

PA1417: Basic System Verification

Tutorial 3: GUI Testing with Cypress



BLEKINGE
INSTITUTE OF
TECHNOLOGY

Julian Frattini

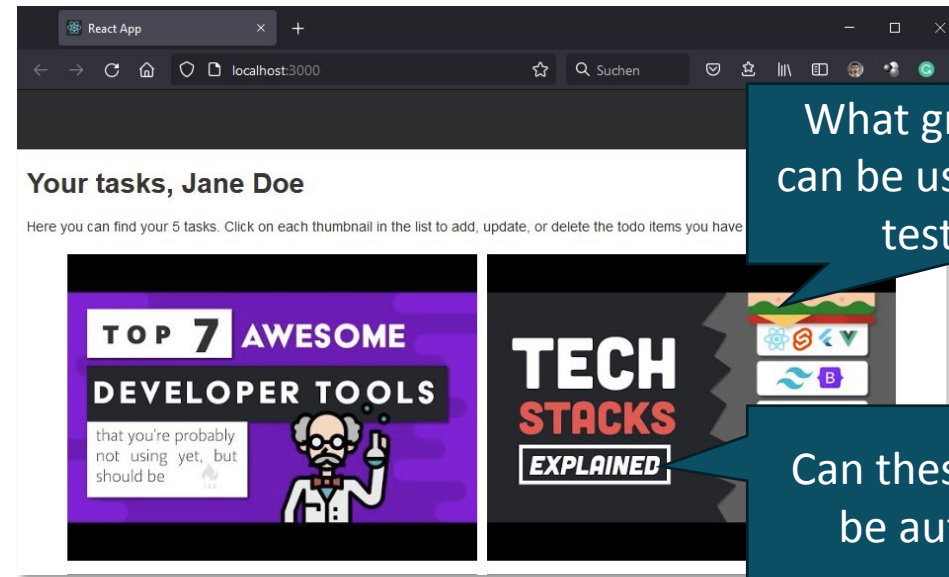
Challenges of Test Automation

When testing (parts of the) **backend**, the interface of each function is clearly defined.

```
11 def get_user_by_email(self, email: str):
12     """Given a valid email address of an existing account, return the user object contained in the
13     database associated
14     to that user. For now, do not assume that the email attribute is unique. Additionally print a
15     warning message containing the email
16     address if the search returns multiple users.
17
18     parameters:
19         email -- an email address string
20
21     returns:
22         user - user object
23         None - if no user is found
24
25     raises:
26         ValueError - if email is not a string
27         Exception - if email is empty
28
29     """
30     if not re:
31         raise
32     try:
33         users = self.get_users(email)
34         if len(users) > 1:
35             print(f"Warning: Multiple users found for email {email}")
36         elif len(users) == 1:
37             return users[0]
38         else:
39             return None
40     except Exception as e:
41         print(f"Error: {e}")
42         raise
```

```
> python .\main.py
Connecting to collection user on MongoDB at url mongodb://localhost:27017
Connecting to collection task on MongoDB at url mongodb://localhost:27017
Connecting to collection video on MongoDB at url mongodb://localhost:27017
Connecting to collection todo on MongoDB at url mongodb://localhost:27017
Map[<Rule '/users/create' (POST, OPTIONS) -> user_blueprint.create_user>,
<Rule '/tasks/create' (POST, OPTIONS) -> task_blueprint.create>,
<Rule '/todos/create' (POST, OPTIONS) -> todo_blueprint.create>,
<Rule '/users/all' (GET, HEAD, OPTIONS) -> user_blueprint.get_users>,
<Rule '/populate' (POST, OPTIONS) -> populate>,
<Rule '/' (GET, HEAD, OPTIONS) -> ping>,
<Rule '/users/byemail/<email>' (GET, HEAD, OPTIONS) -> user_blueprint.get_user_by_email>,
<Rule '/tasks/ofuser/<id>' (GET, HEAD, OPTIONS) -> task_blueprint.get_tasks_of_user>,
<Rule '/tasks/byid/<id>' (GET, HEAD, PUT, OPTIONS) -> task_blueprint.get>,
<Rule '/todos/byid/<id>' (DELETE, GET, HEAD, PUT, OPTIONS) -> todo_blueprint.get_todo>,
<Rule '/static/<filename>' (GET, HEAD, OPTIONS) -> static>,
<Rule '/users/<id>' (GET, HEAD, PUT, OPTIONS) -> user_blueprint.get_user>]]
* Serving Flask app 'todoapp' (lazy loading)
* Environment: production
```

When testing (parts of the) **frontend**, the interface of each function requires the identification of elements in a user interface.



