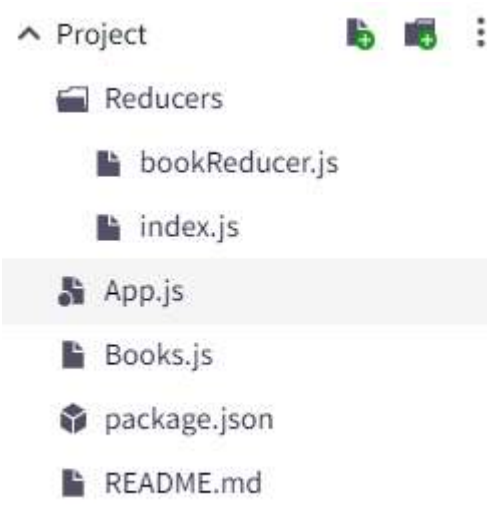
Bài thực hành Tạo ứng dụng BookApp

- Truy cập https://snack.expo.dev/

- Trong thư mục dự án tạo các thư mục và tập tin như sau:



Thư mục Reducers chứa hai tập tin là **bookReducer.js** và **index.js**. Nội dung tập tin **bookReducer.js** chứa một reducer (bookReducer) như sau:

```
const initialState = {
  books: [{ name: 'East of Eden', author: 'John Steinbeck' }]
}
const bookReducer = (state = initialState) => {
  return state
}

export default bookReducer
```

Đối tượng **initialState** sẽ giữ trạng thái bắt đầu. Trong trường hợp này, đó là một mảng các cuốn sách mà chúng ta sẽ nạp vào các đối tượng có chứa các props là name và author. Một hàm được tạo (bookReducer) lấy một đối số là state và đặt giá trị mặc định là initialState. Khi hàm này được gọi đầu tiên, state sẽ không được xác định và sẽ trả về initialState. Lúc này, mục đích duy nhất của hàm là trả về trạng thái state.

Sau khi tạo reducer đầu tiên (bookReducer), trong **index.js** chúng ta sẽ kết hợp tất cả các reducer để tạo trạng thái toàn cục (global state). Nội dung **index.js**

```
import { combineReducers } from 'redux'
import bookReducer from './bookReducer'

const rootReducer = combineReducers({
   bookReducer
})

export default rootReducer
```

Chúng ta phải import reducer từ **bookReducer** và hàm **combineReducers** từ Redux để kết hợp các reducer.

```
import { combineReducers } from 'redux'
import bookReducer from './bookReducer'
```

Trong trường hợp chúng ta chỉ có một reducer là **bookReducer**

```
const rootReducer = combineReducers({
   bookReducer
})
```

Trong phần này, chúng ta sẽ thêm **provider** vào ứng dụng. Một **provider** thường là một component cha truyền dữ liệu thuộc loại nào đó cho tất cả các component con. Trong Redux, provider chuyển state/store toàn cục đến phần còn lại của ứng dụng. Nội dung tập tin **App.js** như sau

```
import React from 'react'
import Books from './Books'
import rootReducer from './Reducers'
import { Provider } from 'react-redux'
import { createStore } from 'redux'


const store = createStore(rootReducer)

export default class App extends React.Component {
 render() {
   return (
      <Provider store={store} >
         <Books />
      </Provider>
    )
  }
```

```
}
```

Chúng ta import component **Books**, reducer **rootReducer**, **Provider** và hàm **createStore** để tạo store

```
import Books from './Books'
import rootReducer from './Reducers'
import { Provider } from 'react-redux'
import { createStore } from 'redux'
```

