{() => fs}

f Exercises: extending the bloglist

In addition to the eight exercises in the React router and custom hooks sections of this seventh part of the course material, 13 exercises continue our work on the Bloglist application that we worked on in parts four and five of the course material. Some of the following exercises are "features" that are independent of one another, meaning that there is no need to finish the exercises in any particular order. You are free to skip over a part of the exercises if you wish to do so. Quite many of the exercises are applying the advanced state management technique (Redux, React Query and context) covered in part 6.

If you do not want to use your Bloglist application, you are free to use the code from the model solution as a starting point for these exercises.

Many of the exercises in this part of the course material will require the refactoring of existing code. This is a common reality of extending existing applications, meaning that refactoring is an important and necessary skill even if it may feel difficult and unpleasant at times.

One good piece of advice for both refactoring and writing new code is to take *baby steps*. Losing your sanity is almost guaranteed if you leave the application in a completely broken state for long periods while refactoring.

Exercises 7.9.-7.21.

7.9: automatic code formatting

In the previous parts, we used ESLint to ensure that code follows the defined conventions. Prettier is yet another approach for the same. According to the documentation, Prettier is *an opinionated code formatter*, that is, Prettier not only controls the code style but also formats the code according to the definition.

Prettier is easy to integrate into the code editor so that when the code is saved, it is automatically formatted correctly.

Take Prettier to use in your app and configure it to work with your editor.

State management: Redux

There are two alternative versions to choose for exercises 7.10-7.13: you can do the state management of the application either using Redux or React Query and Context. If you want to maximize your learning, you should do both versions!

7.10: Redux, step1

Refactor the application to use Redux to manage the notification data.

7.11: Redux, step2

Note that this and the next two exercises are quite laborious but incredibly educational.

Store the information about blog posts in the Redux store. In this exercise, it is enough that you can see the blogs in the backend and create a new blog.

You are free to manage the state for logging in and creating new blog posts by using the internal state of React components.

7.12: Redux, step3

Expand your solution so that it is again possible to like and delete a blog.

7.13: Redux, step4

Store the information about the signed-in user in the Redux store.

State management: React Query and context

There are two alternative versions to choose for exercises 7.10-7.13: you can do the state management of the application either using Redux or React Query and Context.

7.10: React Query and context step1

Refactor the app to use the useReducer-hook and context to manage the notification data.

7.11: React Query and context step2

Use React Query to manage the state for blogs. For this exercise, it is sufficient that the application displays existing blogs and that the creation of a new blog is successful.

You are free to manage the state for logging in and creating new blog posts by using the internal state of React components.

7.12: React Query and context step3

Expand your solution so that it is again possible to like and delete a blog.

