Testing remote access to e-resource with CodeceptJS

Abstract:

At the Badische Landesbibliothek Karlsruhe (BLB) we offer a variety of e-resources with different access requirements. On the one hand, there is free access to open access material, no matter where you are. On the other hand, there are e-resources that you can only access when you are in the rooms of the BLB. We also offer e-resources that you can access from anywhere, but you must have a library account for authentication to gain access. To test the functionality of these access methods, we have created a project to automatically test the entire process from searching our catalogue, selecting a hit, logging in to the provider's site and checking the results. For this we use the End 2 End Testing Framework CodeceptJS.[1]

by Ralf Weber

Introduction and Motivation

The Badische Landesbibliothek Karlsruhe (BLB)[2] offers their users a variety of e-resources with different access possibilities. Some of these are freely accessible, but many require a login with username and password to ensure that you are a user of the library. In the BLB we use Shibboleth[3] to login. The procedure is always the same: the user usually selects the library as an institution and identifies himself with his user number and password. However, the implementation on the individual portals is not always obvious and often there are functional problems; sometimes access to a source that is actually licensed is denied. From time to time we get feedback from users about problems, but many don't report them because they suspect the problem to be their own. Therefore we thought about the possibility of automatically testing selected sources for their correct operation by mapping the complete workflow of information retrieval. For instance:

- Search for a specific term in the OPAC
- Expecting to find a specific match
- Login as user
- Access to the full text at the provider

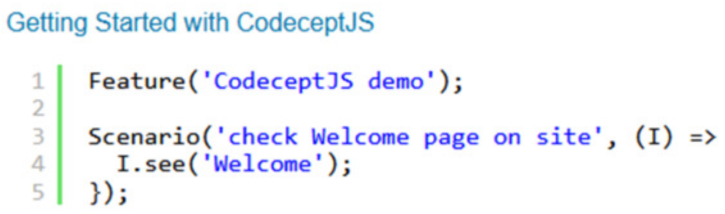
By automating these processes, we hope to discover problems or errors in access to the full text in order to react promptly and correct the error as quickly as possible, either on our side or by contacting the provider.

[1] https://codecept.io/
[2] https://www.blb-karlsruhe.de/
[3] https://www.shibboleth.net/

<u>What is CodeceptJs? Why did we choose this framework?</u>

CodeceptJS is a modern end to end testing framework with a special behavior-driven development (BDD) style syntax for NodeJS. The tests are written as a linear scenario of the user's action on a site.

It abstracts browser interaction to simple steps that are written from a user perspective. A simple test that verifies that "Welcome" text is present on a main page of a site will look like this:

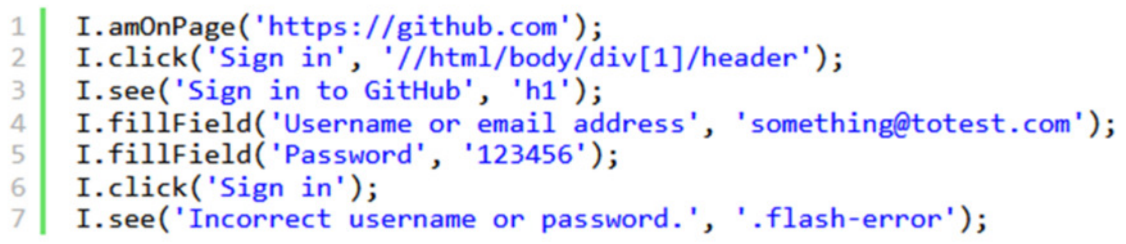

Getting Started with CodeceptJS

```
Feature('CodeceptJS demo');

Scenario('check Welcome page on site', (I) =>
  I.see('Welcome');
});
```

CodeceptJS passes execution commands to helpers. Depending on what you intend to test, you choose one of the helpers. In our case, we do not need cross-browser support, we are simply interested in faster testing and therefore chose the Chrome-based Puppeteer Helper.

Writing the tests for our scenarios can be done by librarians with little to no knowledge of programming. CodeceptJS is simple to use and tests are written from a user's perspective. There is an actor (represented as I) which contains actions taken from helpers. A test is written as a sequence of actions performed by an actor. Here's an example:



```
I.amOnPage('https://github.com');
I.click('Sign in', '//html/body/div[1]/header');
I.see('Sign in to GitHub', 'h1');
I.fillField('Username or email address', 'something@totest.com');
I.fillField('Password', '123456');
I.click('Sign in');
I.see('Incorrect username or password.', '.flash-error');
```

It's readable and simple and works using the Puppeteer API.