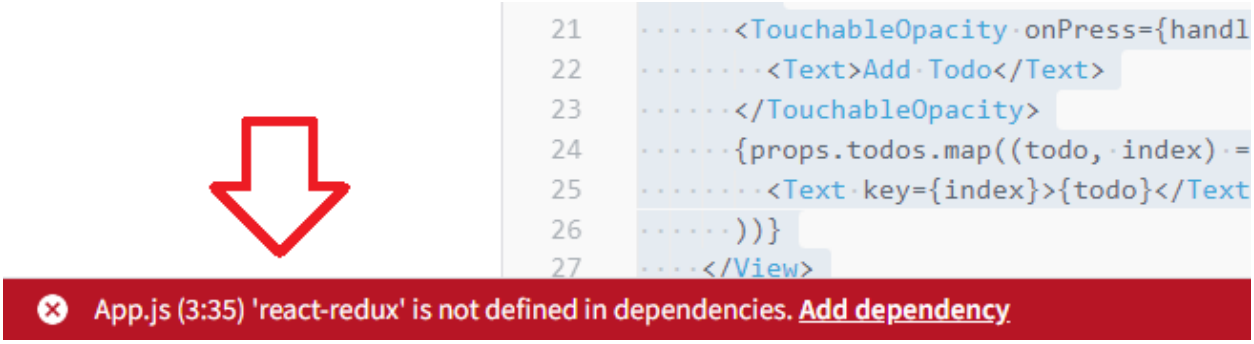
Khởi tạo store dùng hàm **createStore** với đối số là **rootReducer**

```
const store = createStore(rootReducer)
```

Kết quả kết xuất là component **Books** được bọc bởi **Provider** với *store* đóng vai trò là một *prop*

```
<Provider store={store} >
   <Books />
</Provider>
```

Chú ý rằng, chúng ta cần thêm đầy đủ các gói Redux và React Redux bằng cách nhấn vào **Add dependency**



Bây giờ chúng ta sẽ thêm component **Books** đến tập tin **Books.js**

```
import React from 'react'
import {
  Text,
  View,
  ScrollView,
  StyleSheet
} from 'react-native'

import { connect } from 'react-redux'
```

```
class Books extends React.Component<{}> {
render() {
  const { books } = this.props
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Books</Text>
      <ScrollView
        keyboardShouldPersistTaps='always'
        style={styles.booksContainer}
      >
      {
      books.map((book, index) => (
       <View style={styles.book} key={index}>
         <Text style={styles.name}>{book.name}</Text>
         <Text style={styles.author}>{book.author}</Text>
       </View>
       ))
      }
     </ScrollView>
    </View>
  )
 }
}

const styles = StyleSheet.create({
 container: {
  flex: 1
 },
 booksContainer: {
 borderTopWidth: 1,
 borderTopColor: '#ddd',
  flex: 1
 },
 title: {
  paddingTop: 30,
  paddingBottom: 20,
  fontSize: 20,
  textAlign: 'center'
 },
 book: {
  padding: 20
 },
 name: {
  fontSize: 18
 },
```

```
 author: {
  fontSize: 14,
  color: '#999'
 }
})
const mapStateToProps = (state) => ({
  books: state.bookReducer.books
})
export default connect(mapStateToProps)(Books)
```

Chúng ta truy cập Redux store từ một component con bằng cách dùng hàm ***connect*** từ **react-redux**. Đối số đầu tiên của hàm ***connect*** là một hàm cho phép chúng ta truy cập vào toàn bộ trạng thái Redux. Sau đó, chúng ta có thể trả lại một đối tượng với bất kỳ phần nào của store chúng ta muốn truy cập vào.

connect là một hàm *curried*, nghĩa là, một hàm trả về một hàm khác. Nguyên mẫu hàm *connect* với hai bộ đối số trông như sau *connect(args)(args)*. Các thuộc tính trong đối tượng được trả về từ đối số đầu tiên đến *connect* sau đó được cung cấp cho component dưới dạng props.

