Test plan

Test objectives for this iteration

The objective for this iteration is to verify and validate that the implemented functionality is working as expected as well as trying to identify as many faults as possible in the system. The idea is to build up confidence that the project so far is heading at the right direction. But since this is still development testing, the main objective is to find as many defects as possible.

Test objects

To make sure that the correct content is rendered and presented as it should we need to test the client. Since not all features are implemented yet this iteration will focus on use cases 1 – 3. List books, view book information and delete book.

The delete book use case is dependent on a predicate function (**`getAllBooksBut`**) to filter the list of books. This needs to be tested to verify that it will return the right result.

The next step is to implement the functionality to add and edit books. The book data needs to be validated before being added to the system. One function (**`validateTitle`**) has already been implemented and needs to be tested. Another (**`validatePublishDateFormat`**) has not been implemented yet but to enable a test first approach there is a need to create a test for this function as well.

The API need to be tested to make sure that it fulfils its purpose and to give a clear picture of what's been implemented.

Testing techniques

The design of all the tests will have equivalence partition in mind to reduce the number of tests but still having as large coverage as possible.

Testing the implemented features and the client will be done using manual tests since this not only has to do with functionality but also has to do with navigating on the website and to make sure that everything is presented as it should.

Automatic unit tests will be used to test the functions to reduce the time spent on testing as well as allow for automatic regression testing. This will reduce the risk to make changes that will result in

faults in the system. The tests of the implemented functions will have a decision coverage to verify the function.

The API will be tested using automatic API tests for the same reasons as the unit tests.

Using reviews on the code to find defects and to make sure that the code is following the code standard.

Test matrix for manual test cases

	Req. 1. UC 1.1	Req. 1. UC 1.2	Req. 1. UC 2.1	Req. 1. UC 3.1	Req. 1. UC 4.1	Req. 1. UC 4.2	Req. 1. UC 5.1	Req. 1. UC 5.2	Req. 1. UC 6.1	Req. 1. UC 6.2
reqs. Tested	1	1	2	3	0	0	0	0	0	0
1.1.1	x									
1.2.1		x								
2.1.1			x							
2.1.2			x							
3.1.1				x						
3.1.2				x						
3.1.3				x						

Manual test cases

1.1.1. List existing books

To be able to show the available books in the system all of the books that are in books.xml should be presented in a list. The title and author of each books should be shown in the list.

Pre-conditions

The content in **`books.xml`** must be replaced with the content in **`test/data/111books.xml`**.
The server must be running.

Steps

1. Go to http://localhost:9090/.
2. Click on the **`Books`** button in the menu.

Expected output

	Pass	Fail
4 books are listed	**x**	
Each book has a **`Title`** and **`Author`**	**x**	
All book information corresponds to the items in **`books.xml`**	**x**	
There is a **`New book`** button	**x**	

Comments

The information for all 4 books matches the one in the XML and there is a button for adding new books.

1.2.1. List books without any saved books

This is a test to make sure that an error doesn't occur and that the right information is shown in case the system has no books.

Pre-conditions

The content in **`books.xml`** must be replaced with the content in **test/data/121books.xml**.
The server must be running.

Steps

1. Go to http://localhost:9090/.
2. Click on **`Books`** in the menu.

Expected output

	Pass	Fail
No books are listed	**x**	
There is a **`New book`** button	**x**	
There is a text saying **`No books available.`**		**x**

Comments

As expected there were no books listed. But instead of having the information message, the list headline with **`Title`** and **`Author`** is visible. It seems as if the client is trying to render the empty book list instead of presenting the message.

2.1.1. View book by clicking on the title

The user should be able to view the book by clicking on the title. This will present all the information about the book as well as the possibility to alter the book.

Pre-conditions

The test **1.1.1. List existing books** must have been performed.

Steps

1. Click on the title of the first element in the list **A Brief History of Time**.

Expected output

	Pass	Fail
Title is **A Brief History of Time**	**X**	
Author is **Stephen Hawking**	**X**	
Genre is **Science**	**X**	
Price is **199**	**X**	
Publish date is **1988-09-01**	**X**	
Description is **Hawking attempts to explain** […] **the beginning of time.**	**X**	
There is a **Back** button	**X**	
There is a **Save** button	**X**	
There is a **Delete** button	**X**	

Comments

All information matches and everything is working as expected.

2.1.2 View book by clicking on the author

The user should be able to view the book by clicking on the Author. This will present all the information about the book as well as the possibility to alter the book.

Pre-conditions

The test **1.1.1. List existing books** must have been performed.

Steps

1. Click on the author of the second element in the list **`Astrid Lindgren`**.

Expected output

	Pass	Fail
`Title` is **`Ronia the Robbers Daughter`**	X	
`Author` is **`Astrid Lindgren`**	X	
`Genre` is **`Fantasy`**	X	
`Price` is **`79`**	X	
`Publish date` is **`1981-04-23`**	X	
`Description` is **`Ronia is a girl growing`** […] **`the two bands.`**	X	
There is a **`Back`** button	X	
There is a **`Save`** button	X	
There is a **`Delete`** button	X	

Comments

Everything passed and the rendered information is the same as the expected result.

