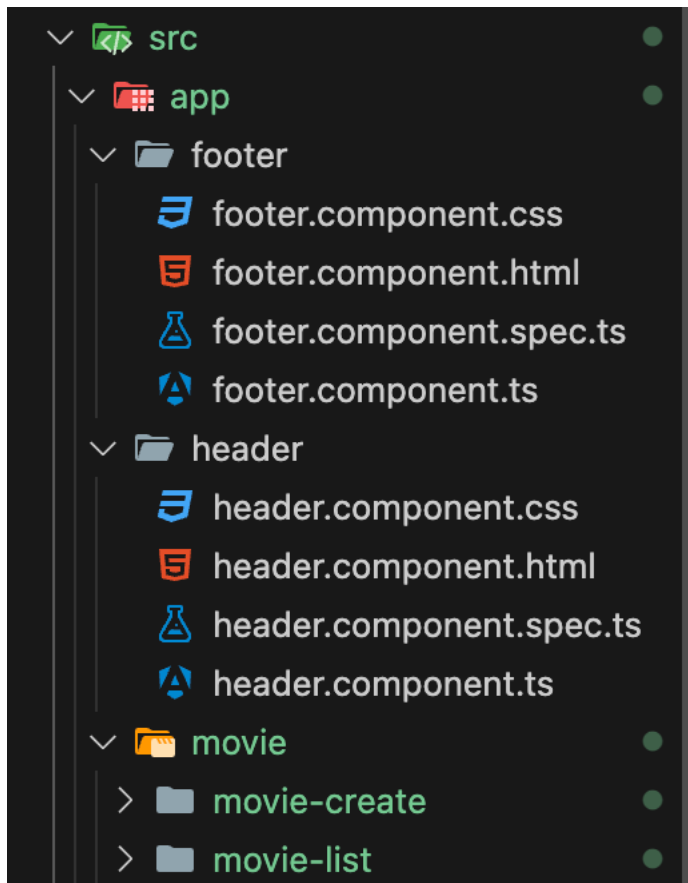


Austin McLain
Tyler Harsell
Adi Pillai


1. Angular

For this requirement, we built a web application using Angular as our frontend. We included 4 components: header, footer, movie, and movie list. Our header component includes routes for a home page and a new review page. Our footer includes our names. The two body components send information between each other.



2. NodeJS

For this requirement, we used NodeJS and HTTP for sending information between the database and the frontend.

```
ackend >  app.js > ...
40 const express = require("express");
39 const cors = require("cors");
38 const bodyParser = require("body-parser");
37 const mongoose = require("mongoose");
36
35 const app = express();
34
33 app.use(cors());
32 app.use(bodyParser.json());
31 app.use(bodyParser.urlencoded({ extended: false }));
30
29 const mongoURI =
28   "mongodb+srv://webdev2:webdev2@cluster1.ugvyl6x.mongodb.net/MovieList?retryWrites=true&w=majority&appName=Cluster1";
27
26 mongoose
25   .connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })
24   .then(() => console.log("MongoDB Connected to MovieList Database"))
23   .catch((err) => console.log(err));
22
21 const movieSchema = new mongoose.Schema({
20   title: String,
19   description: String,
18   rating: Number,
17 });
16
15 // Create a Model connected to 'movies' collection in the 'MovieList' database
14 const Movie = mongoose.model("Movie", movieSchema, "movies");
13
```

3. Express.js

For this requirement, we used Express.js in our backend for the API. For example, our application uses a “/movies” endpoint.

```
5
4 app.get("/movies", async (req, res) => {
3   try {
2     const movies = await Movie.find();
1     res.status(200).json(movies);
0   } catch (error) {
9     res.status(500).send(error);
8   }
7 });
6
5 app.post("/movies", async (req, res) => {
4   const newMovie = new Movie({
3     title: req.body.title,
2     description: req.body.description,
1     rating: req.body.rating,
0   });
9
8   try {
7     const savedMovie = await newMovie.save();
6     res.status(201).json(savedMovie);
5   } catch (error) {
4     res.status(400).json({ message: error.message });
3   }
2 });
1
module.exports = app;
```

4. MongoDB

For this requirement, we used MongoDB for our database.

```
ackend > js app.js > ...
40 const express = require("express");
39 const cors = require("cors");
38 const bodyParser = require("body-parser");
37 const mongoose = require("mongoose");
36
35 const app = express();
34
33 app.use(cors());
32 app.use(bodyParser.json());
31 app.use(bodyParser.urlencoded({ extended: false }));
30
29 const mongoURI =
28   "mongodb+srv://webdev2:webdev2@cluster1.ugvyl6x.mongodb.net/MovieList?retryWrites=true&w=majority&appName=Cluster1";
27
26 mongoose
25   .connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })
24   .then(() => console.log("MongoDB Connected to MovieList Database"))
23   .catch((err) => console.log(err));
22
21 const movieSchema = new mongoose.Schema({
20   title: String,
19   description: String,
18   rating: Number,
17 });
16
15 // Create a Model connected to 'movies' collection in the 'MovieList' database
14 const Movie = mongoose.model("Movie", movieSchema, "movies");
13
```

5. Routing

For this requirement, we added routing to our application so that users can switch between a home page and a page to create new reviews. These links are accessible via the header component.

```
0 > ts app-routing.module.ts > ...
16 import { NgModule } from '@angular/core';
15 import { RouterModule, Routes } from '@angular/router';
14
13 import { PostListComponent } from './app/post/post-list/post-list.component';
12 import { PostCreateComponent } from './app/post/post-create/post.component';
11
10 const routes: Routes = [
9   { path: '', component: PostListComponent },
8   { path: 'create', component: PostCreateComponent }
7 ];
6
5 @NgModule({
4   imports: [RouterModule.forRoot(routes)],
3   exports: [RouterModule]
2 })
1 export class AppRoutingModule { }
7
```

[Home](#) [New Review](#)

6. Service.js

For this requirement, we added a service file to handle data transfer between components.

```
import { Injectable } from '@angular/core';
import { Post } from '../post.model';
import { HttpClient } from '@angular/common/http';
import { Subject } from 'rxjs';

@Injectable({ providedIn: 'root' })
export class PostService {
  private posts: Post[] = [];
  private postsUpdated = new Subject<Post[]>();

  constructor(private http: HttpClient) {}

  getPosts() {
    this.http.get<{message: string, posts: Post[]}>('http://localhost:3000/api/posts')
      .subscribe((postData) => {
        this.posts = postData.posts;
        this.postsUpdated.next([...this.posts]);
      });
  }

  getPostUpdateListener() {
    return this.postsUpdated.asObservable();
  }

  addPost(content: string) {
    const post: Post = { content: content };
    this.posts.push(post);
    this.postsUpdated.next([...this.posts]);
  }
}
```

7. Component.css

For this requirement, we added CSS for the styling.

```
mat-card {
  width: 50%;
  margin: auto;
  float: left;
}
mat-form-field, textarea{
  width: 100%;
}
```

8. Subjects

For this requirement, we used subjects and subscriptions to handle the data from the database.

```
private postsUpdated = new Subject<Post[]>();
```

```
export class PostListComponent implements OnInit, OnDestroy {  
  ⚡posts: Post[] = [];  
  private postsSub: Subscription;  
  
  constructor(public postService: PostService) {}  
  
  ngOnInit() {  
    this.postService.getPosts();  
    this.postsSub = this.postService.getPostUpdateListener()  
      .subscribe((posts: Post[]) => {  
        this.posts = posts;  
      });  
  }  
  
  ngOnDestroy() {  
    this.postsSub.unsubscribe();  
  }  
}
```