Since our tests are always more or less similar, we decided to create a template that can be used as a basis for each query. CodeceptJS allows us to store often used interactions like common locators and methods in "page objects." This way the code can be made even more simple and readable. For example, we have created a module to login with Shibboleth in which the access data of a test user is stored, saved as shibboleth.js:

```
/**
 * Modul für das Shibboleth Login
 */

const { I } = inject();

module.exports = {
  login () {
    I.fillField("#username", "XXXXXXXXXX");
    I.fillField("#password", "XXXXXXXXXX");
    I.click(".bBtn");
  },
  loginInTab () {
    I.switchToNextTab();
    I.fillField("#username", "XXXXXXXXXX");
    I.fillField("#password", "XXXXXXXXXX");
    I.click(".bBtn");
    I.wait(10);
  }
};
```

So for the whole login process, in the test file you only have to enter

```
ShibbolethPage.login();
```

Or, if it is necessary to switch to the next tab in the browser

```
ShibbolethPage.loginInTab();
```

We have created a second module, search.js, which contains all further interactions regarding the search for a hit and the access to the full text at the provider. This is best explained using an example.

Hands On – A practicle example

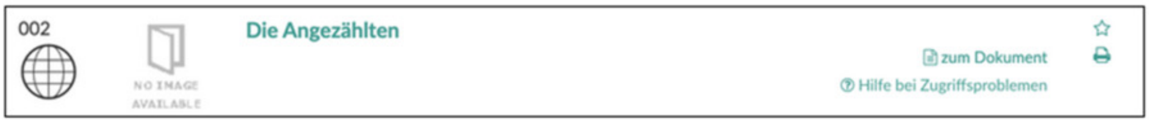
A simple example

If there are no special features necessary, a simple test proceeds as follows:

- Load start page of the OPAC
- Log in via Shibboleth
- Search a chosen title by ID
- Check if title matches expected title
- Click on link to full text
- Log in again
- Test for a text you expect to see at the provider

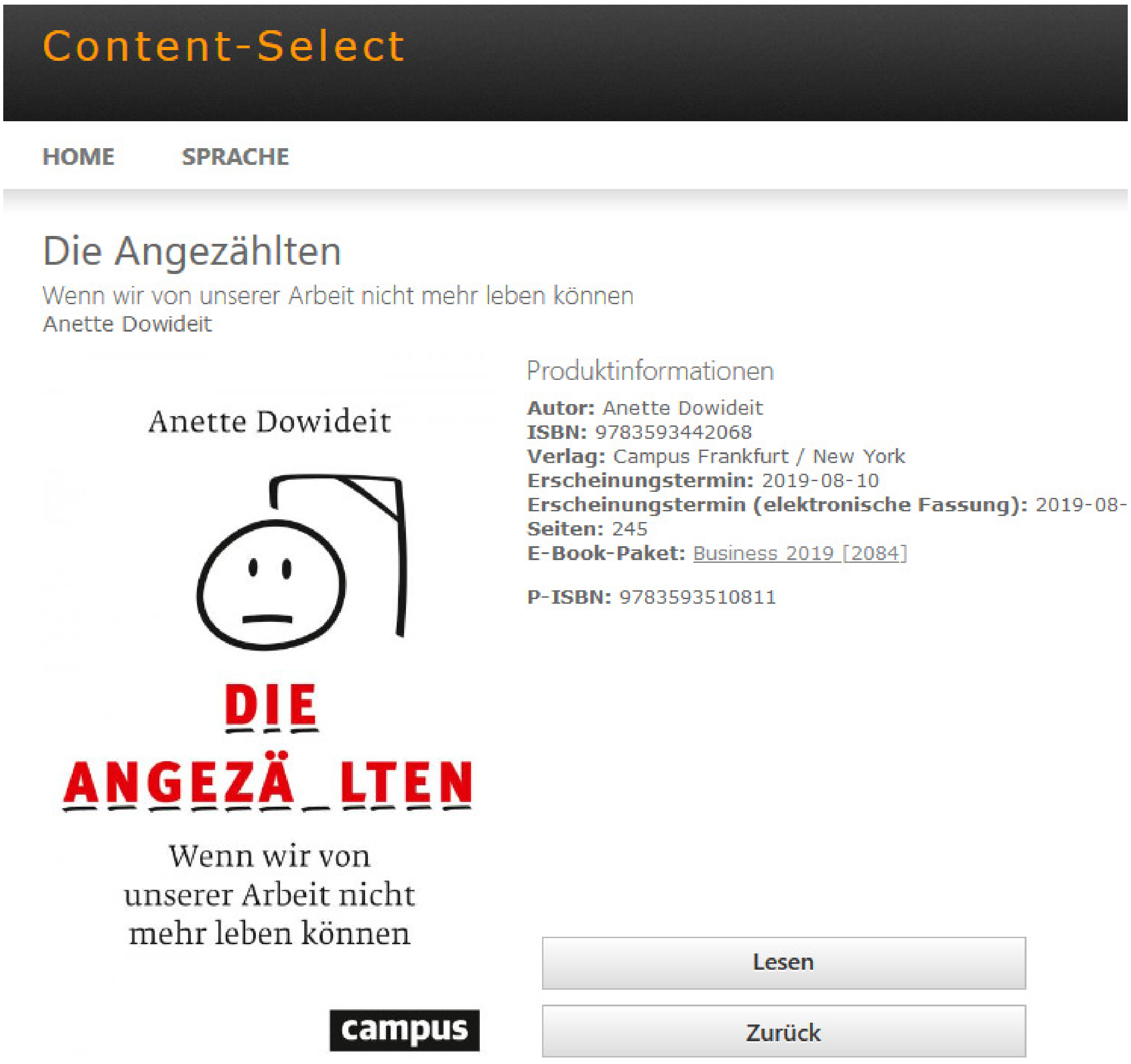
You can try this live:

- Go to https://rds-blb.ibs-bw.de/opac/
- Search for the id 1668499266



- We expect in our test the title “Die Angezählten” to be there.
- Now we click on the link to the full text (“zum Dokument”)
- Now, as you are not able to login, your journey ends on the Shibboleth Login page
- But in our test, we automatically login and then expect to see the text ‘Lesen’ on the page from the provider

This is a screenshot what the page from the provider looks like after successful login

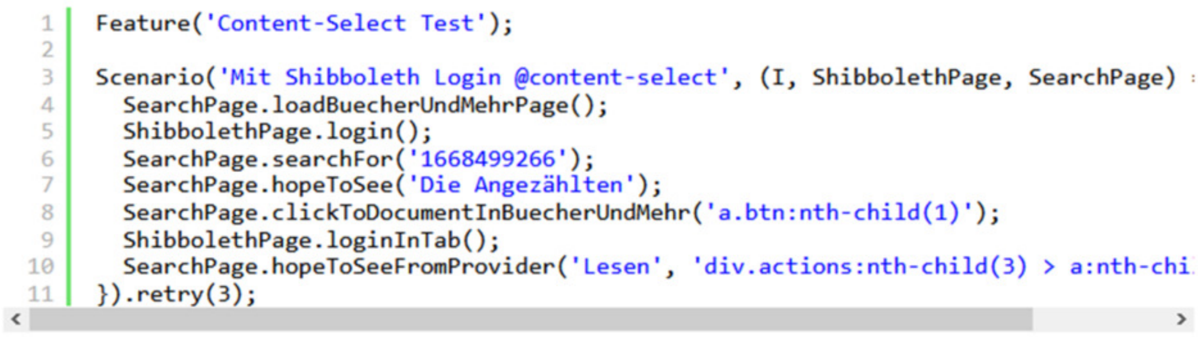


Die Angezählten
Wenn wir von unserer Arbeit nicht mehr leben können
Anette Dowideit

Produktinformationen
Autor: Anette Dowideit
ISBN: 9783593442068
Verlag: Campus Frankfurt / New York
Erscheinungstermin: 2019-08-10
Erscheinungstermin (elektronische Fassung): 2019-08-
Seiten: 245
E-Book-Paket: Business 2019 [2084]

P-ISBN: 9783593510811

You can see the text “Lesen”.
The test itself looks like this:



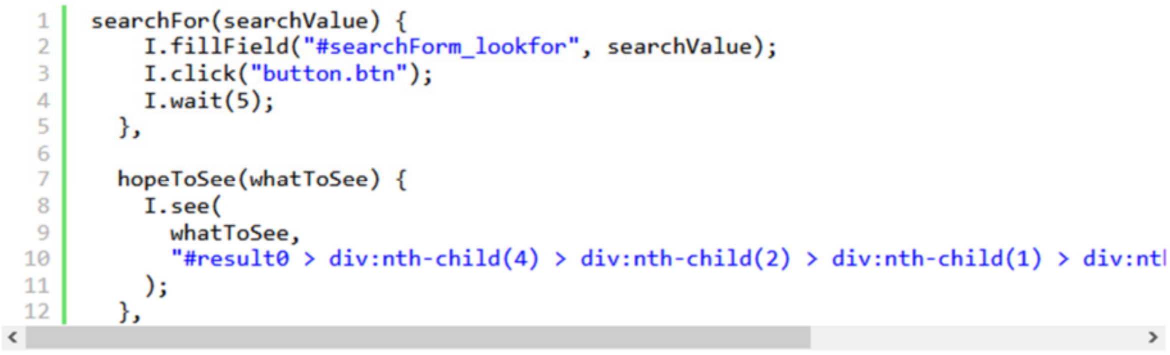
```
Feature('Content-Select Test');

Scenario('Mit Shibboleth Login @content-select', (I, ShibbolethPage, SearchPage)
  SearchPage.loadBuecherUndMehrPage();
  ShibbolethPage.login();
  SearchPage.searchFor('1668499266');
  SearchPage.hopeToSee('Die Angezählten');
  SearchPage.clickToDocumentInBuecherUndMehr('a.btn:nth-child(1)');
  ShibbolethPage.loginInTab();
  SearchPage.hopeToSeeFromProvider('Lesen', 'div.actions:nth-child(3) > a:nth-chi
}).retry(3);
```

Each line corresponds to one of the instructions listed above

As the identifiers (CSS/XPath) for the search string and the title in the result list are always the same, you only have to specify their values, so no further CSS/XPath rules are necessary.

In the search.js file these functions look like this:



```
searchFor(searchValue) {
    I.fillField("#searchForm_lookfor", searchValue);
    I.click("button.btn");
    I.wait(5);
  },

  hopeToSee(whatToSee) {
    I.see(
      whatToSee,
      "#result0 > div:nth-child(4) > div:nth-child(2) > div:nth-child(1) > div:nt
    );
  },
```

In summary, a librarian writing a simple test only needs to take this template and adjust three things:

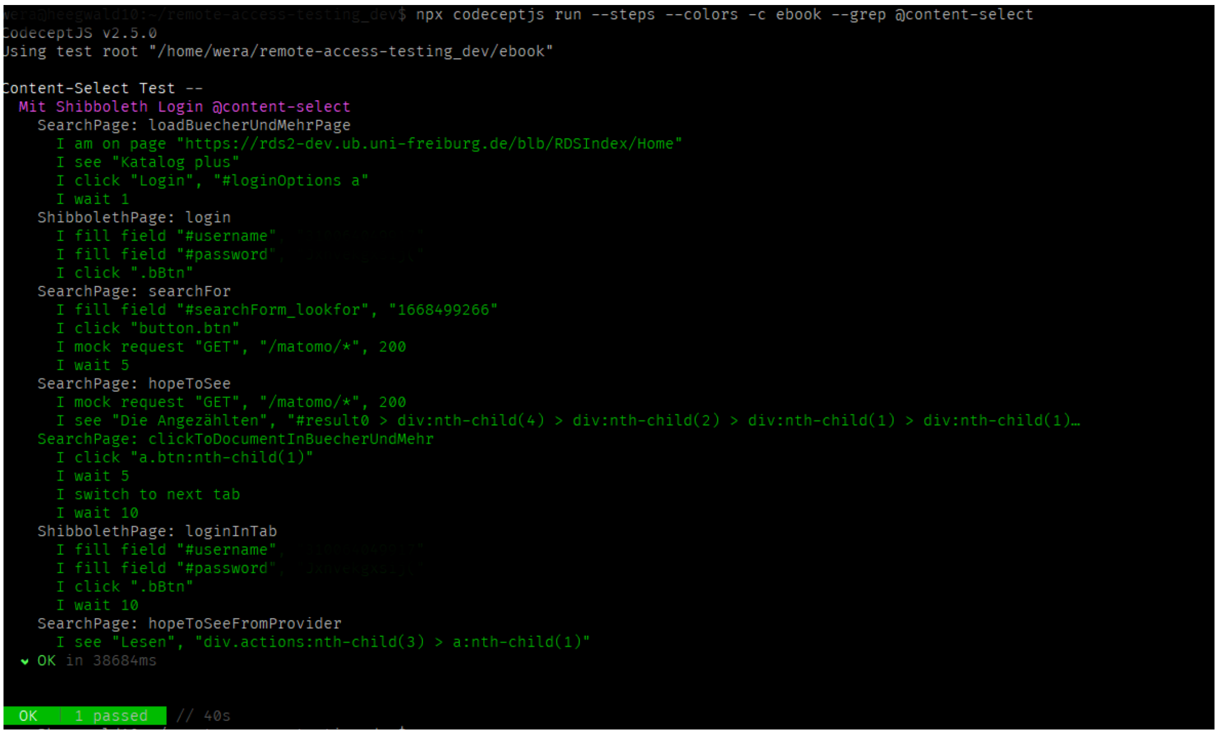
1. the values for the Id to be searched
2. the value of the expected title in the result list
3. the text that can only be seen on the provider’s site after successful login, including the (CSS/XPath) identifier that he gets with Firefox/Chrome DevTools.

