

Final Project Documentation

Maya Bowman, Stella Craig, Phyllis Kan, Michael Sturm, Upasana Roy

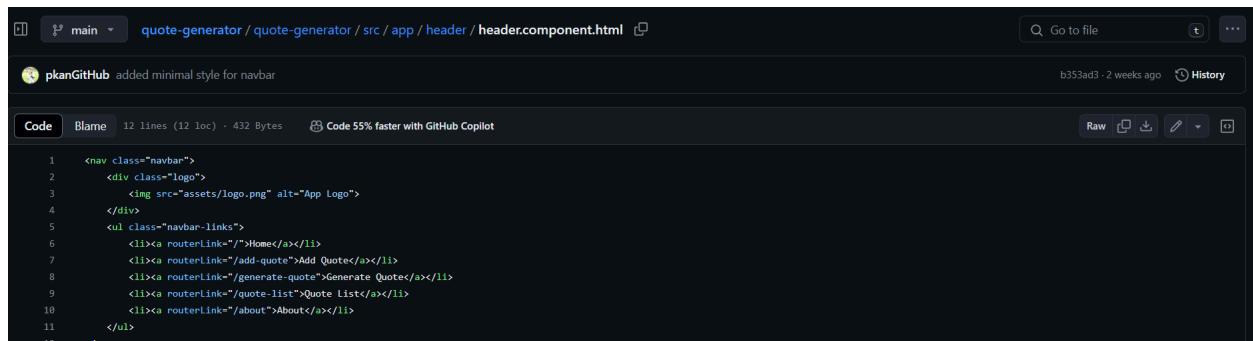
Github: <https://github.com/pkanGitHub/quote-generator>

Presentation: https://youtu.be/R2lwU_YLw6o

To run the project use “npm run dev”.

Angular

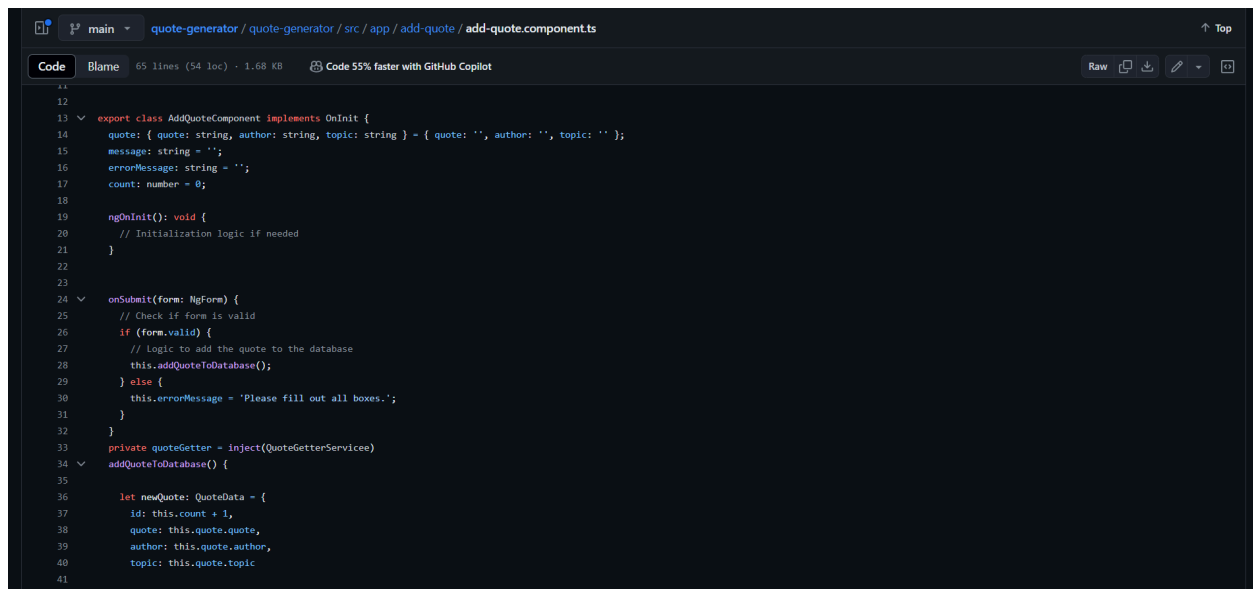
In order to meet the Angular requirement, we created a header component, footer component, and various other body components like about, add-quote, gen-quote, home, quote-list, and profile card. We used the header to route between all of the different pages as shown here:



The screenshot shows a GitHub repository for 'quote-generator' with the file 'header.component.html' selected. The file contains HTML code for a navigation bar with links to Home, Add Quote, Generate Quote, Quote List, and About. The code is as follows:

```
1 <nav class="navbar">
2   <div class="logo">
3     
4   </div>
5   <ul class="navbar-links">
6     <li><a routerLink="/">Home</a></li>
7     <li><a routerLink="/add-quote">Add Quote</a></li>
8     <li><a routerLink="/generate-quote">Generate Quote</a></li>
9     <li><a routerLink="/quote-list">Quote List</a></li>
10    <li><a routerLink="/about">About</a></li>
11  </ul>
12 </nav>
```

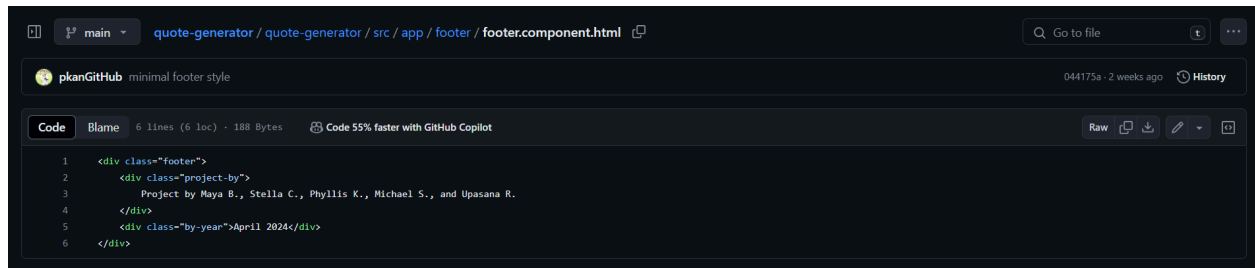
We used the body components to send information to each other shown here in the onSubmit function:



The screenshot shows a GitHub repository for 'quote-generator' with the file 'add-quote.component.ts' selected. The file contains TypeScript code for the AddQuoteComponent, including an ngOnInit method and an onSubmit method. The code is as follows:

```
11
12
13 export class AddQuoteComponent implements OnInit {
14   quote: { quote: string, author: string, topic: string } = { quote: '', author: '', topic: '' };
15   message: string = '';
16   errorMessage: string = '';
17   count: number = 0;
18
19   ngOnInit(): void {
20     // Initialization logic if needed
21   }
22
23   onSubmit(form: NgForm) {
24     // Check if form is valid
25     if (form.valid) {
26       // Logic to add the quote to the database
27       this.addQuoteToDatabase();
28     } else {
29       this.errorMessage = 'Please fill out all boxes.';
30     }
31   }
32
33   private quoteGetter = inject(QuoteGetterService)
34   addQuoteToDatabase() {
35
36     let newQuote: QuoteData = {
37       id: this.count + 1,
38       quote: this.quote.quote,
39       author: this.quote.author,
40       topic: this.quote.topic
41     }
42   }
```

In our footer we put the names of each of the group members:

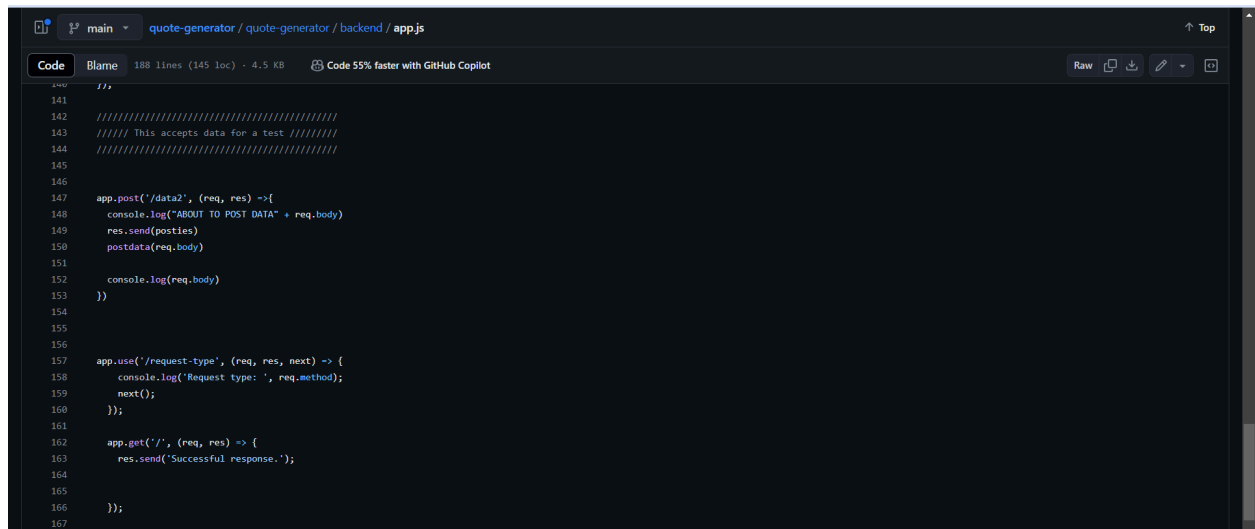


A screenshot of a GitHub code viewer interface. The breadcrumb path is 'quote-generator / quote-generator / src / app / footer / footer.component.html'. The file name is 'minimal footer style', last updated '044175a · 2 weeks ago'. The code is 6 lines (6 loc) and 188 Bytes. The code content is:

```
1 <div class="footer">
2   <div class="project-by">
3     Project by Maya B., Stella C., Phyllis K., Michael S., and Upasana R.
4   </div>
5   <div class="by-year">April 2024</div>
6 </div>
```

