Paste House

https://github.com/Syd-/paste

Introduction

Paste House allows users to paste snippets of text into a textbox. They are then given a convenient short link to share the text they just pasted. This allows people to share text, code, etc. on online messengers like Facebook Messenger, Whatsapp, Twitter, etc.

The features I provide are:
1. A text area to paste text into, and a submit button.
2. Being given a link to the paste upon hitting the submit button.
3. Being able to visit the link and be presented the pasted text.
4. Expiring the paste in 2 days (To reuse shortlinks, as there are only so many characters).

Design and Implementation

The **Front End** is a single page app written in Angular.

It is hosted at http://siddhant.name/paste/, on an apache based server.
I had to create a **.htaccess** file (found in aws_lambda/) that would rewrite all urls to **index.html**, but didn't do so for .js, .css, .ico and .txt files in that directory.
It follows the UI Wireframe design, and implements the minimum features I had chosen.

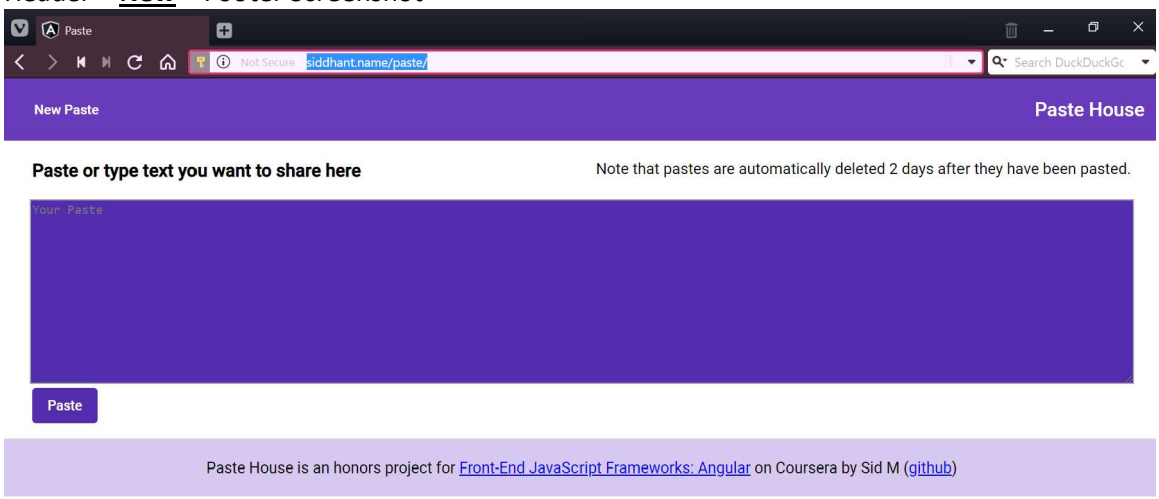
It has 4 main components:
1. **Header**: Here, I learned how to use router in the html template to show & hide a button.
2. **Footer**: Bog-standard. Nothing much to see here.
3. **New**: I learned to use router.navigate to move away on success.
4. **Existing**: This uses the **ng-clipboard** modue to allow us to copy to clipboard. It also displays ids, text, loading spinner and error messages rather interestingly.

It has 1 main services:
1. **PasteService**: An issue I bumped into here was that the **http.get** was expecting a json, while S3 was returning text. The solution was to use { r esponseType: ' t ext ' as ' t ext ' } .

It has 3 other providers:
1. **ProcessHttpMsgService**: Standard. Nothing much to see here.
2. **ReadURL**: exported from app/shared/urls.ts, which points to S3.
3. **WriteURL**: exported from app/shared/urls.ts, which points to API gateway to PUT into.