3.1.1. Delete book

The user has to be able to remove books from the system. This test will make sure that it's possible.

Pre-conditions

The test **`2.1.1. View book by clicking on the title`** must have been performed.

Steps

1. Click on the **Delete** button

Expected output

	Pass	Fail
Got redirected to **`http://localhost:9090/#/books`**	X	
There are 3 books listed	X	
`A Brief History of Time` is not in the list	X	
There is a **`New book`** button	X	

Comments

Everything went as expected. The Brief History of Time is not in the list and the remaining books are listed after the page redirected.

3.1.2. Delete a second book

The user has to be able to remove books from the system. This test will make sure that it's possible.

Pre-conditions

The test **`3.1.1. Delete book`** must have been performed.

Steps

1. Click on the author of the first element in the list **`Astrid Lindgren`**.
2. Click on the **`Delete`** button.

Expected output

	Pass	Fail
Got redirected to **`http://localhost:9090/#/books`**	**X**	
There are 2 books listed	**X**	
`A Brief History of Time` is not in the list	**X**	
`Ronia the Robbers Daughter` is not in the list	**X**	
There is a **`New book`** button	**X**	

Comments

Both of the books are removed and the remaining books are unaltered.

3.1.3. Delete the last book

In case the user removes the last book in the system we have to make sure nothing unexpected happens.

Pre-conditions

The content in **`books.xml`** must be replaced with the content in **test/data/313books.xml**.
The server must be running.

Steps

1. Click on the author of the first element in the list **`Astrid Lindgren`**.

2. Click on the **`Delete`** button.

Expected output

	Pass	Fail
Got redirected to **`http://localhost:9090/#/books`**	**X**	
No books are listed	**X**	
There is a text saying **`No books available.`**		**X**
There is a **`New book`** button	**X**	

Comments

I got redirected and no books were available but instead of having a text saying so the list headline saying “Title” and “Author”.

Unit tests

Remove book resource

`getAllBooksBut` is a function in **`RemoveBookResource`**. It returns a predicate function to filter out the book with the provided ID.

```
var expect             = require("chai").expect;
var RemoveBookResource = require("../app/resources/RemoveBookResource");

describe("RemoveBookResource", function () {

    describe("getAllBooksBut", function () {

        it("The return function returns true if ID:s are different", function () {
            var id = 1;
            var bookObj = {$:{id:2}}

            var result = RemoveBookResource.getAllBooksBut(id)(bookObj);

            expect(result).to.equal(true);
        });

        it("The return function returns false if ID:s are identical", function () {
            var id = 1;
            var bookObj = {$:{id: id}}

            var result = RemoveBookResource.getAllBooksBut(id)(bookObj);

            expect(result).to.equal(false);
        });

    });
});
```

Validation helpers

The public functions from **`ValidationHelpers`**. **`validatePublishDateFormat`** has not been implemented yet.

```
var expect          = require("chai").expect;
var validationHelpers = require("../app/helpers/validationHelpers");

describe("validationHelpers", function () {

    describe("validateTitle", function () {

        var testCases = [
            { case: "The title is too short", title: '', expection: false },
            { case: "The title is too long", title: 'This title has more than forty characters.', expection: false },
            { case: "The title is has the right length", title: 'Happy path title', expection: true },
            { case: "The title is null", title: null, expection: false },
        ]

        testCases.forEach(function(test) {
            it(test.case, function () {
                var result = validationHelpers.validateTitle(test.title)
                expect(result).to.equal(test.expection);
            });
        });
    });

    describe("validatePublishDateFormat", function () {

        it('Correct formating', function () {
            var result = validationHelpers.validatePublishDateFormat('1865-11-26')
            expect(result).to.equal(true);
        });
    });
});

```

Unit testing results

```
default: > lnu-book-library@1.0.0 test /vagrant
default: > mocha --reporter spec
default:   RemoveBookResource
default:     getAllBooksBut
default:       ✓ The return function returns true if ID:s are different
default:       ✓ The return function returns false if ID:s are identical
default:   validationHelpers
default:     validateTitle
default:       ✓ The title is too short
default:       ✓ The title is too long
default:       ✓ The title is has the right length
default:       ✓ The title is null
default:     validatePublishDateFormat
default:       1) Correct formating
default:   6 passing (22ms)
default:   1 failing
default:   1) validationHelpers
default:        validatePublishDateFormat
default:          Correct formating:
default:
default:       AssertionError: expected false to equal true
default:       + expected - actual
default:
default:       -false
default:       +true
default:
default:       at Context.<anonymous> (test/validationHelpers.unit.spec.js:27:31)
```

