Alex Mishev & Elian Konedarev



Sprint 3 Report

Prof. Vladimir Georgiev INF440

# User Stories

Spring #3 User Stories

|  |  |  |
| --- | --- | --- |
| Story 1 | As a logged-in user, I want to be able to search all books | * Logged-in users should see a “Search” menu link, which redirects to the search page * The search screen allows the user to enter a string and show all books that contain this string in their title, description and/or author * The results are shown in a list/table of books, each row showing the book information * Results should be sorted by the number of occurrences of the search string in the book info, in descending order |
| Story 2 | As a logged-in user I want to be able to delete a book | * In the list of most-recent and search results, display a “Delete” button next to each book * Before a book is deleted, the system should display a prompt to the user: “Are you sure you want to delete this book?” * After deleting the respective book entry the user is redirected to the home page * The button should be visible to logged-in users only |
| Story 3 | As a logged-in user I want to be able to edit an existing book | * In the list of most-recent and search results, display an “Edit” button next to each book * When clicked, the user is redirected to a page where he/she can edit the respective book’s information * The edit page contains two button – Save and Cancel * Proper validation of the entered data should be done by the system and any errors reported to the user * After the book is updated, the user is redirected to the home page * The “Edit” button should be visible to logged-in users only |

User Story - Status Code - Contributor

* **Story 1 – done – Aleks Mishev and Elian Konedarev**
* **Story 2 – done – Alex Mishev**
* **Story 3 – done – Elian Konedarev**

\*Disclaimer: It might seem that the work was not distributed evenly. Even though, the unpleasant circumstances we managed to get on a call and work together through all of the tasks.

# Individual contributions

**User Story:** As a logged-in user, I want to be able to search all books - Aleks Mishev and Elian Konedarev

**Acceptance Tests:**

* Logged-in users should see a “Search” menu link, which redirects to the search page;
* The search screen allows the user to enter a string and show all books that contain this string in their title, description and/or author;
* The results are shown in a list/table of books, each row showing the book information;
* Results should be sorted by the number of occurrences of the search string in the book info, in descending order.

**Alex:** To make sure we cover this user’s story, we had to do two things. First we made a new method in the BooksController. But since it’s not provided as the default GET method we used the custom routing, and the custom routing decorators - [Route("api/search/{term}")]. Using LINQ search queries, we managed to get all the records that contained the search term in either or both the title, description or author fields.

public IQueryable<Book> Search(string term)

{

var searchResult = db.Books.Where(o => o.Title.Contains(term) ||

o.Author.Contains(term) ||

o.Description.Contains(term));

return searchResult;

}

**Elian**: Second, we had to make a GET request to the API with the current search term. Using an observable, we tracked the changes in the input field and made a request to the API.

  search(terms: Observable<string>) {

    return terms.pipe(debounceTime(800),

    distinctUntilChanged(),

    switchMap(term => this.searchEntries(term)))

  }

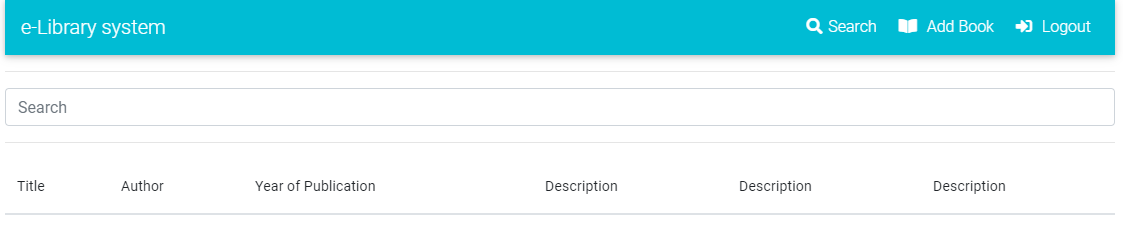
  searchEntries(searchTerm){

    return this.http.get(this.rootURL + '/search/' + searchTerm)

