

CS312 - Assignment Two

Initial Due Date: 20223-03-02 8:15AM

Final Due Date: 2023-03-31 4:15PM

Goals

- Implement basic React components with state and callbacks
- Use a linter and code formatter to write more consistent, more maintainable, higher quality, code

The goal of this assignment is to learn the basics of creating a single page web application (SPA) with React. In this and the next few assignments, you will be developing a miniature version of Wikipedia named “Simplepedia”. Implementing Simplepedia will provide hands-on experience developing a full single page web application in preparation for completing your large project. Simplepedia was first developed with Davin Chia in Fall 2016 as an independent study.

As with previous assignments, you will need to do additional research (online) to successfully complete the assignment.

Prerequisites

1. Click the GitHub classroom [link](#) and then clone the repository GitHub classroom creates to your local computer (in your shell, execute `git clone` followed by the address of the repository).
2. Update the `package.json` file with your name and e-mail
3. Install the package dependencies with `npm install`

Once you have the dependencies installed you can start the development server with `npm run dev`.

I recommend reviewing [introduction to JSX](#) before starting your assignment.

Background

In this assignment, you will be constructing the basic interface for viewing a collection of articles. Here is the goal you are aiming for:

Simplepedia

1 2 4 5 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z


- La Neuville-en-Hez
- Lachenalia
- Lacrymosa (song)
- Lake Tengiz
- Langlade, Gard
- Languages of Singapore
- Lateral rotator group
- Lateral sural cutaneous nerve
- Laura Dupuy Lasserre
- Lecce
- Lee Jae-joung
- Leonid Kuchma
- Liiva, Tallinn
- Lionel Jefferson
- Lonan (parish)
- Lord of the World
- Los Cardones National Park
- Louis Lepic
- Love, Illinois
- Luis Mansilla

Lateral rotator group

The lateral rotator group is a group of six small muscles of the hip which all externally (laterally) rotate the femur in the hip joint. It consists of the following muscles: Piriformis, gemellus superior, obturator internus, gemellus inferior, quadratus femoris and the obturator externus. All muscles in the lateral rotator group originate from the hip bone and insert on to the upper extremity of the femur. The muscles are innervated by the sacral plexus (L4-S2), except the obturator externus, which is innervated by the lumbar plexus.

2016-11-17T20:23:40.299Z

Along the top of the page, there is a list of sections. Clicking on one of the sections displays the appropriate list of articles. Clicking on an article title displays the contents of the article at the bottom of the page. All of the articles have a title, some contents, and a modification date.

As with previous assignments an initial set of (failing) tests are provided as part of the skeleton (run with  `npm test`). These tests are intended to help you as you develop your application and to ensure that the grading scripts can successfully test your assignment. Code that passes all of the provided tests is not guaranteed to be correct. However, code that fails one or more these tests does not meet the specification. Testing React applications is a future topic and so you are not expected to write any additional tests for this assignment.

You will notice that there is already some code present in `index.js` and in `components`. I have provided starter code for all of the required components. In addition, you will find that I have provided some data for your application. The data can be found in `data/seed.json`, and we are importing it directly into `index.js`. I have loaded it into state for you to get you started.

Assignment

Design

With React, the first step is to break the interface down into components:

Simplepedia

1 2 4 5 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- Easter Road
- Eastern Partnership
- Ed Roberts (computer engineer)
- Edward Neville Syfret
- Edward W. North
- Eels (band)
- Einar Lönnberg
- El País
- El Toro Y
- Elaine D. Kaplan
- Eliot A. Jardines
- Ellistown
- Emil Fey
- Endiandra compressa
- Erasmus M. Weaver, Jr.
- Eric Illayapparachchi
- Ernesto De Pascale
- Esophageal atresia
- Estalak
- Eugene Levy