The test gets called by the following command:

The test gets called by the following command:

```
npx codeceptjs run --steps -colors filename_test.js
```

The output looks like this (green when everything is fine, red on errors)



```
wera@heegwald10:~/remote-access-testing_dev$ npx codeceptjs run --steps --colors -c ebook --grep @content-select
CodeceptJS v2.5.0
Using test root "/home/wera/remote-access-testing_dev/ebook"

Content-Select Test --
  Mit Shibboleth Login @content-select
    SearchPage: loadBuecherUndMehrPage
      I am on page "https://rds2-dev.ub.uni-freiburg.de/blb/RDSIndex/Home"
      I see "Katalog plus"
      I click "Login", "#loginOptions a"
      I wait 1
    ShibbolethPage: login
      I fill field "#username", [redacted]
      I fill field "#password", [redacted]
      I click ".bBtn"
    SearchPage: searchFor
      I fill field "#searchForm_lookfor", "1668499266"
      I click "button.btn"
      I mock request "GET", "/matomo/*", 200
      I wait 5
    SearchPage: hopeToSee
      I mock request "GET", "/matomo/*", 200
      I see "Die Angezählten", "#result0 > div:nth-child(4) > div:nth-child(2) > div:nth-child(1) > div:nth-child(1)…
    SearchPage: clickToDocumentInBuecherUndMehr
      I click "a.btn:nth-child(1)"
      I wait 5
      I switch to next tab
      I wait 10
    ShibbolethPage: loginInTab
      I fill field "#username", [redacted]
      I fill field "#password", [redacted]
      I click ".bBtn"
      I wait 10
    SearchPage: hopeToSeeFromProvider
      I see "Lesen", "div.actions:nth-child(3) > a:nth-child(1)"
  ✔ OK in 38684ms


OK  | 1 passed   // 40s
```

Troubleshooting and Limitations

Normally the tests run as shown in the example above, with no problems.

We have found that it makes sense to take a “break” between the individual steps. Otherwise the test may fail because the page is not yet completely loaded and access to the DOM will not succeed. CodeceptJS offers the command wait(seconds) for this. This ensures that the page is loaded and the complete DOM is available before the next test starts. We use this extensively. The test runs longer, but that doesn’t matter.

