Lab exercise 11 React App Using Hooks (long)

1. Installing and starting the REST server. We will get its data from a REST server based on the 'json-server' package. We will install the package and start up the REST server.
2. Create a directory for the REST server: rest-server
3. Open a command prompt and navigate into the directory you just created. Create a package.json file using the “npm init” command:

npm init

1. Enter to take default settings.
2. Install the 'json-server' package:

npm install --save json-server

1. Ignore any warnings. Copy the books.json data file to rest-server.
2. Edit the rest-server/package.json file, add the following line to the scripts section, and save the file:

"start": "json-server --port 4000 --watch books.json"

1. From the project directory rest-server, start up the rest server:

npm start

1. This calls the script you just added to package.json. If you have any problems check to make sure the script was added correctly.
2. Verify that the REST server comes up without errors and works as expected. Try these URLs in a browser:

http://localhost:4000/books http://localhost:4000/books/0 http://localhost:4000/books?isbn=123

1. Navigate back to the react-books-project folder. Open the package.json file in your editor. We need to add a proxy setting so that the REST server can be accessed from the same server as the application. Add the following line into the package.json after the "name": "react-books- project", line, and save the file:

"proxy": "http://localhost:4000/",

1. From the cmd prompt where the react-books-project server is running stop and restart the application server.
2. Test the proxy: In this part you will add a button that triggers an HTTP request to the REST server in order to test that it is available from within the application. Copy the ApiTestComponent.js file from into the src directory of your project as shown below:
3. Open ApiTestComponent.js in an editor. Note how it implements a React component and how it makes an HTTP call to the REST server. Open App.js and add the component so that it displays in the app. Import it:

import ApiTest from './ApiTestComponent';

1. Replace the App() function with:

function App() { return (

<div className="App">

<h3>React Book Project</h3>

<ApiTest />

</div>

);

}

1. Save the file. The app should rebuild and update the browser on its own at which point a "test api" button should appear. Pressing the "test api" button should cause book data from the REST service to appear.
2. If you have no data:
   1. Make sure the proxy changes to the package.json were saved.
   2. Make sure that the REST service is up and running on localhost and that its "server:port" matches that of the proxy setting in package.json
   3. Try restarting the embedded development server.
3. Next, create the book list with sample data. Modify the React web application to display a hard coded list of books. Create a directory called components in the project under src to hold components.
4. Create the file src/components/booklist.js to hold the books list component.
5. Edit booklist.js to include the following content and then save the file.

import React from 'react';

export default function BookList() { return (<div id={'book-list'}>

<h4>Book List</h4>

<ul>

<li>book one</li>

<li>book two</li>

<li>book three</li>

</ul>

<input type={'button'} value="New Book" />

</div>

);

}

1. Save the file. Create the file src/components/bookform.js to hold the books list component. Edit bookform.js to include the following content and then save the file:

import React from 'react';

export default function BookForm() { return (

<div id='book-form' >

<h4>Book Form</h4>

<form >

<table><tbody>

<tr>

<td>Isbn:</td>

<td><input type={'text'} name={'isbn'} /></td>

</tr>

<tr>

<td>Title:</td>

<td><input type={'text'} name={'title'} /></td>

</tr>

<tr>

<td>Price:</td>

<td><input type={'number'} name={'price'}

/></td>

</tr>

</tbody></table>

<input type={'button'} value="Cancel" />

<input type={'button'} value="Save" />

<input type={'button'} value="Delete" />

</form>

</div>

)

}

1. Replace the src/index.css and src/App.css files.
2. Open src/App.js in your editor.
3. Add imports at the top of the file to bring in the BookList and BookForm components:

import BookList from './components/booklist'; import BookForm from './components/bookform';

1. Update the App() function's return statement so that it shows the BookList and BookForm components:

function App() { return (

<div className="App">

<h3>React Book Project</h3>

<BookList />

<BookForm />

</div>

);

