

Single Page Application State Management

Maintaining State in React Applications with Redux and Routing

Introduction

A previous assignment used React to implement a Single Page Application (SPA) called WhiteBoard, an online Learning Management System (LMS). This assignment will expand on that assignment to allow faculty to add various content widgets such as Headings, Paragraphs, Images and Links.

Learning Objectives

The learning objectives of this assignment are

- Maintain state with Redux in React applications
- Use a store to keep track of a history of states
- Map application state to stateless React components
- Navigating with React Router DOM library

Features

The features implement it in this assignment are

- Navigating from Course List to Course Editor
- Navigating between Modules, Lessons, and Topics
- Maintaining state for current Course, Module, Lesson, and Topic

Install Redux in an existing React application

In the root directory of the React application created in an earlier assignment, install the Redux library and its React integration library

```
npm install redux --save
npm install react-redux --save
```

Implement navigation with routing

Refactor CourseTable and Grid components to navigate to the CourseEditor component by clicking on the title of course. Encode the course ID into the URL using the pattern: ***/courses/:layout/edit/:courseId*** where ***courseId*** is the ID of the course selected and layout is either ***table*** or ***grid***.

Implement a course editor component

In ***/src/components/course-editor/course-editor.js***, create a functional component called ***CourseEditor*** which displays the modules, lessons, and topics for a selected course. The ***CourseEditor*** is shown below

displaying the name of the currently selected course. The component should be mapped to **`/courses/:layout/edit/:courseId`** and display the modules, lessons, and topics for a course whose ID is **`courseId`**. Clicking on the close button on the **`CourseEditor`** (the X next to the course title) navigates to the previous page.

✕ Web Dev Selected Course

The screenshot shows the 'CourseEditor' interface. On the left is a sidebar titled '✕ Web Dev Selected Course'. It contains a list of modules: 'Unselected Module 1', 'Selected Module 2' (highlighted in blue), 'Unselected Module 3', 'Editing Module 4' (with a close and checkmark icon), and 'New Module'. At the bottom of the sidebar is a blue plus icon. The main area on the right has two rows of tabs. The top row represents lessons: 'Unselected Lesson 1', 'Selected Lesson 2' (highlighted), 'Lesson 3', 'Editing Lesson 4' (with close and checkmark icons), and 'New Lesson'. The bottom row represents topics: 'Unselected Topic 1', 'Selected Topic 2' (highlighted), 'Topic 3', 'Editing Topic 4' (with close and checkmark icons), and 'New Topic'. Both rows end with a blue plus icon.

Implementing the module list component

In **`/src/components/course-editor/module-list.js`**, create a functional react component called **`ModuleList`**. The component should render a list of modules from the currently selected course. At the bottom of the module list there's a button that allows adding new modules to the list. New modules appear at the end of the list with a default new title of "New Module". Each module is rendered using an **`ItemEditor`** component described later in this document. Each module has a label, an input field, edit button, save button, and delete button. By default each module item only displays the edit button and label with the module's title. Clicking on the edit button hides the label and edit button, and shows the other buttons, input field, and highlights the module. Clicking on the delete button removes the module from a remote server and the list of module updates. Clicking on the save button saves the changes to a remote server and updates the list of modules. Clicking on the title makes it currently selected module, and highlights the module.

Implementing the lesson tabs component

In **`/src/components/course-editor/lesson-tabs.js`**, create a functional React component called **`LessonTabs`**. The component should render a list of lesson tabs from the currently selected module. Encode the selected module in the URL as **`/courses/:layout/edit/:courseId/modules/:moduleId`**. At the end of the list of tabs there's a button that allows adding new lesson tabs. New lessons appear at the end of the list with a default title of "New Lesson". Each lesson is rendered using an **`ItemEditor`** component described later in this document. Each tab has a label, an input field, edit button, save button, and delete button. By default each tab item only displays the edit button and label with the lesson's title. Clicking on the edit button hides the label and edit button, and shows the other buttons and input field, and highlights the lesson. Clicking on the delete button removes the lesson from a remote server and the list of lessons updates. Clicking on the save button saves the changes to a remote server and updates the list of lessons. Clicking on the title makes it currently selected lesson, and highlights the lesson.

Implementing the topic pills component

In **`/src/components/course-editor/topic-pills.js`**, create a functional React component called **`TopicPills`**. The component should render a list of topic pills from the currently selected lesson. Encode the selected lesson

in the URL as `/courses/:layout/edit/:courseId/modules/:moduleId/lessons/:lessonId`. At the end of the list of pills there's a button that allows adding new topic pills. New topics appear at the end of the list with a default title of "New Topic". Each topic is rendered using an `ItemEditor` component described later in this document. Each pill has a label, an input field, edit button, save button, and delete button. By default each pill item only displays the edit button and label with the topic's title. Clicking on the edit button hides the label and edit button, and shows the other buttons and input field, and highlights the topic. Clicking on the delete button removes the topic from a remote server and the list of topics updates. Clicking on the save button saves the changes to a remote server and updates the list of topics. Clicking on the title makes it currently selected topic, and highlights the topic. If needed encode the selected topic in the URL as `/courses/:layout/edit/:courseId/modules/:moduleId/lessons/:lessonId/topics/:topicId`.