We also wanted to avoid recording these test accesses in statistics. CodeceptJS offers MockRequest for this.
This helper allows you to use mock requests while running tests, so you can block calls to 3rd-party services like Matomo, Google Analytics etc.

```
I.mockRequest('GET', '/matomo/*', 200);
```

These calls are also integrated in the functions

Some sites expect the consent of the privacy policy, which appears in a pop-up window. Access to this was not possible with CodeceptJS via the DOM. But we could solve this by setting the required cookies manually. CodeceptJS also provides a simple method for this:

```
I.clearCookie();
  I.setCookie({
    name: "AccessSessionSignature",
    value: "2df0a0adc0a705630e5f62be9f2efdf4b87835094e077dc76e6799dcbdbae627"
  });
  I.setCookie({
    name: "AccessSessionTimedSignature",
    value: "b2a0050b0df65bab1b16bad39f08507302c0218db8a0c18a55dbd48f8744984c"
  });
```

Fortunately, these cookies do not have to be created each time but can be loaded with fixed values directly from the test file. Therefore no further work or preparation is necessary and these tests can be started by cronjob without problems too.

On some pages you have to select the institution from a dropdown list, sometimes this happens dynamically via AJAX. CodeceptJS is well equipped for this also as you can see here on these examples[4]:

Filling Fields

```
I.selectOption('Choose Plan', 'Monthly'); // select by label
I.selectOption('subscription', 'Monthly'); // match option by text
I.selectOption('subscription', '0'); // or by value
I.selectOption('//form/select[@name=account]','Premium');
I.selectOption('form select[name=account]', 'Premium');
I.selectOption({css: 'form select[name=account]'}, 'Premium');
```

To make sure that there are no temporary network problems or other technical failures, we optionally run each test three times. CodeceptJS offers the possibility of RetrySteps[5].
You simply append the function *retry(amount)* to the scenario.

```
Scenario('Really complex', (I) => {
  // test goes here
}).retry(2);
```

The execution is only repeated if the test has failed, very nice!

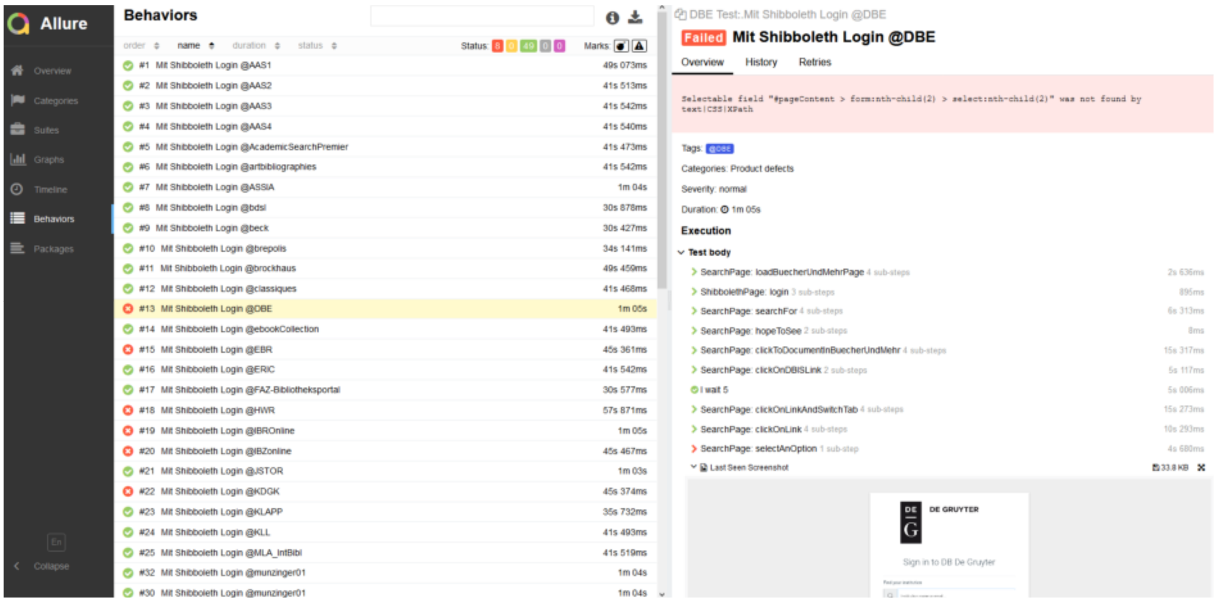
There is one thing we have not yet been able to solve.
There's a provider who requests a ReCAPTCHA in addition to the login. Maybe we can think of a solution for this in the future.

