BookVerse: Add New Book and Manage Data Flow

1. Introduction

BookVerse is a Single Page Application (SPA) built using React and Vite. This project is an extension of the earlier BookVerse application and focuses on implementing form handling, validation, and predictable state management using the Flux architecture. The goal of this activity is to allow an admin to add new books to the application while maintaining a clear and unidirectional data flow.

2. User Story

As an admin,
I want to add new books through a form with validation,
So that I can expand the book collection easily.

3. Project Scope (Extension)

This activity extends the existing BookVerse application. All previously implemented features such as routing, reusable components, hooks, lifecycle methods, HOCs, and render props are retained. The current activity adds the following capabilities:

- A validated form to add new books
- Flux-based state management for adding books
- Dependency Injection for store management
- Clear demonstration of SPA data flow

4. Technologies Used

- React (with Vite)
- React Router (SPA navigation)
- Formik (form handling)
- Yup (form validation)
- Flux architecture (Action, Dispatcher, Store)
- json-server (mock backend for persistence)
- Bootstrap (UI styling)

5. Form Handling and Validation

The Add Book feature is implemented using Formik for form state management and Yup for validation. The form contains the following fields:

- Title (required)
- Author (required)
- Price (required, numeric)

Formik manages the form state and submission, while Yup ensures that invalid data cannot be submitted. This results in a clean and reliable form-handling mechanism.

6. Flux Architecture Implementation

The application follows the Flux pattern to ensure predictable state management.

6.1 Action

An action is triggered when the admin submits the Add Book form. The action describes what happened (adding a new book) and carries the book data.

6.2 Dispatcher

The dispatcher acts as a central hub. It receives actions and broadcasts them to all registered stores. Components do not directly update the store.

6.3 Store

The BookStore maintains the list of newly added books in memory. It listens to dispatched actions, updates its internal state, and emits change events when data is modified.

7. Dependency Injection

The BookStore instance is created in the application entry point and injected into the App component via props. This ensures loose coupling between components and the store. Components consume the store without being responsible for its creation, improving modularity and testability.

8. Component Lifecycle and Data Flow

When the Home component mounts, it:

1. Fetches existing books from the backend
2. Reads any books present in the Flux store
3. Subscribes to store updates

When a new book is added:

- The action is dispatched
- The store updates its state
- The store emits a change
- The Home component re-renders automatically

This demonstrates unidirectional data flow and effective use of the component lifecycle within the Flux context.

9. Backend Persistence

To ensure that added books persist across page refreshes, the application sends a POST request to the json-server backend when a new book is added. The Flux store is still used to update the UI immediately, while the backend ensures data persistence.

10. SPA Behavior

The application uses React Router for navigation. Pages such as Home, Book Details, and Add Book are navigated without full page reloads. This confirms that the application behaves as a true Single Page Application.

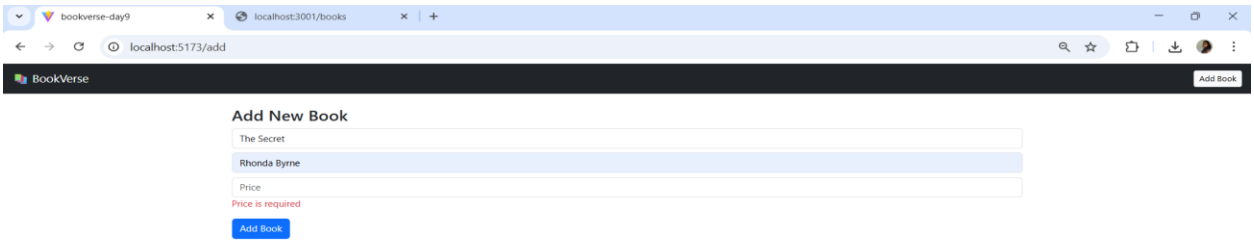
11. Data Flow Summary

1. Admin submits Add Book form
2. Formik validates input using Yup
3. Action is triggered with book data
4. Dispatcher forwards action to the store
5. Store updates state and emits change