}

1. Save App.js and fix any compile errors if they come up. This provides the basic UI for our application. Next we will get live data from our REST service.
2. Update the book list with data from the REST service: You will modify the Book application to display a list of books obtained from the REST service that was created earlier.
   1. Create the directory /src/rest
   2. Create the file /src/rest/index.js
   3. Open /src/rest/index.js in an editor.
   4. Add the following getBooks() method to the file:

export function getBooks() {

let myHeaders = new Headers({ "Content-Type": "application/json" }); var myInit = { method: 'GET', headers: myHeaders, mode: 'cors' }; let promise = fetch("/books", myInit);

return promise.then((response) => { return response.text();

});

}

1. Save the file. Open src/App.js in an editor. Add an import at the top of the file for the getBooks method:

import { getBooks } from './rest';

1. Remove unused imports. Update the React import statement to include the useState and useEffect Hook methods:

import React, { useState, useEffect } from 'react';

1. Add a useState statement that creates a 'books' state variable to hold the list of books. Add it as the first line in the App() function:

const [books, setBooks] = useState([]);

1. Add a call to useEffect to retrieve book data from the REST server. This can go right after the useState line added in the previous step:

useEffect(() => {

let promise = getBooks(); promise.then(

(text) => {

let bookArray = JSON.parse(text); setBooks(bookArray);

}

)

},[]);

1. Update the App function's JSX code to pass 'books' as a property to the booklist element:

<BookList books={books} />

1. This passes the book array into the BookList component which will then display it. Save the App.js file. Open components/booklist.js file in an editor. Add the parameter 'props' to the BookList function signature:

export default function BookList(props) {

1. Replace the three hard coded <li> elements with the following code that displays books from the list we got from the REST server:

{props.books.map( (book, index) => {

return <li key={index} >

{book.isbn} - {book.title} - ${book.price}</li>

}

)}

1. Save all files and let the browser update. The BookList component should now show data from the REST service instead of the initial hard coded data.
2. Highlight a selected book in the book list: In this part you will update the application to respond to a user clicking on a book to select it. Information about the selected book is then used to render the book list so that the selected book is visually distinguished.
3. Later on we will need to share the selected Book with the BookForm so we are going to maintain it at the App component level and pass it into the BookList. We will also need to pass in a method so that the BookList can update the selected book when someone clicks on the list.
4. Open App.js in an editor.
5. Add the following useState statement as the first line in the App() function:

const [selectedBook, setSelectedBook] = useState(null);

1. We set the selectedBook to null whenever we want to show the list without a selected item.
2. Now edit the JSX code to pass selectedBook and setSelectedBook to the BookList:

<BookList books={books} selectedBook={selectedBook} setSelectedBook={setSelectedBook} />

1. Save the App.js file.
2. Open booklist.js in an editor. We are going to add some JSX code that selects the book and highlights it in bold text when it is selected.
3. Add the following inside the opening tag of the <li>:

className={(isSelected(book)) ? 'selected' : ''} onClick={(event) => props.setSelectedBook(book)}

1. The onClick statement uses the setSelectedBook function that was passed in as a property earlier. The className statement uses an 'isSelected' method to determine if the book is selected. We will add a function for that in the next step.
2. Add the following function as the first line in the BookList function (before the return statement):

const isSelected = function(book){

return (props.selectedBook != null && props.selectedBook === book);

}

1. The function uses the selectedBook property we passed in earlier. It checks if selectedBook is null before comparing it against the current book.
2. Save the booklist.js file.
3. Let the browser update. At this point you should be able to click on an item in the list to select it.
4. Display the selected book in a book form: In this part you will update the application so that book form only displays when a book is selected. To do this we are going to modify the App component so that it only displays BookForm if the selectedBook is not null.
5. Open the App.js file in an editor. Add the following functional component into the App() function just before the return statement:

const Conditional = function(){ if(selectedBook != null){

return <BookForm book={selectedBook} setBook={setSelectedBook}

/>

}

return <div/>

}

1. If selectedBook is null this component returns an empty <div>. If selectedBook is not null then it returns a BookForm component. We are passing the selectedBook object and the setSelectedBook method so that they can be used as props inside the BookForm component.
2. Next we will replace the line in the App() function's return statement that displays BookForm with the conditional component we just created.

