Cypress

AID 119

ADITO Academy

Version 1.0 | Last changed 22.04.2022

Versions	Changes
1.0	Release Version
0.9	Internal Review Version

Index

Character Formatting

The following signs will point you to specific sections:

Hints and notes.

Tips and tricks.

This is important!

Warning! These actions are dangerous and can result in data loss!

The following font formatting applies:

Font type	Meaning
Mask	The mask, table or button to which the section refers
"Mask"	Terms that originate from the system and that need to be emphasized in the reading flow
`code();`	Code and program parts

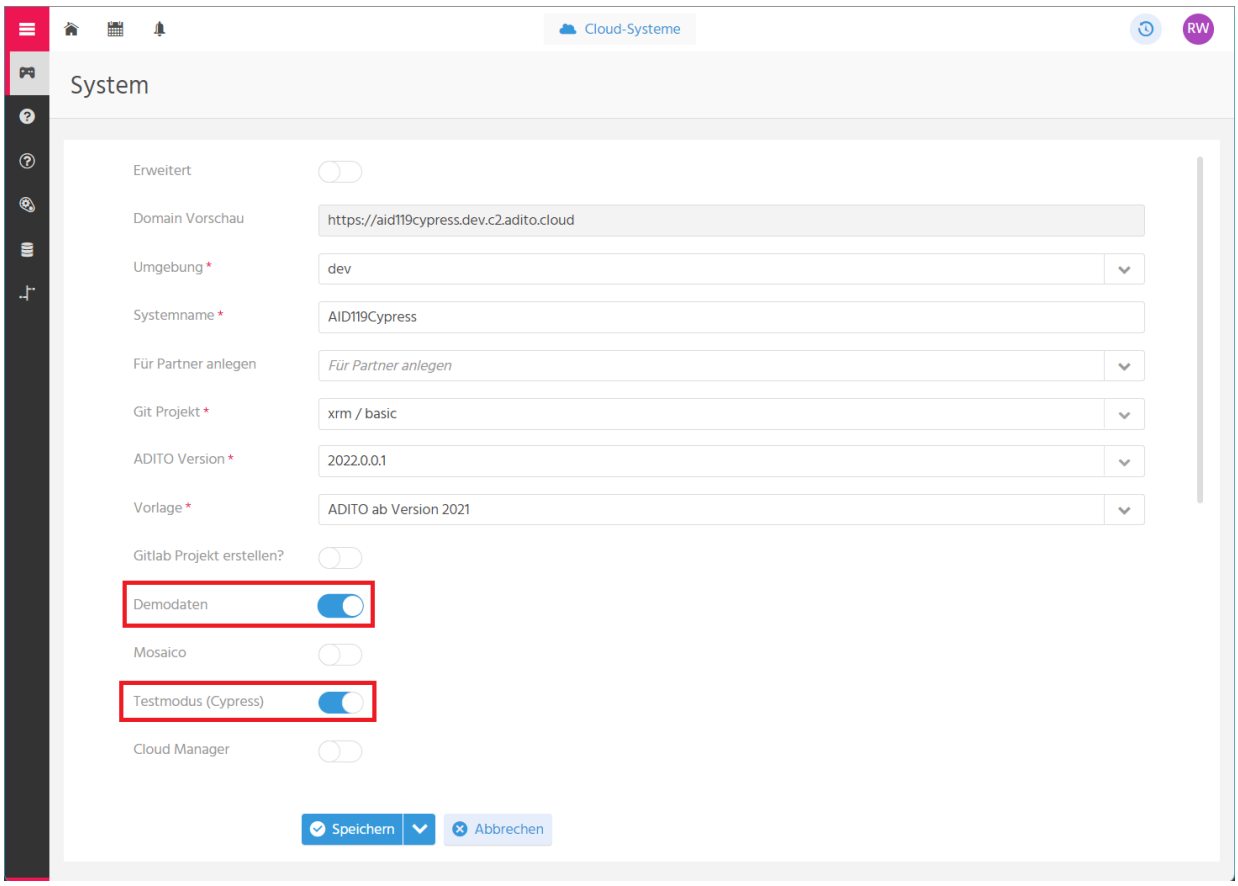
1. Purpose of this Document

The AID for cypress covers the necessary steps to install all required tools for writing and executing **frontend tests**.
This includes the setup of a special cloud system in the ADITO SSP which will be used to run all tests and all required steps in the ADITO Designer for writing tests.
After the setup and basic functions, commands and the writing of tests is covered.

2. Setting up a Cloud System for testing

To execute the cypress tests, an ADITO system is needed. The easiest way to create a working system is to use the ADITO SSP.
While creating a new Cloud system the "testmode" has to be activated:

Since tests can rely on existing data to execute tasks, it is necessary to check the "demodata" as well.