  }

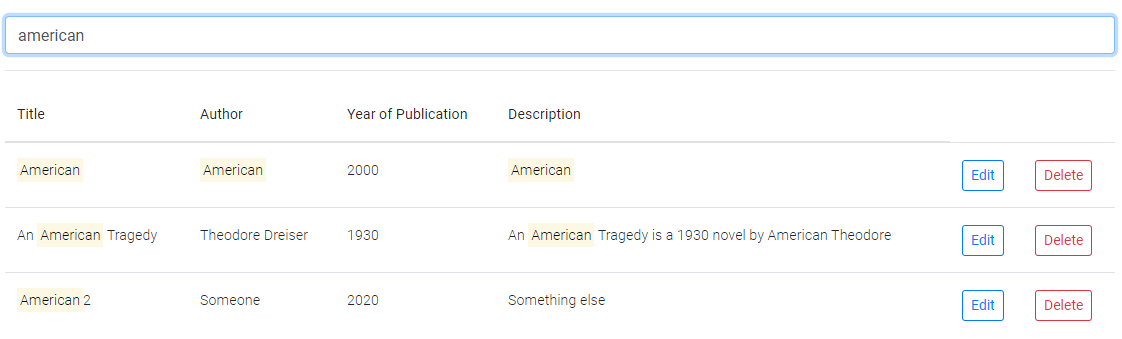
**Acceptance Test:** Logged-in users should see a “Search” menu link, which redirects to the search page – Elian Konedarev.

**Acceptance Test:** The search screen allows the user to enter a string and show all books that contain this string in their title, description and/or author – Elian Konedarev.



**Acceptance Test:** The results are shown in a list/table of books, each row showing the book information – Alex Mishev.

**Acceptance Test:** Results should be sorted by the number of occurrences of the search string in the book info, in descending order – Alex Mishev.



**User Story:** As a logged-in user I want to be able to delete a book – Alex Mishev

**Acceptance Tests:**

* In the list of most-recent and search results, display a “Delete” button next to each book;
* Before a book is deleted, the system should display a prompt to the user: “Are you sure you want to delete this book?”
* After deleting the respective book entry the user is redirected to the home page;
* The button should be visible to logged-in users only.

Now that the user is able to search for books and add them to the DB, the user is also able to delete a desired book from the DB. To achieve that had to make a new method in the BooksController to handle a Delete request.

public IHttpActionResult DeleteBook(int id)

{

Book book = db.Books.Find(id);

if (book == null)

{

return NotFound();

}

db.Books.Remove(book);

db.SaveChanges();

return Ok(book);

}

Then handle the on-click event of the user when he clicks the delete button.

  deleteBook(id: number) {

    return this.http.delete(this.rootURL + '/Books/' + id);

  }

The above method is part of the book service, which handles all the http actions. It’s called by the book component that has the delete button.

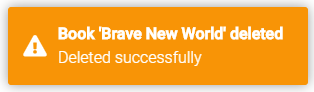
**Acceptance Test:** In the list of most-recent and search results, display a “Delete” button next to each book.

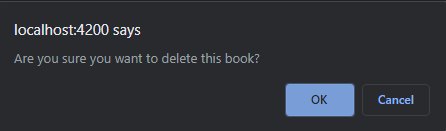
|  |  |
| --- | --- |
| <td \*ngIf="isLoggedIn"><button (click)="onDelete(book.BookID, book.Title)" | |
| class="btn btn-sm btn-outline-danger">Delete</button></td> |

**Acceptance Test:** Before a book is deleted, the system should display a prompt to the user: “Are you sure you want to delete this book?”

**Acceptance Test:** After deleting the respective book entry the user is redirected to the home page.

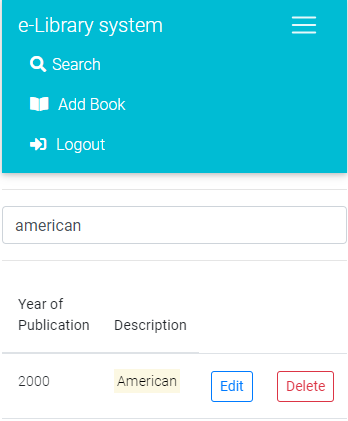
|  |
| --- |
| onDelete(id: number, title: string) { |
| if (confirm('Are you sure you want to delete this book?')) { | |
| this.service.deleteBook(id).subscribe(res => { | |
| this.service.refreshList(); | |
| this.router.navigate(['/welcome']); | |
| this.toastr.warning('Deleted successfully', `Book '${title}' deleted`); | |
| }); | |
| } | |
| } | |

We added ${title} in the message to look more pleasant.



