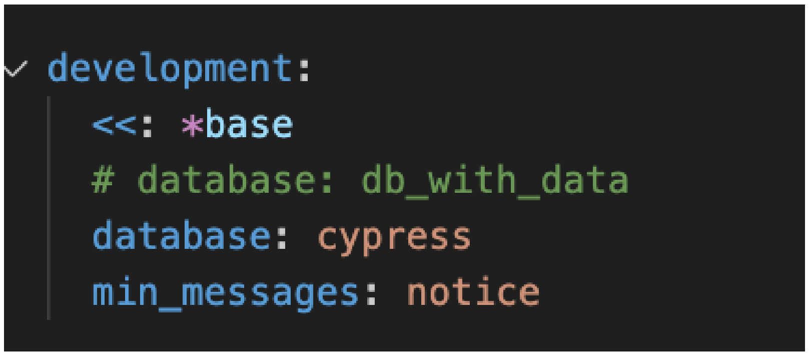
Cypress Introduction

Set Up

1) Set up ManageIQ and ManageIQ UI:
https://ibm.ent.box.com/notes/721845308562

2) For now all Cypress tests will be run on an empty database. In order to match the test environment you should write all your Cypress tests in an empty database as well. To do this:

- In ManageIQ → config/database.yml under development create a new database by commenting out the database with data and creating a new line for your cypress database. Example:



```
development:
  <<: *base
  # database: db_with_data
  database: cypress
  min_messages: notice
```

- Then in the ManageIQ repo terminal run the command:

    rake evm:db:reset

*** NOTE: If you ever need to change between databases just comment the line for the database you don't want and uncomment the database you want to change to. Then run bin/update and start the server. This will only work after you have done the initial set up for each database.

*** NOTE: If you ever need to reset your Cypress database back to default for example you added data and don't remember where it was located or you can't delete it in the UI then stop the server and run the command: rake evm:db:reset. This will reset the database back to default and then you can restart your server.

3) Start the server and confirm that you are running on an empty database. This can be done by navigating through the UI and ensuring that the tables are empty. Commands you should know:

- To start the UI/server:
  `bin/rails s`
- Optional: Not needed for Cypress tests but when making changes to the UI code if you want to reload the UI without restarting the server you can run the command:
  `bin/webpack --watch --follow`
- Optional: Not needed for Cypress tests but some pages will need this command running in order to load or function correctly. It is good to keep this running at all times to ensure all features are working as intended.
  - To start the rails console:
    `rails c`
  - Then in the rails console run:
    `simulate_queue_worker`

*** NOTE: It is good practice to run all these commands in the ManageIQ-UI-Classic repo terminal. The bin/rails s command can be run from the ManageIQ repo terminal but some commands like the bin/webpack command and the Cypress commands will only work when run from the ManageIQ-UI-Classic repo terminal. By running all these commands in one place it will allow you keep organized and minimize/close your ManageIQ repo terminal.

