Test Plan

Test objectives

The objectives for the tests in this stage of the development process are threefold. One is to prepare manual tests for the most important parts of the functionality, to make sure that not only the technical aspects of the system is on par, but that it is also usable from the client side. Two is to implement unit tests for both one of the implemented functions, to continuously check that I don't break what's working as I go forth, and for one of the next planned functions, being the functionality to get a specific book rather than a list of all. The third is to write API tests to check that these work, both ones already implemented and at least one of the ones to be implemented.

Static testing

Standard JS for code structure evaluation, check before each commit.

Manual test cases

Test 2.1

Test ID: MT5.1. Add new book, The GraveYard Book by Neil Gaiman. Test of use case ‘User adds new book’.

This manual test is meant to test the Add new book functionality of the library system, this being one of the primary use cases for the system and a prerequisite of most other uses of the system.

Prerequisites: Server up and running on port 9090.

Test steps:

1. User goes to library page
2. Click on Book
3. Click on New Book
4. Write in “The Graveyard Book” in Title field
5. Write “Neil Gaiman” in Author field
6. Click on Save

Expected result: Book should be added to database, and user should be returned to main library page.

☐ Test succeeded
☐ Test failed

Comments by tester:____

____

____

Test 3.1

Test ID MT3.1. Testing use case 'find a specific book', with The Graveyard Book by Neil Gaiman. This seems to me to be one of the most important, and most used, functions of a system like this - to check if a certain book is in there.

Prerequisites: Server up and running on port 9090; Test MT2.1.

Test steps:

1. User goes to library page
2. Click in search field
3. Type “the graveyard book” in field
4. Click submit

Expected result: System returns a page listing The Graveyard book by Neil Gaiman.

☐ Test succeeded
☐ Test failed

Comments by tester:____
____
____

Manual test cases: results

Manual test cases

Test 2.1

Test ID: MT5.1. Add new book, The GraveYard Book by Neil Gaiman. Test of use case ‘User adds new book’.

This manual test is meant to test the Add new book functionality of the library system, this being one of the primary use cases for the system and a prerequisite of most other uses of the system.

Prerequisites: Server up and running on port 9090.

Test steps:

1. User goes to library page
2. Click on Book
3. Click on New Book
4. Write in “The Graveyard Book” in Title field
5. Write “Neil Gaiman” in Author field
6. Click on Save

Expected result: Book should be added to database, and user should be returned to main library page.

☐ Test succeeded
☒ Test failed

Comments by tester: Steps 1-5 works smoothly, but test fails on last step, possible because book is never added to database.

Test 3.1

Test ID MT3.1. Testing use case 'find a specific book', with The Graveyard Book by Neil Gaiman. This seems to me to be one of the most important, and most used, functions of a system like this - to check if a certain book is in there.

Prerequisites: Server up and running on port 9090; Test MT2.1.

Test steps:

1. User goes to library page
2. Click in search field
3. Type “the graveyard book” in field
4. Click submit

Expected result: System returns a page listing The Graveyard book by Neil Gaiman.

☐ Test succeeded
☒ Test failed

Comments by tester: Test fails on step 4, when nothing happens on clicking submit.

Automated unit tests

Unit test #1: Test the xmlToJson function to make sure it is returning an object that has an ID. This because the ID is used for almost all other functions in the system, and thus very important to make sure it gets right.

Unit test #2: Test xmltoJson with empty input to make sure it doesn’t crash.