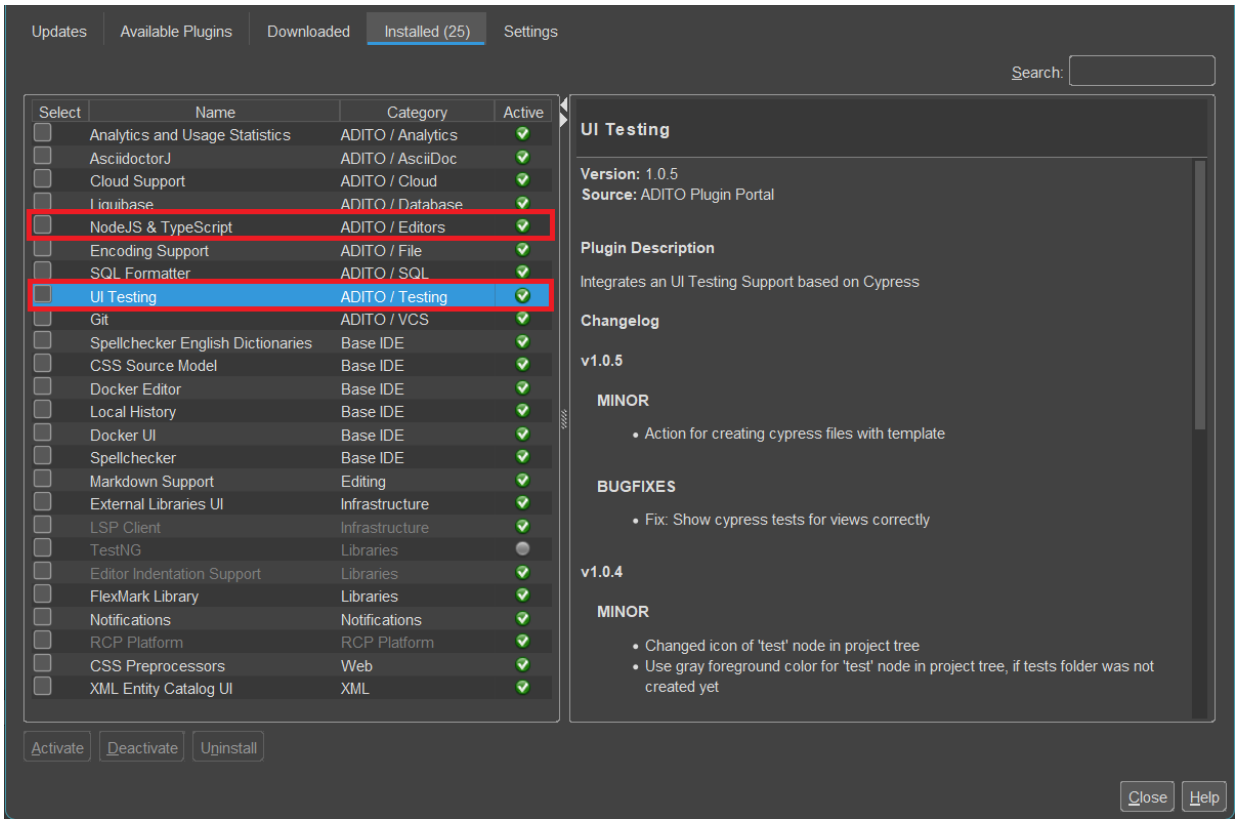
3. Setting up the ADITO Designer

To use cypress tests a ADITO Designer with at least the Version 2022.0.0 is required.

3.1. Required Plugins

The Designer needs to have the following plugins installed and activated:

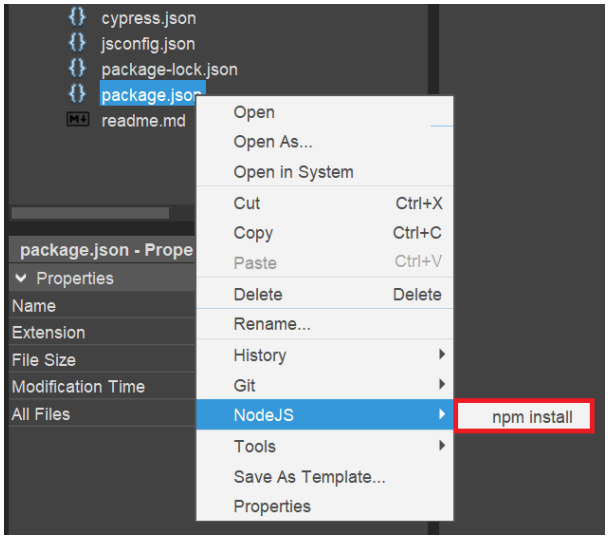
1. NodeJS & Typescript
2. UI Testing



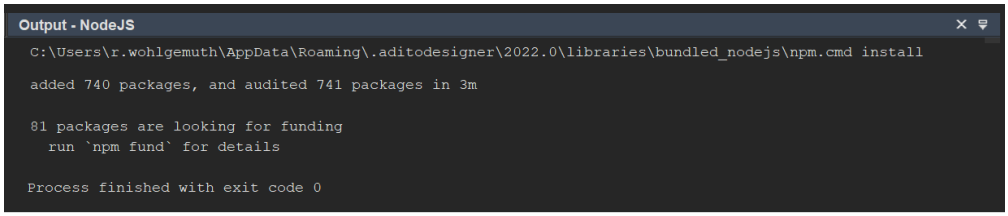
Your xRM/ADITO Project must be at least version 2021.2.4, since this version the files needed for cypress and its dependencies are present.

3.2. Installing cypress and all dependencies

Since npm is used to install cypress and all needed dependencies, a "npm install" has to be executed before cypress can be used. To achieve this, the designer provides an action inside the context-menu of the "package.json"-file in your project-tree.



