Personal Blog Post Manager

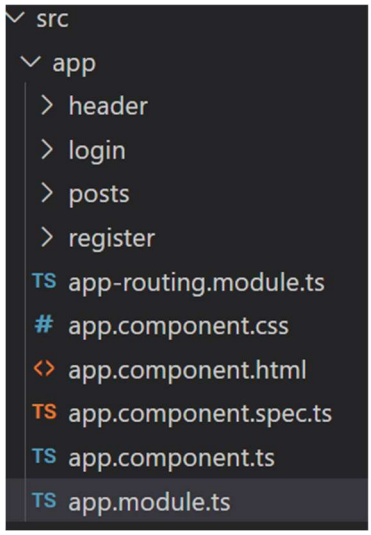
Abstract: -

The blog post manager is a single page application created with Angular JavaScript and TypeScript it is a robust and dynamic with CRUD operations and image verification. There are 4 main components Post, Header, Login, Register. Under post we have post list and post create sub component. In order to edit the created post, we have reused the create component itself for better optimization and efficiency but in edit case we send and Objectid created by Mongo Db which is like and unique and primary key every document in mongo which made the task easier.

Understanding the file directories: -

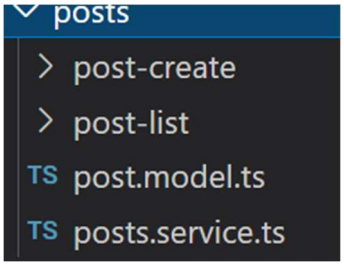
- Module import in the app.module.ts | File directory of angular

```
imports: [
  BrowserModule,
  AppRoutingModule,
  ReactiveFormsModule,
  BrowserAnimationsModule,
  MatInputModule,
  MatCardModule,
  MatButtonModule,
  MatToolbarModule,
  MatExpansionModule,
  MatProgressSpinnerModule,
  MatPaginatorModule,
  HttpClientModule,
  BrowserModule,
  AppRoutingModule,
  BrowserAnimationsModule,
  MatCardModule,
  MatInputModule,
  MatButtonModule,
  FormsModule,
  ReactiveFormsModule,
  MatToolbarModule,
  MatSnackBarModule
],
```



- The file directory has sub components in post component

➔ The post service is used to pass the variable from post create to post list like title and other form fields this plays major role calling the api services from the node server.js

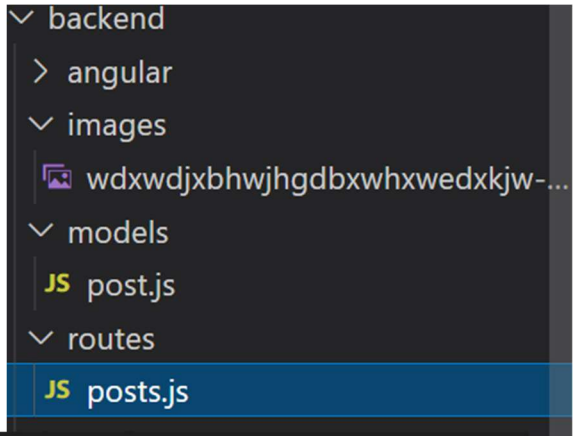


- Node and Mongo server directory: -

➔ Over here we have app.js which contains the routes where the api's are created and the connection to mongo db is made in app.js file it self and then imported to post.js

➔ In the images folder we have the images where the user is selected through the file directory and the file path is stored with respect to ObjectId.

