Aleks Mishev & Elian Konedarev



Sprint 2 Report

Prof. Vladimir Georgiev INF440

# User Stories

Spring #2 User Stories

|  |  |  |
| --- | --- | --- |
| Story 1 | As a user, I want to see all pages of the web app being responsive and visualize well on PC and mobile devices | * The Bootstrap framework should be integrated and used for rendering all pages of the web application * When viewed on a small screen (e.g. cell phone or tablet), page elements should be properly aligned, horizontal scroll should NOT appear * Existing menu links should be moved in the top navigation menu * The top navigation menu should be shown on all pages * Login, Logout, Register and other links should be visible accordingly to the current visitor’s logged-in status |
| Story 2 | As a logged-in user, I want to be able to add books to the e-Library application | * Logged-in users should see an extra item in the main menu – “Add Book” which redirects them to the book entry page * A book should have a title, description, author, and publication year – all required * Book entry page should validate all of the above properties, and save the book in the database * All errors (validation, application, etc.) should be properly reported to the user * After a successful entry, the user is redirected to the home screen |
| Story 3 | As a user, I want to see a list of the 5 most-recently added books | * The home screen should be updated to contain a list of 5 most-recently inserted books, in addition to the existing welcome message * Both types of users should see the list, ordered by insertion time in descending order (newest one at the top) |

User Story - Status Code - Contributor

* **Story 1 – done – Aleks Mishev**
* **Story 2 – done – Aleks Mishev & Elian Konedarev**
* **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 user, I want to see all pages of the web app being responsive and visualize well on PC and mobile devices – Aleks Mishev

**Acceptance Tests:**

* The Bootstrap framework should be integrated and used for rendering all pages of the web application
* When viewed on a small screen (e.g. cell phone or tablet), page elements should be properly aligned, horizontal scroll should NOT appear
* Existing menu links should be moved in the top navigation menu
* The top navigation menu should be shown on all pages
* Login, Logout, Register and other links should be visible accordingly to the current visitor’s logged-in status

In order to achieve the acceptance tests for the user story we had to take advantage of Bootstrap in Angular. Luckily, there is a UI KIT – Material Design for Bootstrap (Angular version) that allows us to build responsive, mobile-first websites and applications free of charge.

Thanks to MDB we took advantage of all features of the latest version of Bootstrap. MDB is updated & integrated every two weeks.

**Acceptance Test:** The Bootstrap framework should be integrated and used for rendering all pages of the web application

We integrated the UI KIT in our application using **npm.**

**Installing the @angular/cdk package**

**npm install @angular/cdk –save**

**Installing MDB Angular:**

**npm install angular-bootstrap-md --save**

**Updating the app.module.ts file with the following code:**

import { NgModule } from '@angular/core';

import { MDBBootstrapModule } from 'angular-bootstrap-md';

@NgModule({

  imports: [

    MDBBootstrapModule.forRoot()

  ]

})

* **Updating the styles and scripts arrays in angular.json file:**

"styles": [

              "node\_modules/ngx-toastr/toastr.css",

              "node\_modules/@fortawesome/fontawesome-free/scss/fontawesome.scss",

              "node\_modules/@fortawesome/fontawesome-free/scss/solid.scss",

              "node\_modules/@fortawesome/fontawesome-free/scss/regular.scss",

              "node\_modules/@fortawesome/fontawesome-free/scss/brands.scss",

              "node\_modules/angular-bootstrap-md/assets/scss/bootstrap/bootstrap.scss",

              "node\_modules/angular-bootstrap-md/assets/scss/mdb.scss",

              "node\_modules/animate.css/animate.css",

              "src/styles.scss"