Ground Truth for Test Design

To test the backend, we used docstrings as our oracle. For UI tests, we need to resort to a different ground truth documentation.

Backend Tests

```
def hasAttribute(obj: dict, attribute: str):  
    """Check whether a given dict contains a specific attribute  
  
    attributes:  
        obj -- a dict object  
        attribute -- the key which potentially occurs in the obj dict  
  
    returns:  
        True -- if the dict contains the attribute as a key  
        False -- if the dict does not contain the attribute as a key or is None  
    """  
    return (attribute in obj)
```

Frontend Tests

ID	R3UC1
Primary Actor	End user
Preconditions	-
Main Success Scenario	<ol style="list-style-type: none">1. When the user enters the website, the system prompts the user to enter his/her credentials.2. When the user enters a valid email-password combination and clicks "Login", the system authenticates the user.
End Condition	The user is forwarded to an overview of the tasks associated to him/her.
Alternative Scenarios	<ol style="list-style-type: none">2.b When the email is not yet registered, the system prompts the user to signup instead.2.c When the password is incorrect, the system prompts the user to try again.

4-step Test Design Technique

Application to a Use Case

Quiz: Identify actions, expected outcomes, and conditions from use cases.

ID	R3UC1
Primary Actor	End user
Preconditions	-
Main Success Scenario	<ol style="list-style-type: none"> 1. When the user enters the website, the system prompts the user to enter his/her credentials. 2. When the user enters a valid email-password combination and clicks "Login", the system authenticates the user.
End Condition	The user is forwarded to an overview of the tasks associated to him/her.
Alternative Scenarios	<p>2.b When the email is not yet registered, the system prompts the user to signup instead.</p> <p>2.c When the password is incorrect, the system prompts the user to try again.</p>

The quiz will be available after the lecture at
https://docs.google.com/forms/d/e/1FAIpQLSdTy_7kWzuyFLENkQ3dHDF09BxwEt2IObB2AN_Dw3wondRGtg/viewform?usp=sf_link

4-step Test Design Technique

Application to a Use Case

Quiz: Identify actions, expected outcomes, and conditions from use cases.

ID	R3UC1
Primary Actor	End user
Preconditions	-
Main Success Scenario	<ol style="list-style-type: none"> When the user enters the website, the system prompts the user to enter his/her credentials. When the user enters a valid email-password combination and clicks "Login", the system authenticates the user.
End Condition	The user is forwarded to an overview of the tasks associated to him/her.
Alternative Scenarios	<p>2.b When the email is not yet registered, the system prompts the user to signup instead.</p> <p>2.c When the password is incorrect, the system prompts the user to try again.</p>

ID	-	User entering the website
1.1	-	The system prompts the user to enter his/her credentials

ID	email	Password	User clicks "Login"
2.1	registered	correct	system authenticates the user and forwards user to task overview
2.2	registered	incorrect	system prompts to try again
2.3	not registered	-	system prompts to signup

Testing Frameworks

Automating GUI Tests

There are several frameworks that allow the implementation of automatic GUI tests, for example:

1. Selenium (<https://www.selenium.dev/>)
2. Cypress (<https://www.cypress.io/>)

Test Level

Cypress supports both testing paradigms:

- **Component Tests:** unit tests of UI elements
- **E2E (end-to-end) Tests:** functional/scenario tests of the whole system

In assignment 4, we will focus on E2E tests.