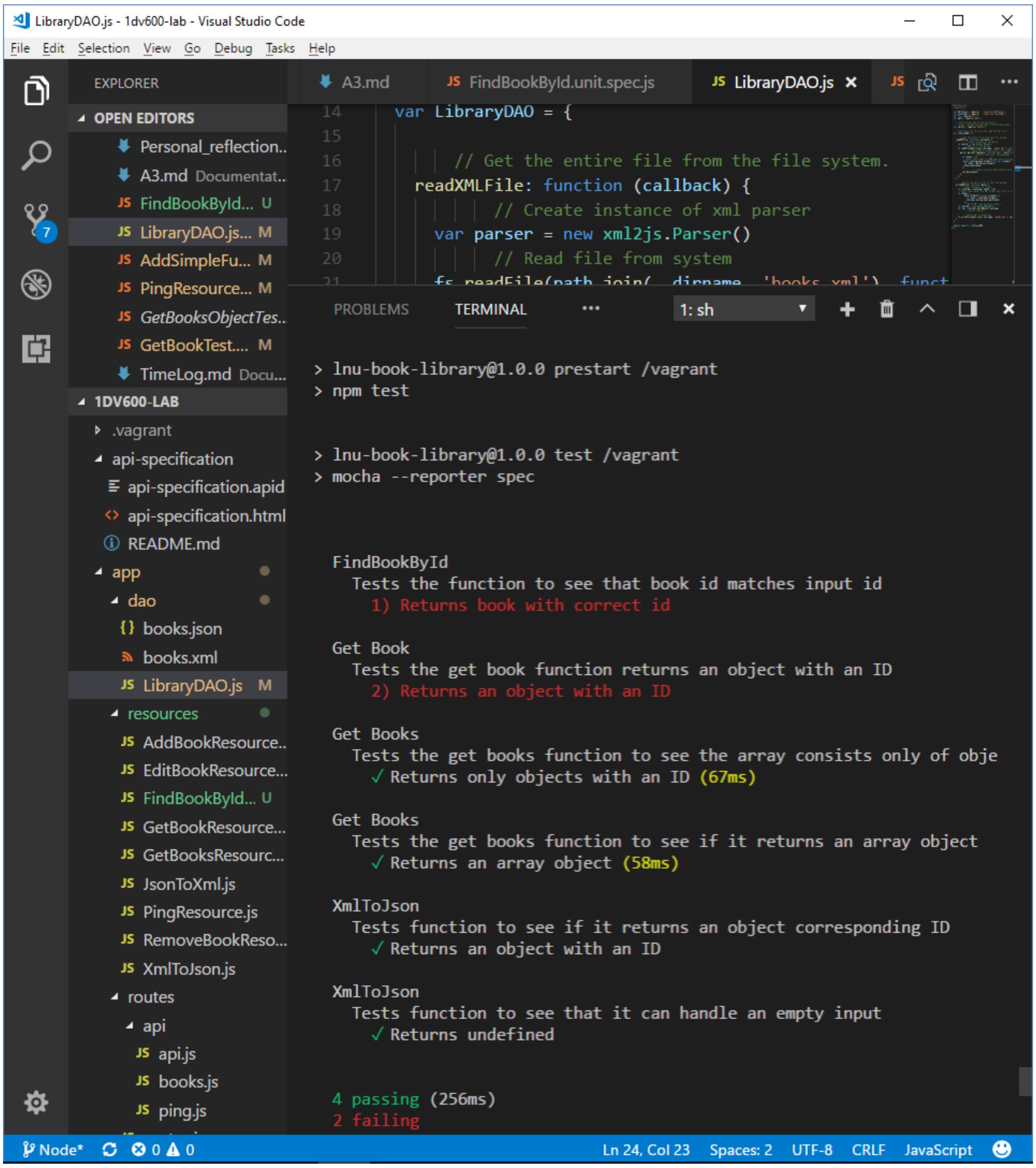
Unit test #3: Test FindBookById to see if it returns book with correct id. Unfinished method.

Automated API tests

API test #1: Get books method, being a more important method than the delete method of the ones already implemented as you generally want to find a book in the system before deleting it. First test: Check that list books returns json type.

API test #2: Get book, being the next method to implement. Test that Get Book returns a json object with correct id.