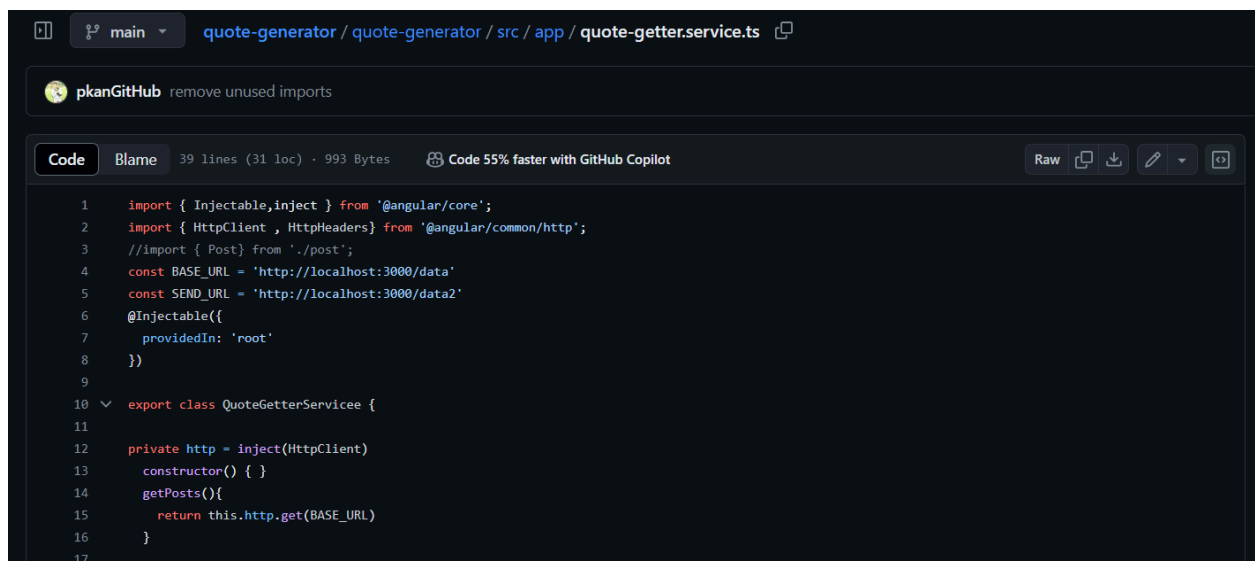
NodeJS

In order to meet the requirements for NodeJS, we built our backend in NodeJS and used HTTP for the connection to the data between our frontend and backend.



A screenshot of a GitHub code viewer interface. The breadcrumb path is 'quote-generator / quote-generator / backend / app.js'. The file name is 'app.js', last updated '044175a · 2 weeks ago'. The code is 188 lines (145 loc) and 4.5 KB. The code content is:

```
141 //,
142 //,
143 //,
144 //,
145 //,
146 //,
147 app.post('/data2', (req, res) => {
148   console.log("ABOUT TO POST DATA" + req.body)
149   res.send(postles)
150   postdata(req.body)
151
152   console.log(req.body)
153 })
154
155
156
157 app.use('/request-type', (req, res, next) => {
158   console.log("Request type: ", req.method);
159   next();
160 });
161
162 app.get('/', (req, res) => {
163   res.send('Successful response. ');
164
165
166 });
167
```