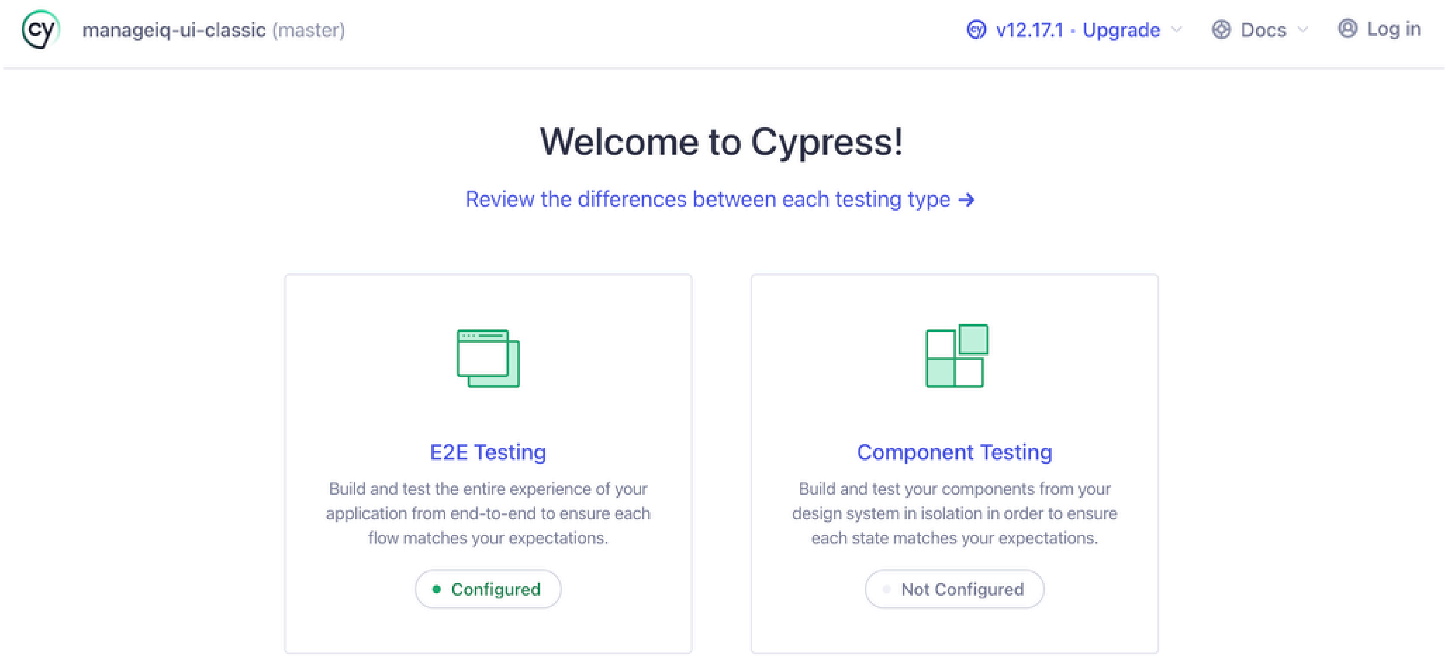
4) There are 2 ways to run the Cypress tests, through the Cypress UI or headless. In order to write, run, and debug your tests easier use the Cypress UI. If you ever want to run the tests in a similar format to the Github actions environment you can run the tests in headless mode. The headless mode will run the tests in the terminal without opening any UI or browser.

- Cypress UI:
  ```
  yarn cypress:open
  ```
- Headless mode (select which browser you want to run the tests on):
  ```
  yarn cypress:run:chrome
  yarn cypress:run:edge
  yarn cypress:run:firefox
  ```