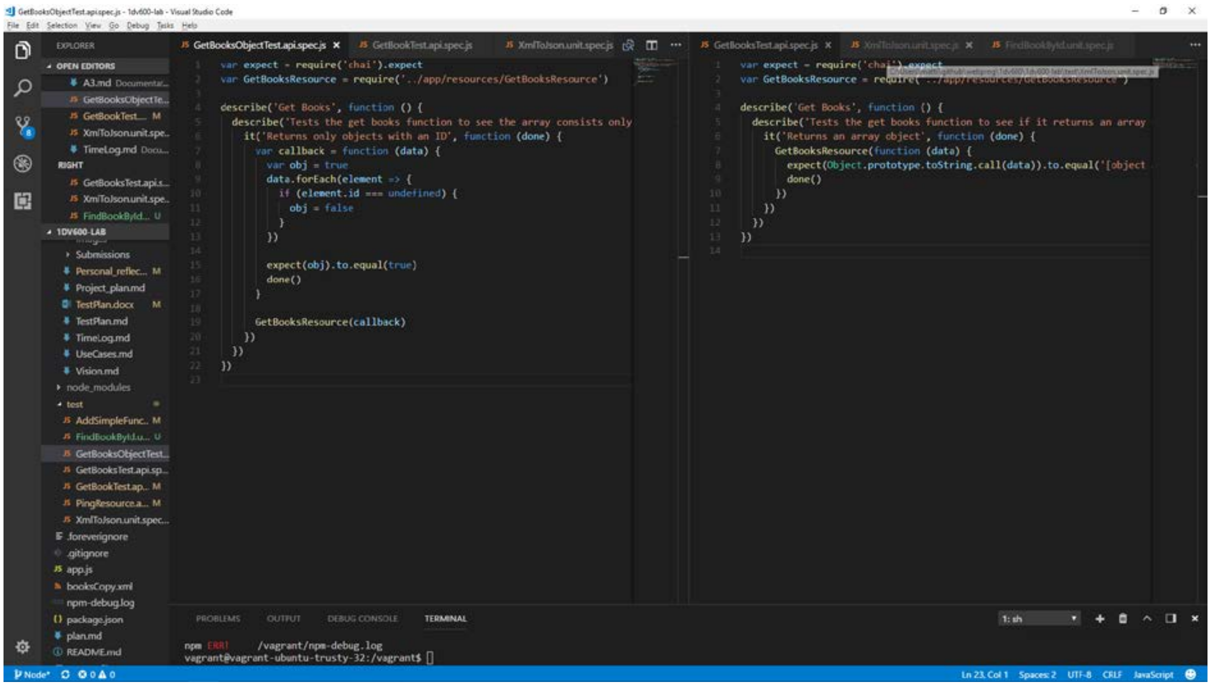
Screenshot showing unit tests and API tests.



```
var LibraryDAO = {

    // Get the entire file from the file system.
  readXMLFile: function (callback) {
      // Create instance of xml parser
    var parser = new xml2js.Parser()
      // Read file from system
```

```
> lnu-book-library@1.0.0 prestart /vagrant
> npm test


> lnu-book-library@1.0.0 test /vagrant
> mocha --reporter spec



  FindBookById
    Tests the function to see that book id matches input id
      1) Returns book with correct id

  Get Book
    Tests the get book function returns an object with an ID
      2) Returns an object with an ID

  Get Books
    Tests the get books function to see the array consists only of obje
      ✓ Returns only objects with an ID (67ms)

  Get Books
    Tests the get books function to see if it returns an array object
      ✓ Returns an array object (58ms)

  XmlToJson
    Tests function to see if it returns an object corresponding ID
      ✓ Returns an object with an ID

  XmlToJson
    Tests function to see that it can handle an empty input
      ✓ Returns undefined


  4 passing (256ms)
  2 failing
```

Test code screenshots



GetBooksObjectTest.api.spec.js - 1dv600-lab - Visual Studio Code

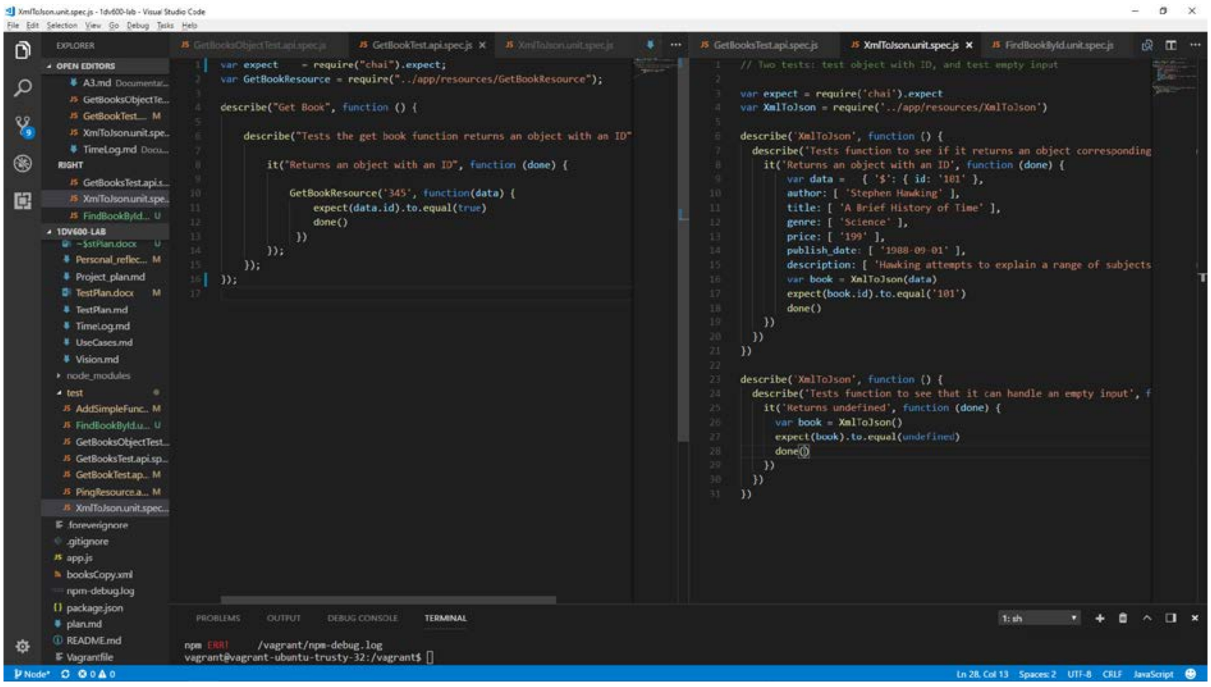
GetBooksObjectTest.api.spec.js

```javascript
var expect = require('chai').expect
var GetBooksResource = require('../app/resources/GetBooksResource')

describe('Get Books', function () {
  describe('Tests the get books function to see the array consists only
    it('Returns only objects with an ID', function (done) {
      var callback = function (data) {
        var obj = true
        data.forEach(element => {
          if (element.id === undefined) {
            obj = false
          }
        })

        expect(obj).to.equal(true)
        done()
      }

      GetBooksResource(callback)
    })
  })
})
```