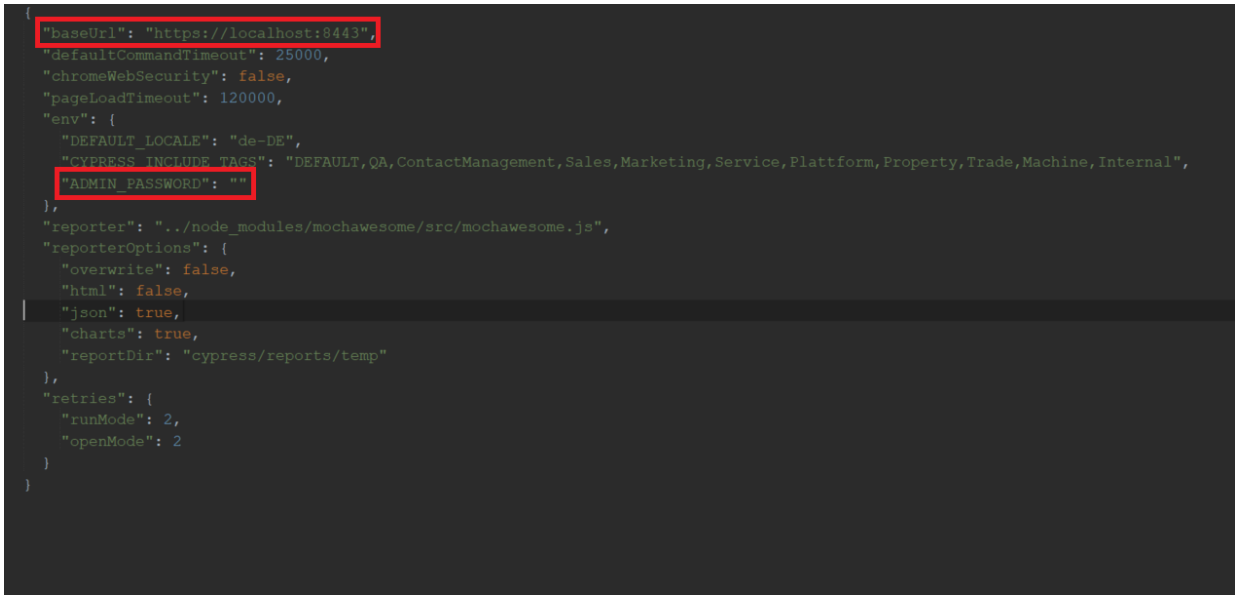
After the "npm install" is executed, the progress can be seen inside the new output-window.



When the installation is finished, nodejs will output "Process finished with exit code 0". (0 means no errors accused)

3.3. Connecting cypress to the created cloud system

To let cypress know which server to connect to, the credentials have to be altered inside the "cypress.json" of your project.
For this, the **URL** and **admin-password** has to be entered, like in the following screenshot:



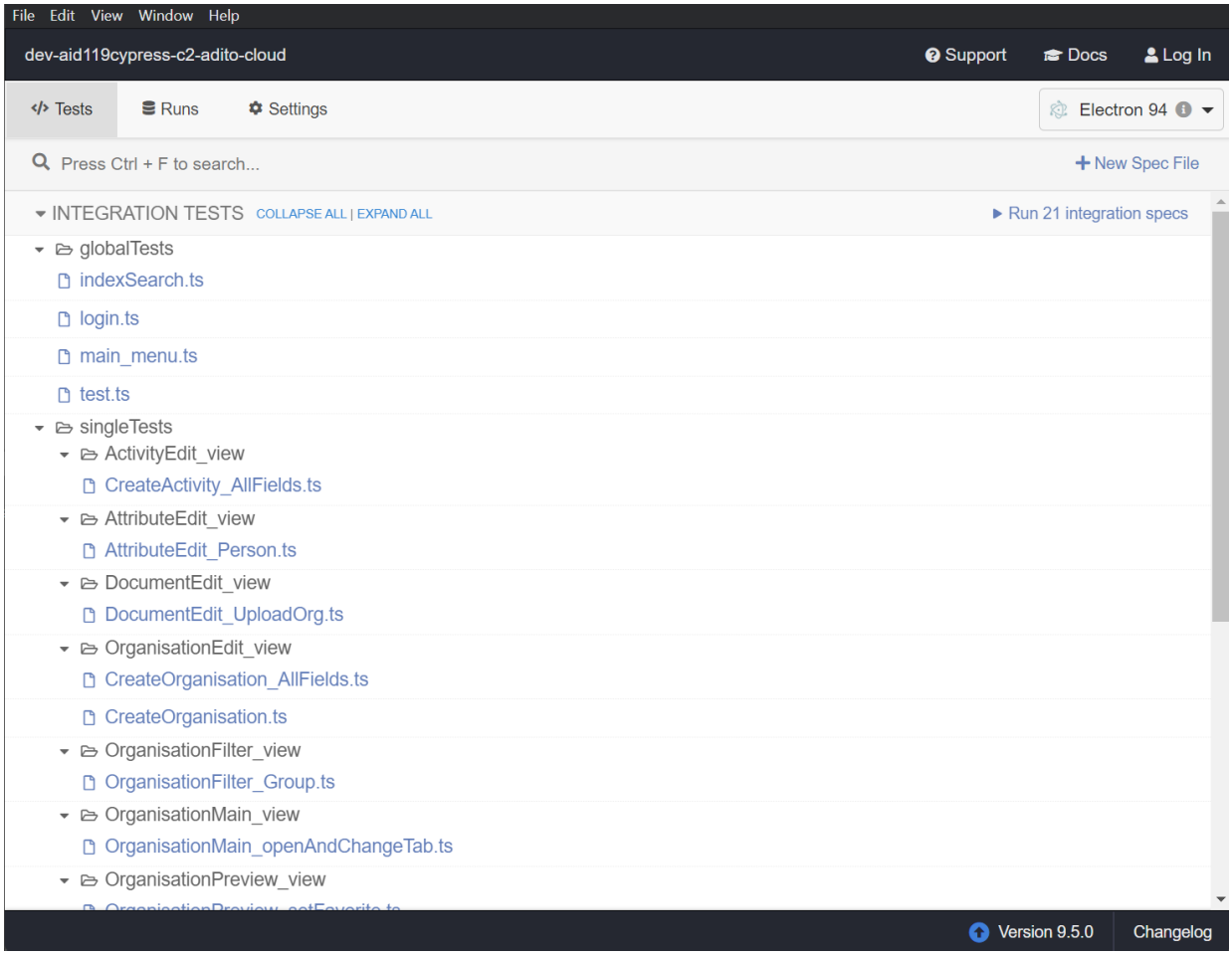
The baseurl must not have the "/client" ending.