Setting up Cypress

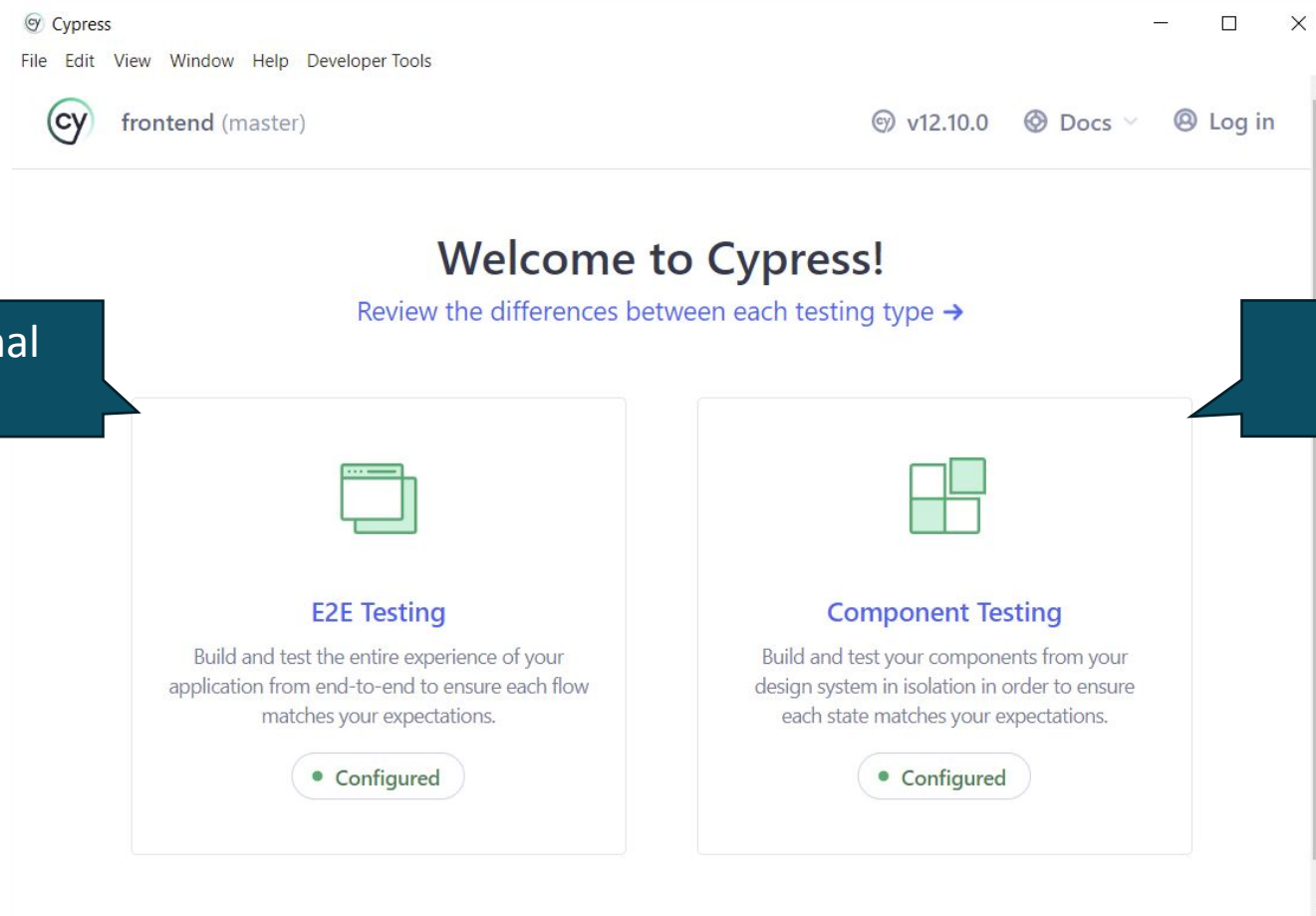
Requirements: the npx package must be installed (npm install npx), which usually comes automatically with npm

Setup:

1. Ensure that the system is running (either locally or via Docker)
2. Within the *frontend* folder, install cypress via `npm install -D cypress`
3. Open the cypress test runner via `npx cypress open`

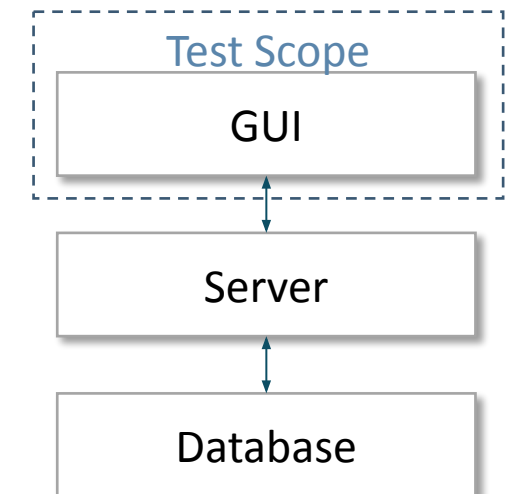
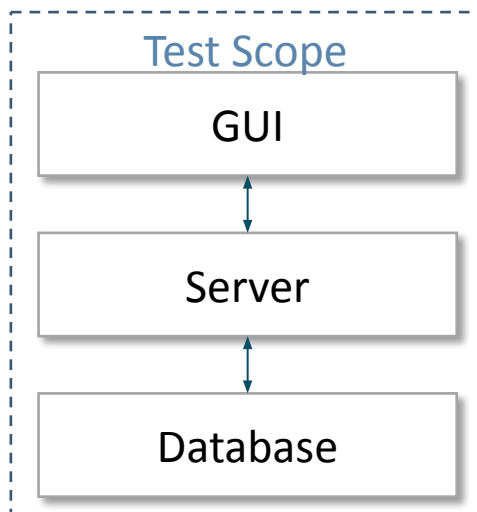
Step 2 has already been prepared in the current system: when you set up the system via `npm install`, cypress was included.