GetBooksTest.api.spec.js

```javascript
var expect = require('chai').expect
var GetBooksResource = require('../app/resources/GetBooksResource')

describe('Get Books', function () {
  describe('Tests the get books function to see if it returns an array
    it('Returns an array object', function (done) {
      GetBooksResource(function (data) {
        expect(Object.prototype.toString.call(data)).to.equal('[object
        done()
      })
    })
  })
})
```

```
npm ERR!     /vagrant/npm-debug.log
vagrant@vagrant-ubuntu-trusty-32:/vagrant$
```



XmlToJson.unit.spec.js - 1dv600-lab - Visual Studio Code

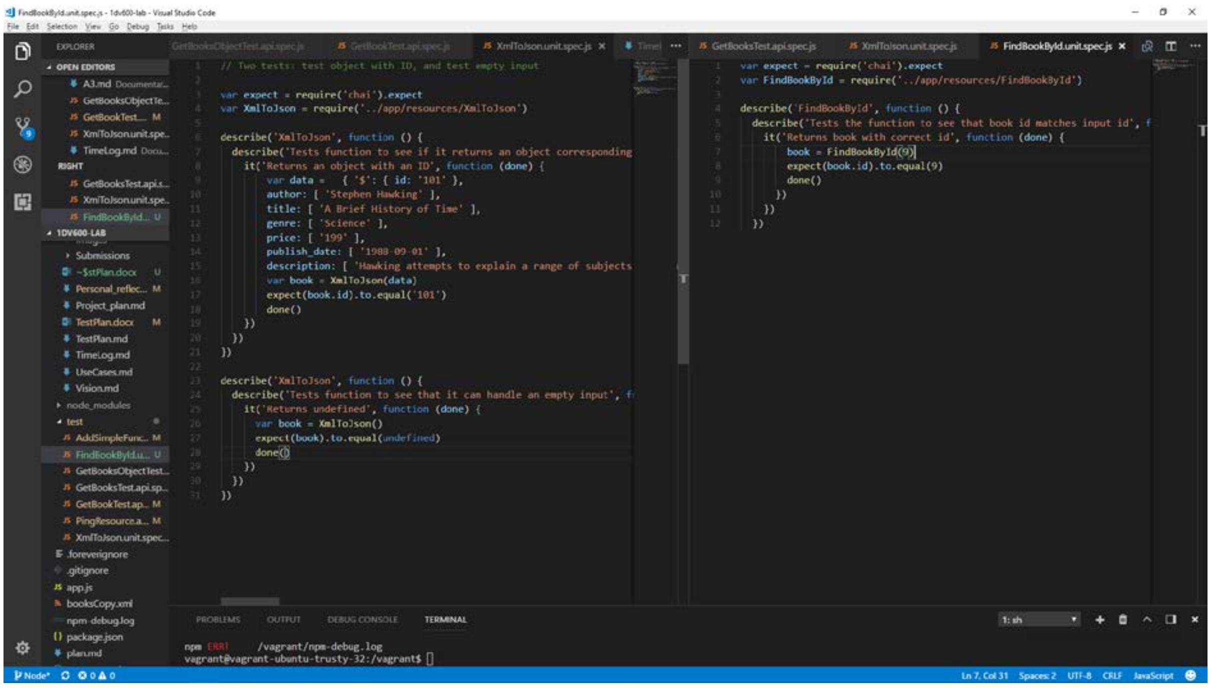
GetBookTest.api.spec.js

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

describe("Get Book", function () {

    describe("Tests the get book function returns an object with an ID"

        it("Returns an object with an ID", function (done) {

            GetBookResource('345', function(data) {
                expect(data.id).to.equal(true)
                done()
            })
        });
    });
});
```

XmlToJson.unit.spec.js