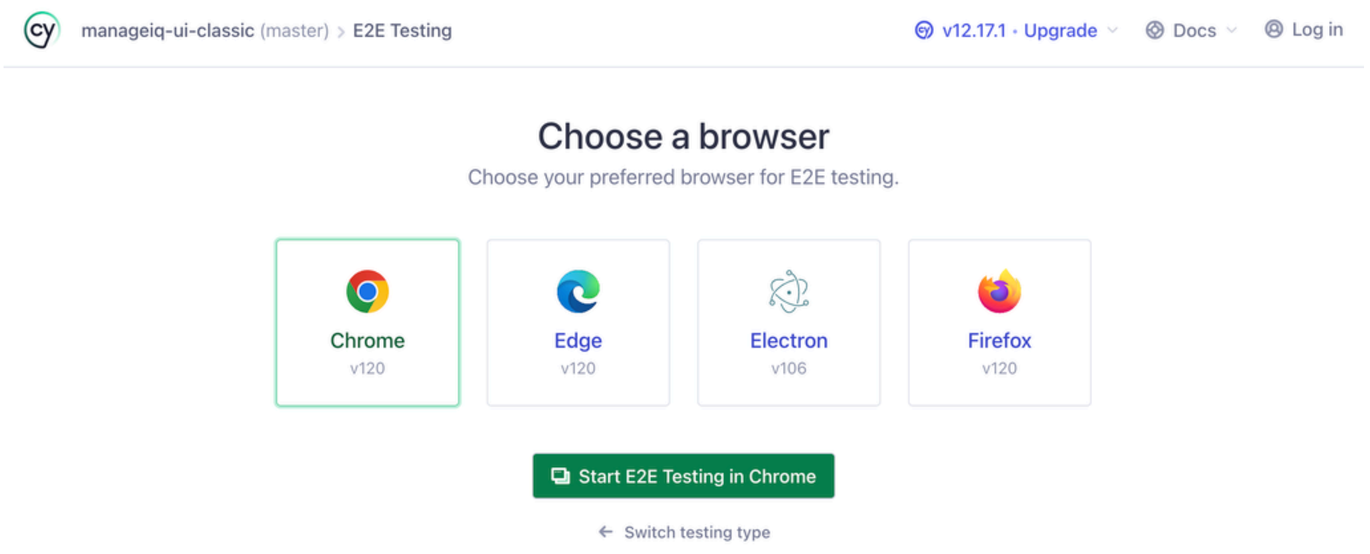
*** NOTE: Since the Cypress UI will be the main method of writing, running and debugging your tests this guide will focus on instructions for the Cypress UI.

*** NOTE: You must have all these browsers installed on your computer in order to run the Cypress tests on them.

5) While running the UI/server with bin/rails s, open the Cypress UI using yarn cypress:open. This will open the Cypress UI which will look like this:

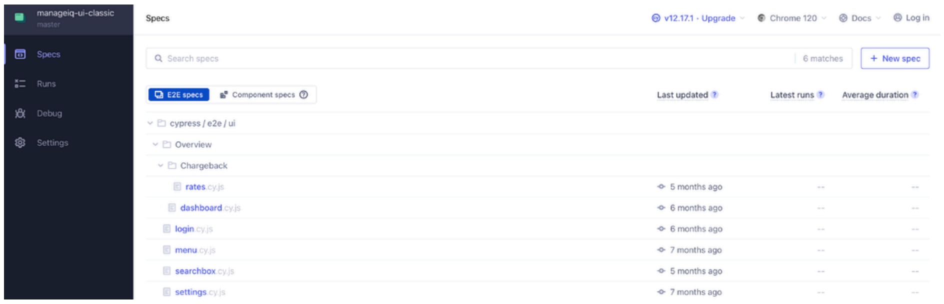

manageiq-ui-classic (master)
v12.17.1 · Upgrade
Docs
Log in
Welcome to Cypress!
Review the differences between each testing type →
E2E Testing
Build and test the entire experience of your application from end-to-end to ensure each flow matches your expectations.
Configured
Component Testing
Build and test your components from your design system in isolation in order to ensure each state matches your expectations.
Not Configured

Select E2E testing and then select the browser to open the tests on. Chrome is the most commonly used so it is best to write your tests for Chrome and then run your tests on all browsers to ensure compatibility with all browsers.


manageiq-ui-classic (master) > E2E Testing
v12.17.1 · Upgrade
Docs
Log in
Choose a browser
Choose your preferred browser for E2E testing.
Chrome v120
Edge v120
Electron v106
Firefox v120
Start E2E Testing in Chrome
Switch testing type