[4] https://codecept.io/acceptance/#filling-fields
[5] https://codecept.io/basics/#retry-step

Conclusion

Meanwhile our tests are running productively and automated. We only run them from off campus, because if they work there, they also work inside the rooms of the library. For a better organization we have divided them into three groups: database, e-book and e-journals. At the time of writing, we have 83 tests and more will be added.

We have set up a cronjob for each group which calls up the test files once a month, with the result is logged and sent by e-mail to the responsible employees. Additionally, we have activated the Allure plugin CodeceptJS.cThis is a reporting tool that shows a representation of what has been tested in a web report form.



Status: 8 0 49 0 0 Marks:

order	name	duration
#1	Mit Shibboleth Login @AAS1	49s 073ms
#2	Mit Shibboleth Login @AAS2	41s 513ms
#3	Mit Shibboleth Login @AAS3	41s 542ms
#4	Mit Shibboleth Login @AAS4	41s 540ms
#5	Mit Shibboleth Login @AcademicSearchPremier	41s 473ms
#6	Mit Shibboleth Login @artbibliographies	41s 542ms
#7	Mit Shibboleth Login @ASSIA	1m 04s
#8	Mit Shibboleth Login @bdsl	30s 878ms
#9	Mit Shibboleth Login @beck	30s 427ms
#10	Mit Shibboleth Login @brepolis	34s 141ms
#11	Mit Shibboleth Login @brockhaus	49s 459ms
#12	Mit Shibboleth Login @classiques	41s 468ms
#13	Mit Shibboleth Login @DBE	1m 05s
#14	Mit Shibboleth Login @ebookCollection	41s 493ms
#15	Mit Shibboleth Login @EBR	45s 361ms
#16	Mit Shibboleth Login @ERIC	41s 542ms
#17	Mit Shibboleth Login @FAZ-Bibliotheksportal	30s 577ms
#18	Mit Shibboleth Login @HWR	57s 871ms
#19	Mit Shibboleth Login @IBROnline	1m 05s
#20	Mit Shibboleth Login @IBZonline	45s 467ms
#21	Mit Shibboleth Login @JSTOR	1m 03s
#22	Mit Shibboleth Login @KDGK	45s 374ms
#23	Mit Shibboleth Login @KLAPP	35s 732ms
#24	Mit Shibboleth Login @KLL	41s 493ms
#25	Mit Shibboleth Login @MLA_IntBibl	41s 519ms
#32	Mit Shibboleth Login @munzinger01	1m 04s
#30	Mit Shibboleth Login @munzinger01	1m 04s

DBE Test:.Mit Shibboleth Login @DBE

Failed **Mit Shibboleth Login @DBE**

```
Selectable field "#pageContent > form:nth-child(2) > select:nth-child(2)" was not found by
text|CSS|XPath
```

Tags: @DBE
Categories: Product defects
Severity: normal
Duration: 1m 05s

Execution

Test body

SearchPage: loadBuecherUndMehrPage 4 sub-steps	2s 636ms
ShibbolethPage: login 3 sub-steps	895ms
SearchPage: searchFor 4 sub-steps	6s 313ms
SearchPage: hopeToSee 2 sub-steps	8ms
SearchPage: clickToDocumentInBuecherUndMehr 4 sub-steps	15s 317ms
SearchPage: clickOnDBISLink 2 sub-steps	5s 117ms
I wait 5	5s 006ms
SearchPage: clickOnLinkAndSwitchTab 4 sub-steps	15s 273ms
SearchPage: clickOnLink 4 sub-steps	10s 293ms
SearchPage: selectAnOption 1 sub-step	4s 680ms
Last Seen Screenshot	33.8 KB

DE GRUYTER
Sign in to DB De Gruyter

This allows details of the tests to be tracked, and in case of an error there is a screenshot, which often makes troubleshooting extremely easy.

Now that we have been in the production phase for a few months, we can say that we have achieved what we wanted. We can automatically control whether external access to full text works or not, we notice long response times on the part of the providers and can ask them to improve. We can better understand the problems of the users to reach the full text, because now we have to write the whole process as a test. Here we can see how complex the user guidance is. And we notice changes to the user interface of the website on the part of the provider, because then our tests fail, which fortunately is not often.

About the Author

Ralf Weber is the Information Technologies Librarian at the Badische Landesbibliothek Karlsruhe.
He cares and develops tools for automation of library workflows and for new services within the context of the library.