logged in user.

- goToSignUp()

1. Ensure new users are able to navigate to sign up page.
2. Use a html element which acts as a button to go to sign up page.
3. On click of this button invoke the above function and use router object to go to sign up page.

Signup

Present a page for new users to register into the network of Offbeat Escape.

Various functions of the signup component are as follows:

- onSignUpClick()

1. Declare an object to capture values from the signup form.
2. Use this credential to make an http request to sign up API.
3. Ensure the API request is written in another service file inside a method.
4. Receive The signup payload from the client to the backend.
5. Create a new User in the database with unique ID and encrypted password.
6. Ensure proper error handler is written inside the service file.
7. Once response is received show a message to user that signed up successfully.
8. Use Angular Material's snack-bar component to show the message once http response is successfully received.

Dashboard

Represents the main dashboard of the application. Contains sub-components like Trending Posts, Users List, Reading List, Header and General Feed.
Various functions of the dashboard component:

- showFriendList():

  1. Add an html element to show the remaining list of users if the length of users list is more than 5.
  2. Use the router object to navigate to the component to show remaining users.

- getUsersList():

  1. Call the above-mentioned function when the dashboard component is initialized.
  2. Use a separate service to initiate the http request.
  3. Inside the service write a separate method and call the users list API.
  4. Perform search on Mongo database for all active users
  5. Send the response payload of all active users to client.
  6. Make sure to write a method for error handling in case the API fails and does not send a response.
  7. After receiving a successful response from API call the filterFriendsList() method to filter the users list. User's list will not show the other users if the logged in user has sent them a friend request.

- filterFriendsList()

  1. After receiving the initial users list filter, the users list array.
  2. Declare another array to store other users to whom the current logged in user has already sent a friend request.
  3. Use the friend's array inside users list response to determine all other users who have been sent a friend request.
  4. Traverse through the users list array and already sent friend request array to mark a boolean field "alreadySentFriendRequest" to true or false.
  5. Use filter function to store a new temporary user list based on the boolean value. If alreadySentFriendRequest is false assign it to the temp user list array.
  6. Store the temp user list array to user list array.
- getFriendsPosts()

1. Use the dashboard service to make http request to fetch friend's posts.
2. Make sure you call the error handler method in case the request fails to fetch a response.
3. Use mongoDB aggregate to get the desired output.
4. Match the logged in user's username in users collection to reach the specific user doc
5. For the above match's output, get posts of all users in friends array in user document using mongoDb Lookup and project them as feed.
6. Once the API sends a response subscribe to it inside component to show the general feed posts.

- goToDetailedPost()

1. Whenever, a click on a post redirects the user to the specific post.
2. Use the whole div structure of post as an html element. On click of this element called the above-mentioned function.
3. Use router object to navigate to post-heading component.

- sendFriendRequestToUser()

1. Use the dashboard service to call the send friend request API.
2. Declare the payload for this request.
3. Payload will contain the user id to which a friend request is being sent.
4. Inside the service declare a method and call the send friend request API.
5. Check if friend array exists for the current user and, add the friend ID to the current user's friends array.
6. If friends array does not exist, create a friends array and add the friend ID with status as pending to the current user's friends array. The same step is performed for the friend ID to whom the friend request is sent where current user ID status is set as pending.
7. Make sure you call the error handler method.
8. After successful response show the message friend request sent successfully using snack bar component of Angular Material.

- getLoggedInUser()

1. Get the current from localStorage.
2. Return the stringified and parsed json response.

- getSavedPostId()

1. Get the user details from the getLoggedInUser() method.
2. Retrieve the id of saved posts from the savedPosts field in user.
3. Store the savedPosts in an array.

- getSavedPostContent()

1. For every saved post id stored in array, call the dashboard service getSavedPosts() and give the id as parametres to the function.
2. This function will return the saved post.
3. Push this returned result in an array which will be the reading list.