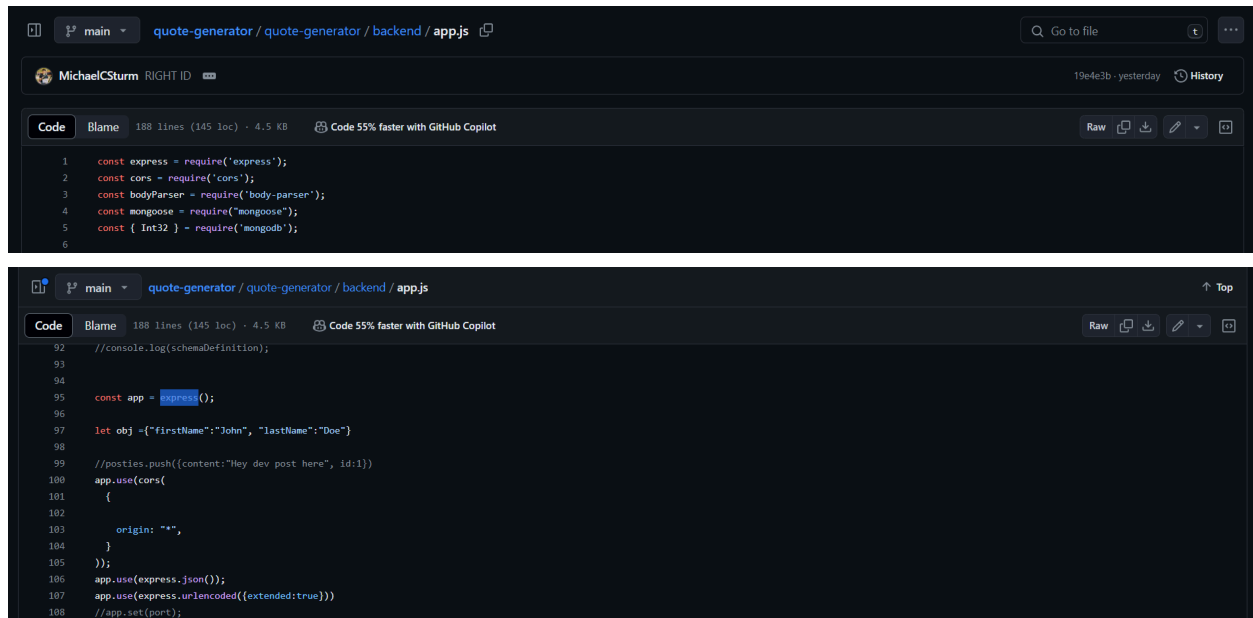


A screenshot of a GitHub code viewer interface. The breadcrumb path is 'quote-generator / quote-generator / src / app / quote-getter.service.ts'. The file name is 'quote-getter.service.ts', last updated '044175a · 2 weeks ago'. The code is 39 lines (31 loc) and 993 Bytes. The code content is:

```
1 import { Injectable, inject } from '@angular/core';
2 import { HttpClient, HttpHeaders } from '@angular/common/http';
3 //import { Post } from './post';
4 const BASE_URL = 'http://localhost:3000/data'
5 const SEND_URL = 'http://localhost:3000/data2'
6 @Injectable({
7   providedIn: 'root'
8 })
9
10 export class QuoteGetterService {
11
12   private http = inject(HttpClient)
13   constructor() { }
14   getPosts(){
15     return this.http.get(BASE_URL)
16   }
17 }
```

Express.js

For the Express.js requirement, we used express to create a restful API between our backend and service.ts:



```
1  const express = require('express');
2  const cors = require('cors');
3  const bodyParser = require('body-parser');
4  const mongoose = require('mongoose');
5  const { Int32 } = require('mongodb');
6
92  //console.log(schemaDefinition);
93
94
95  const app = express();
96
97  let obj = {"firstName": "John", "lastName": "Doe"}
98
99  //posties.push({content: "Hey dev post here", id:1})
100  app.use(cors({
101    {
102      origin: "",
103    }
104  }));
105
106  app.use(express.json());
107  app.use(express.urlencoded({extended:true}))
108  //app.set(port);
```

All of our communication and data transfer was done in our http from node as shown in the requirements for NodeJS above.

MongoDB



```
45  async function postdata(newData){
46    posties[0].push(newData)
47
48    const newestData = new Quotes({
49
50      id: posties[0].length+1,
51      quote: newData.quote,
52      author: newData.author,
53      topic: newData.topic})
54
55    ///console.log(JSON.stringify(newestData))
56    //newestData.id = posties.length+1
57    try{
58      await client.connect() => {
59        const database = client.db("Quote_Generator");
60        const collection = database.collection("Quotes");
61        db.collection.insertOne(newestData);
62        console.log("WE IN THE CLINET BBY")
63      })
64      disquotes = await collection.insertOne(newestData);
65
66      newestData.save().then(savedData => {
67        console.log('Data saved successfully:', savedData)})
68    }
69    catch(e){
70      console.log(e)
71    }
```

Using mongo we allowed all IP's to be able to get and post to the db. We set it up so the service automatically calls the database to put into an array. We use this array to avoid the Onlnit async waiting for db error. We also use this array to find the number of quotes in quotes to assign an id that counts up. We also have a quotes Schema to format the data to a pushable state.