4. Using cypress

4.1. Start cypress

To start the cypress interface and start executing tests, the "start cypress" script must be started.

After the execution of the run-command a new cypress-window will open:



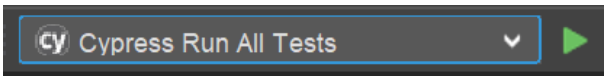
At first all tests that are currently present in your project will be listed.

4.2. Executing a test in cypress

To execute a specific test, click on the corresponding file in the provided list inside the cypress-window.

At the top of this window a search bar is located, you can use that to search for a desired filename.

If you want to execute all tests, the run-script "Cypress Run All Tests" in the Designer can be executed.



4.3. Analyzing tests in cypress

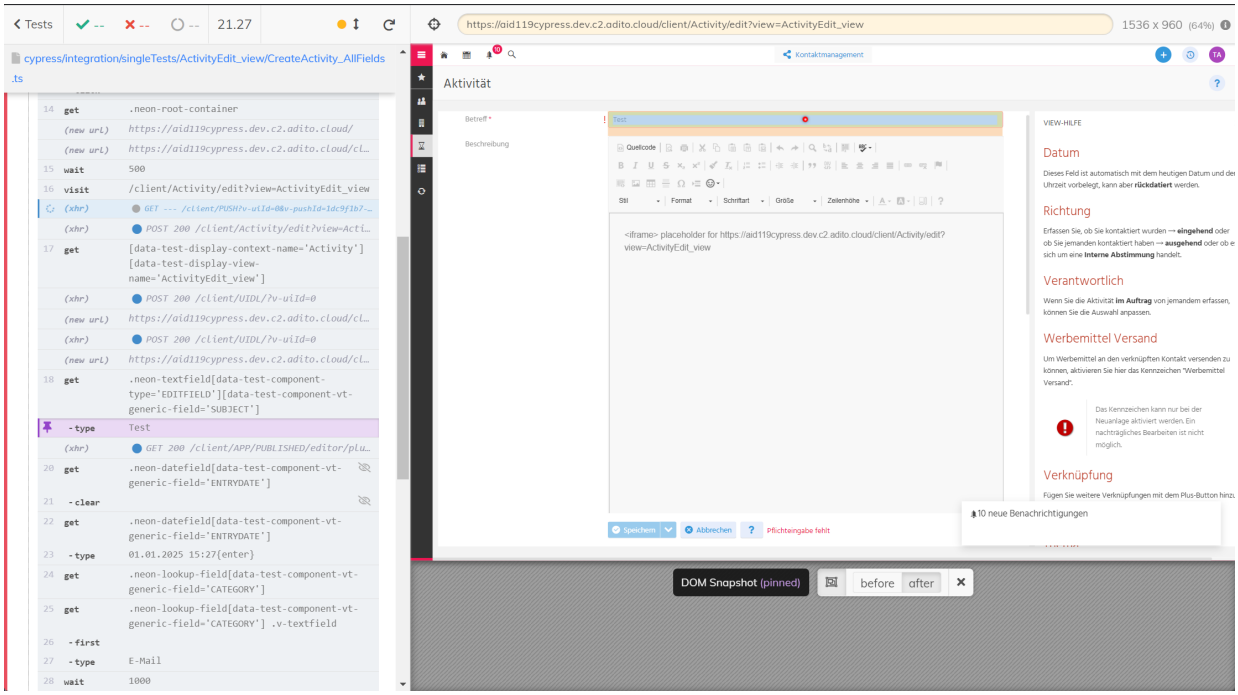
While a test is executed, you can watch everything that happens in the preview on the right side of the "run"-window.

To see or get more information about specific steps of the running test, you can click on a step in the list on the left side.
If you do not want to get interrupted while looking at a step, wait for the test to finish or use the "stop"-button above the left panel.

Every step can be hovered over, and the preview will show the snapshot from when this step was executed.
It is also possible to pin a step by clicking on it.

Additional information can be found in the official cypress documentations segment "The Test Runner".

