Blog API Project Ideation

This blog API project requires you to create your personal full-stack blog.

# Vision statement

**The problem:** Keeping an organization’s fans in sync with internal communications and products development.

**The solution:** A blogging API that allows fans to read a company’s articles and comments between staff members.

# Goals

* Practice and see the benefits of creating an [API only backend](https://www.brightspot.com/cms-resources/technology-insights/api-first-content-management-system-design-benefits#:~:text=Unlike%20no%2DAPI%20CMSs%2C%20the,end%20interface%20to%20input%20data.).
* Learn how to set up an API and access it from an external source.
* Experience the power and flexibility of separating your backend code from your frontend code.
* Create a backend
* Create two frontends
* The first front end is for accessing blog posts. This is a blog consumption website for people who want to read and comment on your posts.
* The second front end is for editing the blog posts. This is a blog editing website for you to write, edit and publish your posts.

**Note:** How much work you want to put into the front-end code is up to you. Technically this is a backend focused course so if you would prefer, feel free to focus on the REST API.

# Structure

You can structure the project’s repo using either of the following options:

* Create three polyrepos (separate GitHub repos) for each of the three apps (the backend, the consumption frontend, and the editing frontend).
* Create a [monorepo](https://en.wikipedia.org/wiki/Monorepo) in which each of the three apps are kept in their own directory within the same parent repo.

**Structure choice:**

I prefer to use polyrepos for this project to:

* Separate concerns.
* Differentiate the frontends as two independent projects that can get data from backend services. This project’s backend is just one of many options that the frontends can use.
* Think of the backend as a service providing APIs that blogs can use to create, display, and manage posts and users’ comments.

# Task

1. Design the back-end’s model and schema. Consider the following during the design process:

* Your blog should have posts and comments, so think about the fields you are going to want to include for each of those.
* Are you going to require users to leave a username or email with their comments?
* Are you going to display a date or a timestamp for posts and comments?
* Posts should probably have a title, but should comments?
* A useful feature for a blog is the ability to have posts that are in the database but not published for the public to read. How might you designate published vs unpublished posts in your DB?
* You will want a user model that will contain any blog authors and any normal user accounts. Even if you decide to only have a single author and no normal user accounts, a minimal user model will still be helpful to allow for easier route protection via authentication.

1. Set up your Express app
2. Use Prisma to define your models.
3. Use RESTful organization to set up your routes and controllers.
4. Test your routes using the curl cli-tool or route testing web applications like Postman.
5. Use JWT authentication to protect specific routes like the edit path and the routes for only logged-in users. Note the following:

* You can use [jsonwebtoken](https://github.com/auth0/node-jsonwebtoken) to create and verify JWTs.
* You can use [Passport’s JWT strategy](https://github.com/mikenicholson/passport-jwt) for verifying JWTs, especially if you are handling logging in with Passport’s local strategy.
* A successful login will grant the user a JWT. That user can then attach their JWT to any future requests, where your backend can verify the JWT to allow or deny access to the rest of the protected route. When the user logs out, you can have the client remove the JWT from storage.
* There are many ways to send and store JWTs, such as via cookies, storing in localStorage, using access/refresh tokens etc. Some of these methods are more complicated (though with the right implementation, potentially more secure), especially once you deploy both ends. For example, cross-site cookies can be a real headache if you aren’t aware of certain extra details. You may wish to explore some of these alternatives in the future. For now, keep it simple and send your JWTs via an “Authorization” header with “Bearer” schema, and have the client store a JWT in localStorage.

1. Once your API is working you can focus on your front-end code by creating the blog consumption website that allows people to read and comment on your posts. How you go about this is up to you. If you are comfortable with a front-end framework then go for it! If you’re happier using plain HTML and CSS that’s fine too. All you need to do to get your posts onto a website is to fetch the correct API endpoint and then display the results.
2. Create a second website (the blog editing website) for authoring and editing your posts. You can set this up however you like but the following features might be useful:

* A list of all posts that shows each one’s publication status (published or unpublished).
* A button to publish unpublished posts, or to unpublish published ones!
* A ‘NEW POST’ form. If you want to get fancy, you could use a rich text editor such as [TinyMCE](https://www.tiny.cloud/docs/tinymce/6/cloud-quick-start/).
* The ability to manage comments (i.e. delete or edit them).