Cypress Opening Window



Scenario/functional
tests

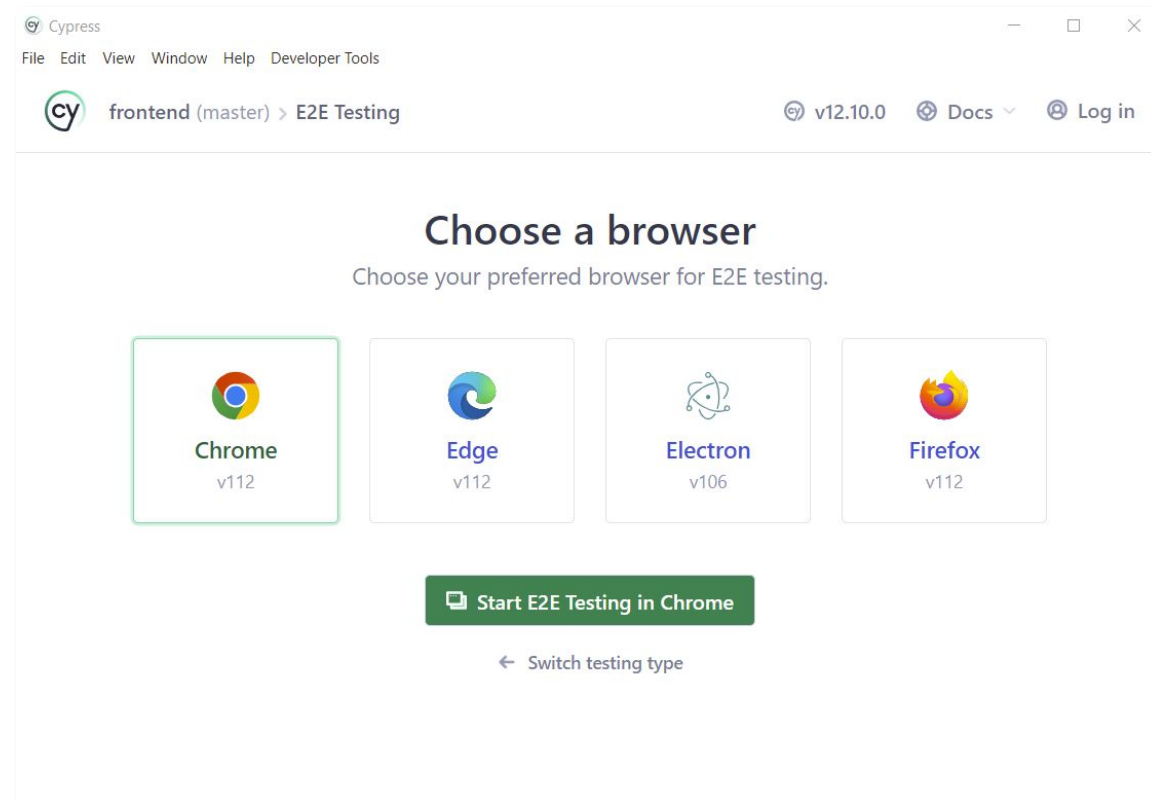
Unit Tests on GUI
Elements



Selecting a Test Browser



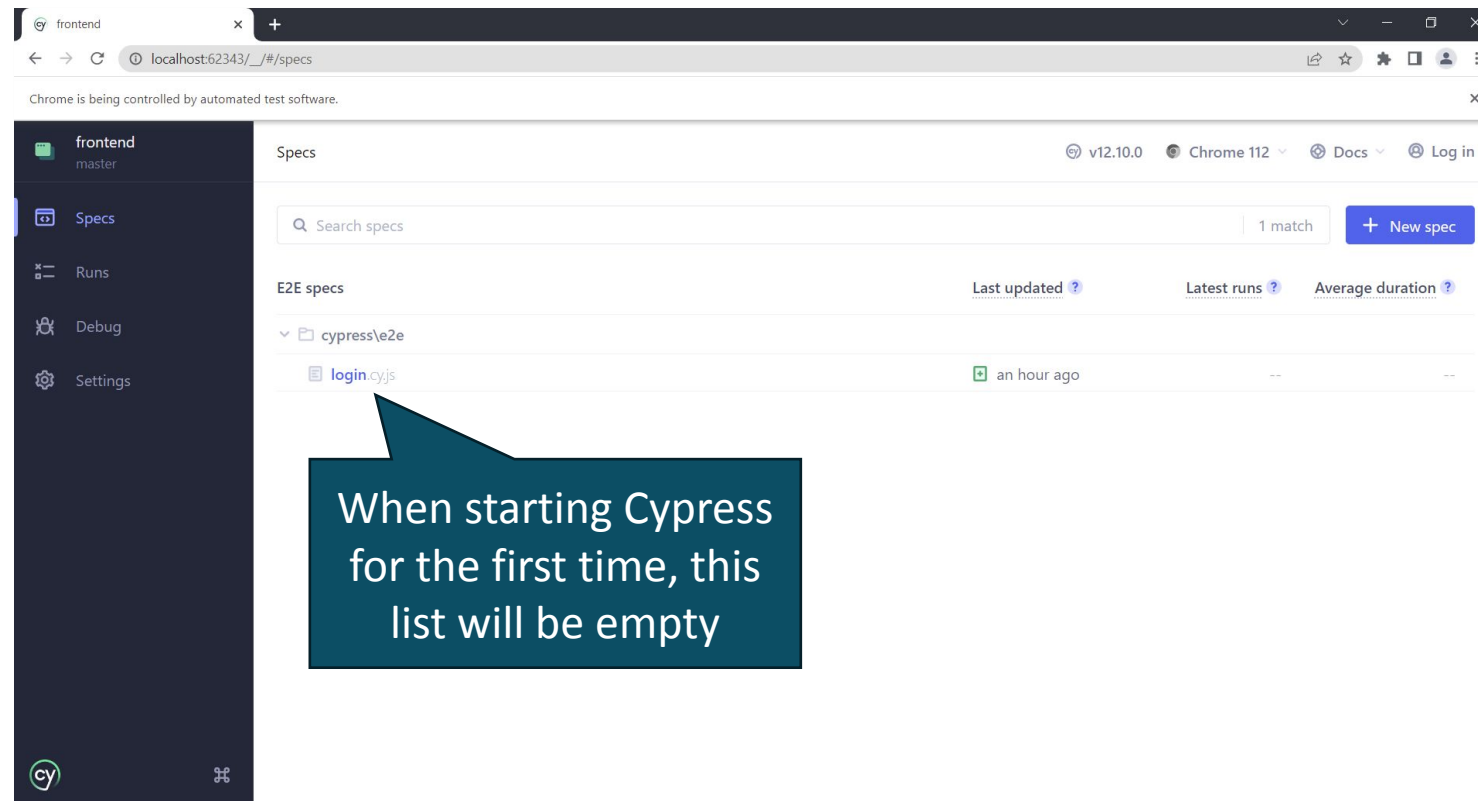
BLEKINGE
INSTITUTE OF
TECHNOLOGY



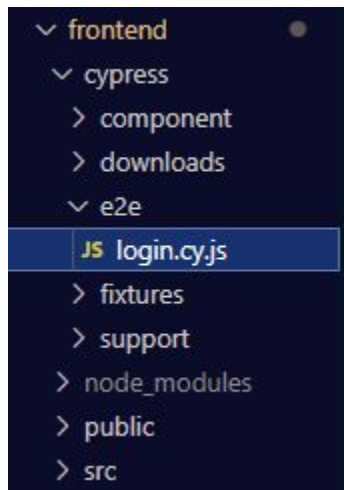
The Cypress Test Runner



BLEKINGE
INSTITUTE OF
TECHNOLOGY



Creating a new Spec



```
frontend > cypress > e2e > JS login.cy.js > ...
```

```
1 describe('Logging into the system', () => {  
2   it('starting out on the landing screen', () => {  
3     // enter the main main page  
4     cy.visit('http://localhost:3000')  
5  
6     // make sure th  
7     cy.get('h1')  
8     .should('contain.text', 'Login')  
9   })  
10 })
```

Name the test suite

Name a test case

Execute Cypress
commands via **cy**.

