**03. Books**

You can check your solutions in [Judge.](https://judge.softuni.org/Contests/Compete/Index/5189#2)

**Your Task**

Using **Mocha** and **Chai** write **JS Unit Tests** to test a variable namedbookService, which represents an object. You may use the following code as a template:

|  |
| --- |
| describe(**"*Tests* …"**, **function**() {  describe(**"*TODO* …"**, **function**() {  ***it***(**"*TODO …*"**, **function**() {  *//* ***TODO:*** …  });  });  *//* ***TODO:*** …  }); |

The bookServiceobject that should have the following functionality:

* getBooks**()** - A function that returns an object with a response from a book service.
  + If the request is successful, return an object with status **200** and an array of **3 books**, each containing **id**, **title**, **author**, **year** and **genre**.
* addBook**(book)** - A function that adds a new book to the list of books.
  + **book** is an object that must contain the fields **id**, **title**, **author**, **year** and **genre**.
  + If the book is successfully added, return an object with status **201** and a success message: "**Book added successfully**."
  + If any of the fields are missing or invalid, return an object with status **400** and an error message: "**Invalid Book Data**!"
* deleteBook**(**bookID**)** - A function that deletes a book by its id.
  + bookID is a string with the Id of the book to be deleted
  + If the book is successfully deleted, return an object with status **200** and a success message: "**Book deleted successfully**."
  + If the book is not found, return an object with status **404** and an error message: "**Book Not Found**!"
* updateBook**(**oldId**,** newBook**) -** A function that updates the information of an existing book.
  + oldId is a string with the id of the book to be updated.
  + newBook is an object containing the new book data.
  + If the book is successfully updated, return an object with status **200** and a success message: "**Book updated successfully**."
  + If the book is **not found**, return an object with status **404** and an error message: "**Book Not Found**!"
  + If any property of the new book is **empty or missing**, return a **400** status and an error message: "**Invalid Book Data!**"

**JS Code**

To ease you in the process, you are provided with an implementation that meets all of the specification requirements for the bookService object:

|  |
| --- |
| bookService.js |
| const bookService = {  books: [  { id: "1", title: "1984", author: "George Orwell", year: 1949, genre: "Dystopian" },  { id: "2", title: "To Kill a Mockingbird", author: "Harper Lee", year: 1960, genre: "Fiction" },  { id: "3", title: "The Great Gatsby", author: "F. Scott Fitzgerald", year: 1925, genre: "Classic" }  ],    // getBooks() - Returns a list of books with status 200  getBooks() {  return {  status: 200,  data: this.books  };  },  // addBook(book) - Adds a new book to the list. If any field is missing or invalid, returns status 400  addBook(book) {  if (!book.id || !book.title || !book.author || !book.year || !book.genre) {  return {  status: 400,  error: "Invalid Book Data!"  };  }  this.books.push(book);  return {  status: 201,  message: "Book added successfully."  };  },  // deleteBook(bookID) - Deletes a book by ID. If not found, returns status 404  deleteBook(bookID) {  const bookIndex = this.books.findIndex(book => book.id === bookID);  if (bookIndex === -1) {  return {  status: 404,  error: "Book Not Found!"  };  }  this.books.splice(bookIndex, 1);  return {  status: 200,  message: "Book deleted successfully."  };  },  // updateBook(oldId, newBook) - Updates an existing book by ID. If not found, returns status 404  // If new book data is invalid, returns status 400  updateBook(oldId, newBook) {  const bookIndex = this.books.findIndex(book => book.id === oldId);  if (bookIndex === -1) {  return {  status: 404,  error: "Book Not Found!"  };  }  if (!newBook.id || !newBook.title || !newBook.author || !newBook.year || !newBook.genre) {  return {  status: 400,  error: "Invalid Book Data!"  };  }  this.books[bookIndex] = newBook;  return {  status: 200,  message: "Book updated successfully."  };  }  }; |

**Submission**

Submit your tests inside a describe**()** statement, as shown above.