API tests

Remove book resource

Testing to remove a book twice as well as removing the last book in the list.

```
var request = require('supertest');
var libraryDAO = require('./../app/dao/LibraryDAO');
var app = require("../app");

var testData = [
    {
        $: {id: '1'},
        author: 'Isaac Asimov',
        title: 'Foundation',
        genre: 'Science Ficition',
        price: 164,
        publish_date: '1951-08-21',
        description: 'Foundation is the first novel in Isaac Asimovs Foundation Trilogy (later expanded into The Foundation Se
    },
    {
        $: {id: '2'},
        author: 'Isaac Asimov',
        title: 'Foundation and Empire',
        genre: 'Science Ficition',
        price: 79,
        publish_date: '1952-10-12',
        description: 'Foundation and Empire is a novel written by Isaac Asimov that was published by Gnome Press in 1952. It i
    }
];

describe("Remove book Resource", function () {

    // Setup
    var orignalXML;
    libraryDAO.readXMLFile(function(xml){
        orignalXML = xml;
        libraryDAO.writeXMLFile(testData);
    });

    var testCases = [
        { case: "existing book is removed",    id: 1, expectedLength: 1 },
        { case: "remove the same book twice", id: 1, expectedLength: 1 },
        { case: "remove the last book",        id: 2, expectedLength: 0 }
    ];

    // Exercise
    testCases.forEach(function(test, i){
        it(test.case, function (done) {
            request(app)
                .delete('/api/books/' + test.id)
                .set('Accept', 'application/json')
                .expect(200)
                .expect(function(res){
                    libraryDAO.readXMLFile(function(xml){
                        var result = xml.find(function(book){
                            book.$.id === test.id;
                        });
                        if (xml.length !== test.expectedLength) {
                            throw new Error('The number of books are incorrect.');
                        }
                        if (result !== undefined) {
                            throw new Error('The book was NOT removed.');
                        }
                    });
                })
                .end(function(err, res){
                    // Teardown if it is the last test
                    if(i === testCases.length - 1) {
                        libraryDAO.writeXMLFile(orignalXML);
                    }
                    if (err) done(err);
                    else done();
                });
        });
    });
});
```

Get books resource

Test if the response has status code 200 and is of type JSON as well as making sure that the book is formatted as it's supposed and the data is strings.

```
var request = require('supertest');
var libraryDAO = require('./../app/dao/LibraryDAO');
var app = require("../app");

describe("Get books Resource - API", function () {

    // Exercise
    it('response with 200 and json', function (done) {
        request(app)
            .get('/api/books/')
            .set('Accept', 'application/json')
            .expect(200)
            .expect('Content-Type', /json/)
            .end(done);
    });

    it('Has correct format if not empty', function (done) {
        request(app)
            .get('/api/books/')
            .set('Accept', 'application/json')
            .expect(200)
            .expect(function(res) {
                if (res.body && res.body.length > 0) {
                    var book = res.body[0];
                    if (
                        !(
                            book.id && typeof book.id === 'string' &&
                            book.author && typeof book.author === 'string' &&
                            book.title && typeof book.title === 'string' &&
                            book.genre && typeof book.genre === 'string' &&
                            book.price && typeof book.price === 'string' &&
                            book.publishDate && typeof book.publishDate === 'string' &&
                            book.description && typeof book.description === 'string'
                        )
                    ) {
                        throw new Error('Missing key or data is not of type string');
                    }
                }
            })
            .end(done);
    });
});
```

API testing results

```
default:
default: > lnu-book-library@1.0.0 prestart /vagrant
default: > npm test
default:
default: > lnu-book-library@1.0.0 test /vagrant
default: > mocha --reporter spec
default:
default:   Get books Resource - API
default:     ✓ response with 200 and json (83ms)
default:     ✓ Has correct format if not empty
default:   Ping Resource
default:     GET /api/ping
default:       ✓ respond with pong
default:   Remove book Resource
default:     ✓ existing book is removed
default:     ✓ remove the same book twice
default:     ✓ remove the last book
default:
default:   6 passing (149ms)
```

Time log

Activity	Estimated time	Date	Effective time
Write test plan	180 min	2018-02-26	176 min
Write manual test cases	300 min	2018-02-27	283 min
Create unit tests	120 min	2018-03-01	73 min
Create automated API tests	300 min	2018-03-04	247 min
Write reflections	15 min	2018-03-05	28 min

Reflections

My plan, in the beginning, was to create more API tests to get a better coverage. The best solution would have been to use the API to control the test but since I haven’t implemented the functionality to add books yet this wasn't really possible. Instead, I had to override the XML as I do in the Remove Book Resource test but when the tests are in different modules it becomes hard to control the synchronized order. A solution to this is to have all the API test in one module but that can easily become quite messy. Therefore, I decided to wait with the rest of the test until "add books" is implemented.