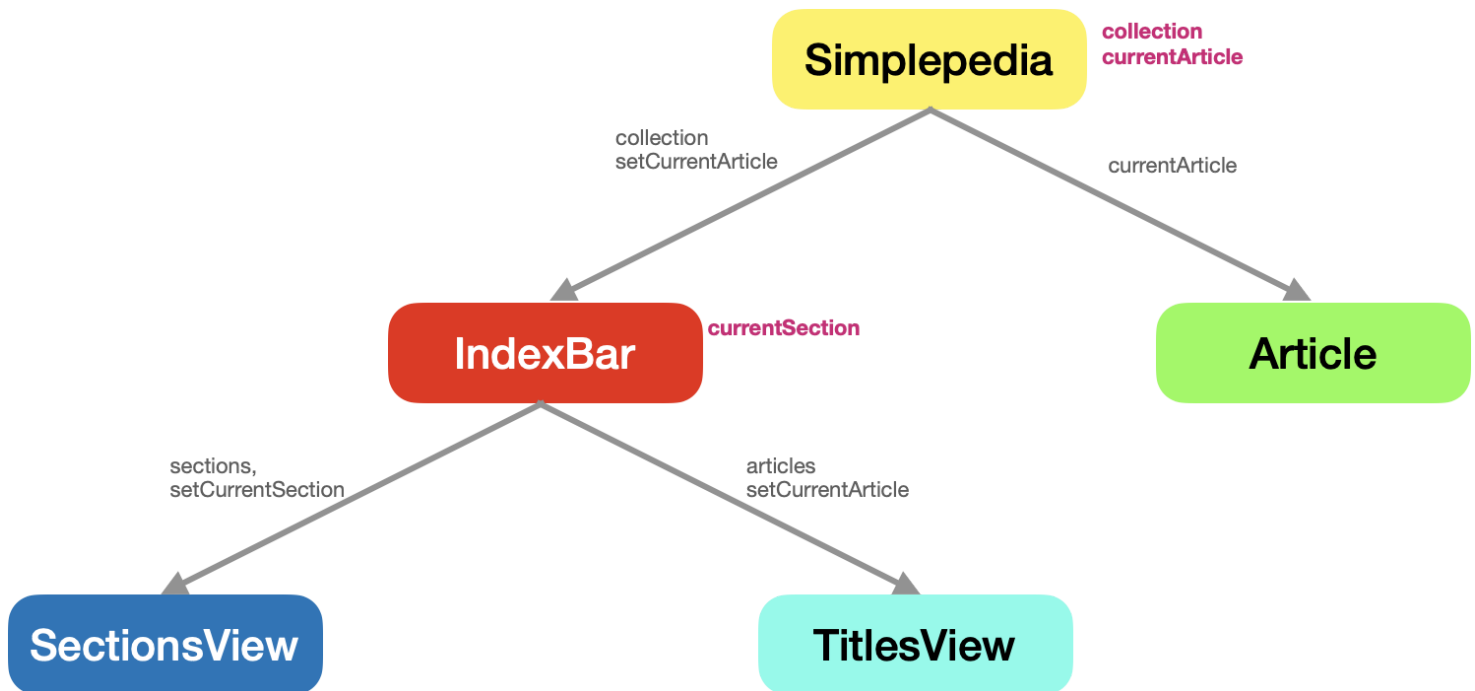
Eels (band)

Eels (often typeset as eels or EELS) is an American indie rock band, formed in California in 1995 by singer/songwriter and multi-instrumentalist Mark Oliver Everett, known by the stage name E. Band members have changed across the years, both in the studio and on stage, making Everett the only official member for most of the band's work. Eels' music is often filled with themes about family, death and lost love. Since 1996, Eels has released eleven studio albums, seven of which charted in the Billboard 200.

12/16/2016, 8:14:32 PM

At the top of the hierarchy, we have the **Simplepedia** component (yellow). We will break this top-level interface down into two main components: IndexBar (red rectangle) and Article (green rectangle). We will further break down the IndexBar into IndexSections (blue) and IndexTitles (cyan).