            ],

            "scripts": [

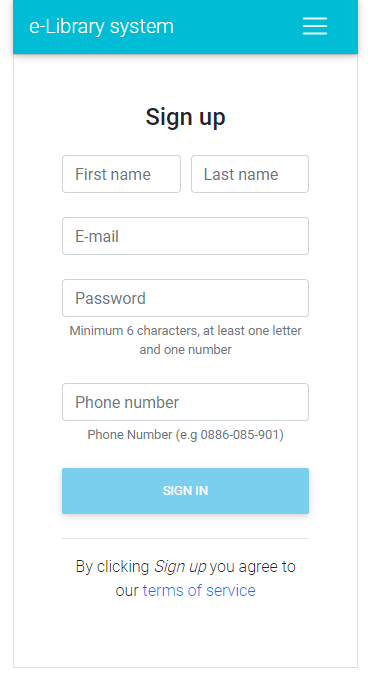
              "node\_modules/chart.js/dist/Chart.js",

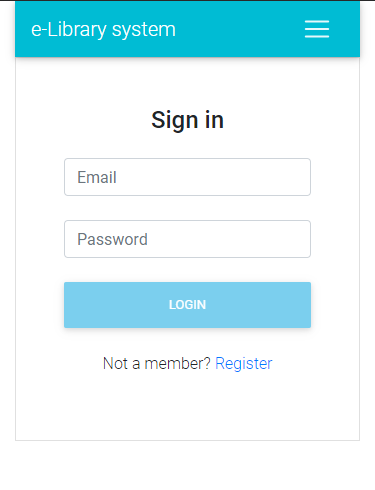
              "node\_modules/hammerjs/hammer.min.js"

            ]

* **Installing external libraries for the proper function of the KIT**

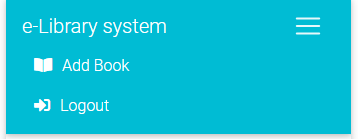
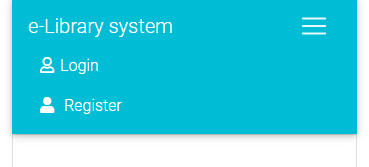
**npm install -–save chart.js@2.5.0 @types/chart.js @fortawesome/fontawesome-free hammerjs animate.css**

**Acceptance Test:** When viewed on a small screen (e.g. cell phone or tablet), page elements should be properly aligned, horizontal scroll should NOT appear

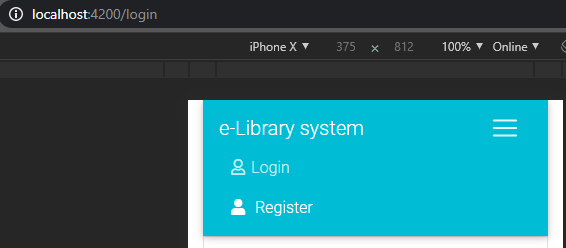


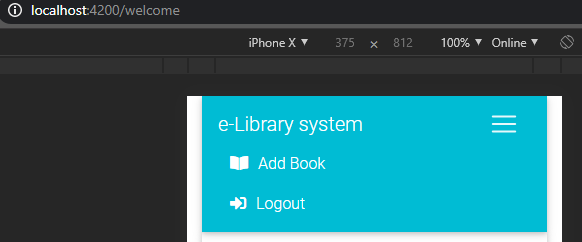
**Acceptance Test:** Existing menu links should be moved in the top navigation menu

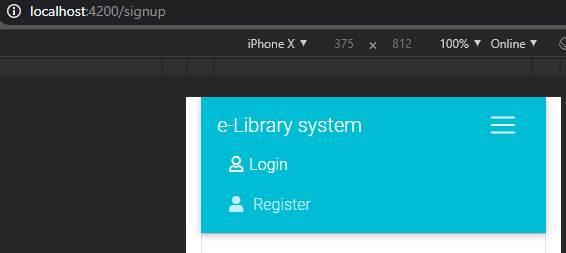
When user is not logged-in When user is logged-in



**Acceptance Test:** The top navigation menu should be shown on all pages & Login, Logout, Register and other links should be visible accordingly to the current visitor’s logged-in status

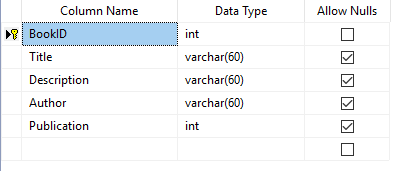






**User Story:** As a logged-in user, I want to be able to add books to the e-Library application – Aleks Mishev & Elian Konedarev