Để bắt đầu dùng *connect*, chúng ta import

```
import { connect } from 'react-redux'
```

Vì mảng ***books*** đã được trả về từ hàm *connect* nên chúng ta có thể truy cập vào nó như props

```
const { books } = this.props
```

Ánh xạ qua mảng và hiển thị thông tin về *name* và *author* của mỗi cuốn sách

```
books.map((book, index) => (
       <View style={styles.book} key={index}>
         <Text style={styles.name}>{book.name}</Text>
         <Text style={styles.author}>{book.author}</Text>
       </View>
))
```

Lấy trạng thái Redux và trả về một đối tượng với một khóa chứa mảng *books* từ *bookReducer*

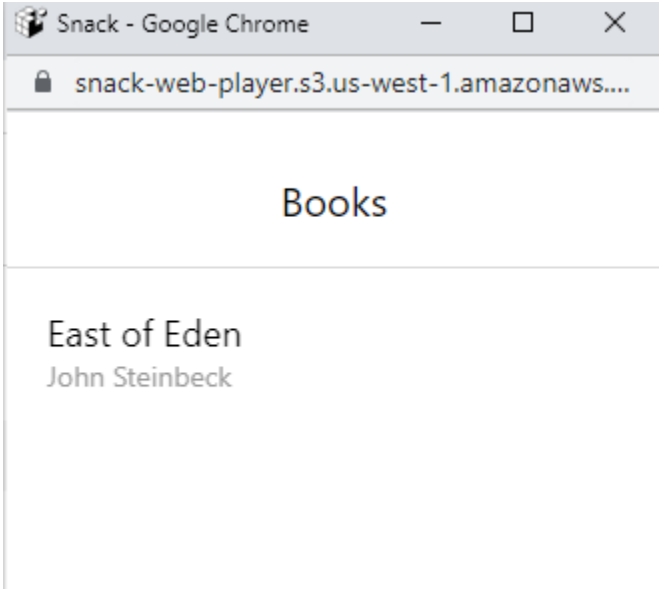
```
const mapStateToProps = (state) => ({
  books: state.bookReducer.books
})
```

Cuối cùng là export hàm connect

```
export default connect(mapStateToProps)(Books)
```

chuyển vào *mapStateToProps* làm đối số đầu tiên và Books là đối số duy nhất trong bộ đối số thứ hai của *connect*.

Lúc này giao diện ứng dụng như sau



Chúng ta đã truy cập vào trạng thái Redux, bước hợp lý tiếp theo là thêm một số chức năng cho phép chúng ta thêm sách vào mảng ***books***. Đây là lúc chúng ta dùng *actions*.

actions về cơ bản là các hàm trả về các đối tượng gửi dữ liệu đến *store* và cập nhật reducer; actions là cách duy nhất để thay đổi *store*. Mỗi action chứa một thuộc tính *type* để reducer có thể sử dụng chúng.

Trong thư mục dự án chúng ta tạo tập tin tên **actions.js** có nội dung như sau:

```
export const ADD_BOOK = 'ADD_BOOK'

export function addBook (book) {
```

```
  return {
    type: ADD_BOOK,
    book
  }
}
```

Chúng ta tạo một hằng ADD_BOOK để dùng lại trong reducer (bookReducer). Hàm addBook chứa một đối số là đối tượng book và trả về một đối tượng chứa thuộc tính type và đối số book.

Kế tiếp chúng ta kết nối đến **bookReducer** để sử dụng action **addBook** bằng cách cập nhật lại nội dung tập tin **bookReducer.js** như sau:

```
import { ADD_BOOK } from '../actions'

const initialState = {
   books: [{ name: 'East of Eden', author: 'John Steinbeck' }]
}

const bookReducer = (state = initialState, action) => {
   switch(action.type) {
      case ADD_BOOK:
         return {
           books: [
             ...state.books,
             action.book
           ]
         }
      default:
        return state
    }
}

export default bookReducer
```

Chúng ta cần import hằng ADD_BOOK

```
import { ADD_BOOK } from '../actions'
```

Hàm **bookReducer** thêm một đối số thứ hai là **action**

```
const bookReducer = (state = initialState, action) => {
```