```javascript
// Two tests: test object with ID, and test empty input

var expect = require('chai').expect
var XmlToJson = require('../app/resources/XmlToJson')

describe('XmlToJson', function () {
  describe('Tests function to see if it returns an object corresponding
    it('Returns an object with an ID', function (done) {
        var data =   { '$': { id: '101' },
        author: [ 'Stephen Hawking' ],
        title: [ 'A Brief History of Time' ],
        genre: [ 'Science' ],
        price: [ '199' ],
        publish_date: [ '1988-09-01' ],
        description: [ 'Hawking attempts to explain a range of subjects
        var book = XmlToJson(data)
        expect(book.id).to.equal('101')
        done()
    })
  })
})

describe('XmlToJson', function () {
  describe('Tests function to see that it can handle an empty input', f
    it('Returns undefined', function (done) {
      var book = XmlToJson()
      expect(book).to.equal(undefined)
      done()
    })
  })
})
```

```
npm ERR!     /vagrant/npm-debug.log
vagrant@vagrant-ubuntu-trusty-32:/vagrant$
```

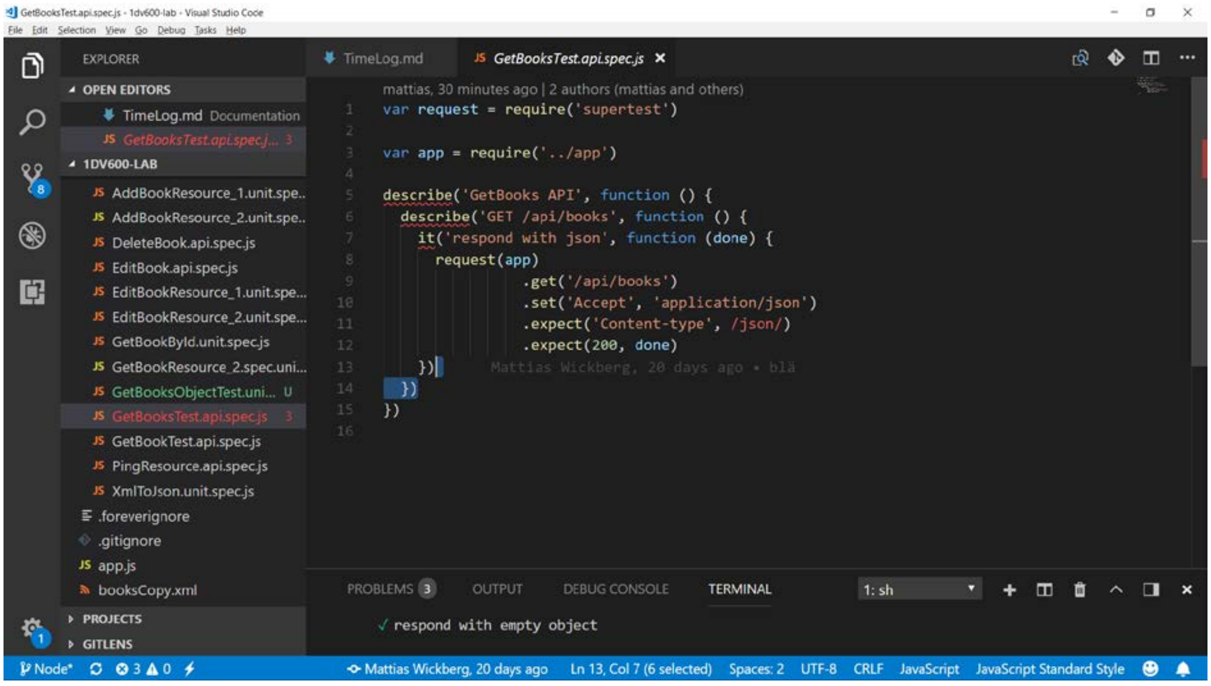


```
// Two tests: test object with ID, and test empty input

var expect = require('chai').expect
var XmlToJson = require('../app/resources/XmlToJson')

describe('XmlToJson', function () {
  describe('Tests function to see if it returns an object corresponding
    it('Returns an object with an ID', function (done) {
      var data =   { '$': { id: '101' },
      author: [ 'Stephen Hawking' ],
      title: [ 'A Brief History of Time' ],
      genre: [ 'Science' ],
      price: [ '199' ],
      publish_date: [ '1988-09-01' ],
      description: [ 'Hawking attempts to explain a range of subjects
      var book = XmlToJson(data)
      expect(book.id).to.equal('101')
      done()
    })
  })
})

describe('XmlToJson', function () {
  describe('Tests function to see that it can handle an empty input', f
    it('Returns undefined', function (done) {
     var book = XmlToJson()
     expect(book).to.equal(undefined)
     done()
    })
  })
})
```