In order to add books in the e-Library we made a new table in our newly created BooksDB, again using the SQL Server Manager. – Aleks Mishev



Then using the Entity Model Wizard, we created our Model.

public partial class Book

{

public int BookID { get; set; }

public string Title { get; set; }

public string Description { get; set; }

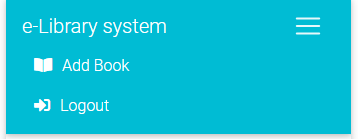
public string Author { get; set; }

public Nullable<int> Publication { get; set; }

}

Then we created the books controller for API requests.

**Acceptance Test:** Logged-in users should see an extra item in the main menu – “Add Book” which redirects them to the book entry page – Elian Konedarev



**Acceptance Test:** A book should have a title, description, author, and publication year – all required – Aleks Mishev

As you can see we made a small mistake when declaring the Book model and not all fields are required. However, we fixed that problem on the client-side, not allowing the user to submit a post request if fields are empty or the input is invalid. See acceptance tests below.

**Acceptance Test:** Book entry page should validate all of the above properties, and save the book in the database – Aleks Mishev

We use the disabled directive to make sure that the from is valid, it’s only then when the user is allowed to press the submit button. When the submit button is clicked we call the insertRecord method, which respectively calls the book service that makes the post request.

How the post Service look.

  readonly rootURL ="http://localhost:51377/api"

  constructor(private http : HttpClient) { }

  postBook(formData : Book){

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

   }

Further we made an interface book, that looks like this:

export class Book {

  BookID : number;

  Title : string;

  Description : string;

  Author : string;

  Publication : number

}

The post ActionResult in the BooksController adds the record to the database and saves it.

[ResponseType(typeof(Book))]

public IHttpActionResult PostBook(Book book)

{

db.Books.Add(book);

db.SaveChanges();

return CreatedAtRoute("DefaultApi", new { id = book.BookID }, book);

}

**Acceptance Test:** All errors (validation, application, etc.) should be properly reported to the user

After a successful entry, the user is redirected to the home screen – Elian Konedarev

  insertRecord(form: NgForm) {

      this.service.postBook(form.value).subscribe(res => {

        this.toastr.success('Inserted successfully', 'Book Added');

…

        this.router.navigate(['/welcome']);

      });

  }

That’s the HTML5 validation implementation:

  <div class="form-group">

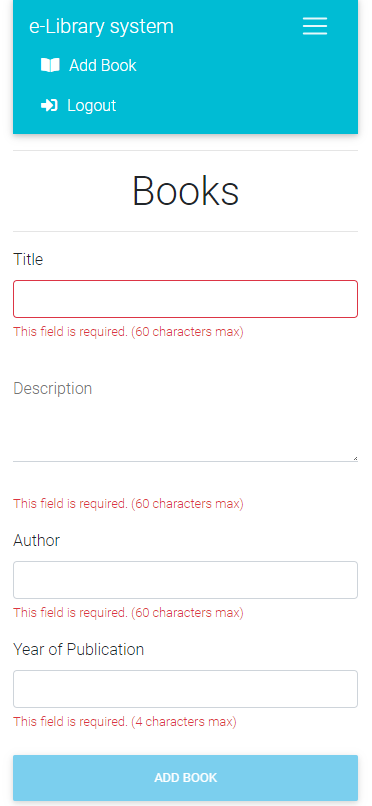
    <label>Year of Publication</label>

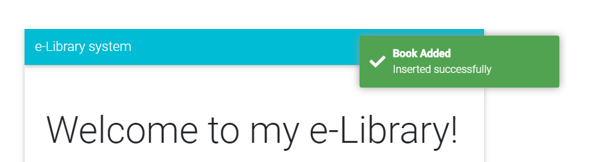
    <input name="Publication" #Publication="ngModel" maxlength="4" [(ngModel)]="service.formData.Publication"

      class="form-control" required>

    <div class="validation-error" \*ngIf="Publication.invalid && Publication.touched">This field is required. (4 characters max)</div>

  </div>





**User Story:** As a user, I want to see a list of the 5 most-recently added books – Elian Konedarev

Now that we have the user story #2 in implemented. We only had to do an ActionResult for the GET request, and of call it in the Angular application.