Sử dụng lệnh switch với đối số là action.type kiểm tra thuộc tính type từ action addBook. Nếu type là ADD_BOOK thì một mảng books mới được tạo; ngược lại, trạng thái ban đầu được trả về

```
switch(action.type) {
      case ADD_BOOK:
         return {
           books: [
             ...state.books,
             action.book
           ]
         }
      default:
        return state
    }
```

Bây giờ chúng ta cần cập nhật lại giao diện từ component Books. Đầu tiên, chúng ta cần thêm các component là TextInput và TouchableOpacity, hàm addBook và khởi tạo một đối tượng InitialState với các thuộc tính là name và author:

```
import React from 'react'
import {
  Text,
  View,
  ScrollView,
  StyleSheet,
  TextInput,
  TouchableOpacity
} from 'react-native'

import { connect } from 'react-redux'
import { addBook } from './actions'

const initialState = {
  name: '',
  author: ''
}
```

Kế tiếp, trong lớp Books, chúng ta thêm đoạn mã sau phía trên hàm **render()**:

```
state = initialState

updateInput = (key, value) => {
   this.setState({
      ...this.state,
      [key]: value
   })
}

addBook = () => {
   this.props.dispatchAddBook(this.state)
   this.setState(initialState)
}
```

Chúng ta cần tạo 3 thứ:

- Trạng thái component gán đến InitialState.

- Một phương thức (updateInput) cập nhật trạng thái component khi các giá trị TextInput thay đổi.

- Một phương thức (addBook) sẽ gửi action tới Redux chứa các giá trị của book (name và author) khi nhấn nút gửi. Ở đây, hàm *addBook* gọi hàm *dispatchAddBook* có thể truy cập như *props* từ hàm ***connect***.

Bây giờ, chúng ta cập nhật đoạn mã trong hàm render() phía dưới </ScrollView>

```
    ...
    </ScrollView>
    <View style={styles.inputContainer}>
      <View style={styles.inputWrapper}>
       <TextInput
         value={this.state.name}
         onChangeText={value => this.updateInput('name', value)}
         style={styles.input}
         placeholder='Book name'
       />
       <TextInput
         value={this.state.author}
         onChangeText={value => this.updateInput('author', value)}
         style={styles.input}
         placeholder='Author Name'
       />
      </View>
      <TouchableOpacity onPress={this.addBook}>
```

```
        <View style={styles.addButtonContainer}>
          <Text style={styles.addButton}>+</Text>
        </View>
      </TouchableOpacity>
    </View>
```

Cập nhật style phía trên ***container***

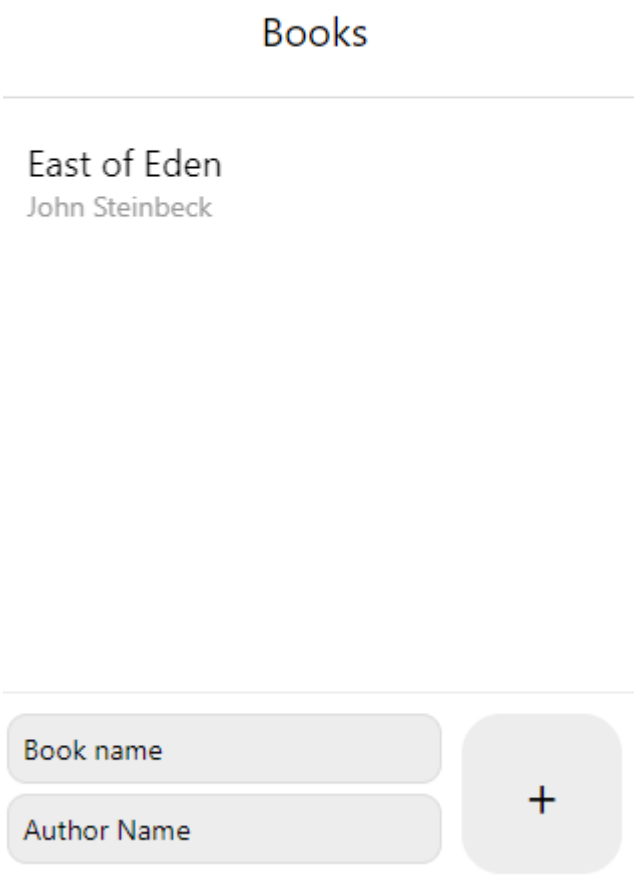
```
const styles = StyleSheet.create({
 inputContainer: {
  padding: 10,
  backgroundColor: '#ffffff',
  borderTopColor: '#ededed',
  borderTopWidth: 1,
  flexDirection: 'row',
  height: 100
 },
 inputWrapper: {
  flex: 1
 },
 input: {
  height: 44,
  padding: 7,
  backgroundColor: '#ededed',
  borderColor: '#ddd',
  borderWidth: 1,
  borderRadius: 10,
  flex: 1,
  marginBottom: 5
 },
 addButton: {
  fontSize: 28,
  lineHeight: 28
 },
 addButtonContainer: {
  width: 80,
  height: 80,
  backgroundColor: '#ededed',
  marginLeft: 10,
  justifyContent: 'center',
  alignItems: 'center',
  borderRadius: 20
 },
 container: {
```

```
...
```