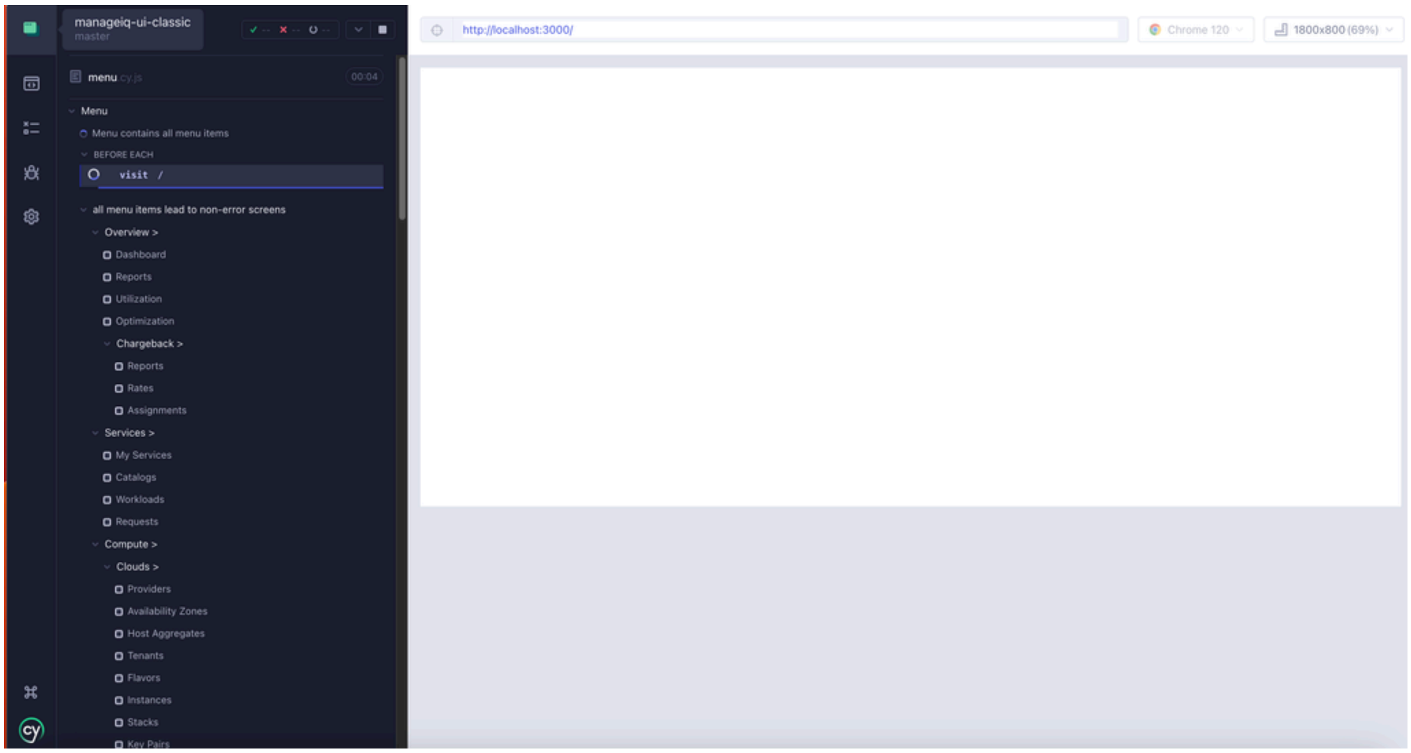
When you select a browser it will open this page in that selected browser. The file structure here is the same as the file structure in the

ManageIQ-UI-Classic code. Each file represents a separate spec.



	Last updated	Latest runs	Average duration
cypress / e2e / ui			
Overview			
Chargeback			
rates.cy.js	5 months ago	--	--
dashboard.cy.js	6 months ago	--	--
login.cy.js	6 months ago	--	--
menu.cy.js	7 months ago	--	--
searchbox.cy.js	5 months ago	--	--
settings.cy.js	7 months ago	--	--