**Acceptance Test:** The home screen should be updated to contain a list of 5 most-recently inserted books, in addition to the existing welcome message

[HttpGet]

[AllowAnonymous]

public IQueryable<Book> GetBooks()

{

var lastFiveBooks = db.Books.OrderByDescending(b => b.BookID).Take(5);

return lastFiveBooks;

}

We call the API when the user submits the new book record, because after that he is redirected to the welcome screen, hence the list of books should be updated. We resolve the promise and save it in an array of type Book.

   refreshList(){

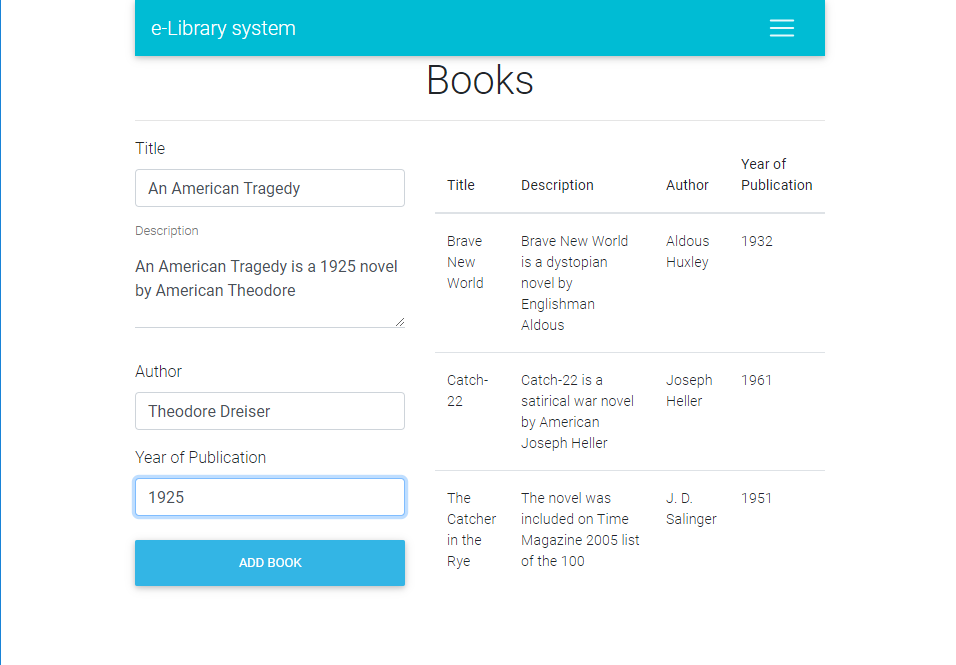
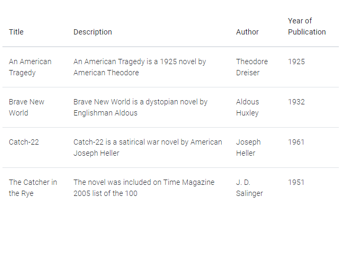
    this.http.get(this.rootURL+'/Books')

    .toPromise().then(res => this.list = res as Book[]);

  }

Since we needed only the 5 most recent record ordered in descending order, we used the LINQ OrderByDescending and Take methods to accomplish that requirement. See previous acceptance test. Screenshots on next page.

Before inserting the new Book: After:



**Acceptance Test:** Both types of users should see the list, ordered by insertion time in descending order (newest one at the top)

This was a problem at first, since in the first sprint we added an http interceptor in order to prevent unauthorized requests. However, we managed to overcome that issue by adding the [AllowAnonymous] attribute to the GET request for the books. Screenshots on next page.