➔ This is a snapshot how the connection is made.




```
const app = express();
const mongoURI =
 "mongodb+srv://root:fnAYHT3kOEebCQS5@cluster0.zjx3h.mongodb.
mongoose.connect(mongoURI, {
useNewUrlParser: true,
useUnifiedTopology: true,
});
if(mongoose){console.log('connected to tata lohitasya db')}
```

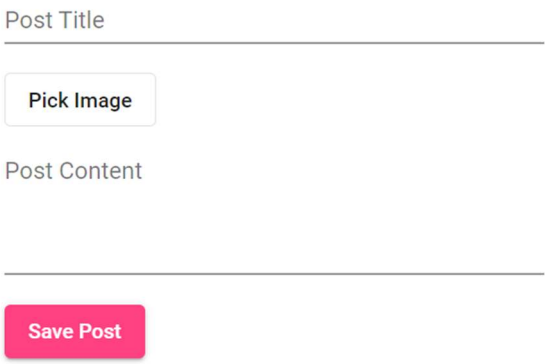
- Understanding the code of each component

➔ The post component

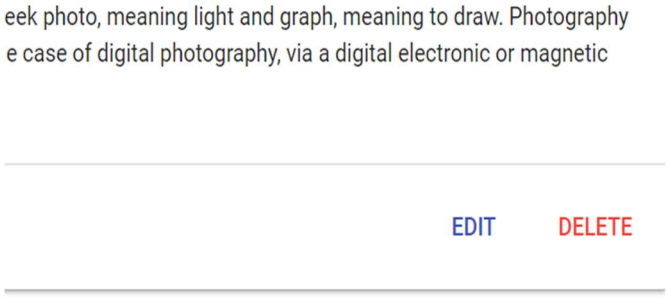
Post list component we have a display of the posts created and fetched form the data base the Api is `http://localhost:3000/api/posts`

- From this api the post create data is saved by post function and then retrived by post list component

New post



Editing the old post



Code for adding post

```
addPost(title: string, content: string, image: File) {
  const postData = new FormData();
  postData.append("title", title);
  postData.append("content", content);
  postData.append("image", image, title);
  this.http
    .post<{ message: string; post: Post }>(
      "http://localhost:3000/api/posts",
      postData
    )
    .subscribe(responseData => {
      this.router.navigate(["/postlist"]);
    });
}
```

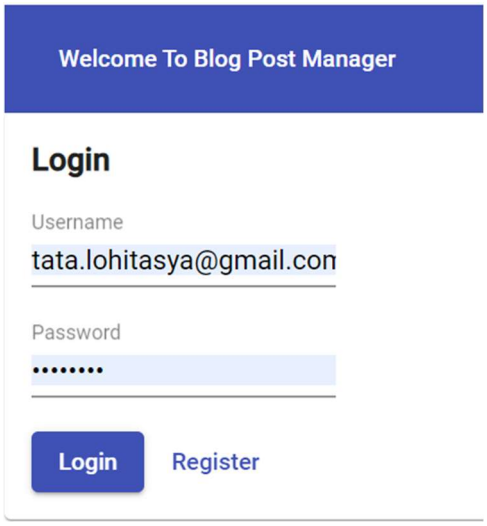
Code for editing the post

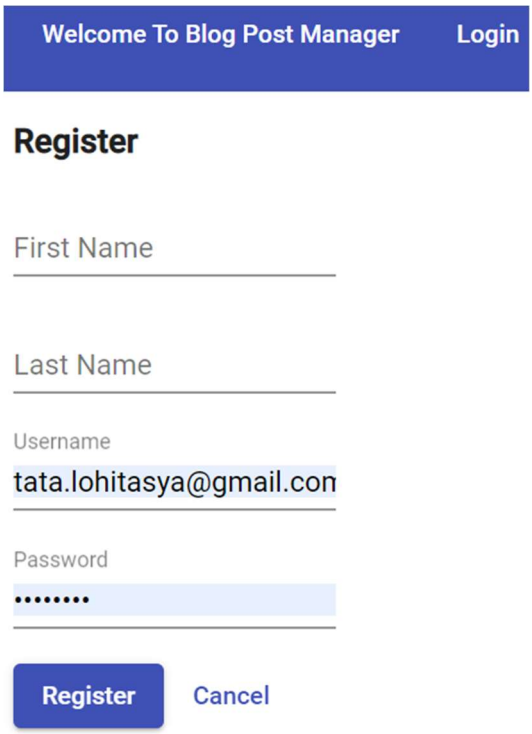
```
updatePost(id: string, title: string, content: string, image: File | string) {
  let postData: Post | FormData;
  if (typeof image === "object") {
    postData = new FormData();
    postData.append("id", id);
    postData.append("title", title);
    postData.append("content", content);
    postData.append("image", image, title);
  } else {
    postData = {
      id: id,
      title: title,
      content: content,
      imagePath: image
    };
  }
  this.http
    .put("http://localhost:3000/api/posts/" + id, postData)
    .subscribe(response => {
      this.router.navigate(["/"]);
    });
}
```

➔ Displaying the created posts

- There is pagination also in order to display the number of posts according to requirement.