More abstractly, we can draw the following tree:



In this diagram, I have also included the state and the props. In the **Simplepedia** component, we will have two pieces of state: the collection of all articles (**collection**) and the article currently being displayed (**currentArticle**).

If there is an article to display, the **Simplepedia** component displays an **Article** component, and passes it a prop called **currentArticle** containing the article object to display.

The main controls are provided by the **IndexBar** component at the top of the page. **Simplepedia** hands **IndexBar** two props: **collection** (the collection of all articles) and **setCurrentArticle** (a callback function that **IndexBar** can call when the user selects an article to view).

As described above, the idea is that the user clicks one of the sections along the top (the letters), and **IndexBar** presents a list of the titles available in that section. The user clicks on a title of interest, and that article is displayed. **IndexBar** will use the **collections** prop to determine the sections to display, and when an article is selected, the **setCurrentArticle** callback should be called with the relevant article as the argument. Notice in the diagram that I have given the **IndexBar** a piece of state called **currentSection**. This will help to make sure that the correct list of article titles is displayed.

IndexBar

The first component you should create is the **IndexBar**. Implement the **IndexBar** in the provided file `src/components/IndexBar.js`. Internally, you will have two sub-components: a list of sections (**SectionsView**), and a list of titles in the current section (**TitlesView**). Create these in their own files (i.e., `src/components/SectionsView.js` and `src/components/TitlesView.js`).

The **SectionsView** component should take two props: **sections**, a list of the sections, and **setCurrentSection**, a callback for reporting when a section is selected. The list itself should be implemented as an unordered list (``). You should add an `onClick` callback to each `` in the list which sets the current section as state for the **IndexBar**. You will want to make sure that the sections are sorted, but make sure that you don't modify the props. Note that the *only* thing that the **SectionView** component should do is sort and display the sections. Let **IndexBar** do the heavy lifting of figuring out what the sections are.

Wrap the `` in a `<div>`. I've provided you with styling to turn the list into the horizontal bar. You can enable this by adding a `className` to the `div` with the value `{styles.sectionList}`. To make that style available, import the relevant CSS module in `SectionsView.js`, e.g.,

```
import styles from "../styles/SectionsView.module.css";
```

TitlesView will be very similar to **SectionsView**, albeit with different props. It should purely be concerned with displaying a sorted list of titles. However, to facilitate communication, **IndexBar** should provide **TitlesView** a prop **articles**, a list of full article objects (not just titles), and "pass through" the prop **setCurrentArticle**, which is the callback passed into **IndexBar** to set the current article (since the selection made from the titles will determine which article gets displayed).

TitlesView should also be implemented as an unordered list ("unordered" just means that the list is displayed with bullets instead of numbers, it is not referring to the contents, which you should sort). For these ``, make sure to invoke the callback that was passed down with the props when the user clicks on a title, returning the article corresponding to the title.

The primary job of the `IndexBar` component will be to break up the collection for the two sub-components and to maintain state (the currently selected section). When the section list reports a new section selection, `IndexBar` should set its local state, which will cause a re-render (and thus a new selection of titles for the title list). In addition, when a new section is selected, you should invoke the `setCurrentArticle` callback with a missing (thus `undefined`) argument to indicate that the current article should be “cleared”, i.e. not shown.

Finally, if there isn’t a currently selected section, a message asking the user to select a section should be displayed instead of the `TitlesView` sub-component.