While hovering over a step, the preview will cycle through the before- and after-snapshot.
If the Step is pinned, you can choose between the before- and after-snapshot with the buttons below the preview.



If a test fails to run without error, it will automatically try to run again.

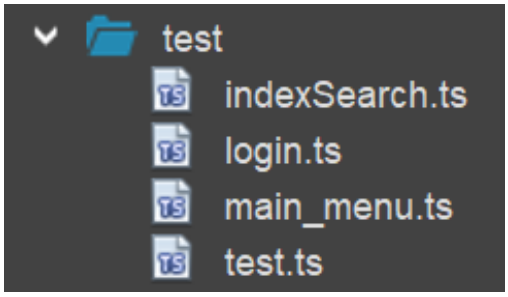
5. Writing tests

5.1. Locating the needed files and libraries for testing

5.1.1. Tests

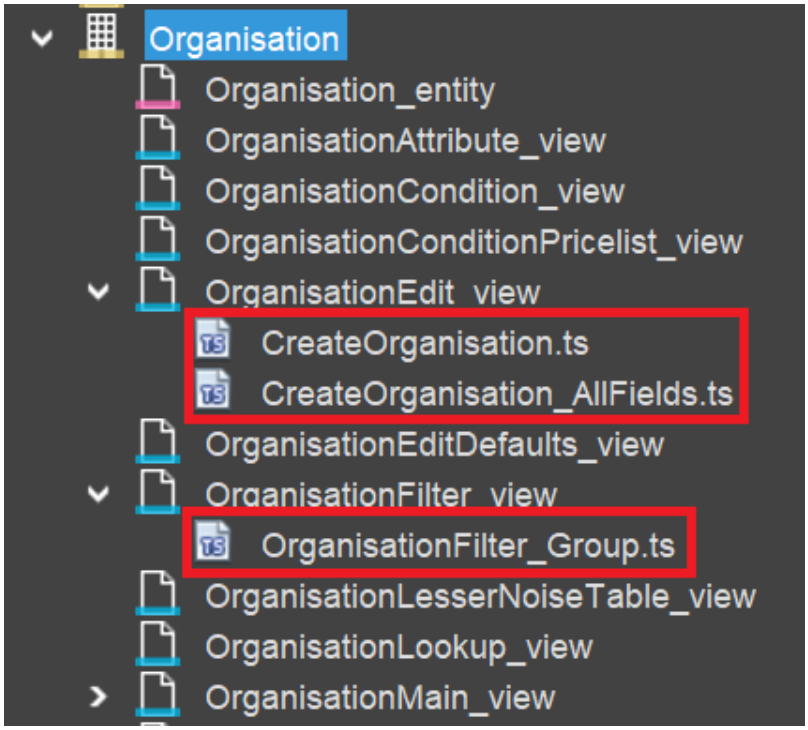
5.1.1.1. General tests

Tests that do not correspond to a specific context or view are located under the "tests"-folder inside the project-tree (project-window).



5.1.1.2. Tests for specific views

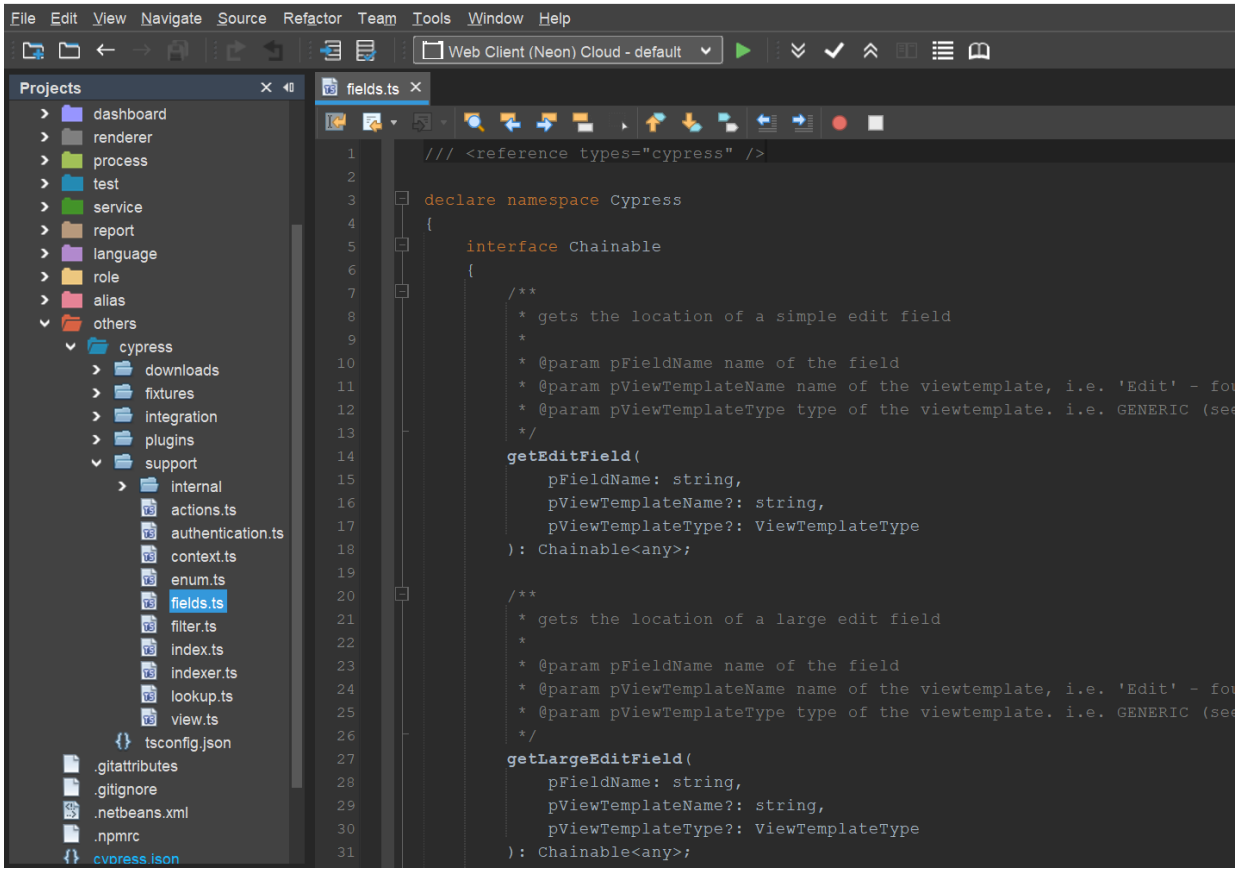
If you want to look at all tests that are for a view, you can find these under the context and then view inside the project-tree on the left ot the Designer.