Obtain a GUI element

Assert a property about the
element with **should(...)**

Cypress Commands

Cypress offers a variety of commands to simulate user interaction with a website. Some of the most important being:

- **get**: get one or more DOM elements by selector or alias (similar to CSS)
- **contains**: get the DOM element containing the given text
- **find**: get the descendent DOM elements of a specific selector
- **should**: create an assertion
- **trigger**: trigger an event on a DOM element

... and many more

<https://docs.cypress.io/api/table-of-contents>

E2E Test with Cypress

Test Case Execution



BLEKINGE
INSTITUTE OF
TECHNOLOGY

List of specifications, the test cases they contain, and the steps that these contain

Execution summary

Manual rerun

The screenshot shows the Cypress test runner interface. On the left, a dark sidebar contains a list of test cases under the heading "login.cjs". The selected test case is "Logging into the system", which has a green checkmark and a duration of 00:02. Below this, the "TEST BODY" is visible, showing three steps: 1. visit http://localhost:3000, 2. get hl, and 3. -assert expected <hl> to contain text Login. On the right, a light-colored panel displays the "Execution summary" for the selected test case. It shows a green checkmark, a red 'x' for a failed assertion, and a "Manual rerun" button. Below the summary, a "Login" form is visible, with fields for "Email Address" and "Password". The "Password" field has a message: "Password functionality not supported yet. Proceed the login/signup with the email address." and a "Login" button. At the bottom of the form, there is a link: "Have no account yet? Click here to sign up."

E2E Test with Cypress Playback



BLEKINGE
INSTITUTE OF
TECHNOLOGY

The screenshot shows the Cypress test runner interface. On the left, the 'Specs' panel displays a test suite named 'login.cjs' with a duration of 00:02. Under the 'Logging into the system' section, the 'TEST BODY' is visible, showing three steps: 1. 'visit http://localhost:3000', 2. 'get h1', and 3. 'assert expected <h1> to contain text Login'. The third step is highlighted. On the right, the browser window shows a 'Login' page with an 'Email Address' input field, a 'Password' input field with a message 'Password functionality not supported yet. Proceed the login/signup with the email address.', and a 'Login' button. A 'DOM Snapshot' button is visible at the bottom right of the browser window. A dark blue callout box with white text points to the 'assert' step in the test body, stating: '... to see which element Cypress is referring to.' Another dark blue callout box with white text points to the 'assert' step, stating: 'Hover over a step ...'. The browser's address bar shows 'localhost:3000/_/#/specs/runner?file=cypress/e2e/login.cjs'.

E2E Test with Cypress

Defining multiple Test Cases

```
frontend > cypress > e2e > JS login.cy.js > ...
1  describe('Logging into the system', () => {
2    it('starting out on the landing screen', () => {
3      // enter the main main page
4      cy.visit('http://localhost:3000')
5
6      // make sure the landing page contains a header with "login"
7      cy.get('h1')
8        .should('contain.text', 'Login')
9    })
10
11  it('email field enabled', () => {
12    // enter the main main page
13    cy.visit('http://localhost:3000')
14
15    // find the html element with the id email
16    cy.get('.inputwrapper #email')
17      .should('be.enabled')
18  })
19 }
```

E2E Test with Cypress

Defining multiple Test Cases



BLEKINGE
INSTITUTE OF
TECHNOLOGY

The image shows a Cypress test runner interface on the left and a web application on the right. The test runner shows a test suite named "login.cy.js" with a spec "Logging into the system". The test body includes three steps: 1. visit http://localhost:3000, 2. get .inputwrapper #email, and 3. -assert expected <input#email> to be enabled. The application on the right is a login page with fields for "Email Address" and "Password", a "Login" button, and a link to "sign up".

frontent x +

localhost:3000/_/#/specs/runner?file=cypress/e2e/login.cy.js

Chrome is being controlled by automated test software.

Specs

login.cy.js 277ms

Logging into the system

- starting out on the landing screen
- email field enabled

TEST BODY

```
1 visit http://localhost:3000
2 get .inputwrapper #email
3 -assert expected <input#email> to be enabled
```

http://localhost:3000/ Chrome 112 1000x660 (93%)

Login

Email Address

Password

Password functionality not supported yet. Proceed the login/signup with the email address.