|  |
| --- |
| import { expect } from "chai";  import {bookService} from "../functions/bookService.js";  describe("Book Service Tests", function() {      describe("getBooks()", function() {          // Test: Should return a status 200 and an array of books          // 1. Verify that the response status is 200.          // 2. Check that the first book includes the required keys: 'id', 'title', 'author', 'year', 'genre'.          it("Should return a status 200 and an array of books", function(){              const response = bookService.getBooks();              expect(response.status).to.equal(200);              expect(response.data).to.be.an("array")              expect(response.data.length).to.equal(3)              expect(response.data[0]).to.have.keys('id', 'title', 'author', 'year', 'genre')              expect(response.data[1]).to.have.keys('id', 'title', 'author', 'year', 'genre')              expect(response.data[2]).to.have.keys('id', 'title', 'author', 'year', 'genre')          })        });      describe("addBook()", function() {          // Test: Should add a new book successfully          // 1. Create a new valid book object.          // 2. Verify the response status is 201 and the success message is correct.          // 3. Verify that the newly added book is present in the book list.          it("Should add a new book successfully", function(){              const newBook =                  {                      id: "4",                      title: "QA advance",                      author: "Ivan Orwell",                      year: 2024,                      genre: "Action"              }              const response = bookService.addBook(newBook);              expect(response.status).to.equal(201);              expect(response.message).to.equal("Book added successfully.")              const allBook = bookService.getBooks().data;                expect(allBook).to.deep.include(newBook);          })          // Test: Should return status 400 when adding a book with missing fields          // 1. Create an invalid book object with missing fields.          // 2. Check if the response status is 400 and the error message is "Invalid Book Data!".          it("Should return status 400 when adding a book with missing fields", function(){              const newBook =                  {                      id: "4",                      title: "QA advance",                      author: "Ivan Orwell",                  }              const response = bookService.addBook(newBook);              expect(response.status).to.equal(400);              expect(response.error).to.equal("Invalid Book Data!")            })      });      describe("deleteBook()", function() {          // Test: Should delete a book by id successfully          // 1. Add a book and then delete it by its ID.          // 2. Verify the response status is 200 and the success message is correct.          // 3. Ensure the book count returns the sum of the initial count of the books and the count of the added books from the tests          it("Should delete a book by id successfully", function(){              const newBook =              {                  id: "5",                  title: "QA advance",                  author: "Ivan Orwell",                  year: 2024,                  genre: "Action"          }          const initialBookLength = bookService.getBooks().data.length;          const response = bookService.addBook(newBook);          expect(response.status).to.equal(201);          expect(response.message).to.equal("Book added successfully.")          const allBook = bookService.getBooks().data;          expect(allBook).to.deep.include(newBook);          // delete book and check          const newResponse = bookService.deleteBook("5")          expect(newResponse.status).to.equal(200);          expect(newResponse.message).to.equal("Book deleted successfully.")          const allBookAfterDell = bookService.getBooks().data;          const foundBook =  allBookAfterDell.filter(book => book.id === "5")          expect(foundBook.length).to.be.equals(0)             //// check book length before and after delete          expect(allBookAfterDell.length).to.equal(initialBookLength)          })          // Test: Should return status 404 when deleting a book with a non-existent id          // 1. Attempt to delete a book with a non-existent ID.          // 2. Check that the response status is 404 and the error message is "Book Not Found!".          it("Should return status 404 when deleting a book with a non-existent id", function(){              const response = bookService.deleteBook("50000")              expect(response.status).to.equal(404);              expect(response.error).to.equal("Book Not Found!")          })      });      describe("updateBook()", function() {          // Test: Should update a book successfully          // 1. Create updated data for an existing book.          // 2. Verify the response status is 200 and the success message is correct.          // 3. Ensure that the updated book fields reflect the new data.          it("Should update a book successfully", function(){              const oldId = "1";              const newBook =                   {                  id: "6",                  title: "QA advance",                  author: "Ivan Orwell",                  year: 2024,                  genre: "Action"                   }              const response = bookService.updateBook(oldId, newBook)              expect(response.status).to.equal(200);              expect(response.message).to.equal("Book updated successfully.");              const allBook = bookService.getBooks().data;             expect(allBook).to.deep.include(newBook);          })          // Test: Should return status 404 when updating a non-existent book          // 1. Attempt to update a book that doesn't exist.          // 2. Check that the response status is 404 and the error message is "Book Not Found!".          it("Should return status 404 when updating a non-existent book", function(){              const oldId = "100000";              const newBook =                   {                  id: "6",                  title: "QA advance",                  author: "Ivan Orwell",                  year: 2024,                  genre: "Action"                   }              const response = bookService.updateBook(oldId, newBook)              expect(response.status).to.equal(404);              expect(response.error).to.equal("Book Not Found!");          })          // Test: Should return status 400 when updating with incomplete book data          // 1. Provide an incomplete book object with missing fields.          // 2. Verify that the response status is 400 and the error message is "Invalid Book Data!".          it("Should return status 400 when updating with incomplete book data", function(){              const oldId = "2";              const newBook =                   {                  id: "6",                  title: "QA advance",                     }              const response = bookService.updateBook(oldId, newBook)              expect(response.status).to.equal(400);              expect(response.error).to.equal("Invalid Book Data!");          })      });  }); |