By clicking on one of the specs it runs the spec. From this screen there is a section on the right which shows the spec running in real time on the browser and a section on the right which shows the individual subheadings and tests in the spec. The subheadings are just used to organize related tests and create common functions for them using things like beforeEach, they are not tests themselves however. The tests are represented by the items that have a checkbox beside them on the left section of the page. In the top bar you can also pause and rerun the spec as well as see how many of the tests in the spec passed and failed.



```
menu.cy.js                                   00:04
Menu
  Menu contains all menu items
  BEFORE EACH
    visit /
  all menu items lead to non-error screens
    Overview >
      Dashboard
      Reports
      Utilization
      Optimization
      Chargeback >
        Reports
        Rates
        Assignments
    Services >
      My Services
      Catalogs
      Workloads
      Requests
    Compute >
      Clouds >
        Providers
        Availability Zones
        Host Aggregates
        Tenants
        Flavors
        Instances
        Stacks
        Key Pairs
```

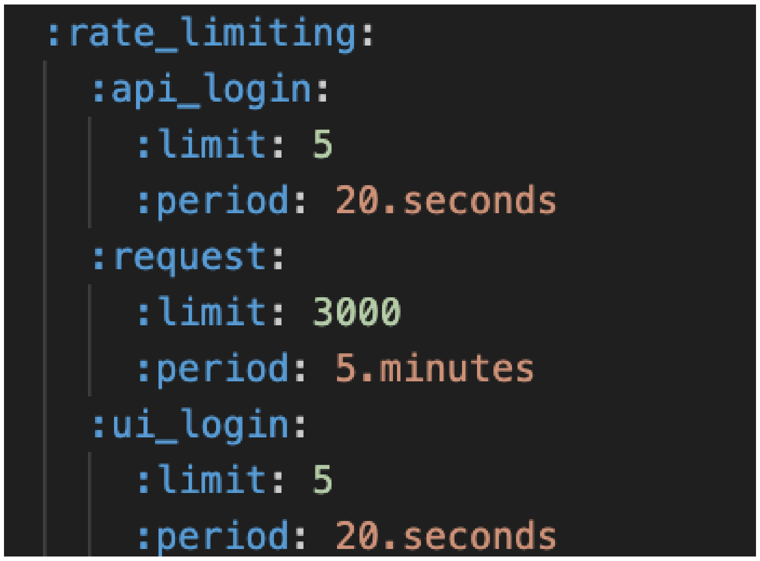
To create a section in the spec for related tests you can use:
describe('subsection title', () => { … }

To create a test in the spec you can use:

it('test name', () => { ... }

6) Final set up steps before writing and running tests:

- 1) When writing and running Cypress tests it will often require making a lot of requests in a short period of time. By default this will cause a "Too Many Requests" error. In order to prevent this navigate to the file ManageIQ/config/settings.yml. Under the section for :rate_limiting: and :request: change the value for :limit: to any large number, example: 3000.



```
:rate_limiting:
  :api_login:
    :limit: 5
    :period: 20.seconds
  :request:
    :limit: 3000
    :period: 5.minutes
  :ui_login:
    :limit: 5
    :period: 20.seconds
```

- 2) In order to prevent the Cypress test UI from crashing due to being very large, for example the menu.cy.js file which causes crashes since it requires visiting every page in the UI, the cypress.config.js file contains a value:
  numTestsKeptInMemory: 0

  This means that while debugging your tests you can not select a specific step in the test and view its snapshot history. This may make debugging difficult so to avoid this you can comment out this line or change the number to anything greater than 0 to help with debugging tests.

Important Files

Before writing new tests there are some important common files used in Cypress that you should be aware of.

1) cypress.config.js - This file contains various Cypress configuration settings for the Cypress tests. This includes the base url where the app is located, the viewport size for the Cypress test UI, the number of tests history to store in memory and settings for recording videos of the Cypress tests.

2) cypress/support/e2e.js - This file contains the import statements for all Cypress commands and assertions so that they can be called in the Cypress e2e spec files such as cy.login() which is an example of a command that will visit the base url and log in to the account specified by the command arguments. The e2e.js file also contains any logic that can run in certain circumstances. For example:

This function runs whenever Cypress encounters a "uncaught:exception" error. This function will check the error message and in certain situations it returns false, meaning that the test is not a fail due to this error. This was created due to certain browsers throwing various exceptions that do not affect the UI or the tests but were causing the Cypress tests to fail since Cypress thought these we real errors. If the error message does not meet these any of these circumstances then the function returns nothing and the Cypress test will fail with whatever error Cypress has run into.