- The Login and sign component





➔ This involves the basic functionality with field validation and verification.

- Routing Module

```
const routes: Routes = [
  { path: '', component: RegisterComponent },
  { path: 'postlist', component: PostListComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'create', component: PostCreateComponent },
  { path: 'edit/:postId', component: PostCreateComponent },
  { path: 'login', component: LoginComponent }
];
```

  - These are all the routed and valid pages used in this case study.

BACKEND

➔ App,js file

```
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use("/images", express.static(path.join("backend/images")));

app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept"
  );
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET, POST, PATCH, PUT, DELETE, OPTIONS"
  );
  next();
});

app.use("/api/posts", postsRoutes);

module.exports = app;
```

This functions is used to perform basic operations like CRUD with post, get, put, patch.

➔ Understanding the API in routes

To create the image

```
➔ const storage = multer.diskStorage({
➔   destination: (req, file, cb) => {
➔     const isValid = MIME_TYPE_MAP[file.mimetype];
➔     let error = new Error("Invalid mime type");
➔     if (isValid) {
➔       error = null;
➔     }
➔     cb(error, "backend/images");
➔   },
➔   filename: (req, file, cb) => {
➔     const name = file.originalname
➔       .toLowerCase()
➔       .split(" ")
➔       .join("-");
➔     const ext = MIME_TYPE_MAP[file.mimetype];
➔     cb(null, name + "-" + Date.now() + "." + ext);
➔   }
➔ });
```

Post creation

```
router.post(
  "",
  multer({ storage: storage }).single("image"),
  (req, res, next) => {
    const url = req.protocol + "://" + req.get("host");
    const post = new Post({
      title: req.body.title,
      content: req.body.content,
      imagePath: url + "/images/" + req.file.filename
    });
    post.save().then(createdPost => {
      res.status(201).json({
        message: "Post added successfully",
        post: {
          ...createdPost,
          id: createdPost._id
        }
      });
    });
  }
);
```

Updating the post

```
router.put(
  "/:id",
  multer({ storage: storage }).single("image"),
  (req, res, next) => {
    let imagePath = req.body.imagePath;
    if (req.file) {
      const url = req.protocol + "://" + req.get("host");
      imagePath = url + "/images/" + req.file.filename;
    }
    const post = new Post({
      _id: req.body.id,
      title: req.body.title,
      content: req.body.content,
      imagePath: imagePath
    });
    console.log(post);
    Post.updateOne({ _id: req.params.id }, post).then(result => {
      res.status(200).json({ message: "Update successful!" });
    });
  }
);
```

Fetching the posts

```
router.get("", (req, res, next) => {
  const pageSize = +req.query.pagesize;
  const currentPage = +req.query.page;
  const postQuery = Post.find();
  let fetchedPosts;
  if (pageSize && currentPage) {
    postQuery.skip(pageSize * (currentPage - 1)).limit(pageSize);
  }
  postQuery
    .then(documents => {
      fetchedPosts = documents;
      return Post.count();
    })
    .then(count => {
      res.status(200).json({
        message: "Posts fetched successfully!",
        posts: fetchedPosts,
        maxPosts: count
      });
    });
});
```

Deleting the posts

```
router.delete("/:id", (req, res, next) => {
  Post.deleteOne({ _id: req.params.id }).then(result => {
    console.log(result);
    res.status(200).json({ message: "Post deleted!" });
  });
});
```

API testing link

https://documenter.getpostman.com/view/16807360/UVeGrkuk