return (

<div className="App">

<h3>React Book Project</h3>

<BookList books={books}

selectedBook={selectedBook} setSelectedBook={setSelectedBook} />

<Conditional />

</div>

1. Save the App.js file. You should notice that the BookForm is no longer displayed. Try selecting an item in the list. The BookForm re-appears but there is no item displayed in the form. Lets fix that.
2. Open the components\bookform.js file in an editor.
3. Add a props parameter to the BookForm function signature:

export default function BookForm(props) {

1. Locate the first <input> field (the one for isbn) and add the following code in bold:

<input type={'text'} name={'isbn'} value={props.book.isbn} />

1. Do the same for the other two input fields:

<input type={'text'} name={'title'} value={props.book.title} />

<input type={'text'} name={'price'} value={props.book.price} />

1. When we passed selectedBook in we assigned it to a property named 'book'. That allows us to access it using props.book.
2. Save the bookform.js file. Let the browser update.
3. Try selecting an item in the list. The BookForm should appear displaying the selected item.
4. Right now clicking on the 'New Book' button on the BookList doesn't do anything. We would like it to display the BookForm with empty input fields that we can edit and save as a new book. We can do this by setting the selectedBook to an empty version of the book object.
5. Open the components\booklist.js file in an editor.
6. Add this const expression as the first line in the BookList function:

const newBook = { "id":-1, "isbn": "", "title": "", "price": 0 };

1. This is the 'empty version' of the book object that we are going to set as the selectedBook when we click the 'New Book' button.
2. Add an onClick statement to the JSX code for the 'New Book' button input element:

<input type={'button'}

value="New Book"

onClick={()=> props.setSelectedBook(newBook)}

/>

1. In the onClick handler we are calling the setSelectedBook that was passed into the BookList from the App component which maintains the selectedBook state.
2. Save the file.
3. Add a BookForm usage property: The BookForm allows us to either enter a new record or edit an existing record. But to save our results we need to call two different REST server endpoints. To save a new record we call a POST endpoint. To update and existing record we call a PUT endpoint. We will need to know how the BookForm is being used so that we can send the request to the right endpoint when we click the 'Save' button.
4. For this purpose we will add a const to the BookForm that we can use to determine the form's current use.
5. Open the bookform.js file in an editor.
6. Add the following statement as the first line in the BookForm function:

const usage = (props.book.id < 0 )?'add':'edit';

1. This creates a constant named 'usage' with the one of two values: 'add' or 'edit'. The empty book we pass in with the 'New Book' button always has an id value of -1. We use this to determine when the form is being used to add a new book. Any books with non- negative ids are assumed to be ones that we are editing and plan to update.
2. Display the current form usage by adding the usage value to the title of the BookForm like this:

<h4>Book Form - {usage} </h4>

1. Save the bookform.js file. Let the browser update.
2. Try clicking on the 'New Book' button and then on a book in the list. You should see that the usage value that appears in the BookList title section matches the intended usage: add or edit.
3. Add input field edit capability: You may have noticed that the BookForm input fields do not yet accept any new data. Since components are immutable in React we need to add some extra code to make sure that input fields are updated on-screen when the user enters new data.
4. Open the bookform.js file in an editor. Update the React import statement at the top of the file to also import useState:

import React, { useState } from 'react';

1. Add the following useState statement as the first line in the BookForm function:

const [ book, setBook ] = useState(props.book);

1. This will allow us to maintain the state of the book that we are creating/editing in the BookForm. To make this work we will also have to change the value attributes of the input fields to get the book from this state variable as opposed to the props object.
2. Edit the value attributes of the input statements to remove 'props.' like this:
3. Change This: value={props.book.isbn} /> To This: value={book.isbn} />
4. Make sure to make this same change in the other two input fields.
5. Add the following function into the BookForm function right before the return statement:

const handleChange = function (event) { const name = event.target.name; const value = event.target.value; book[name] = value;

setBook({ ...book });

}

1. This function will be hooked up to the onChange event of each input field. When called it will update the appropriate property in the book object to include the last keystroke entered into its input field.
2. Edit the JSX code of the BookForm() function to add onChange like this:

<input type={'text'} name={'isbn'} onChange={(e)=>handleChange(e)} value={book.isbn}/></td>

1. Make sure to do this for the other two input fields as well.
2. Save the file.
3. Add save functionality: you will modify the application so that the selected book shown in the Book Form can be edited and saved to the the REST service.
4. Also in order to save or update records we need to add functions that make POST and PUT calls to the REST service. We will make these changes now.
5. Open the src/rest/index.js file in an editor. Add the following addBook and updateBook functions to the file:

export function addBook(book) { let url = "/books/";

let myHeaders = new Headers({ "Content-Type": "application/json" }); delete book.id;

let body = JSON.stringify(book); var myInit = {

method: 'POST', body: body, headers: myHeaders, mode: 'cors'

};

let promise = fetch(url, myInit); return promise.then((response) => {

return response.text();

});

}

export function updateBook(book) { let url = "/books/" + book.id;

let myHeaders = new Headers({ "Content-Type": "application/json" }); let body = JSON.stringify(book);

var myInit = { method: 'PUT', body: body, headers: myHeaders, mode: 'cors'

};

let promise = fetch(url, myInit); return promise.then((response) => {

return response.text();

});

}

1. These functions are very similar to the getBooks function we added earlier. The call itself is executed in the 'let promise' line and then a Promise is returned that lets the caller subscribe to the results.
2. There is one more REST related function 'deleteBook' that we will need later on. Lets add that one now as well. ( text for this functions is also available to cut and paste from LabFiles\books-project\rest.js )

export function deleteBook(book) {

let myHeaders = new Headers({ "Content-Type": "application/json" }); var myInit = { method: 'DELETE', headers: myHeaders, mode: 'cors' }; let promise = fetch("/books/" + book.id , myInit);

return promise.then((response) => { return response.text();

});

}

1. Save the src/rest/index.js file. Open the bookform.js file in an editor. Add the following import statement at the top of the file. This will allow us to access the required REST functions.

import { addBook, updateBook, deleteBook } from '../rest/index';

1. Add the following handleSaveBook function into the BookForm function right before the return statement:

const handleSaveBook = function (){ if(usage === 'add'){

let promise = addBook(book); promise.then(function (text) {

console.log('handleSaveBook.add');

});

}

if(usage === 'edit'){

let promise = updateBook(book); promise.then(function (text) {

console.log('handleSaveBook.edit');

});

}

}

1. The usage value (add|edit) is used to make sure that saving of new records gets sent to the addBook() method which uses POST and update of records gets send to the updateBook() method that uses PUT. When each call comes back from the REST server we output a short message.
2. Add the following onChange attribute to the 'Save' input button element.

<input type={'button'} value="Save" onClick={()=>handleSaveBook()} />

1. Save the bookform.js file and wait for the app to update in the browser. Click on the 'New Book' button and type in some data into the fields in the BookForm. Save. It will be blank. Refresh the browser manually. The new book record you added should appear in the BookList. This proves that the REST service is being updating successfully. At this point we need the application to refresh the list from the back end data after any operations that change data.
2. Adding auto-refresh capability: To update the list we need to repeat the call to the background REST service. Let’s take a look at the code that is already being used to retrieve the data initially. Open the App.js file in an editor. Take a look at the existing useEffect() method:

useEffect(() => {

let promise = getBooks(); promise.then(

(text) => {

let bookArray = JSON.parse(text); setBooks(bookArray);

}

)

}, []);

1. The second parameter to useEffect is an empty array []. When the empty array is used the code in useEffect will only be called once at the start of the application. What want to do is to have this code called any time we expect the back end data to have changed, for example after a record is added, updated or deleted. To do this we need to provide a variable inside the array. After each render React will check the value of the variable and if it has changed since the last render React will call the useEffect code which in this case means it will make a new call to the back end and get and data that's been updated. We will add a state variable named refreshFlag to the App component for this purpose.
2. Add the following useState statement as the first line in the App() function:

const [refreshFlag, setRefreshFlag] = useState(0);