```
var expect = require('chai').expect
var FindBookById = require('../app/resources/FindBookById')

describe('FindBookById', function () {
  describe('Tests the function to see that book id matches input id', f
    it('Returns book with correct id', function (done) {
        book = FindBookById(9)
        expect(book.id).to.equal(9)
        done()
      })
    })
  })
```

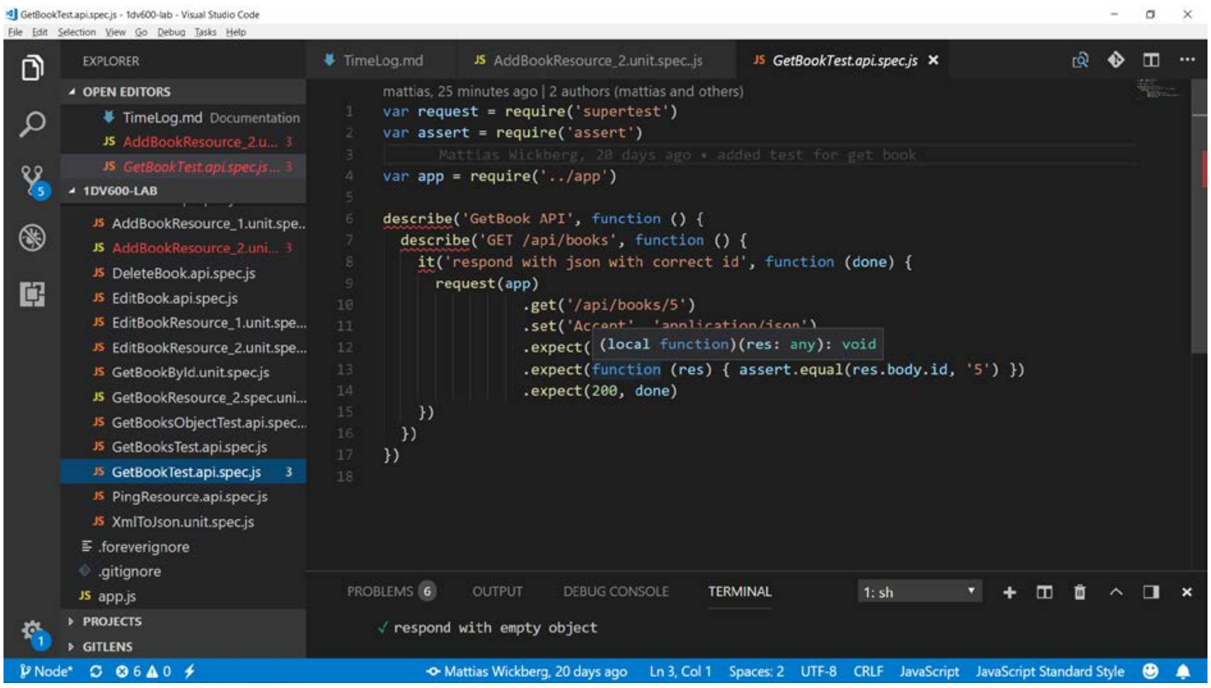
```
npm ERR!     /vagrant/npm-debug.log
vagrant@vagrant-ubuntu-trusty-32:/vagrant$
```

API tests:



```
var request = require('supertest')

var app = require('../app')

describe('GetBooks API', function () {
  describe('GET /api/books', function () {
    it('respond with json', function (done) {
      request(app)
          .get('/api/books')
          .set('Accept', 'application/json')
          .expect('Content-type', /json/)
          .expect(200, done)
    })
  })
})
```

```
✓ respond with empty object
```


GetBookTest.api.spec.js - 1dv600-lab - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
TimeLog.md Documentation
AddBookResource_2.u... 3
GetBookTest.api.spec.js... 3
1DV600-LAB
AddBookResource_1.unit.spe...
AddBookResource_2.uni... 3
DeleteBook.api.spec.js
EditBook.api.spec.js
EditBookResource_1.unit.spe...
EditBookResource_2.unit.spe...
GetBookById.unit.spec.js
GetBookResource_2.spec.uni...
GetBooksObjectTest.api.spec...
GetBooksTest.api.spec.js
GetBookTest.api.spec.js 3
PingResource.api.spec.js
XmlToJson.unit.spec.js
.foreverignore
.gitignore
app.js
PROJECTS
GITLENS
TimeLog.md
AddBookResource_2.unit.spec.js
GetBookTest.api.spec.js
mattias, 25 minutes ago | 2 authors (mattias and others)
Mattias Wickberg, 20 days ago • added test for get book
(local function)(res: any): void
PROBLEMS 6
OUTPUT
DEBUG CONSOLE
TERMINAL
1: sh
Node*
6 0
Mattias Wickberg, 20 days ago
Ln 3, Col 1
Spaces: 2
UTF-8
CRLF
JavaScript
JavaScript Standard Style