7.13: React Query and context step4

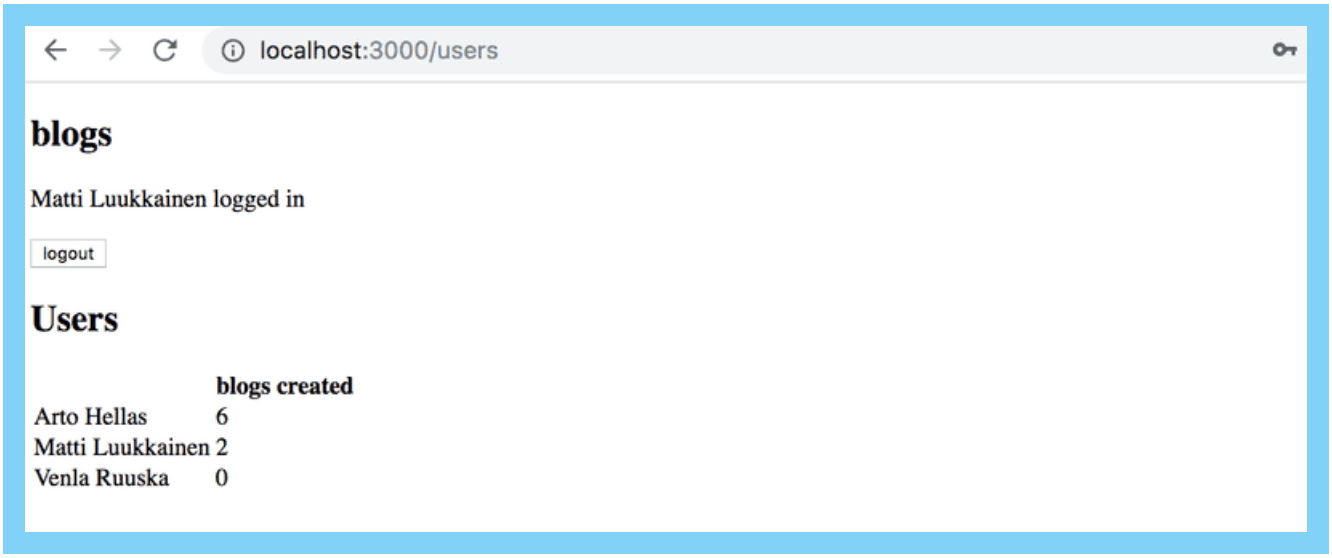
Use the useReducer-hook and context to manage the data for the logged in user.

Views

The rest of the tasks are common to both the Redux and React Query versions.

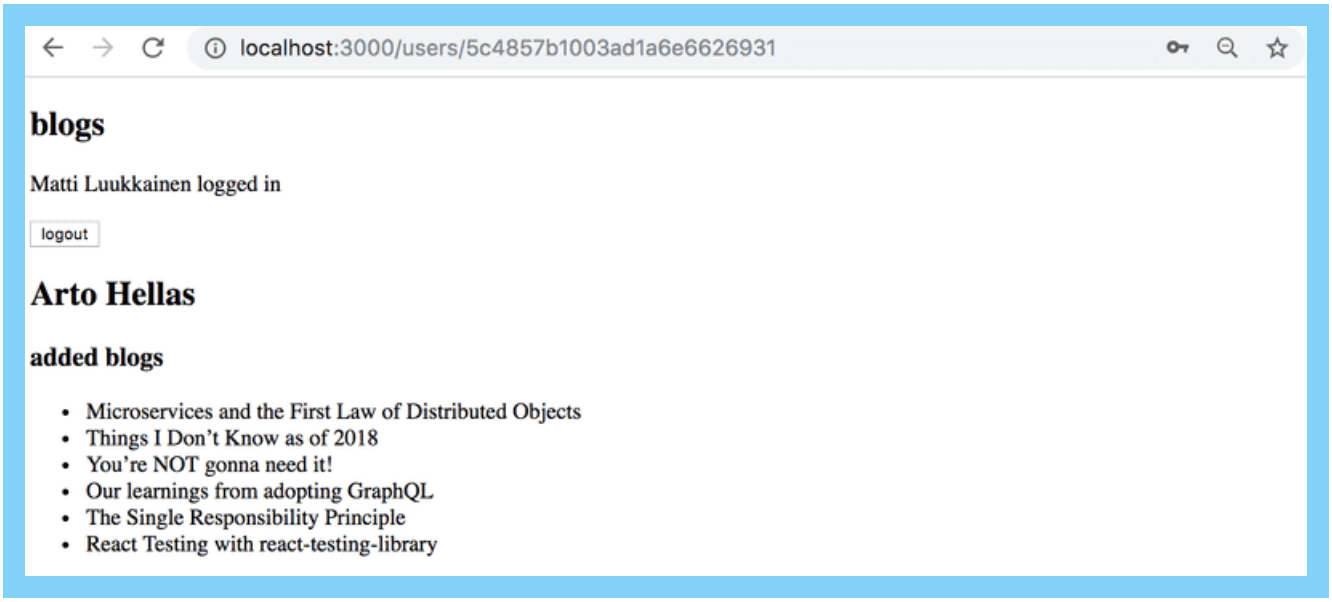
7.14: Users view

Implement a view to the application that displays all of the basic information related to users:

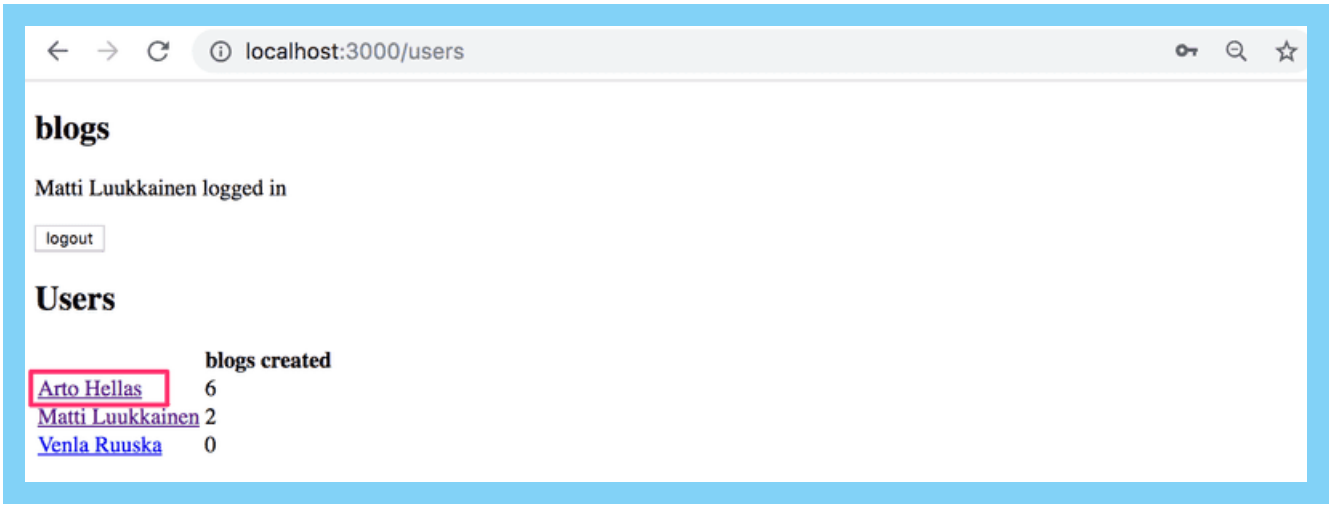


7.15: Individual user view

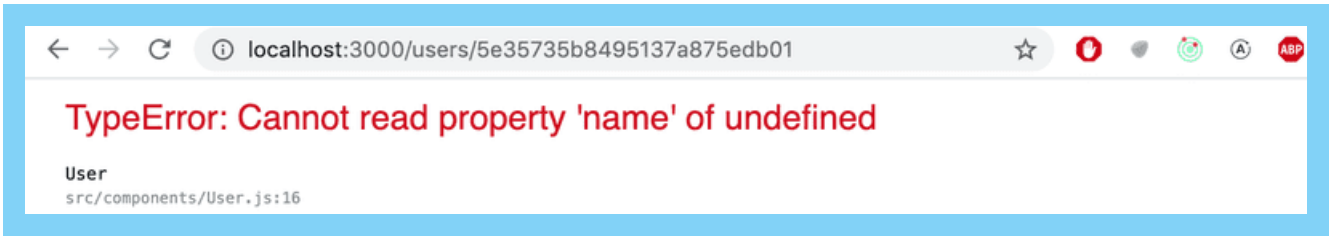
Implement a view for individual users that displays all of the blog posts added by that user:



You can access the view by clicking the name of the user in the view that lists all users:



NB: you will almost certainly stumble across the following error message during this exercise:



The error message will occur if you refresh the page for an individual user.