- getTopTrendingPosts()

1. Use the dashboard service to initiate a http request to get the top trending posts.
2. Inside the service write a separate method to make the http request to trending post API.
3. Match the logged in user's username in users collection to reach the specific user doc
4. For the above match's output, get posts of all users in friends array in user document using mongoDb Lookup and project as feed.
5. For the above feed, unwind results to get separate docs
6. Project the unwinded results as original data format and add another field count that keeps track of size of savedBy subdocument to check the no. of saves for a post
7. Sort the above output on the basis of count field.
8. Limit output to 6
9. Ensure error handler is called in case the API fails to deliver a response.
10. Once response is received subscribe to the response inside component and assign an array to store the top trending posts and show it in UI.

Header

The header of the application. Contains various components like user settings and the notification.

Various functions of the header component are as follows:

- logOutFromApplication()

  1. Develop this function so that the user is able to log out from the application.
  2. Keep an html element in the DOM and on click call the above-mentioned function.
  3. Remove the current user from local storage.
  4. Use the router object to route back to login page.

- goToAddPost()

  1. Use the router object to route the user to add post page

- goToMyPosts()

  1. Use the router object to route the user to my posts page

- goToSettings()

  1. Use the router object to route the user to settings page

- goToPost()

  1. Use the router object to route the user to detailed post page
  2. Make sure you pass the post id in the route
  3. Make sure you pass the notifier of the post. This is done for creating the inspiration cycle.

- routeToDashboard()

  1. Use the router object to route the user to dashboard from any other page.

- acceptFriendRequest()

1. Write a function to make http request to the accept friend request API.
2. Use a separate service to make the API call. Accept the payload from the component.
3. Make the HTTP request to friend request API using the payload.
4. Accept the payload response for the current userID.
5. Check if friend array exists for the current user and, add the friend ID to the current user's friends array and set status as friends.
6. If friends array does not exist, create a friends array and add the friend ID to the current user's friends array and set status as friends.
7. Ensure you call error handler in case the API fails to return a response.
8. Once you get a successful response subscribe to it inside component.

Detailed Post

Show a post in detail including all functionalities like commenting, saving and reporting the post. Also shows the inspiration cycle of the post.
Various functions of the detailed post component are as follows:

- addComment()

1. Develop a method to add comment in a post.
2. Use the add comment API to add a comment in a post.
3. Develop the payload of this API inside the method.
4. Use the service file to accept payload and call the add comment API.
5. At the backend, this comment is stored in each post.
6. Each comment has three fields, the username of the person who commented, the actual comment, and the time of the comment.
7. When the comment is added, the whole post is returned in the response from backend.
8. Use the error handler method in case API response fails for some reason.
9. Once response is received from the API subscribe to the response inside component and update the UI.
10. Make a temporary array to store comments. Once API response is successful use the temporary array to push a comment into the temporary array to keep the UI updated.

- savePost()

    1. Develop a method to save post for the reading list and also to show the inspiration cycle.
    2. Develop the necessary payload for passing to save post API and also for the graph API.
    3. Develop two methods inside service to make HTTP request to save post API and also for the graph API.
    4. API shall receive "notifier" in payload. This notifier will be used to track if a user reached the post via a notification or directly via the general feed.
    5. Push entry of User saving a post i.e. savedBy : [{user : A, inspired : [B]}, { user : B, inspired : [] } , { user : C, inspired : [] }]
    6. Push entry for "C was inspired by B" i.e. savedBy : [{user : A, inspired : [B]}, { user : B, inspired : [C] } , { user : C, inspired : [] }]
    7. Call error handler to handle if response fails.
    8. Once the response is successful subscribe it inside component.
    9. Use the snack bar component to show the message that post is successfully saved.

- reportPost()

    1. Use this function to report a post for violating terms and conditions of the application.
    2. Write a method inside service to call the report post API.
    3. Make the payload inside component and pass it to the service.
    4. Initiate the HTTP request to report post API by passing the payload.
    5. At the backend, each post has a timesReported field which is initialized to zero..
    6. Each time a post is reported, and the api is called, this timesReported field is incremented one time.
    7. When a post is reported, the backend sends the full post as a response.
    8. When the timesReported reaches 10, the post is deleted.
    9. Make sure you call the error handler properly in case response fails from the API.
    10. Once the API return a response use that response to check the number of times a post has been reported.
    11. If the post has been reported more than 10 times call another API to delete the post.
    12. Navigate to my posts to show the user that the post has been deleted.