5.1.2. Functions

To help and make writing tests easier and avoid repetitive code, a number of helper-functions are included. These functions are located in the "others - cypress - support"-folder inside the project-tree.

The basic helper-functions are listet in the chapter "Basic functions/commands → Custom for ADITO"



5.2. Basic functions/commands

This chapter just contains the basic needs to get started. For a full list of commands that cypress includes, go visit the official documentation.
Custom functions provided via the helper-libraries of ADITO will expand in the future.

5.2.1. Cypress

Command	Description	Example
.get	Tries to find a component	`cy.get(".neon-root-container")`
.click / .dblclick	Clicks on a through .get specified component	`cy.get("[...]").click()`
.type / .clear	Type into a component / clear existing text	`cy.get("[...]").type("admin")`
.should	A given assertion is checked to be true or not	`cy.get("[...]").should("be.visible")`
.contains	Checks if a component contains a text	`cy.get("[...]").contains("Kontakt")`
.visit	Opens a specified sup-url	`cy.visit("/client/login")`
.wait	Browser will wait for given time (in ms)	`cy.wait(2000)`
.first / .last	Chooses the first / last element	`cy.get("[...]").first().click()`

5.2.2. Custom for ADITO

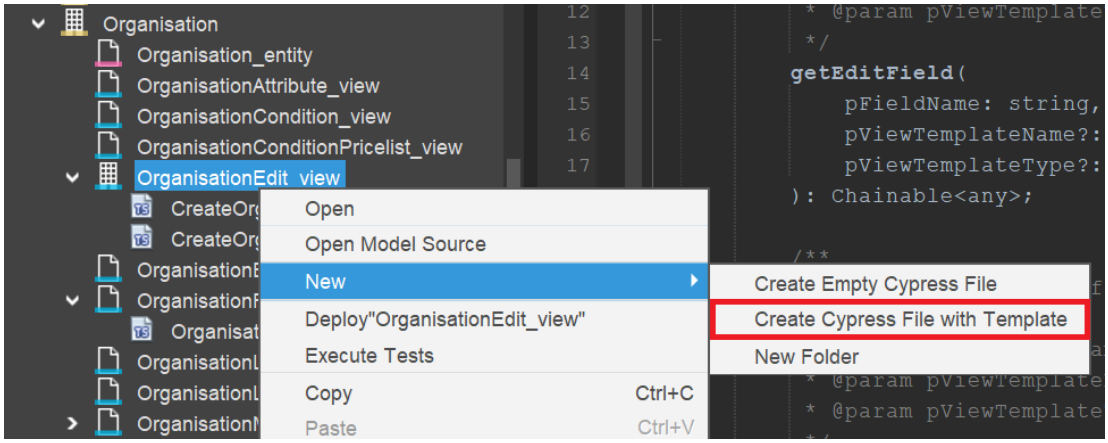
Command	Description	Example
.login	Log in (optional as given user with password)	`cy.login()`
.openContext	Open a context and view	`cy.openContext("Organisation", "OrganisationFilter_view", PresentationMode.FILTER)`
.fillLookUpField / .fillComboField	Fill a lookup or combo-field with a specified value	`cy.fillLookUpField("SALUTATION", "Herr")`
.getEditField / .getMultiEditField	Jumps to a specified (multi-) edit-field	`cy.getEditField("LASTNAME").type("Maier")`
.saveEdit	Save a edit view	`cy.saveEdit(true, true)`
.pressButton	Presses a button depending on the given parameters	`cy.pressButton(1)`

Further information about the functions can be found in the documentation inside the helper lib itself.

5.3. Writing a new test

5.3.1. Preparing

The easiest way to create a new test is by finding the corresponding context and view. On the wanted view the option "New - Create Cypress File with Template" is to be used. After you input the desired filename in the dialog, the new file will be opened.