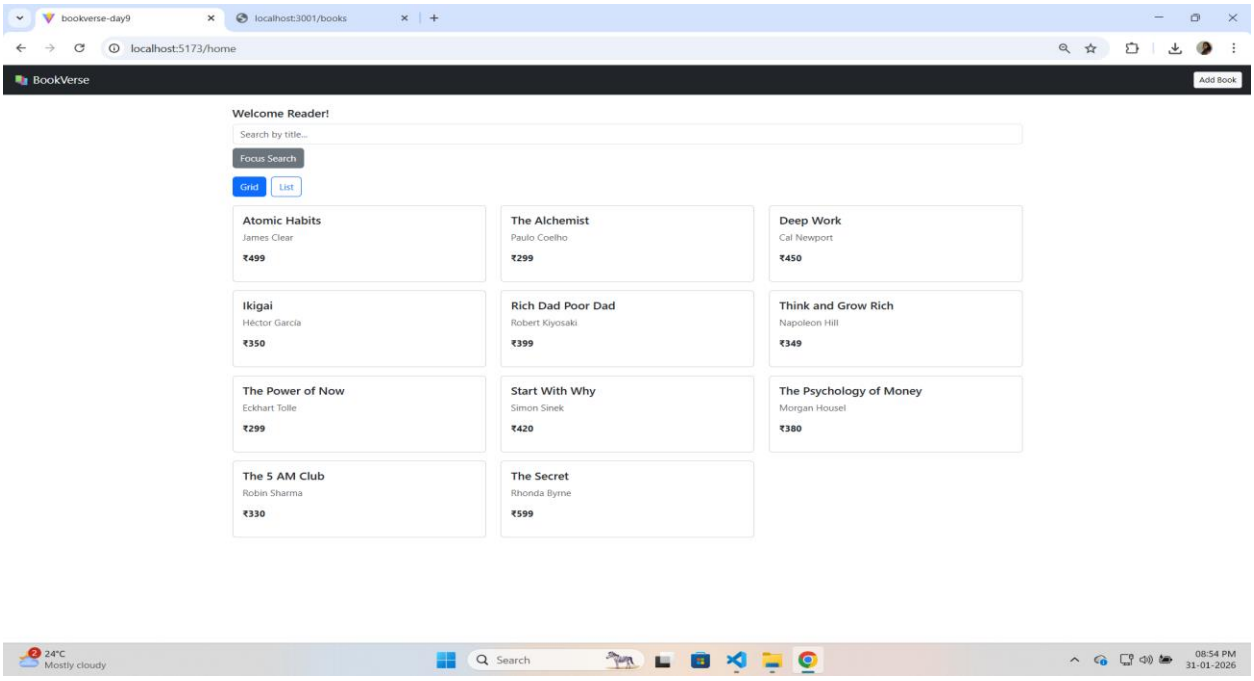
6. Home component re-renders
7. Backend stores the new book