- getInspirerHistory()

1. Develop this function to call the inspirer history API and show the cycle in UI.
2. Declare a variable for the request payload.
3. Pass the payload to a method inside service.
4. Ensure that this service is calling the get inspirer history API.
5. Run Mongoose's aggregate function over the posts collection.
6. Match post's title and owner to get the post for which cycle is to be created
7. Unwind the document to get savedBy subdocuments into separate docs.
8. Match "current user" with SavedBy.inspired to fetch the document where user was inspired.
9. Project output to return SavedBy.user as inspirer.
10. If return values is not null, Match the output with owner to check if end of cycle is reached.
11. If yes, set cycle completed to true and return the cycle.
12. If false, set current user to return value of aggregate query else set cycle completed to true and return empty cycle.
13. Store the response to an array by subscribing to the response inside component.
14. Show the array as inspirer history in UI.

Settings

Contains the option to upload profile image for the user.
Various functions of the settings component are as follows:

- uploadFile()

1. Develop a function to upload images from your local machine.
2. Use file handler and image snippet services for this functionality.
3. Display the file name beside the upload button in DOM.

- addProfileImage()

1. Develop this function to add profile image to a user profile.
2. Make an instance of form data and reference the profile image from DOM.
3. Append the profile image into form data and make HTTP request to add profile image API.

4. If the response is successful image is uploaded successfully for the current logged in user.
5. Make sure to handle error conditions while calling the API.

Add Posts

Contains the option to add the posts of the logged in user.
Various functions of the add post component are as follows:

- addPosts()

1. Check if the title and description of the post are given.
2. If not, return an error.
3. If given, check if any image is selected.
4. If not, make an instance of form data.
5. Append the title and description to this instance.
6. Make an http request and call the addPost api and send this form data instance.
7. If the response is successful, the post will be added.
8. Make sure to handle error conditions while calling the API.
9. If, no image is selected, create a form data instance.
10. Append the title, description and image to the instance.

- uploadFile()

1. Create a new file reader instance.
2. If any image is selected, add an eventlistener to this instance.
3. Create an image snippet in this instance which will contain the image file.
4. Use the readAsDataUrl method of the file Reader instance which will create the image file into a blob.
5. Return the blob.

My Posts

Contains the option to view, edit and delete the posts of the logged in user.
Various functions of myposts component are as follows:

- deletePost()

  1. Make an http delete request to the posts api to delete the post.
  2. If the response is successful, the post will be deleted.
  3. Make sure to handle error conditions while calling the API.
  4. Also, make sure to update the posts array using the updatePostsArray() method.
  5. Navigate to myposts page.

- getLoggedInUser()

  1. Get the current from localStorage.
  2. Returns the stringified and parsed json response.

- showPost()

  1. Get the current user details using the getLoggedInUser() function.
  2. Navigate to the Detailed Post page to show the post.

- editPost()

  1. Navigate to the edit post page.

Edit Post

Edits the posts of the logged in user.
Various functions of the edit post component are as follows:

- ngOnInit()

    1. This function runs whenever this component is initialized.
    2. Get the postId from routeParametres.
    3. If there is no post Id, give an error response.
    4. If there is post id, do the following steps.
    5. Call the editPostService method getPost().
    6. If, the post has image, return the title, description and image/
    7. Store them in an array.
    8. If the posts does not have an image, return the title and description.
    9. Store them in array.
    10. Use these title and description and image to prefill the text box of the post and make it editable.

- editPost()

    1. Fill the title, description and image url in the edit post payload.
    2. Make a call to editpost service's method editPost which will make an http patch request to edit the post.
    3. Send the edit post payload along with this request.
    4. If the request is successful, navigate to myPosts page.
    5. If the request is not successful, give an error.