Test-files must always be named with the following rules in mind:

Is the test using the **filter view** and …

1. an action is pressed → [Context]Action.ts
2. the data is filtered → [Context]Filter.ts
3. the data is grouped → [Context]Group.ts

Is the test using the **edit view** and …

1. a new record is created → [Context]Create.ts
2. a record is altered → [Context]Edit.ts

Is the test using the **main view** → [Context]Main.ts

is the test using the **preview** and …

1. an action is pressed → [Context]PreviewAction.ts
2. a record ist altered → [Context]PreviewEdit.ts
3. entries are checked (nothing changed or pressed) → [Context]PreviewFields.ts

To start of, we add the login-function in the beforeEach-function and name the use- and testcase (describe and it). Next we want to open the edit view of the context organisation (to create a new company).

The result will look something like this:

```
beforeEach(() => {
    cy.resetData();
    cy.login();
});

describe("Usecase", () => {
    it([Tag.DEFAULT], "Testcase", () => {
        cy.openContext("Organisation", "OrganisationEdit_view",
PresentationMode.EDIT);
    });
});
```

If you run this test now, the client will be opened and cypress will log in as the admin user (with the password from the "cypress.json" of your project) and open a new edit view of the organisation context.

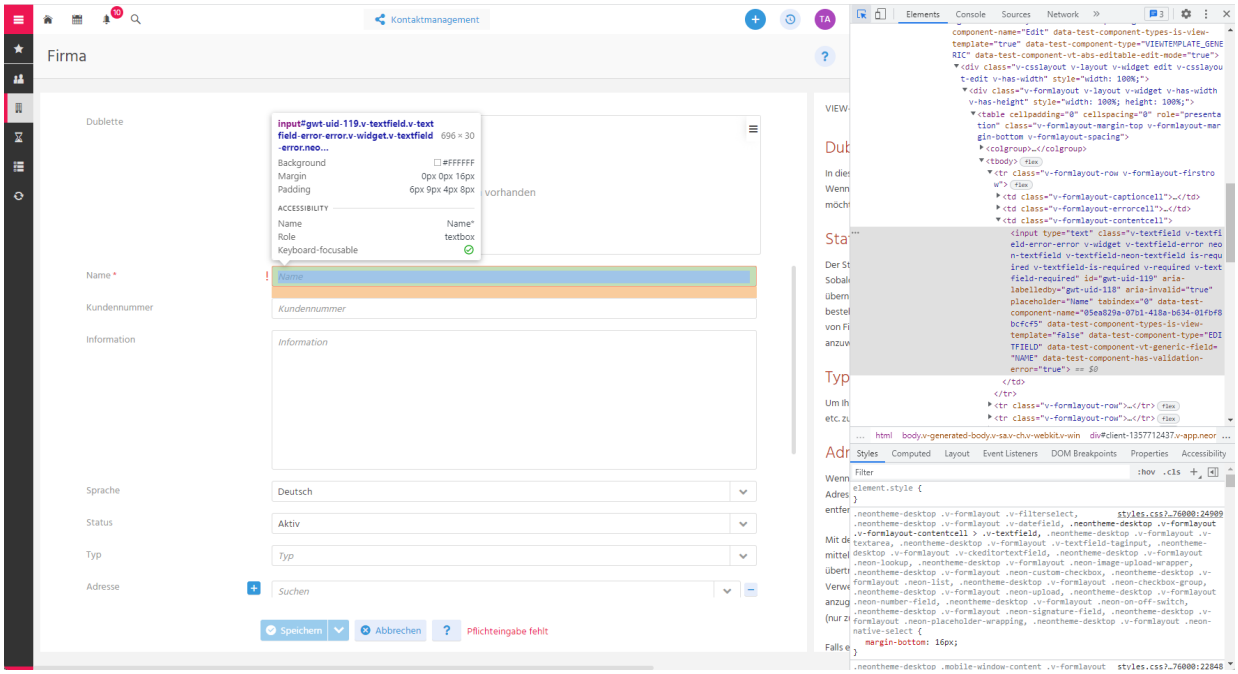
5.3.2. Examples

5.3.2.1. Fill and check a field in an edit view

To get elements with the cypress-get command we need to know how an element is identifiable.

The official cypress documentation provides a best-practise reference for selecting elements.

The best way to search for identifiers is to use the inspect-element function of any browser.



Here we find out that the "Name"-field of the organisation edit view has the attribute *data-test-component-vt-generic-field="NAME"* which we can use in combination with a custom ADITO command:

```
cy.getEditField("NAME").type("MyCustomCompany 123");
```

Now we can check if the input we typed is still present. To achieve this we can use the "contains" command provided by cypress:

```
cy.getEditField("NAME").contains("MyCustomCompany 123");
```