1. Deploy your backend using PaaS (Platform as a service) hosting platforms like Koyeb or Railway.
2. Deploy your frontends using PaaS hosting platforms like Netlify or Vercel.

# User privileges

* **Fan:** Unauthenticated user (Read-only privileges)
* **Staff:** Employed member (Read, compose, and personal content management privileges)
* **Admin:** An administrator (All privileges except updating staff comments)

|  |  |  |  |
| --- | --- | --- | --- |
| Privilege | Fan | Staff | Admin |
| Create posts | No | Yes | Yes |
| Read published posts | Yes | Yes | Yes |
| Update personal posts | No | Yes | Yes |
| Update any post | No | No | Yes |
| Delete personal posts | No | Yes | Yes |
| Delete any post | No | No | Yes |
| Create comments | No | Yes | Yes |
| Read comments | Yes | Yes | Yes |
| Update personal comments | No | Yes | Yes |
| Update any comment | No | No | No |
| Delete personal comments | No | Yes | Yes |
| Delete any comment | No | No | Yes |
| Delete users | No | No | Yes |

# Applications objectives

## Consumption website

The consumption website is for displaying all the published posts in the database while allowing authenticated users to add comments to the posts. It also gives signed-in users direct access to their personal dashboard where they can manage their own posts.

* **Homepage:**
* List all published posts as clickable items that links to each post’s page for reader to view or add comments.
* Allow user to sign up/login/logout
* **Sign up:** Allow user to create a new account as an author or admin.
* First name
* Lastname
* Username (unique)
* Email (unique)
* Password
* Admin passcode
* **Log in:** Allow users to log in to their account.
* Email
* Password
* **Blog page**
* Allow all users to view blog and comments
* Allow authenticated user to add comments
* **Create dashboard link/button**
* Link to editing website to allow user to create new post or manage existing ones

## Editing website

The editing website is a management space where authors can manage their posts and admins can manage roles and posts.

* **Homepage:**
* List all the user’s posts
* Add delete buttons to each post item
* Add editing UI to each post item that links to each post’s editing page
* Provide a navigation UI to the new post creation page
* Provide a navigation UI to the members management page
* **Blog creation page**
* Allow user to create new posts as draft or live publication
* Add publish/unpublish UI to each post item
* **Blog Editing page**
* Allow user to edit their draft or published posts
* Add publish/unpublish UI to each post item
* **Members page (admin only)**
* List all members and their statuses
* Allow admins to manage staff privileges

## Backend

Provide RESTful APIs that blogs can use to:

* Create users with statuses that determines what they can do on the blog.
* Allow authenticated users to contribute to the blog.
* Allow administrators to manage users’ access to the restricted sections of blog.

# Models

## users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | first\_name | last\_name | email | password |
| 1 | Oluwatobi | Sofela | contact@codesweetly.com | test |
| 2 | Sarah | Precious | sp@example.com | example |
| 3 | Dav | Emma | de@sample.com | sample |

## statuses

|  |  |
| --- | --- |
| id | name |
| 1 | Staff |
| 2 | Admin |

## posts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | title | text | published | published\_date |
| 1 | The beginning | Video provides a powerful way to help you prove your point. | false | null |
| 2 | Something wonderful | Lorem ipsum dolor sit amet, consectetuer adipiscing elit. | true | 2024-11-29 |
| 3 | Time and seconds | When you click Online Video, you can paste in the embed code for the video you want to add. | false | null |

### comments

|  |  |  |  |
| --- | --- | --- | --- |
| id | text | date\_created | date\_updated |
| 1 | Lorem ipsum dolor sit amet, consectetuer adipiscing elit. | 2024-11-29 | 2024-12-02 |
| 2 | Maecenas porttitor congue massa. | 2024-12-02 | 2024-12-10 |

### user\_status

|  |  |  |
| --- | --- | --- |
| id | user\_id | status\_id |
| 1 | 3 | 1 |
| 2 | 2 | 2 |

### user\_post

|  |  |  |
| --- | --- | --- |
| id | user\_id | post\_id |
| 1 | 2 | 1 |
| 2 | 3 | 2 |

### user\_comment

|  |  |  |
| --- | --- | --- |
| id | user\_id | comment\_id |
| 1 | 3 | 2 |
| 2 | 1 | 1 |

### post\_comment

|  |  |  |
| --- | --- | --- |
| id | post\_id | comment\_id |
| 1 | 2 | 1 |
| 2 | 1 | 2 |