```javascript
var request = require('supertest')
var assert = require('assert')

var app = require('../app')

describe('GetBook API', function () {
  describe('GET /api/books', function () {
    it('respond with json with correct id', function (done) {
      request(app)
                .get('/api/books/5')
                .set('Accept', 'application/json')
                .expect(
                .expect(function (res) { assert.equal(res.body.id, '5') })
                .expect(200, done)
    })
  })
})
```

```
✓ respond with empty object
```

Mattias Wickberg, 1dv600

Personal reflections on Project 1dv600

#Assignment 1

## Task 1

Subtask A

The most complicated aspect of this task was really that I don't understand what the book object in the dao folder should be? In the end, I just made a file there and went on to create the book list in GetBooksResource. Again, I made the mistake of adding the book list to the callback to make it show on the website instead of console.log, so had to retrace and log it to the terminal instead. As always, the hardest part of any task is usually to actually understand what the task giver is saying/writing. This is something I try to remind myself of every day as a teacher, because it's often the most challenging task for both me and my students.

Subtask B

Converting to JSON is the work of a few minutes, so not much problem with the work this time.

Improvements: Firstly having a list of books in a function is really not a workable strategy for a site where you'll need to add new books through an interface, so first of all I need to separate my books from my functions, and then require that json file into my function. This is probably not good enough for the final version, but it's better than what's there now.

Secondly, I need to comment my code so that I know what I'm doing, and add more details to my books.

Subtask C

Really didn't need much time for this part, and I think maybe I should have come up with more complex improvements - however, without being quite sure what the requirements for the system is, I don't want to rush ahead with something major before I have implemented the basics thoroghly. I realise that my json file is very transitory, as there will be xml in the works, but still, it bugged me having a messy function.

Task 2

The difficulty with writing a vision is that I still have no clear idea of how complex this system can be during these few weeks, or whether to see as a library system for a full-fledged library, as a small scale version mostly used at home, or as a system for a smallish bookshop. They all have different demands. I went with the real library-angle, since I'm guessing that's what the basics of the system was intended for, and it may be difficult to work against what the course idea is. In reality it appeals more to build a system for my home or my school, sicne we don't really have one at the moment, but generally speaking it's better to avoid doing things you actually need in university assignments - they need to meet the requirements of the course first, and your own needs second.

Task 3

The project planning was mostly just a nice was of doing something with a bit more structure than I probably would have otherwise. Planning has been a crucial part of my life for many years now, juggling a complex work, studies and home life. The risk analysis was one part that I wouldn't have put into words had this not been part of the course literature, but it did help me focus on what the real risks are. This of course being a smallish project where I work alone rather than a large project with many people involved, it is a radically more simple plan than the ones I do work with for a living. Time wise I did overestimate the time it would take to do some parts of the first assignment, but the stimates were based on me usually being interrupted while I work, and this last two days I've had perfect calm to focus on this course!

#Assignment 2

##Task 1 ###Subtask A I have encountered UML-diagrams previously, and I do find them very useful in the planning stage for development. Use cases are of course very similar to ways that I've worked in Interaction Design, since I have a lot more experience from that field than programming, and so they do feel quite natural to have here as well. In general, I think the difficult part here is to decide on the scope of the planning stages of this project, knowing that a full scope library system would need more functionality than we can really make during a course like this.

Subtask B

Robustness diagrams in this very case seem a bit overkill for documentation and planning, but I can definitely see the usefulness of it to provide a link between the slightly abstract use cases to the actual implementation of the functionality. Especially for a larger sclae product where the people implementing and the ones designing the system may not be the same people or even in the same location, this seems to provide a good basis for clear technical communication of how the different steps should work.

Subtask C