Building lists of list items

When you are done, the HTML must have this structure:

```
<div>
  <div class="section-list">
    <ul>
      <li data-testid="section">A</li>
      <li data-testid="section">B</li>
    </ul>
  </div>
  <div>
    <ul>
      <li data-testid="title">Apple</li>
      <li data-testid="title">Anteater</li>
      <li data-testid="title">Auton</li>
    </ul>
  </div>
</div>
```

Here is one way to transform an array into an HTML list:

```
const list = ['daleks', 'cybermen', 'ice warriors', 'autons'];
const races = list.map(race => (<li key={race}>{race}</li>));

return (<ul>{races}</ul>);
```

which would produce this HTML:

```
<ul>
  <li>daleks</li>
  <li>cybermen</li>
  <li>ice warriors</li>
  <li>autons</li>
</ul>
```

Note the `key` property in the above example. As discussed in class when we make a list of React components, it is important that we add a `key` property so that React can uniquely identify each component in the collection. You can learn more in the [React documentation on lists](#). Regardless, you need to provide a unique string or value as the key.

Important In addition to adding the `key` property to each ``, I would like you to add a `data-testid` property. For the sections, please set this to “section” (i.e. `data-testid="section"`) and for the titles, please set this to “title” (e.g., `data-testid="title"`). The tests will only pass if these are present, so please make sure that you include them.

Article

The next component you should implement is `Article` (in the provided file `src/components/Article.js`). The `Article` component should take one prop named `article` that is an article record Object. Our articles have three fields: “title”, “extract” (the contents), and “edited” (the time the article was last edited).

In your `Article` component, display the title, text, and date. The entire article should be contained in a `<div>` with a `className` property with value `{styles.article}`. The title should use an `h2` tag, and the contents and timestamp can use simple `p` tags. The date should have the `className` `{styles.timestamp}`. To ensure consistent date formatting, render the date in a locale-relevant format with [Date.toLocaleString](#). Note that this will not necessarily match the picture, it will render the date as appropriate for your computer/browser/environment.

Putting the components together

In `src/pages/index.js` integrate your newly created components. You need to make sure that you pass the appropriate callbacks as props so that clicking on a section opens the title list, and clicking on a title opens the related article for viewing. If there is no article to show at the current moment, best practice is to not render the `Article` component (via [conditional rendering](#)) as opposed to rendering `Article` but having it somehow manage an undefined `article` prop. The latter forces additional complexity into the `Article` component and in the case of more complex components can trigger unnecessary computation.

Finishing up

Your submission should not have ESLint warnings or errors when run with `npm run lint`. Remember that you can fix many errors automatically with `npm run lint -- --fix` (although since ESLint can sometimes introduce errors during this process, we suggest committing your code before running “fix” so you can rollback any changes). The assignment skeleton includes the Prettier package and associated hooks to automatically reformat your code to a consistent standard when you commit. Thus do not be surprised if your code looks slightly different after a commit.

Submit your assignment by pushing all of your committed changes to the GitHub classroom via `git push --all origin` and then submitting your repository to Gradescope as described [here](#). You can submit (push to GitHub and submit to Gradescope) multiple times.

Portions of your assignment will undergo automated grading. Make sure to follow the specifications exactly, otherwise the tests will fail (even if your code generally works as intended). Use the provided test suite (run with `npm test`) to get immediate feedback on whether your code follows the specification. Because of the increased complexity of a React application, Gradescope can take minutes to run all of the tests. Thus you will be more efficient testing locally and only submitting to Gradescope when you are confident your application meets the specification.

Grading

Assessment	Requirements
Revision needed	Some but not all tests as passing.
Meets Expectations	All tests pass, including linter analysis (without excessive deactivations).
Exemplary	All requirements for <i>Meets Expectations</i> and your implementation is clear, concise, readily understood, and maintainable.

FAQ

Do I need to implement unit testing?

We will learn later in the semester how to unit test React components. For this assignment you are not expected to implement any of your own unit tests. The skeleton includes some unit tests to assist you in your development and to ensure that the grading scripts can automatically test your submission.

What if the tests and assignment specification appear to be in conflict?

Please post to Ed so that we can resolve any conflict or confusion ASAP.

© Michael Linderman and Christopher Andrews 2019-2023. Last modified at: 2023-02-28 10:02:15 -0500.