5.3.2.2. Complete example of a cypress-test

```
import { createPerson } from "../../../support/entityHelper/Person";
beforeEach(() => {
    cy.resetData();
    cy.login();
    cy.openContext("Person", "PersonEdit_view", PresentationMode
.EDIT, {});
});

var lastname = "AID_EXAMPLE_NAME";
var salutation = "Herr";
var value = "Unbekannt";

describe("UC2004751 - Person - Edit_View - Person anlegen", () => {
    it([Tag.ContactManagement], "TC2004757 - Speichern nicht
möglich, wenn Eigenschaft 'Beurteilung / Loyalitaet' nicht gefuellt
", () => {
        createPerson(lastname, salutation, null, true);
        cy.checkSaveDisabled("VALUE", "Eigenschaft \"Loyalität\"
muss mindestens 1 mal verwendet werden.", true);
    });
    it([Tag.ContactManagement], "TC2004756 - Speichern nicht
möglich, wenn 'Nachname' nicht gefuellt", () => {
        createPerson(null, salutation, value, true);
        cy.checkSaveDisabled("LASTNAME");
    });
    it([Tag.ContactManagement], "TC2004755 - Speichern nicht
möglich, wenn 'Anrede' nicht mit Auswahl aus Dropdown gefuellt", ()
=> {
        createPerson(lastname, null, value, true);
        cy.checkSaveDisabled("SALUTATION");
    });
    it([Tag.ContactManagement], "TC2004754 - Speichern nicht
möglich, wenn 'Sprache' nicht gefuellt", () => {
```

```
        cy.getLookUpField("LANGUAGE")
          .clear();
        cy.fillLookUpField("SALUTATION", salutation);
        cy.wait(500);
        cy.getEditField("LASTNAME")
          .type(lastname);
        cy.pressMinusButton("AdressMultiEdit_view");
        cy.pressMinusButton("CommunicationMultiEdit_view");
        cy.fillComboField("VALUE", value, true);
        cy.checkSaveDisabled("LANGUAGE");
    });
    it([Tag.ContactManagement], "TC2004753 - Alle Pflichtfelder sind
mit validen Werten gefuellt", () => {
        createPerson(lastname, salutation, value);
    });
    it([Tag.ContactManagement], "TC2004752 - Alle Felder sind mit
validen Werten gefüllt", () => {
        var language = "Deutsch";
        var organisation = "meineFirma"
        var title = "Dr.-Ing.";
        var salutation = "Herr";
        var firstname = "Nico";
        var middlename = "Jamal";
        var lastname = "Testoteles";
        var dateOfBirth = "10.03.2005";
        var status = "Aktiv";
        var dept = "Geschäftsleitung";
        var role = "Geschäftsführer";
        var position = "Geschäftsführer";
        var value = "Gering";
        var blog = "tm-testoteles.de";
        var gender = "Männlich";
        var zip = "84144";
        var city = "Geisenhausen";
        var street = "Gutenbergweg";
        var streetNumber = "4";
        var address = street + " " + streetNumber + " " + zip + " "
+ city;
        cy.getLookUpField("LANGUAGE")
          .find(".neon-textfield.v-textfield-neon-textfield")
          .should("have.value", language);
        cy.get(".neon-lookup-field[data-test-component-vt-generic-
field='ORGANISATION_CONTACTID']")
            .openLookupWithSearch({ query: organisation, select: 0
})
            .wait(1000);
        cy.fillLookUpField("SALUTATION", salutation);
        cy.fillLookUpField("TITLE", title);
        cy.getEditField("FIRSTNAME").type(firstname);
```

```
        cy.getEditField("MIDDLENAME").type(middlename);
        cy.getEditField("LASTNAME").type(lastname);
        cy.getDateField("DATEOFBIRTH")
            .type(dateOfBirth);
        cy.getLookUpField("STATUS")
          .find(".neon-textfield.v-textfield-neon-textfield")
          .should("have.value", status);
        cy.fillComboField("DEPARTMENT", dept);
        cy.fillComboField("CONTACTROLE", role);
        cy.fillComboField("POSITION", position);
        //Address fields
        cy.getMultiEditField("ZIP")
            .type(zip + "{enter}");
        cy.getMultiEditField("CITY")
            .type(city + "{enter}");
        cy.getMultiEditField("ADDRESS")
            .type(street + "{enter}");
        cy.getMultiEditField("BUILDINGNO")
            .type(streetNumber + "{enter}");
        cy.getMultiEditField("STATE")
            .type("Niederbayern{enter}");
        cy.getMultiEditField("ADDRESSADDITION")
            .type("3. Wohnung rechts{enter}");
        cy.getMultiEditField("ADDRIDENTIFIER")
            .type("Test{enter}");
        //Communication fields
        cy.get(".neon-lookup-field[data-test-component-
name='MEDIUM_ID']")
            .type("Blog").wait(500);
        cy.get(".neon-none-layout[data-test-component-
name='DefaultLookup_view']")
            .children()
            .contains("Blog")
            .click()
            .wait(500);
        cy.get(".neon-textfield[data-test-component-name='ADDR']")
            .type(blog)
            .type("{enter}");
        //Attribute fields
        cy.get("[data-test-component-name='VALUE']")
            .first()
            .type(value)
            .wait(200)
            .type("{enter}");
        cy.saveEdit(true, true);
        cy.checkLabel("", salutation + " " + title + " " + firstname
+ " " + lastname, FieldType.TITLE);
        cy.checkLabel("", organisation, FieldType.DESCRIPTION);
        cy.checkLabel("", blog, FieldType.COMMUNICATION);
```

```
        cy.checkLabel("", address, FieldType.ADDRESS);
        cy.checkLabel("GENDER", gender);
        cy.checkLabel("DATEOFBIRTH", dateOfBirth);
        cy.checkLabel("STATUS", status);
        cy.checkLabel("DEPARTMENT", dept);
        cy.checkLabel("CONTACTROLE", role);
        cy.checkLabel("POSITION", position);
        cy.checkLabel("LANGUAGE", language);
        cy.openTabInMain("Eigenschaften");
        cy.checkLabel("", value, FieldType.ATTRIBUTE);
    });
});
```