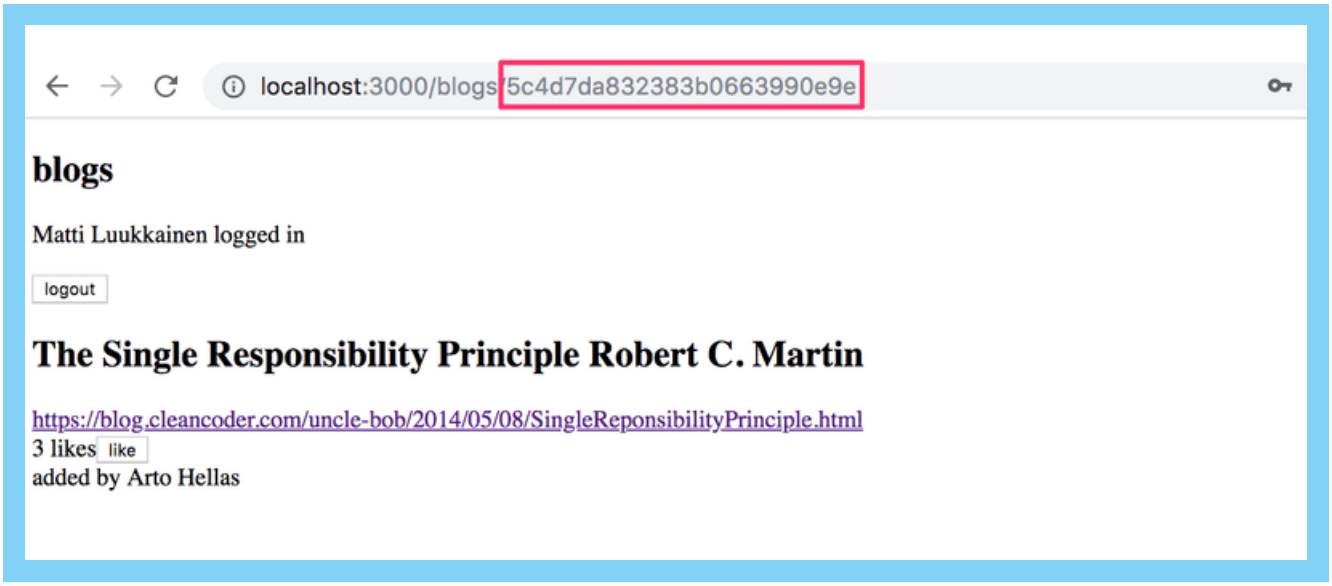
The cause of the issue is that, when we navigate directly to the page of an individual user, the React application has not yet received the data from the backend. One solution for fixing the problem is to use conditional rendering:

```
const User = () => {
  const user = ...
  if (!user) {
    return null
  }

  return (
    <div>
      // ...
    </div>
  )
}
```

7.16: Blog view

Implement a separate view for blog posts. You can model the layout of your view after the following example:

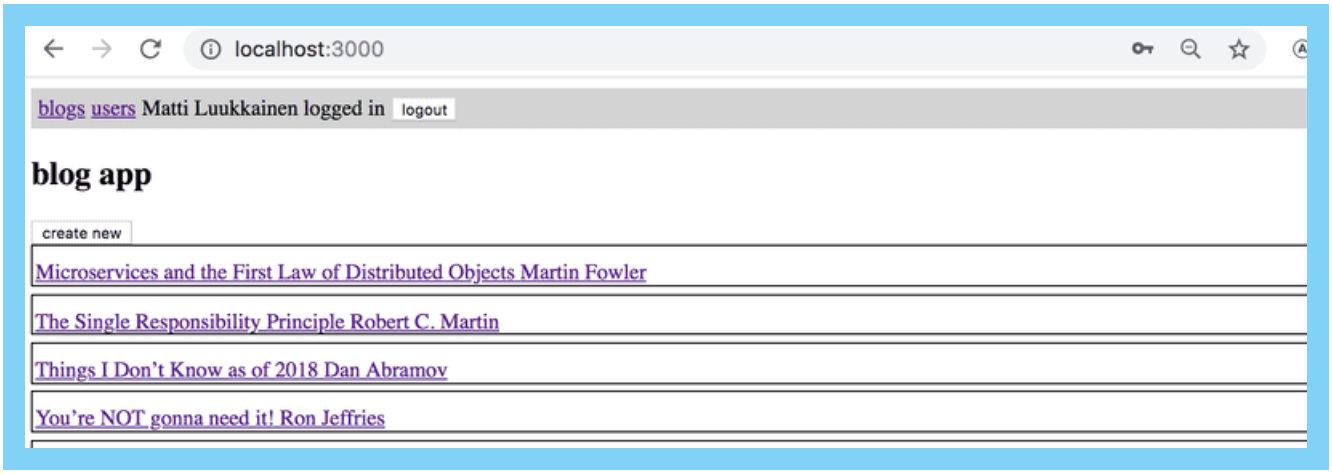


Users should be able to access the view by clicking the name of the blog post in the view that lists all of the blog posts.

After you're done with this exercise, the functionality that was implemented in exercise 5.7 is no longer necessary. Clicking a blog post no longer needs to expand the item in the list and display the details of the blog post.

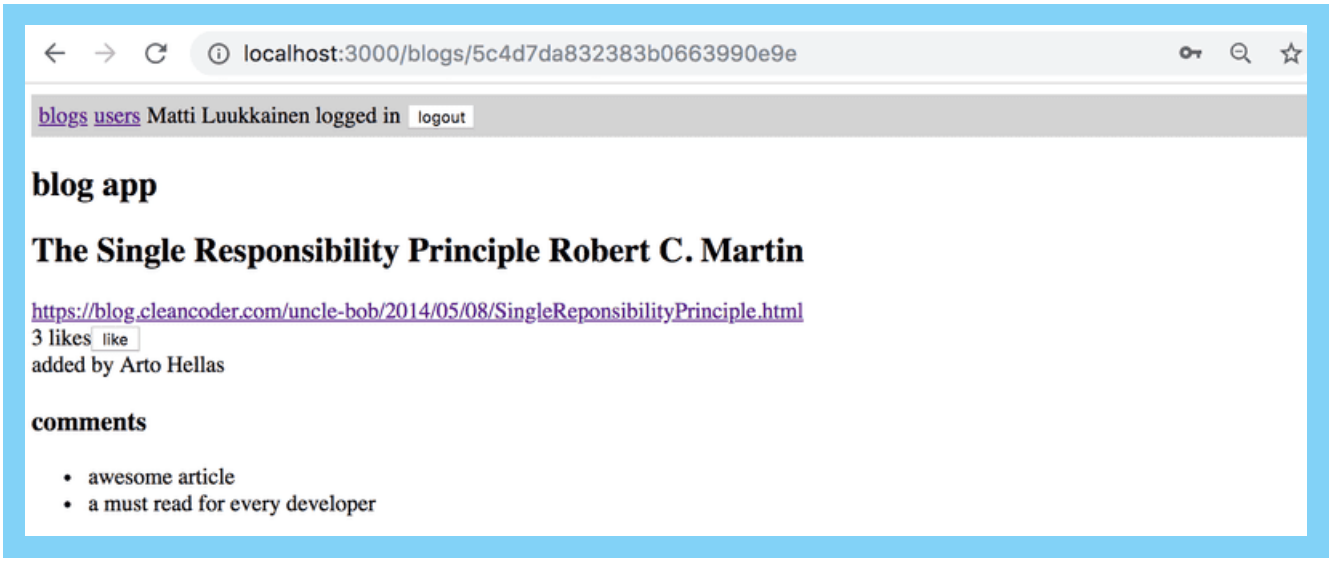
7.17: Navigation

Implement a navigation menu for the application:



7.18: comments, step1

Implement the functionality for commenting on blog posts:



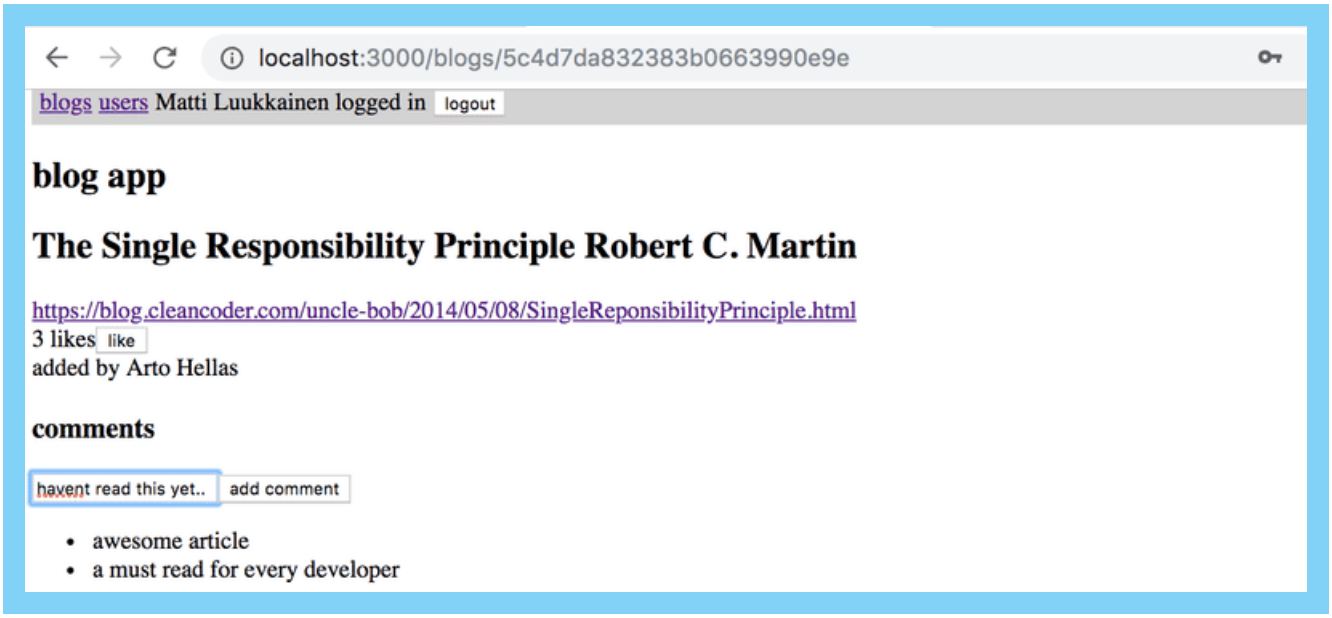
Comments should be anonymous, meaning that they are not associated with the user who left the comment.

In this exercise, it is enough for the frontend to only display the comments that the application receives from the backend.

An appropriate mechanism for adding comments to a blog post would be an HTTP POST request to the *api/blogs/:id/comments* endpoint.

7.19: comments, step2

Extend your application so that users can add comments to blog posts from the frontend:



7.20: Styles, step1

Improve the appearance of your application by applying one of the methods shown in the course material.

7.21: Styles, step2

You can mark this exercise as finished if you use an hour or more for styling your application.

This was the last exercise for this part of the course and it's time to push your code to GitHub and mark all of your finished exercises to the exercise submission system.

Propose changes to material

Part 7e
Previous part

Part 8
Next part

About course

Course contents

FAQ

Partners

Challenge

UNIVERSITY OF HELSINKI

HOUSTON