Implement item editor component

In `/src/components/editable-item.js`, create a functional react component called ***EditableItem*** that can be used to render and edit modules, lessons, and topics as described earlier in this document. ***EditableItem*** has an input field, a hyperlink, an edit button, a save button, and a delete button. By default ***EditableItem*** displays the title of the module, lesson, or topic, as a link, as well as the edit button. Clicking the edit button replaces the title link with an input field with the value of title, the edit button is replaced with a save and delete buttons. Clicking the delete button removes the item from the list. Clicking the save button updates the title, hides the input field, shows the title link, and replaces the save and delete buttons with the edit button. Use the `ItemEditor` to implement each of the items rendered in the module list, lesson tabs, and topic pills.

Implement the services

In `/src/services/` implement the following services: `module-service.js`, `lesson-service.js`, and `topic-service.js` with the following functions

Module service

For the `createModule()` and `findModulesForCourse()` functions below, use the following URL pattern: `https://wbdv-generic-server.herokuapp.com/api/YOUR_NEUID/courses/COURSE_ID/modules`

where `YOUR_NEUID` is your `neuid`, and `COURSE_ID` is the ID of the course the modules belong to. For all other functions use the following URL pattern:

`https://wbdv-generic-server.herokuapp.com/api/YOUR_NEUID/modules/MODULE_ID`

Function	Description
<code>createModule(courseId, module)</code>	creates a new module instance for the course whose ID is <code>courseId</code>
<code>findModulesForCourse(courseId)</code>	retrieves all modules for course whose ID is <code>courseId</code>
<code>findModule(moduleId)</code>	retrieves one module whose ID is <code>moduleId</code> (optional)
<code>updateModule(moduleId, module)</code>	updates one module whose ID is <code>moduleId</code>
<code>deleteModule(moduleId)</code>	removes module whose ID is <code>moduleId</code>

Lesson service

For the `createLesson()` and `findLessonsForModule()` functions below, use the following URL pattern:
`https://wbdv-generic-server.herokuapp.com/api/YOUR_NEUID/modules/MODULE_ID/lessons`

where YOUR_NEUID is your neuid, and MODULE_ID is the ID of the module the lessons belong to. For all other functions use the following URL pattern:

`https://wbdv-generic-server.herokuapp.com/api/YOUR_NEUID/lessons/LESSON_ID`

Function	Description
<code>createLesson(moduleId, lesson)</code>	creates a new lesson instance for the module whose ID is moduleId
<code>findLessonsForModule(moduleId)</code>	retrieves all lessons for course whose ID is moduleId
<code>findLesson(lessonId)</code>	retrieves one lesson whose ID is lessonId (optional)
<code>updateLesson(lessonId, lesson)</code>	updates one lesson whose ID is lessonId
<code>deleteLesson(lessonId)</code>	removes lesson whose ID is lessonId

Topic service

For the `createTopic()` and `findTopicsForLesson()` functions below, use the following URL pattern:
`https://wbdv-generic-server.herokuapp.com/api/YOUR_NEUID/lessons/LESSON_ID/topics`

where YOUR_NEUID is your neuid, and LESSON_ID is the ID of the lesson the topics belong to. For all other functions use the following URL pattern:

`https://wbdv-generic-server.herokuapp.com/api/YOUR_NEUID/topics/TOPIC_ID`

Function	Description
<code>createTopic(lessonId, topic)</code>	creates a new topic instance for the lesson whose ID is lessonId
<code>findTopicsForLesson(lessonId)</code>	retrieves all topics for lesson whose ID is lessonId
<code>findTopic(topicId)</code>	retrieves one topic whose ID is topicId (optional)
<code>updateTopic(topicId, topic)</code>	updates one topic whose ID is topicId
<code>deleteTopic(topicId)</code>	removes topic whose ID is topicId

Implement the reducers

In `/src/reducers`, implement the following reducers with the corresponding actions. Combine the reducers to maintain state across `ModuleList`, `LessonTabs`, and `TopicPills` components. Use `useState()` and `useEffect()` hooks to maintain or share state within components.

module-reducer.js

Function	Description
CREATE_MODULE	updates state variable modules by appending a new module at the end of the list of modules
FIND_MODULES_FOR_COURSE	retrieves all modules for course whose ID is courseId
FIND_MODULE	retrieves one module whose ID is moduleId (optional)
UPDATE_MODULE	updates one module whose ID is moduleId
DELETE_MODULE	removes module whose ID is moduleId

lesson-reducer.js

Function	Description
CREATE_LESSON	creates a new lesson instance for the module whose ID is moduleId
FIND_LESSONS_FOR_MODULE	retrieves all lessons for course whose ID is moduleId
FIND_LESSON	retrieves one lesson whose ID is lessonId (optional)
UPDATE_LESSON	updates one lesson whose ID is lessonId
DELETE_LESSON	removes lesson whose ID is lessonId

topic-reducer.js

Function	Description
CREATE_TOPIC	creates a new topic instance for the lesson whose ID is lessonId
FIND_TOPICS_FOR_LESSON	retrieves all topics for lesson whose ID is lessonId
FIND_TOPIC	retrieves one topic whose ID is topicId (optional)
UPDATE_TOPIC	updates one topic whose ID is topicId
DELETE_TOPIC	removes topic whose ID is topicId

Deliverables

As a deliverable, check in all your work into a github source control repository. Deploy your project to a remote public server on Heroku or AWS. Submit the URL of the repository and the remote server in blackboard. Tag the commit you want graded as assignment4. TAs will clone your repository and grade the tagged commit.