**Acceptance Test:** The button should be visible to logged-in users only.

|  |
| --- |
| constructor(public service: BookService, private toastr: ToastrService, private router: Router, private userService: UserService) { } |
| isLoggedIn: boolean | |

This allows only the logged-in user to see the delete button

**User Story:** As a logged-in user I want to be able to edit an existing book – Elian Konedarev

**Acceptance Tests:**

* In the list of most-recent and search results, display an “Edit” button next to each book;
* When clicked, the user is redirected to a page where he/she can edit the respective book’s information;
* The edit page contains two button – Save and Cancel;
* Proper validation of the entered data should be done by the system and any errors reported to the user;
* After the book is updated, the user is redirected to the home page;
* The “Edit” button should be visible to logged-in users only.

For this section we needed to add a new Book component.

import { EditBookComponent } from './books/edit-book/edit-book.component';

And also into the @NgModule({

EditBookComponent,

The edit book component will get the data from the book service:

  formData: Book;

The formData is set when the user click the edit button on a specific book.

  onEdit(book: Book){

      this.service.formData = Object.assign({}, book);

      this.router.navigate(['/edit-book']);

  }

Hence we have the data ready for submission.

Now we only need a PUT request handler in the API.

[ResponseType(typeof(void))]

public IHttpActionResult PutBook(int id, Book book)

{

if (id != book.BookID)

{

return BadRequest();

}

db.Entry(book).State = EntityState.Modified;

try

{

db.SaveChanges();

}

catch (DbUpdateConcurrencyException)

{

if (!BookExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return StatusCode(HttpStatusCode.NoContent); }

And there we have it.

  putBook(formData: Book) {

    return this.http.put(this.rootURL + '/Books/' + formData.BookID, formData);

  }

**Acceptance Test:** In the list of most-recent and search results, display an “Edit” button next to each book.

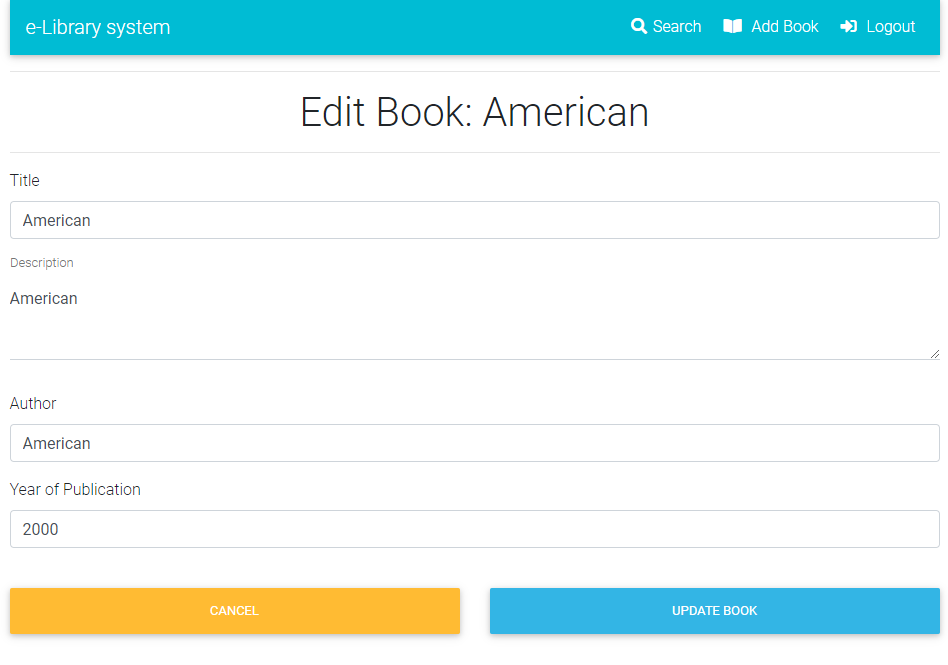
|  |
| --- |
| <td \*ngIf="isLoggedIn"><button (click)="onEdit(book)" |
| class="btn btn-sm btn-outline-primary">Edit</button></td> | |