Sequence diagrams, again feel quite useful. But like with subtask B I'm starting to wonder how many different diagrams are really useful for this fairly limited project. For this kind of project, I do feel that I would do use cases and either sequence diagrams or robustness diagrams, but to add more would start to feel a bit redundant. Again though, I can see that the story is different for larger scale projects. After doing the sequence diagrams I feel a have a better grasp of the system architecture.

##Task 2 This way of doing things does really force me to think through thigs before I do the coding. Of course, it also means I can't test and see if I'm mistaken in some assumption of how things work - but then if I do find out in the end that I made a planning mistake I guess I can go back and redo the sequence diagram. It probably reduces the amount of errors in the end anyways. It is akin to what I tell my students about essay writing (my profession is language teaching), that is is a process that is more about planning than actual writing if you want the end result to be of good quality and reduce the number of errors and unnecessary work you do.

Task 3

Ok so this one took a lot longer than I thought, possibly pasrtly because I started it while I was at the students' LAN party at work, and was quite tired and unfocused, but I really got tangled in how to transform the xml to json. It really should have been ten or twenty minutes work, but it took me two hours to figure that part out. The rest was really quite easy. Since the only real tangle here was the xml, I don't really feel that there was a huge difference between the designed and undesigned approach to implementing a function - the delete function was so much faster and easier to implement, because I had figured out the xml and understood the flow between different functions. I think the lessons I take away from this assignment is:

- What takes time is rarely what you think will take time - it's true about a lot of things in life.
- Converting information between formats is important, but can be time consuming (my partner who works as a bioinfomatician says that this is mostly what working as a scientist is about).
- Planning the flow between functions and what information travels is important, but planning the overall architecture is more important than the details.

Assignment 3

Task 1

OK, so I'm not completely done, but I think this will work better for me if I come back to the plan and fill in some more details after I've done the first two tests and have a clearer view of what I should test. I also think I'll come back and replan the API-tests after I'm done with the Unit tests, instead of planning every test before I start.

Task 2

Wrote two fairly simple manual tests. I first thought of these as real user tests, but I realised after reading about it that this is not the case - coming from the field of linguistics, pedagogy, UX and behavioural sciences testing does mean something a bit different for me than it does in the fireld of development 😃. Anyway, I do think that manual testing is what comes most natural for me, since it tends to be the an integral part of my natural work flow, though I'm not very thorough in documenting it properly. I see the importance of doing more of these, and it feels a bit wrong to stop at just these two - but I'm as always pressed for time.

Task 3

This was really good - I have been meaning to start looking more closely at unit testing and testing frameworks, and mostly this was not too tricky, although I know I have done very simple tests thus far. This is definitely something I want to dig into further, for my own progression, but also because I know that this is one of the weaker areas at our school.

Task 4

The trickiest part of this was that I got a false positive until I figured out how to use the 'done' functionality when testing code with callbacks. This also made me realise I had a bug in one of the previously written tests. Good learning 😃 Other than this not much more to say about testing - it's kind of a vital part of any development, but that feels lika a no-brainer (in theory, I know it's often lacking in practice).

Assignment 1

##Task 1

Subtask A

Expected time: 45 minutes. 2017-02-03, 17.23 - 18.16 Actual time: 53 minutes

Subtask B

Expected total time: 15 minutes 2018-02-03, 18.20 - 18.30 Actual time: 10 minutes

Subtask C

Expected total time: 60 minutes 2018-02-03, 19.15 - 19.55 Actual time: 40 minutes

Task 2

Expected time: 45 minutes 2018-02-04, 14.22 - 14.57 Actual time: 35 minutes

Task 3

Expected time: 75 minutes 2018-02-04, 15.43 - 16.37 Actual time: 54 minutes

Assignment 2

## Task 1

Subtask A

Expected time: 2 hours 2018-02-08, 19.50 - 21.46 Actual time: 1 hr, 56 minutes

Subtask B

Expected time: 45 minutes 2018-02-08, 21.55 - 22.30 Actual time: 35 minutes

Subtask C

Expected time: 60 minutes 2018-02-09, 20.50 - 21.29 Actual time: 39 minutes

Task 2

Expected time: 30 minutes 2018-02-09, 21.51 - 22.15 Actual time: 24 minutes

Task 3

Expected time: 120 minutes 2018-02-09, 22.15 - 23.22 2018-02-10, 10.00 - 11.20 2018-02-11, 09.53 - 11.15 Actual time: 229 minutes

Assignment 3

## Task 1

Expected time: 60 minutes 2018-02-22, 20.51 - 22.01 Actual time: 70 minutes

Task 2

Expected time: 60 minutes 2018-02-25, 10.09- 10.37 Actual time: 28 minutes

Task 3

Expected time: 150 minutes 2018-02-25, 10.40 - 12.14 Actual time: 94 minutes

Task 4

Expected time: 120 minutes 2018-02-26 20.25 - 21.45 Actual time: 80 minutes