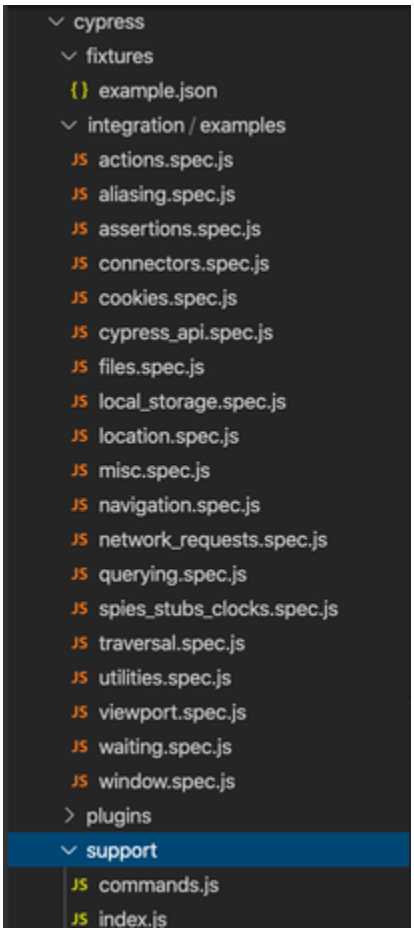
Cypress Standards Document



Project Names

Add the project naming standards here!

Fixtures

Cypress provides a directory named ***fixtures,*** which stores various "*JSON"* files, and these *JSON* files can store the test data which can be read by multiple tests. We store test data in the form of *key-values,* which we can access in the test scripts.

Naming Convention for fixtures will be **Camel Casing**.

Fixture files also help us segregate different environment test data.

drive[APP][Env].json

Examples

driveCrmDev.json, driveCrmStg.json, driveCrmqa.json

It is not recommended to store key-chains and value data

Examples
Token values, Passwords, Email ids.

An important point to note here is that Cypress also relies on user-made plugin/libraries.
There are cases when these plugins tell us to store certain types of files in the fixture folder.
e.g. A cypress file upload plugin stores the .csv extension file in the fixture folder.
The naming convention for these exceptions:

Camel Casing for these types of files and the Entity will be plural

drive[APP][env][Entity].[ext]

Examples
driveCrmDevCustomers.csv
driveCrmStgPostCards.xlsx

Integration

The integration folder includes all the tests files. We can create as many folder as we want and it could be nested folder as well.

Integration folder contains user scenarios that use multiple different methods from the commands folder.

String Conventions for **Describe** and **it** blocks.

Describe:

A Describe block contains a combination of multiple “it” blocks which run in a linear fashion. Hence why the describe block’s string to define it will contain all the “it” block’s functions.

Example:
describe(“LoginUI, Campaign-Regular-Email/Text; Add, Send, Check Inbox, Email Performance”)

The string that will be printed when the above describe block will run represents all the functions which run step by step in this describe block.

Because this is a string that gets printed, we do not need to follow any specific naming convention.
The first function that runs is the loginUI function and afterwards, multiple Campaign-Regular-Email/Text functions run which are written followed by a semicolon.

Folders

Inside the Integration folder, we have multiple folders which will follow the Camel Casing naming convention. If the folder has multiple words there will be no spaces.

Examples:
campaigns, socialAccounts, leadsOthers, leadsWebsite

File Names

The files inside the folders will follow Camel Casing naming convention.

Examples

addCustomer, deleteCustomer

Structure

For example, we have a scenario which checks if the email campaign was sent and opened by the customer.
for this scenario to work, we need certain prerequisites, which are as follows:

Add a customer, add a vehicle and attach it to the customer, add a custom tag and associate it with that customer, add an email campaign and schedule and send it.

So this scenario ensures that all the below mentioned modules/functionalities/screens are tested.

login

Customers

Tags

EmailCampaign

So we need to define scenario names for integration test files.

Customer-->edits, deletes etc-->.

Support

This folder contains **index.js** and **commands Folder.** The index.js file runs before each **test file.** By default we have the commands folder, here we can store the reusable code and use in all the test files. Reusable code can be certain functions, commands, modules which are common and or need to be used throughout the application multiple times, for example, the login function.

File Names

In order to manage the commands.js file, we can create multiple commands.js files and the naming convention for that is Camel Casing.

The file names will use the following format:

[Entity]Commands.js

Examples

customerCommands.js

campaignCommands.js

for common functions/methods, we will use commands.js

Method Names

The method name should include the action that its performing. It should use camel case.
Example

login[type]()

where type is the type of login, example, it can be a User interface login, Api login etc.

loginUI(), loginApi()

The method to do a specific action will be named with the same name as the action in camel case.

E.g. sendCampaign(), scheduleCampaign()

add[Entity]()

edit[Entity]()

delete[Entity]()

[action][entity]()

Below are the common methods that are used in components and their naming convention:
· The method to create an entity will be either create or if the screen/button name exists with add, then we will name it add[Entity], e.g. addCustomers.
· The method to edit an entity will be “edit[Entity]”. editCustomer()
· The method to delete an entity will be “delete[Entity]”, e.g. deleteCustomer()
· The method to do action in common will be “doAction”. [action][entity](), drivePauseCampaign()
· The method to save the data of the screen will be “save”, e.g. save.
· The method to refresh the data of the screen will be “refresh”, e.g. refresh.
· The method to check if the screen has changed will be “isDataChanged”.

PageObjects

Page Object files are used to store the relative and absolute locators for the respective app.

Every screen will have its own respective page object js file, elements which are common like the headers etc will have a common objects js file. The naming convention will be camel casing and the name used will be the screen name.
[screen-Name][Page].js

campaignsPage.js

Buttons should have the following naming convention;

btn[Action]

Button	btn	btnNext

Test Apis

All the cypress testscripts will be mapped manually to drive’s testrail account.
Below is the naming conventions for the apis:

Camel casing for the API names

[action][Entity].ts

createRun.ts