Routing.ts

To meet the Routing requirement, we created the navbar using RouterModule, a built-in Angular module for handling routing. The RouterModule is imported into the main application module. Each route in the RouterModule consists of a path and a component field for all components we created, such as home, gen_quote, add_quote, quote_list, and about.

```
quote-generator > src > app > TS app-routing.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3  import { HomeComponent } from './home/home.component';
4  import { AddQuoteComponent } from './add-quote/add-quote.component';
5  import { GenQuoteComponent } from './gen-quote/gen-quote.component';
6  import { QuoteListComponent } from './quote-list/quote-list.component';
7  import { AboutComponent } from './about/about.component';
8
9  const routes: Routes = [
10     {path: '', component: HomeComponent, pathMatch: 'full'},
11     {path: 'add-quote', component: AddQuoteComponent},
12     {path: 'generate-quote', component: GenQuoteComponent},
13     {path: 'quote-list', component: QuoteListComponent},
14     {path: 'about', component: AboutComponent},
15 ];
16
17 @NgModule({
18   imports: [RouterModule.forRoot(routes)],
19   exports: [RouterModule]
20 })
21 export class AppRoutingModule { }
```

To link up the routes in the header component, we used HTML list elements such as <nav>, , for each route link. And for each element, it would contain an Angular router link directive, pointing to the corresponding path defined in the RouterModule.

```
<nav class="navbar">
  <div class="logo">
    
  </div>
  <ul class="navbar-links">
    <li><a routerLink="/">Home</a></li>
    <li><a routerLink="/add-quote">Add Quote</a></li>
    <li><a routerLink="/generate-quote">Generate Quote</a></li>
    <li><a routerLink="/quote-list">Quote List</a></li>
    <li><a routerLink="/about">About</a></li>
  </ul>
</nav>
```

Service.js

It's an injectable that has loadPosts which posts the quote to the DB using mongo and getPosts which gets the data. This injectable is injected into addQuoteComponent, QuoteListComponent and GenQuoteComponent. This helps the project be very scalable.

Component.css

Our application contains a proper layout using css padding and margins. Our application is visually symmetric using css with the components.

App.component.css:

```
quote-generator > src > app > # app.component.css > ...
Phyllis Kan, 2 weeks ago | 1 author (Phyllis Kan)
1  html, body {
2      height: 100%;
3      margin: 0;
4  }
5
6  main {
7      display: flex;
8      min-height: 100vh;
9      flex-direction: column;
10 }
11
12 .content {
13     flex: 1;
14 }
```

quote-list.component.css:

```
quote-generator > src > app > quote-list > # quote-list.component.css > ...
You, yesterday | 1 author (You)
1  /* Styles for the quote list */
2  .quote-list {
3      width: 80%; /* Set the width to 80% of the viewport width */
4      text-align: center; /* Center align the text */
5      margin: 0 auto; /* Center the list horizontally */
6      list-style: none; /* Remove default list styling */
7      padding: 0; /* Remove default padding */
8  }
9
10 h1 {
11     font-weight: bold; /* Make the text bold */
12     text-decoration: underline; /* Underline the text */
13 }
14
15 .quote-item {
16     text-align: center; /* Center align the text */
17     margin-bottom: 20px; /* Space between quotes */
18 }
19
20 .quote-text {
21     color: black; /* Set the text color to black */
22 }
23
24 .author-text {
25     color: grey; /* Set the author's name to grey */
26 }
27
28 .topic-text {
29     color: lightgray;
30 }
```

And the following examples:

```
▼ about
# about.component.css
```

```
▼ add-quote
# add-quote.component.css
```

```
▼ gen-quote
# gen-quote.component.css
```

```
▼ header
# header.component.css
```

```
▼ profile-card
# profile-card.component.css
```

Subjects

We used subjects and subscriptions to handle the data that will be displayed from our database.

```
18  ✓    loadPosts(CurrPost:JSON) {
19
20
21
22
23      console.log(JSON.stringify(CurrPost) + "AYO THIS IS IN QUOTE GETTER SERVICE")
24      const headers = new HttpHeaders()
25          .set('Authorization', 'my-auth-token')
26          .set('Content-Type', 'application/json');
27      //console.log(body);
28      this.http.post(SEND_URL, CurrPost, { headers:headers })
29          .subscribe(data => {
30              console.log(data)
31          });
32      }
33  }
34  // export class Login {
```