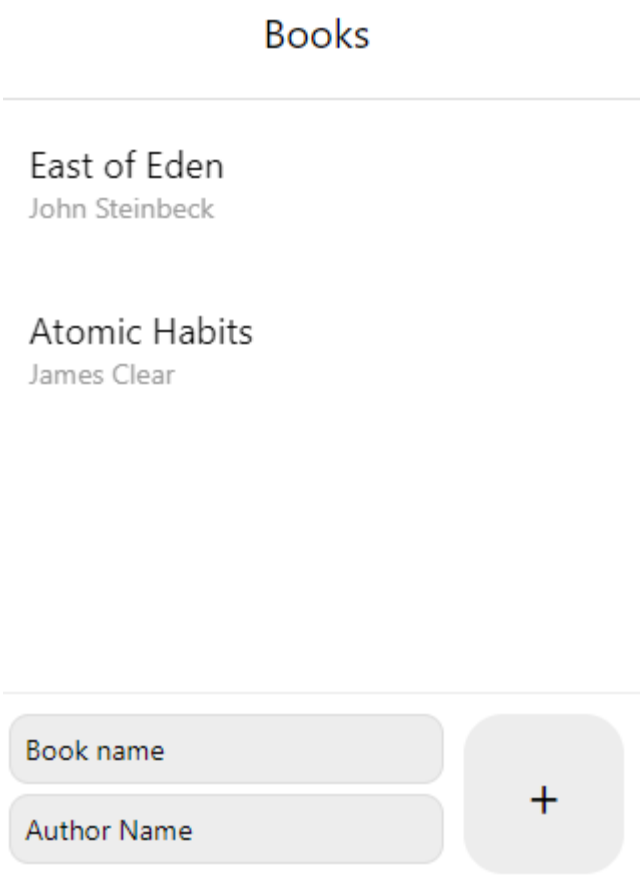
Tạo một đối tượng ***mapDispatchToProps*** và chuyển nó thành tham số thứ hai trong hàm *connect*

```
const mapStateToProps = (state) => ({
  books: state.bookReducer.books
})

const mapDispatchToProps = {
dispatchAddBook: (book) => addBook(book)
}

export default connect(mapStateToProps, mapDispatchToProps)(Books)
```

