Avontece Cabrera

SWDV 691

3/27/2021

Service layer Re-Design

***I added an updated MVP section to include the log in/sign up service and the stretch feature to include the like post service.**

The service layer will be used to listen for and send responses and interact with the database when the administrator is uploading, editing or deleting blog posts.

Technology Stack: JavaScript , React & Redux (make request), Node (parse request), Express (generate database request), MongoDB Atlas (store data), and Heroku (deploy).

The following structure will be used:

• **getPOSTs** – (Retrieve all the posts that we currently have in the database)

o Request and response:

```
export const getPosts = async (req, res) => {
  try {
    const postMessages = await PostMessage.find();
     res.status(200).json(postMessages);
  } catch (error) {
    res.status(404).json({ message: error.message });
  }
}
```

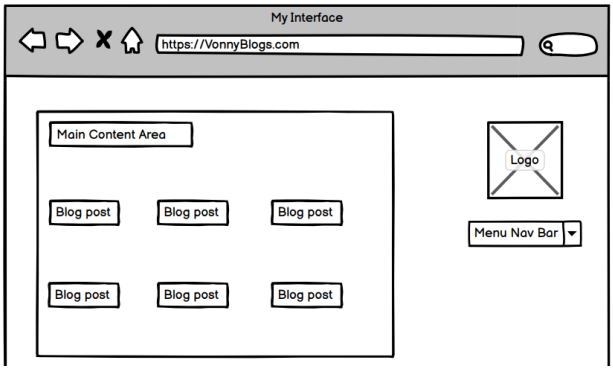
- interact with api (showing the functions returning the actions requested using redux thunk):

```
export const getPosts = () => async (dispatch) => {
 try {
  const { data } = await api.fetchPosts(); ( this is us fetching the data from the api to retrieve all the posts)

  dispatch({ type: FETCH_ALL, payload: data }); ( this is an object that includes the type ( fetch all) , we are sending the data through the action payload aka our actual posts)
 } catch (error) {
  console.log(error.message);
 }
};
```

▪ **Code 200: Successful operation**

▪ **Code 404: The server has not found anything matching the request**

o UI Diagram:



• **createPost** (create a new post)

o Request and response:

```
export const createPost = async (req, res) => {
  const { title, message, selectedFile, creator, tags } = req.body;

  const newPostMessage = new PostMessage({ title, message, selectedFile, creator, tags })

  try {
    await newPostMessage.save();

    res.status(201).json(newPostMessage );
  } catch (error) {
    res.status(409).json({ message: error.message });
  }
}
```

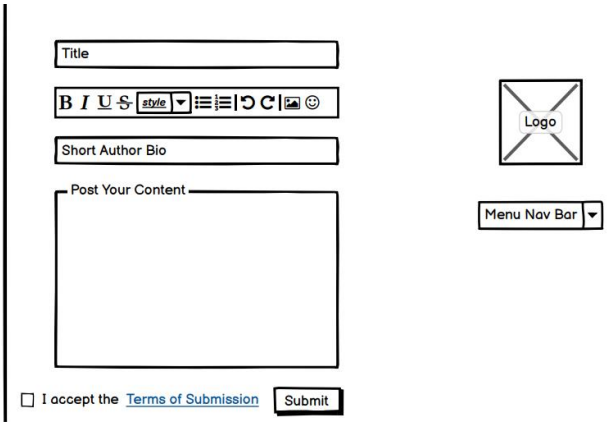
➢ interact with api (showing the functions returning the actions requested using redux thunk):

```
export const createPost = (post) => async (dispatch) => {
 try {
  const { data } = await api.createPost(post); (making a create post api request to our back-end server)

  dispatch({ type: CREATE, payload: data }); (dispatching that CREATE action request)
 } catch (error) {
  console.log(error.message);
 }
};
```

▪ **Code 201: The request has been fulfilled and resulted in a new resource being created.**

▪ **Code 409: The request could not be completed due to a conflict**

o UI Diagram:



• **updatePost** (making an update to a post)

o Request and response:

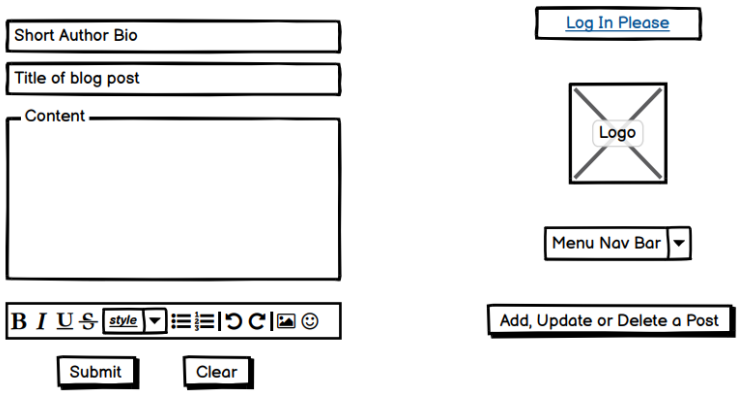
```
export const updatePost = async (req, res) => {
   const { id } = req.params; (extracting the posts mongoose id)
   const { title, message, creator, selectedFile, tags } = req.body;

   if (!mongoose.Types.ObjectId.isValid(id)) return res.status(404).send(`No post with id: ${id}`); (no post with that id in the mongoose database was found)

   const updatedPost = { creator, title, message, tags, selectedFile, _id: id };

   await PostMessage.findByIdAndUpdate(id, updatedPost, { new: true }); (receiving the updated post/ updating in the database)

   res.json(updatedPost);
}
```

- interact with api (showing the functions returning the actions requested using redux thunk):

```
export const updatePost = (id, post) => async (dispatch) => {
 try {
   const { data } = await api.updatePost(id, post);

   dispatch({ type: UPDATE, payload: data });
 } catch (error) {
   console.log(error.message);
 }
};
```

▪ **Code 404: The server has not found anything matching the request**

o UI Diagram:



• **deletePost** (deleting a blog post)

o Request and response:

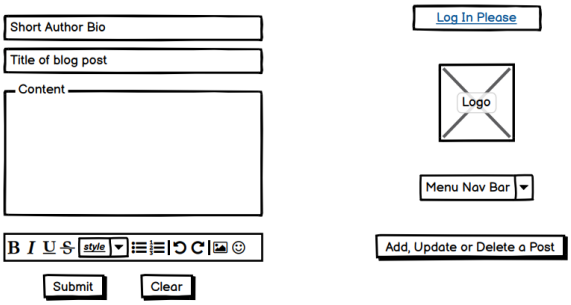
```
export const deletePost = async (req, res) => {
  const { id } = req.params;

  if (!mongoose.Types.ObjectId.isValid(id)) return res.status(404).send(`No post with id: ${id}`);

  await PostMessage.findByIdAndRemove(id);

  res.json({ message: "Post deleted successfully." });
}
```

- interact with api (showing the functions returning the actions requested using redux thunk):

```
export const deletePost = (id) => async (dispatch) => {
 try {
  await api.deletePost(id);

  dispatch({ type: DELETE, payload: id });
 } catch (error) {
  console.log(error.message);
 }
};
```

▪ **Code 404: The server has not found anything matching the request**

UI Diagram:



Additional MVP service for signing up and logging in

• **signUp** (signing the admin user up)

o Request and response:

```
export const signup = async (req, res) => {
  const { email, password, firstName, lastName } = req.body;

  try {
    const oldUser = await UserModal.findOne({ email });

    if (oldUser) return res.status(400).json({ message: "User already exists" });

    const hashedPassword = await bcrypt.hash(password, 12);

    const result = await UserModal.create({ email, password: hashedPassword, name: `${firstName} ${lastName}` });

    const token = jwt.sign( { email: result.email, id: result._id }, secret, { expiresIn: "1h" } );

    res.status(201).json({ result, token });
  } catch (error) {
    res.status(500).json({ message: "Something went wrong" });

    console.log(error);
  }
};
```

- interact with api:

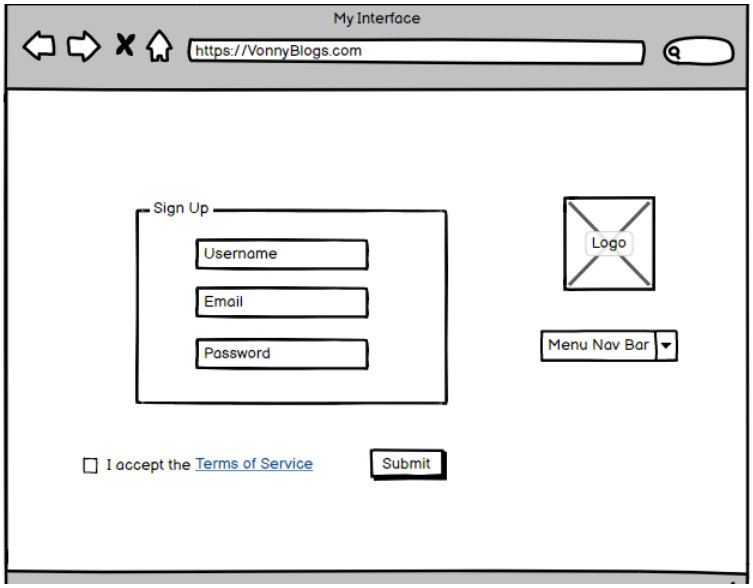
```
export const signup = (formData, router) => async (dispatch) => {
  try {
    const { data } = await api.signUp(formData);

    dispatch({ type: AUTH, data });

    router.push('/');
  } catch (error) {
    console.log(error);
  }
};
```

▪ **Code 201: The request has been fulfilled and resulted in a new resource being created.**

▪ **Code 400: The request could not be understood by the server**

▪ **Code 500: The server encountered an unexpected condition which prevented it from fulfilling the request**

UI Diagram:



• **signIn** (logging in)

o Request and response:

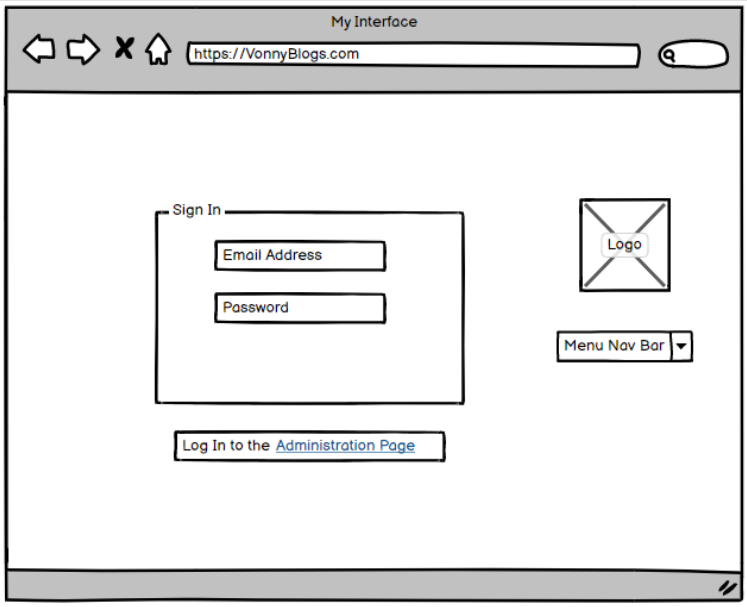
```
export const signin = async (req, res) => {
  const { email, password } = req.body;

  try {
    const oldUser = await UserModal.findOne({ email });

    if (!oldUser) return res.status(404).json({ message: "User doesn't exist" });

    const isPasswordCorrect = await bcrypt.compare(password, oldUser.password);

    if (!isPasswordCorrect) return res.status(400).json({ message: "Invalid credentials" });

    const token = jwt.sign({ email: oldUser.email, id: oldUser._id }, secret, { expiresIn: "1h" });

    res.status(200).json({ result: oldUser, token });
  } catch (err) {
    res.status(500).json({ message: "Something went wrong" });
  }
};
```

- interact with api:

```
export const signin = (formData, router) => async (dispatch) => {
  try {
    const { data } = await api.signIn(formData);

    dispatch({ type: AUTH, data });

    router.push('/');
  } catch (error) {
    console.log(error);
  }
};
```

- **Code 200: Successful operation**
- **Code 400: The request could not be understood by the server**
- **Code 404: The server has not found anything matching the request**
- **Code 500: The server encountered an unexpected condition which prevented it from fulfilling the request**

UI Diagram:



Stretch Feature service

- **likePost** (liking a blog post)

o Request and response:

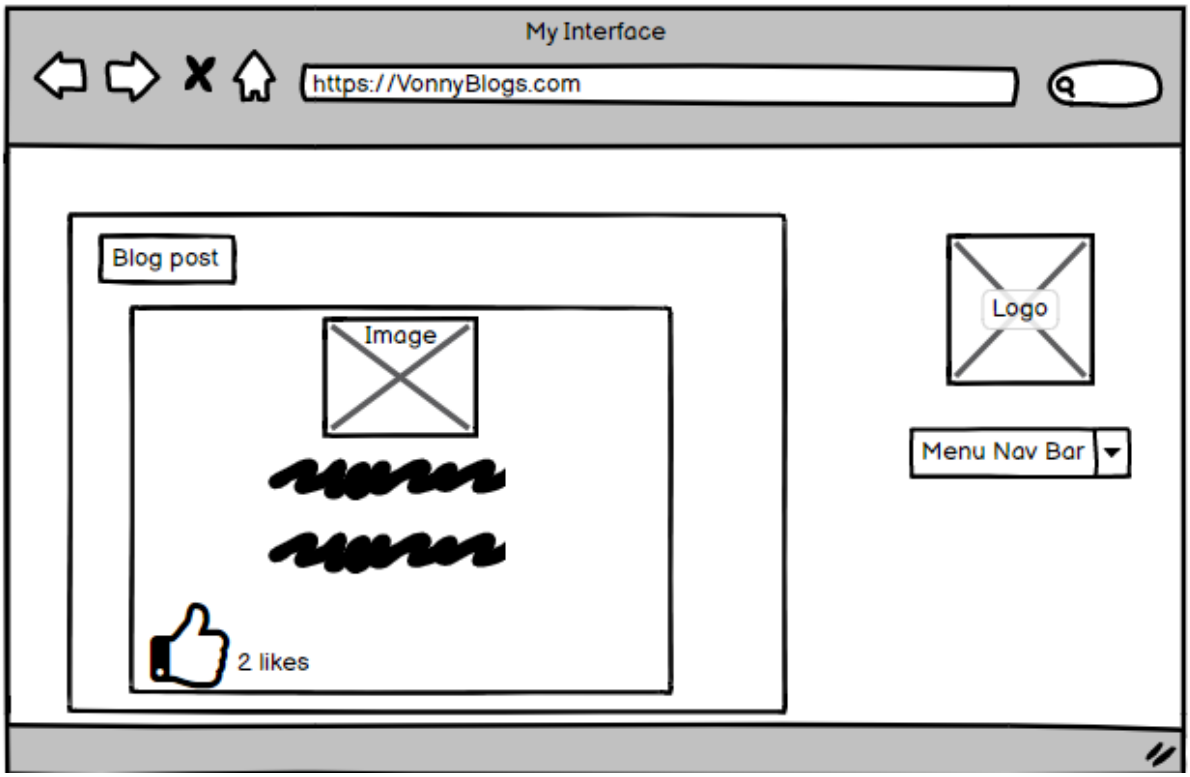
```
export const likePost = async (req, res) => {
   const { id } = req.params;

   if (!req.userId) {
      return res.json({ message: "Unauthenticated" });
    }

   if (!mongoose.Types.ObjectId.isValid(id)) return res.status(404).send(`No post with id: ${id}`);

   const post = await PostMessage.findById(id);

   const index = post.likes.findIndex((id) => id ===String(req.userId));

   if (index === -1) {
     post.likes.push(req.userId);
   } else {
     post.likes = post.likes.filter((id) => id !== String(req.userId));
   }
   const updatedPost = await PostMessage.findByIdAndUpdate(id, post, { new: true });
   res.status(200).json(updatedPost);
}
```

- interact with api:

```
export const likePost = (id) => async (dispatch) => {
  const user = JSON.parse(localStorage.getItem('profile'));

  try {
    const { data } = await api.likePost(id, user?.token);

    dispatch({ type: LIKE, payload: data });
  } catch (error) {
    console.log(error);
  }
};
```

- **Code 200: Successful operation**
- **Code 404: The server has not found anything matching the request**

UI Diagram:


My Interface
https://VonnyBlogs.com
Blog post
Image
2 likes
Logo
Menu Nav Bar