**Acceptance Test:** When clicked, the user is redirected to a page where he/she can edit the respective book’s information.

|  |
| --- |
| onEdit(book: Book){ |
| this.service.formData = Object.assign({}, book); | |
| this.router.navigate(['/edit-book']); | |
| } | |

**Acceptance Test:** The edit page contains two button – Update and Cancel;

|  |
| --- |
| <div class="row"> |
| <div class="col-6"> | |
| <button mdbBtn color="warning" block="true" class="my-4" (click)="Cancel()">Cancel</button> | |
| </div> | |
| <div class="col-6"> | |
| <button [disabled]="!form.valid" mdbBtn color="info" block="true" class="my-4" type="submit" | |
| (click)="updateRecord(form)">Update Book</button> | |
| </div> | |
| </div> | |



**Acceptance Test:** Proper validation of the entered data should be done by the system and any errors reported to the user;

That’s the HTML5 Validation:

|  |  |
| --- | --- |
| <div class="row" \*ngIf="service.formData"> | |
| <div class="col-md-12"> |
| <form #form="ngForm" autocomplete="off"> |
| <input type="hidden" name="BookID" #BookID="ngModel" [(ngModel)]="service.formData.BookID"> |
| <div class="form-group"> |
| <label>Title</label> |
| <input name="Title" maxlength="60" #Title="ngModel" [(ngModel)]="service.formData.Title" |
| class="form-control" required> |
| <div class="validation-error" \*ngIf="Title.invalid && Title.touched">This field is required. (60 characters |
| max)</div> |
| </div> |
| <div class="form-group"> |
| <div class="md-form"> |
| <textarea name="Description" maxlength="60" type="text" id="form7" class="md-textarea form-control" |
| #Description="ngModel" [(ngModel)]="service.formData.Description" mdbInput required></textarea> |
| <label for="form7">Description</label> |
| </div> |
| <div class="validation-error" \*ngIf="Description.invalid && Description.touched">This field is required. (60 |
| characters max)</div> |
| </div> |
| <div class="form-group"> |
| <label>Author</label> |
| <input name="Author" maxlength="60" #Author="ngModel" [(ngModel)]="service.formData.Author" |
| class="form-control" required> |
| <div class="validation-error" \*ngIf="Author.invalid && Author.touched">This field is required. (60 |
| characters max)</div> |
| </div> |
| <div class="form-group"> |
| <label>Year of Publication</label> |
| <input name="Publication" #Publication="ngModel" maxlength="4" [(ngModel)]="service.formData.Publication" |
| class="form-control" required> |
| <div class="validation-error" \*ngIf="Publication.invalid && Publication.touched">This field is required. (4 |
| characters max)</div> |
| </div> |

As we are using the same validation from the previous iterations. We’ll not provide the same screenshots again.

**Acceptance Test:** After the book is updated, the user is redirected to the home page;

|  |
| --- |
| updateRecord(form: NgForm) { |
| this.service.putEmployee(form.value).subscribe(res => { | |
| this.toastr.info('Updated successfully', `Book '${this.service.formData.Title}' Updated`); | |
| this.resetForm(form); | |
| this.router.navigate(['/welcome']); | |
| this.service.refreshList(); | |
| }); | |

**Acceptance Test:** The “Edit” button should be visible to logged-in users only.

|  |
| --- |
| constructor(public service: BookService, private toastr: ToastrService, private router: Router, private userService: UserService) { } |
| isLoggedIn: boolean | |

<td \*ngIf="isLoggedIn"><button (click)="onEdit(book)"

The two buttons “Delete” and “Edit” are visible only to the logged-in User.

