FORM HANDLING AND MUTATION

Jozsef Gal

Nyíregyháza – 2022/05/04

Agenda

01 Formik

02 Exercises – part I

03 Redux Toolkit Query Mutations

04 Exercises – part II

FORMIK

Form data handling in an easy way

Problems when working with form data

- Data representation
- Separate logic from presentation
- Validation issues
- Repetitive work in different fields – tracking values, visited, updated fields
- Handling form submission
- Form state:
  - Local only and unique in each different forms
  - Saving to Redux is an overkill
- React: state represents what you see
  - useState is not the best to work with large data
  - Values has to be followed – usually onChange

Formik

- Helps you out in the 3 most annoying part:
  - Getting values in and out of form state
  - Validation and error messages (with Yup as well)
  - Handling form submission
- Why not redux (redux-form)?
  - Tracking data on this level is not needed
  - Would call reducer multiple times on each keystroke
- Disadvantages:
  - Optimization is highly needed in large forms
  - Dependencies
- Alternative:
  - React-hook-form

```
import { Formik, Form, Field, ErrorMessage } from 'formik';

const Basic = () => (
  <div>
    <h1>Any place in your app!</h1>
    <Formik
      initialValues={{ email: '', password: '' }}
      validate={values => {
        const errors = {};
        if (!values.email) {
          errors.email = 'Required';
        } else if (
          !/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i.test(values.email)
        ) {
          errors.email = 'Invalid email address';
        }
        return errors;
      }}
      onSubmit={(values, { setSubmitting }) => {
        setTimeout(() => {
          alert(JSON.stringify(values, null, 2));
          setSubmitting(false);
        }, 400);
      }}
    >
      {({ isSubmitting }) => (
        <Form>
          <Field type="email" name="email" />
          <ErrorMessage name="email" component="div" />
          <Field type="password" name="password" />
          <ErrorMessage name="password" component="div" />
          <button type="submit" disabled={isSubmitting}>
            Submit
          </button>
        </Form>
      )}
    </Formik>
  </div>
);
```

EXERCISE – EPISODE 1

Formik

Initialize

- Checkout branch: 12-formik
- Start the project together with backend: https://github.com/VarvaraZadnepriak/MoviesAPI.ReactJS
- Add dependencies:
  - formik
  - yup
- Open and analyze files:
  - src/components/pages/movie-form.tsx
  - src/components/pages/genre-selector.tsx
- Create editor slice in redux
- Create provider to open modal window
  - Use the state parameters
  - Add this to layout
- Update components to trigger the modal:
  - PageHeader
  - MovieItemMenu

```
interface MovieEditorState {
  movie: RawMovie | null;
  movieId: Movie["id"];
  showEditor: boolean;
}

const initialState: MovieEditorState = {
  movie: null,
  movieId: undefined,
  showEditor: false,
};

export const movieEditorSlice = createSlice({
  name: "movieEditor",
  initialState,
  reducers: {
    showEditor: (
      state: MovieEditorState,
      action: PayloadAction<Movie["id"]>
    ) => {
      state.showEditor = true;
      state.movieId = action.payload;
      if (!action.payload) {
        state.movie = createRawMovie();
      }
    },
    setMovie: (state: MovieEditorState, action: PayloadAction<RawMovie>) => {
      state.movie = {...action.payload};
    },
    closeEditor: (state: MovieEditorState) => {
      state.movie = null;
      state.showEditor = false;
      state.movieId = undefined;
    }
  },
});
```

Apply Formik

- Create validation rules with Yup
- Apply changes to MovieForm component:
  - useFormik hook or Formik element (context)
  - Add values and onChange handlers (replace console.log with the necessary method)
  - Add error messages and connect validation
  - Disable the submit button when needed

```
export const movieFormValidationSchema = Yup.object({
  title: Yup.string().required("Please insert title"),
  vote_average: Yup.number()
    .min(0, "Must be at least 0")
    .max(10, "Must be at max 10"),
  poster_path: Yup.string()
    .required("Please add a poster path")
    .url("Invalid path"),
  overview: Yup.string().required("Please define movie overview"),
  genres: Yup.array().ensure()
    .typeError("Select at least one genre to proceed")
    .required("Select at least one genre to proceed")
    .min(1, "Select at least one genre to proceed"),
  runtime: Yup.number()
    .integer()
    .typeError("Set the length of the movie")
    .required("Set the length of the movie")
    .min(0, "Must be a positive value"),
  release_date: Yup.date().typeError("Please define the release date")
});


const { values, errors, isSubmitting, isValid, isValidating, setFieldValue,
handleSubmit, handleReset } = useFormik({
    initialValues: movie,
    onSubmit: async (values: Movie, { setSubmitting }) => {
      setSubmitting(true);
      try {
        const validatedMovie = {
          ...values,
          tagline: values.tagLine || undefined,
        };
        await onSubmit(validatedMovie);
      } catch(e) {
        console.error(e);
      }
      setSubmitting(false);
    },
    validationSchema: movieFormValidationSchema,
  });
```

RTQ – MUTATIONS

Sending data with Redux Toolkit Query

Update store with mutations

- Invalidate/reload current data when:
    - Updating item
    - Creating a new item
    - Removing existings items
- Local caching via Tags
    - Get: provide tags
    - Create: list has to be refreshed
    - Update: replace with new data
    - Delete: remove item, refresh list

```
const MovieTag: string = "movie";

...

  providesTags: (result?: GetMoviesResponse) =>
    result?.data && Array.isArray(result.data)
      ?
        [
          ...(result.data as RawMovie[]).map(
            ({ id }) => ({ type: MovieTag, id })
          ),
          { type: MovieTag, id: "LIST" },
        ]
      : [{ type: MovieTag, id: "LIST" }]

...

  invalidatesTags: (_result, _error, { id }) =>
    [{ type: MovieTag, id }]
```

EXERCISE – EPISODE 2

Redux toolkit query

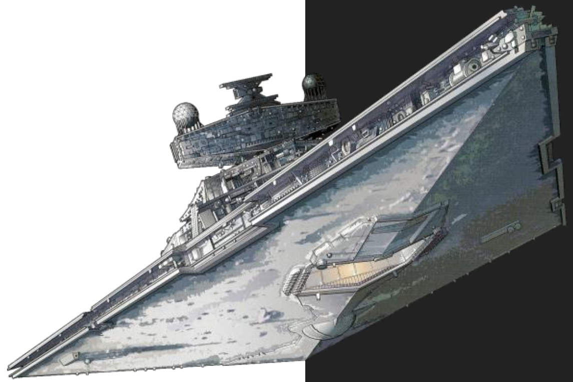
Update API

- Add endpoints to create, delete and update actions
- Create hook for the methods
- Update components to use these ones
  - MovieEditorProvider – update/create
  - MovieItemMenu - delete

```
export const useMovieChanges = () => {
  const [updateMovie] = useUpdateMovieMutation();
  const [deleteMovie] = useDeleteMovieMutation();
  const [createMovie] = useCreateMovieMutation();

  const save = useCallback(async (movie: Movie) => {
    const mutation = movie.id ? updateMovie : createMovie;
    const rawMovie = serializeMovie(movie);
    const result = await mutation(rawMovie);
    if ("error" in result) {
      throw result.error;
    }
    return movieFactory(result.data);
  }, [updateMovie, createMovie]);

  return {
    deleteMovie,
    save,
  };
};
```



MAY THE FORCE BE WITH YOU

For more information, contact

Jozsef Gal
Senior Software Engineer

jozsef_gal@epam.com

EPAM Debrecen

Tüzér street 4.

HU-4028 Debrecen