Login

Have no account yet? Click here to sign up.

The new test case
appears once the
specification is saved

E2E Test with Cypress

Test Case Setup



BLEKINGE
INSTITUTE OF
TECHNOLOGY

```
frontend > cypress > e2e > JS login.cy.js > ...
1  describe('Logging into the system', () => {
2  it('starting out on the landing screen', () => {
3    // enter the main main page
4    cy.visit('http://localhost:3000')
5
6    // make sure the landing page contains a header with "login"
7    cy.get('h1')
8      .should('contain.text', 'Login')
9  })
10
11 it('email field enabled', () => {
12   // enter the main main page
13   cy.visit('http://localhost:3000')
14
15   // find the html element with the id email
16   cy.get('.inputwrapper #email')
17     .should('be.enabled')
18 })
19 })
```

```
frontend > cypress > e2e > JS login.cy.js > ...
1  describe('Logging into the system', () => {
2    beforeEach(function () {
3      // enter the main main page
4      cy.visit('http://localhost:3000')
5    })
6
7    it('starting out on the landing screen', () => {
8      // make sure the landing page contains a header with "login"
9      cy.get('h1')
10        .should('contain.text', 'Login')
11    })
12
13    it('email field enabled', () => {
14      // find the html element with the id email
15      cy.get('.inputwrapper #email')
16        .should('be.enabled')
17    })
18  })
```


Identifying GUI Elements

Black-box vs. White-box approach

A significant part of Cypress test code (or any code for automatic GUI tests) is the identification of GUI/DOM elements. There are two approaches to this:

1. **Imperative** (white-box): clearly specify the element to identify based on its properties in the code
2. **Declarative** (black-box): describe how to identify certain elements the same way a user would search for them

Imperative

```
cy.get('.inputwrapper #email')
```

src/Components/LoginForm.js

```
<div className='inputwrapper'>  
  <label>Email Address</label>  
  <input type='text' id='email' name='email' onChange={e =>  
</div>
```

Declarative

```
cy.contains('div', 'Email Address')  
  .find('input[type=text]')
```

Simulate how a user
would search for the
respective DOM element

E2E Test with Cypress

Failing Test Cases



BLEKINGE
INSTITUTE OF
TECHNOLOGY

```
frontend > cypress > e2e > JS login.cy.js > describe('Logging into the system') callback >
1 describe('Logging into the system', () => {
2   beforeEach(function () {
3     // enter the main main page
4     cy.visit('http://localhost:3000')
5   })
6
7   it('starting out on the landing screen', () => {
8     // make sure the landing page contains a header with "login"
9     cy.get('h1')
10    .should('contain.text', 'Login')
11  })
12
13  it('email field enabled', () => {
14    // find the html element with the id email
15    cy.get('.inputwrapper #email')
16    .should('be.disabled')
17  })
18 })
```

This test case fails on
purpose (for
demonstration)

E2E Test with Cypress

Failing Test Cases



BLEKINGE
INSTITUTE OF
TECHNOLOGY

The screenshot shows the Cypress test runner interface. On the left, the 'Specs' panel displays a test file named 'login.cy.js'. The test suite 'email field enabled' is shown with a failure icon. The test body includes a 'visit' command and an 'assert' command that has failed. The error message is 'AssertionError: Timed out retrying after 4000ms: expected <input#email> to be disabled'. The code snippet shows the test logic: finding the email input and asserting it is disabled. On the right, the browser window shows the 'Login' page. The 'Email Address' field is active, and the 'Password' field contains the message 'Password functionality not supported yet. Proceed the login/signup with the email address.' The 'Login' button is visible at the bottom of the form.

E2E Test with Cypress

Cypress Fixtures for Setup

Define Variables

```
let uid  
let name
```

Read dummy data
located in *fixtures/*

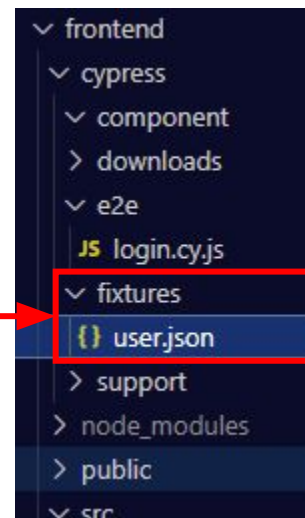
```
cy.fixture('user.json')
```

Issue a call to the
backend

```
cy.request({  
  method: 'POST',  
  url: 'http://localhost:5000/users/create',  
  form: true,  
  body: user  
})
```