Giao diện ứng dụng lúc này



Thêm cuốn sách mới và nhấn vảo nút +



Xóa giá trị từ Redux store trong một reducer

Điều đầu tiên cần nghĩ đến khi xóa một mục khỏi một mảng như thế này là làm thế nào để xác định một cuốn sách là duy nhất. Một người dùng có thể có nhiều sách với cùng một tác giả (author) hoặc nhiều cuốn sách có cùng tên (name), vì vậy việc sử dụng các thuộc tính hiện có sẽ không hiệu quả. Thay vào đó, bạn có thể sử dụng thư viện chẳng hạn như **uuid** để tạo số nhận dạng duy nhất một cách nhanh chóng. Cài đặt thư viện **uuid** như sau:

```
npm i uuid –save
```

Trong Snack Expo tại https://snack.expo.dev/ chúng ta import **uuid** trong tập tin **bookReducer.js** như sau

```
import {v4 as uuidv4} from 'uuid'
```

Tất nhiên sau đó Snack Expo yêu cầu thêm đến Dependencies và chúng ta chỉ việc nhấn vào Add dependency.

Thư viện **uuid** có một vài thuật toán để lựa chọn. Tại đây, chúng ta chỉ nhập thuật toán **v4**, thuật toán tạo **chuỗi 32 ký tự ngẫu nhiên (**random 32-character string**).**

Kế tiếp chúng ta thêm một thuộc tính là id đến đối tượng InitialState

```
const initialState = {
  books: [{ name: 'East of Eden', author: 'John Steinbeck', id: uuidv4() }]
}
```

Giá trị duy nhất id được tạo bằng cách gọi hàm **uuidv4()**

Kế tiếp, trong **actions.js** chúng ta tạo một action mới gọi là **removeBook** và chúng ta cũng cần cập nhật **id** đến **addBook**

```
export const ADD_BOOK = 'ADD_BOOK'
export const REMOVE_BOOK = 'REMOVE_BOOK'
import {v4 as uuidv4} from 'uuid'

export function addBook (book) {
  return {
    type: ADD_BOOK,
    book: {
      ...book,
      id: uuidv4()
    }
  }
}

export function removeBook (book) {
  return {
   type: REMOVE_BOOK,
   book
  }
}
```

Và tất nhiên chúng ta cũng cần cập nhật thêm trong **bookReducer**

```
import { ADD_BOOK, REMOVE_BOOK } from '../actions'
import {v4 as uuidv4} from 'uuid'

const initialState = {
  books: [{ name: 'East of Eden', author: 'John Steinbeck', id: uuidv4() }]
}

const bookReducer = (state = initialState, action) => {
   switch(action.type) {
```

```
      case ADD_BOOK:
         return {
           books: [
             ...state.books,
             action.book
           ]
         }
      case REMOVE_BOOK:
        const index = state.books.findIndex(
                 book => book.id === action.book.id)
        return {
          books: [
           ...state.books.slice(0, index),
           ...state.books.slice(index + 1)
          ]
       }
      default:
        return state
    }
}

export default bookReducer
```

Cập nhật **Books.js**

```
...
import { addBook, removeBook } from './actions'

const initialState = {
  name: '',
  author: ''
}

class Books extends React.Component<{}> {

...

removeBook = (book) => {
  this.props.dispatchRemoveBook(book)
}

render() {
  const { books } = this.props
  return (
    <View style={styles.container}>
```

```
      <Text style={styles.title}>Books</Text>
      <ScrollView
        keyboardShouldPersistTaps='always'
        style={styles.booksContainer}
      >
      {
      books.map((book, index) => (
       <View style={styles.book} key={index}>
         <Text style={styles.name}>{book.name}</Text>
         <Text style={styles.author}>{book.author}</Text>
         <Text onPress={() => this.removeBook(book)}>
           Remove
         </Text>
       </View>
       ))

      }
     </ScrollView>
...
  )
 }
}

const styles = StyleSheet.create({
 ...
})

const mapStateToProps = (state) => ({
  books: state.bookReducer.books
})

const mapDispatchToProps = {
  dispatchAddBook: (book) => addBook(book),
  dispatchRemoveBook: (book) => removeBook(book)
}

export default connect(mapStateToProps, mapDispatchToProps)(Books)
```