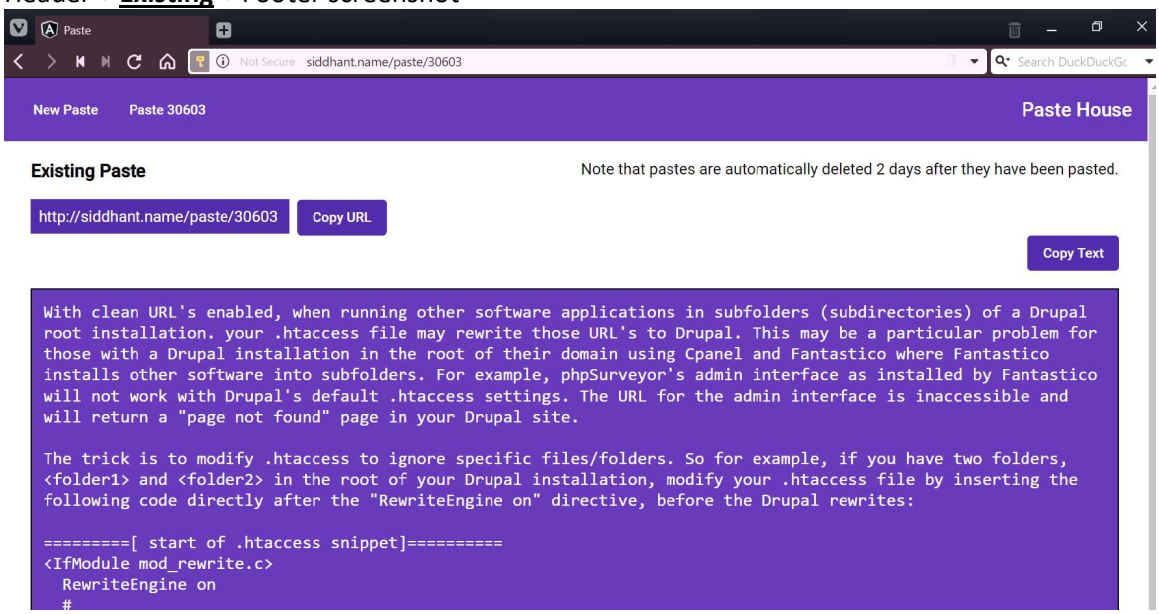
It also defines Paste object with id and text properties, and a file with animations.

Header + **New** + Footer Screenshot



As can be seen in the screenshot above, there is no "Paste :id" button in the header. In addition, your cursor will be automatically in the textbox on page load, making it even more convenient to use. I use simple animations to bring up the text, spinner (during loads) and error message (when there are errors).

Header + **Existing** + Footer screenshot



On a successful paste, the user is navigated to the above page. Note how the header now has an addition button.
The user can now click the **Copy URL** button to copy it to clipboard to share with their friends. Alternatively, they can highlight the URL and copy it (or highlight it from the browser URL bar).
The user can aso **Copy Text** to their clipboard with a single button click. This is more convenient that highlighting all the pasted text and then copying, especially if there is a lot of text being pasted. (As in the example above).

If a user visits the Copied URL, they will get the page above.
Finally, if a user visits a URL that doesn't exist, they will get an error with 403 Forbidden.

A laundry list of modules used here are: MatProgressSpinnerModule, MatButtonModule. ReactiveFormsModule, HttpClientModule, MatToolbarModule, ClipboardModule, AppRoutingModule, BrowserModule, BrowserAnimationsModule

Most of these are for Material UI, or general behaviour, and for routing.

The **Back End** is a bunch of services hosted on AWS

1. **S3** is a convenient and cheap data store. While we could store the text in a DB, by storing it in S3, we can store arbitrarily long text without caring about DB performance. In addition, S3 is a ***lot*** cheaper than a DB. S3 also serves as static site hosting; which is an end point our front-end can hit to get existing pastes.

2. **DynamoDB** gives us a place to store paste id and TTLs. DynamoDB has a built-in expiration that automatically deletes rows when they're older than the timestamp in the TTL. It also triggers an event on deletion, which can be used to delete the corresponding file from S3.

3. **AWS Lambda** serves as the glue with two functions.
3.1 **pasteDelete** listens to events from the DynamoDB table, and when there's a DELETE event, it deletes the corresponding file with the id from S3.
3.2 **pastePost** listens to an event from a source. It generates a unique id, makes an entry in DynamoDB with the TTL set, and saves the text to S3 from the source. Finally, it returns the id.

4. **API gateway** gives us a public endpoint to **PUT** pastes to. It routes the PUT events into 3.2.

We have to set up the CORS for 1. & 4. appropriately, and set up an **IAM** role for 3. that gives Lambda access to S3 and DynamoDB. See docs/backend.md on github for more details.

Conclusions

- I have a working app that provides the basic functionality for a fully operational paste site.
- I have implemented all the features I set out to do (as can be seen in the ideation and the UI).
- I have followed the wireframe almost verbatim.
- I had a lot of issues with getting things to work in their environments, because it wasn't well documented or mentioned. For instance, one had to use

```
ng build --prod --base-href /paste/ --deploy-url /paste/
```

to create a deployment for a subdirectory.
- I had trouble wrangling CORS, .htaccess, other similar issues during testing and deploying.
- I had to downgrade the Node version on AWS Lambda to 8.4 because the 10.x version didn't have the uuid/v4 library anymore (used in **pastePost**).
- I wish I had more time to (1) make more useful error messages (2) write tests for the app.

Given all the above, I believe that I made the right decision not stretching myself and doing just the bare minimum; thus having a working app at the end of 4 weeks!

References

- https://github.com/Syd-/paste
- http://siddhant.name/paste/
- https://www.npmjs.com/package/ngx-clipboard