1. Next add a method called refresh that can be used to trigger a refresh by changing the value of the refreshFlag. The flag variable holds a number so we just need to increment it to trigger the update. Add this function right before return:

const refresh = function(noChanges=false){ if(!noChanges){

setRefreshFlag(refreshFlag + 1 );

}

setSelectedBook(null);

}

1. After refreshing from the REST service we typically also want to reset the selectedBook so we are doing that here as well. In some cases we may ONLY want to reset the selectedBook (for example if we start to edit a record but decide to cancel the edit). In these cases we can use the same refresh method and pass true as a parameter.
2. Now go back to the useEffect function call and add the refreshFlag into the array in the second parameter as shown here in bold:

useEffect(() => {

let promise = getBooks(); promise.then(

(text) => {

let bookArray = JSON.parse(text); setBooks(bookArray);

}

)

}, [refreshFlag]);

1. We also need to pass the refresh method to the BookForm so it can be called after any data changing operations. Do this in the JSX code of the Conditional function:

return <BookForm book={selectedBook} setBook={setSelectedBook}

refresh={refresh} />

1. Save the App.js file. Open the bookform.js file in an editor. First we will take care of the 'Cancel' use case. Add the following onClick attribute to the Cancel input button:

onClick={()=>props.refresh(true)}

1. Save the bookform.js file and let the browser update. Click on the 'New Book' button to open up the BookForm. Now click on the 'Cancel' button in the BookForm. The BookForm should go away and no book should be selected. Refresh the browser manually. Click on one of the books in the list. This will open the BookForm up with the selected record. Now click on 'Cancel'. The BookForm should close and no record should be selected in the BookList. Next we will use the refresh function to refresh after data operations. Open the bookform.js file in an editor. Add the props.refresh() call to both parts of the handleSaveBook function as shown here in bold:

const handleSaveBook = function (){ if(usage === 'add'){

let promise = addBook(book); promise.then(function (text) {

console.log('handleSaveBook.add');

props.refresh();

});

}

if(usage === 'edit'){

let promise = updateBook(book); promise.then(function (text) {

console.log('handleSaveBook.edit');

props.refresh();

});

}

}

1. Save the bookform.js file and let the browser refresh. Click on a book in the list and make a change to its title field. Click the 'Save' button. The BookForm should go away and the BookList should update to show the change you just made. Lets try it also with a new record.
2. Click on the 'New Book' button. Enter some data into the three fields. Click the 'Save' button. The BookForm should go away and the BookList should update to show the added record.
3. Add delete book capability: The 'Delete' button is currently non-functional. In this part we will add a click handler to the button that results in a DELETE call to the REST service.
4. Open the bookform.js in an editor. Add 'deleteBook' to the REST functions import statement as shown here in bold if it is not already added:

import { addBook, updateBook, deleteBook } from '../rest/index';

1. Add the following method to the BookForm() function right before the return statement:

const handleDeleteBook = function (){ if(usage === 'add'){

props.refresh(true);

}

if(usage === 'edit'){

let promise = deleteBook(book); promise.then(function (text) {

props.refresh();

});

}

}

1. If the user clicks on 'Delete' while adding a new record there is no need to call the back end service so we just let the refresh function unset the selectedBook by passing in 'true'.
2. If the user selected a record then we will be in 'edit' mode and clicking on 'Delete' then results in a delete call to the back end using the deleteBook function. When that call returns we use refresh to not only unset the selectedBook but to also refresh the BookList data from the REST service.
3. To make this all work we have to add an onClick attribute the 'Delete' input button element as shown here:

<input type={'button'} value="Delete" onClick={()=>handleDeleteBook()}/>

1. Save the bookform.js file and let the browser update.
2. Click on one of the books in the list to select it. This should display the selected book in the BookForm.
3. Click on the 'Delete' button. The BookForm should go away, the BookList will be refreshed and the deleted book should be gone.
4. Try Clicking on the 'New Book' button. This opens the BookList with an empty book.
5. Then click on 'Delete'. It should have the same effect as 'Cancel' – The BookForm will go away and the selectedBook will be un-set.