Store the returned
variables

```
}).then((response) => {  
  uid = response.body._id.$oid  
  name = user.firstName + ' ' + user.lastName  
})
```



```
{  
  "email": "mon.doe@gmail.com",  
  "firstName": "Mon",  
  "lastName": "Doe"  
}
```

E2E Test with Cypress

Cypress Fixtures for Setup

```
1 describe('Logging into the system', () => {
2   // define variables that we need on multiple occasions
3   let uid
4   let name
5
6   before(function () {
7     // create a fabricated user from a fixture
8     cy.fixture('user.json')
9       .then((user) => {
10       cy.request({
11         method: 'POST',
12         url: 'http://localhost:5000/users/create',
13         form: true,
14         body: user
15       }).then((response) => {
16         uid = response.body._id.$oid
17         name = user.firstName + ' ' + user.lastName
18       })
19     })
20   })
21
22   beforeEach(function() {
23     // enter the main main page
24     cy.visit('http://localhost:3000')
25   })
```

Cleanup method

```
52 after(function () {
53   // clean up by deleting the user from the database
54   cy.request({
55     method: 'DELETE',
56     url: `http://localhost:5000/users/${uid}`
57   }).then((response) => {
58     cy.log(response.body)
59   })
60 })
61 })
```


E2E Test with Cypress

```

1 describe('Logging into the system', () => {
2   // define variables that we need on multiple occasions
3   let uid
4   let name
5
6   before(function () {
7     // create a fabricated user from a fixture
8     cy.fixture('user.json')
9       .then((user) => {
10       cy.request({
11         method: 'POST',
12         url: 'http://localhost:5000/users/create',
13         form: true,
14         body: user
15       }).then((response) => {
16         uid = response.body._id.$oid
17         name = user.firstName + ' ' + user.lastName
18       })
19     })
20   })
21
22   beforeEach(function() {
23     // enter the main main page
24     cy.visit('http://localhost:3000')
25   })
26
27   after(function () {
28     // clean up by deleting the user from the database
29     cy.request({
30       method: 'DELETE',
31       url: `http://localhost:5000/users/${uid}`
32     }).then((response) => {
33       cy.log(response.body)
34     })
35   })
36 })

```

```

27 it('starting out on the landing screen', () => {
28   // make sure the landing page contains a header with "login"
29   cy.get('h1')
30     .should('contain.text', 'Login')
31 })
32
33 it('login to the system with an existing account', () => {
34   // detect a div which contains "Email Address", find the input and type
35   // declarative
36   cy.contains('div', 'Email Address')
37     .find('input[type=text]')
38     .type('mon.doe@gmail.com')
39   // alternative, imperative way of detecting that input field
40   // cy.get('.inputwrapper #email')
41   //   .type('mon.doe@gmail.com')
42
43   // submit the form on this page
44   cy.get('form')
45     .submit()
46
47   // assert that the user is now logged in
48   cy.get('h1')
49     .should('contain.text', 'Your tasks, ' + name)
50 })

```

Find the input form and enter the email address of the dummy user

Submit the input form

Assert that the user is now logged in

E2E Test with Cypress



BLEKINGE
INSTITUTE OF
TECHNOLOGY

The screenshot shows the Cypress test runner interface on the left and the application under test on the right. The browser window at the top shows the URL "localhost:3000/_/#/specs/runner?file=cypress/e2e/login.cy.js". The application interface displays a form titled "Your tasks, Mon Doe" with fields for "Title", "YouTube URL", and "Viewkey of a YouTube video (the part after /watch?v= in the". A "Create new Task" button is at the bottom. The Cypress test runner shows a test suite "login" with a spec "login.cy.js". The test body includes steps for visiting the URL, finding the email input, typing the email, submitting the form, and asserting the response. The test is currently running, as indicated by the "Debug" button and the "Before EACH" section.

GUI Testing Approach Summary

Our approach to testing has not changed:

1. **Test design:** transform a use case into one or more test cases (using the 4-step test design technique)
2. **Test implementation:** translate each test case into Cypress code
3. **Test execution and evaluation:** Run the tests and evaluate the test result

All that has changed is:

- **The ground truth/oracle:** now we use use cases instead of docstrings
- **The testing framework:** now we use Cypress instead of Pytest