3) cypress/support/assertions - This folder contains files that are used to write any common Cypress test assertions that you would want to reuse across various Cypress tests. This allows the reduction in the lines of code required for the spec files. For example:

This function is called using cy.expect_text and you can pass a Cypress element string and text string to verify that the UI element does have

text equal to the desired text. Example:
`cy.expect_text(':nth-child(1) > :nth-child(1) > .cell > .bx--front-line', 'CPU');`

4) cypress/support/commands - This folder contains files that are used to write any common Cypress commands that you would want to reuse across various Cypress tests. These are commands that can be used to login, navigate menus, interact with or retrieve table elements, use search bars, interact with toolbars, etc. Essentially any common command that you would want to repeat in different Cypress test files can be added here in order to reduce the lines of code in the spec files.

*** NOTE: If distinguishing between an assertion or command is not clear just think of assertions as a "test case command" that can be used in the Cypress spec files to test a condition (like if the text in an element matches the expected text) and think of commands as something you would want to do in the UI that is not test related (like navigating the UI, clicking on a button, reading table values, etc.).

*** NOTE: The assertions and commands should be listed on this README: https://github.com/ManageIQ/manageiq-ui-classic/blob/master/cypress/README.md along with a short description of what it does and how to use it. However, some of these may be out of date and no longer functional.

Writing New Tests

These are some things to keep in mind when writing new tests.

1) Avoid Dependent Tests: Currently we have no way of resetting the db in between tests or adding data into the db for a test. This means that if you wanted to test the edit form or delete button on a record you must first use the add form to create the record. Also, instead of creating 3 separate tests for add, edit and delete it is better to combine them into 1 test, see this file as an example:
https://github.com/ManageIQ/manageiq-ui-classic/blob/master/cypress/e2e/ui/Overview/Chargeback/rates.cy.js.
If you write them as 3 separate tests then 1 of the tests failing can lead to the other tests failing as well since they are dependent on each other. However if you write them all as 1 test it would only result in 1

failing test and this allows us to prevent having tests that depend on each other to pass.

2) File Structure: When adding new test files try to create them in a file structure that matches the UI. For example in the UI we have Overview > Chargeback > Rates on the side navigation menu. In the Cypress files this looks like: https://github.com/ManageIQ/manageiq-ui-classic/blob/master/cypress/e2e/ui/Overview/Chargeback/rates.cy.js. At the top level of Cypress we have an Overview folder for all the Overview tests, then a Chargeback folder for all the Chargeback tests. This contains the rates tests file since that is a page in the Chargeback menu. For vert large tests files you can use a folder for the menu items which contain multiple test files. For example this would look like: Overview/Chargeback/Rates/rates1.cy.js and Overview/Chargeack/Rates/rates2.cy.js.

3) No Provider Data: As of right now we have no way to seed real provider to the database. This prevents us from writing tests for anything provider related in the UI. However, there is still lots of tests that can be written for non-provider data. This issue: https://github.com/ManageIQ/manageiq-ui-classic/issues/8859 tracks some of the pages that can be tested without any provider data. This is not all the pages that can be tested but it is what we need to do to finish Phase 2. The rest of these pages can be finished in Phase 3.

4) Default Tests: Try to create base line tests for each spec file. For example in https://github.com/ManageIQ/manageiq-ui-classic/blob/master/cypress/e2e/ui/Overview/Chargeback/rates.cy.js this would be the tests that check that the page loads properly and that the default rates are in the table and contain the correct values.

5) Test all Browsers: Before creating a PR for your new Cypress tests make sure the tests are passing on Chrome, Edge and Firefox first.