12. Output Screenshots

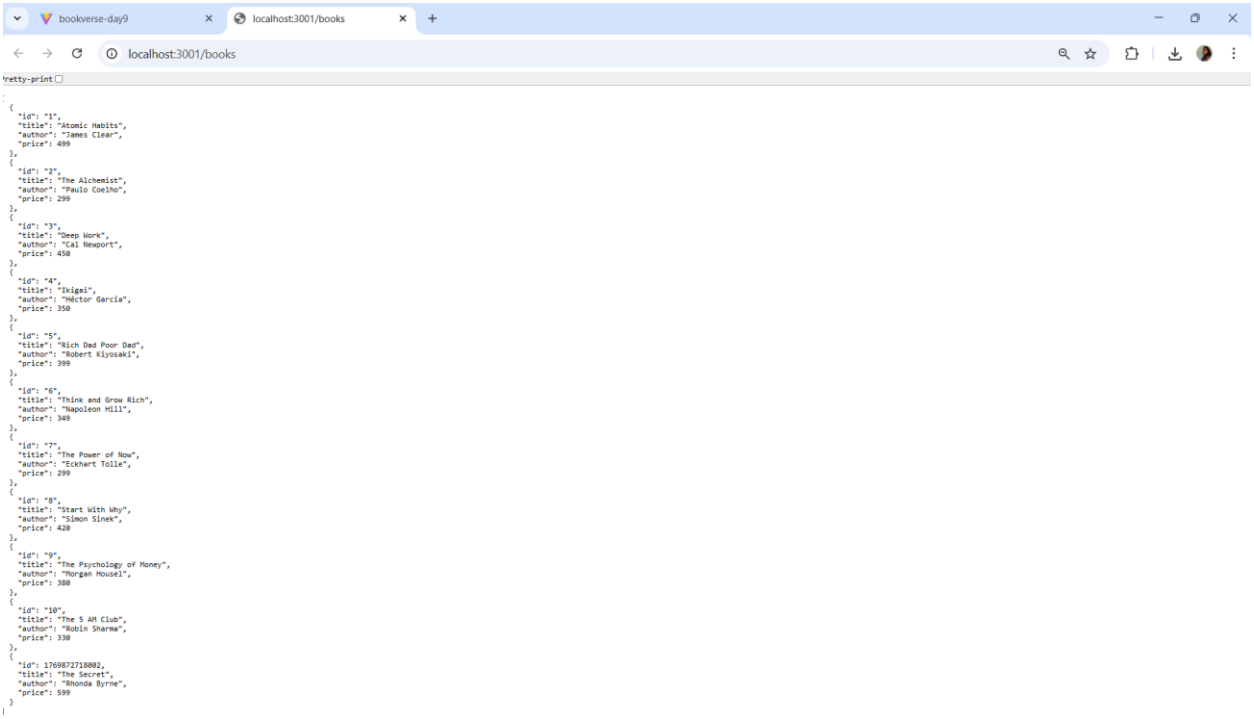
Bookform.png



HomePageAfterAdding.png



updatedJsonServer.png



13. Code

AddBook.jsx

```
import { Formik, Form, Field, ErrorMessage } from "formik";
import * as Yup from "yup";
import { addBook } from "../flux/BookActions";
import { useNavigate } from "react-router-dom";


const BookSchema = Yup.object({
  title: Yup.string().required("Title is required"),
  author: Yup.string().required("Author is required"),
  price: Yup.number()
    .typeError("Price must be a number")
    .required("Price is required")
```

```
});

function AddBook() {
  const navigate = useNavigate();

  return (
    <div className="container mt-4">
      <h3>Add New Book</h3>

      <Formik
        initialValues={{ title: "", author: "", price: "" }}
        validationSchema={BookSchema}
        onSubmit={async (values, { resetForm }) => {
          const newBook = {
            id: Date.now(),
            title: values.title,
            author: values.author,
            price: Number(values.price)
          };

          try {
            await fetch("http://localhost:3001/books", {
              method: "POST",
              headers: {
                "Content-Type": "application/json"
              },
              body: JSON.stringify(newBook)
```

```
      });

      addBook(newBook);
      resetForm();
      navigate("/home");
    } catch (error) {
      console.error(error);
    }
  }}
>
  <Form>
    <div className="mb-2">
      <Field className="form-control" name="title" placeholder="Title" />
      <div className="text-danger">
        <ErrorMessage name="title" />
      </div>
    </div>

    <div className="mb-2">
      <Field className="form-control" name="author" placeholder="Author" />
      <div className="text-danger">
        <ErrorMessage name="author" />
      </div>
    </div>

    <div className="mb-3">
      <Field className="form-control" name="price" placeholder="Price" />
```

```
            <div className="text-danger">
              <ErrorMessage name="price" />
            </div>
          </div>

          <button type="submit" className="btn btn-primary">
            Add Book
          </button>
        </Form>
      </Formik>
    </div>
  );
}

export default AddBook;
```

AppDispatcher.js

```
class AppDispatcher {
  constructor() {
    this.callbacks = [];
  }

  register(callback) {
    this.callbacks.push(callback);
  }
```

```
  dispatch(action) {
    this.callbacks.forEach(cb => cb(action));
  }
}

export default new AppDispatcher();
```

BookActions.js

```
import Dispatcher from "./AppDispatcher";

export const addBook = (book) => {
  Dispatcher.dispatch({
    type: "ADD_BOOK",
    payload: book
  });
};
```

BookStore.js

```
import Dispatcher from "./AppDispatcher";

class BookStore {
  constructor() {
    this.books = [];
    this.listeners = [];

    Dispatcher.register(this.handleActions.bind(this));
  }
```

```
  handleActions(action) {
    if (action.type === "ADD_BOOK") {
      this.books.push(action.payload);
      this.emitChange();
    }
  }

  getBooks() {
    return this.books;
  }

  subscribe(listener) {
    this.listeners.push(listener);
  }

  emitChange() {
    this.listeners.forEach(l => l());
  }
}

export default BookStore;
```

Main.jsx

```
import { createRoot } from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App";
```

```
import BookStore from "./flux/BookStore";
import "bootstrap/dist/css/bootstrap.min.css";
import "./index.css";

const bookStore = new BookStore(); // Dependency Injection

createRoot(document.getElementById("root")).render(
  <BrowserRouter>
    <App bookStore={bookStore} />
  </BrowserRouter>
);
```

App.jsx

```
import { Routes, Route, Navigate } from "react-router-dom";
import Home from "./pages/Home";
import BookDetails from "./pages/BookDetails";
import AddBook from "./pages/AddBook";
import Navbar from "./components/Navbar";

function App({ bookStore }) {
  return (
    <>
      <Navbar />
      <Routes>
        <Route path="/" element={<Navigate to="/home" />} />
        <Route path="/home" element={<Home bookStore={bookStore} />} />
        <Route path="/book/:id" element={<BookDetails />} />
        <Route path="/add" element={<AddBook />} />
```

```
      </Routes>
    </>
  );
}

export default App;
```

14. Conclusion

This project successfully extends the existing BookVerse application by implementing Formik-based form handling, Yup validation, Flux architecture, Dependency Injection, and SPA behavior. The application demonstrates unidirectional data flow, modular design, and persistence using a backend service. All requirements specified in the assignment have been fully satisfied.