

Cypress

Установка и основы



Роман
Горицков



РОДИОН ГОРИЦКОВ

Lead QA Engineer



Цели занятия

- Узнаем, что такое Cypress и для чего он используется
- Научимся использовать Cypress для написания UI-тестов
- Изучим основные команды Cypress
- Напишем первые тесты и сделаем рефакторинг
- Добавим собственные кастомные команды

План занятия

- 1 [Что такое Cypress](#)
- 2 [Запуск Cypress](#)
- 3 [Структура проекта](#)
- 4 [Создаём первый тест](#)
- 5 [Добавляем второй тест](#)
- 6 [Добавляем третий тест](#)
- 7 [Custom commands](#)
- 8 [Итоги](#)
- 9 [Домашнее задание*](#)
- 10 [Дополнительные материалы*](#)

Что такое Cypress



1

Почему Cypress

Cypress — современный фреймворк для автотестов.

Поддерживает различные виды тестирования: как компонентные, интеграционные тесты, так и e2e

Плюсы:

- минимальное время на установку и настройку
- всё «из коробки», в том числе видео и скрины
- хорошая поддержка, документация и развитое сообщество
- удобный дебаггинг

Ограничения:

- только JS
- только одно окно браузера

Установка Cypress

За инструкцией обратимся на официальный сайт Cypress:

<https://docs.cypress.io/guides/getting-started/installing-cypress#System-requirements>

В новой папке создаём новый репозиторий и устанавливаем Cypress:

```
npm init  
npm install cypress
```

Запускаем Cypress:

```
npx cypress open
```

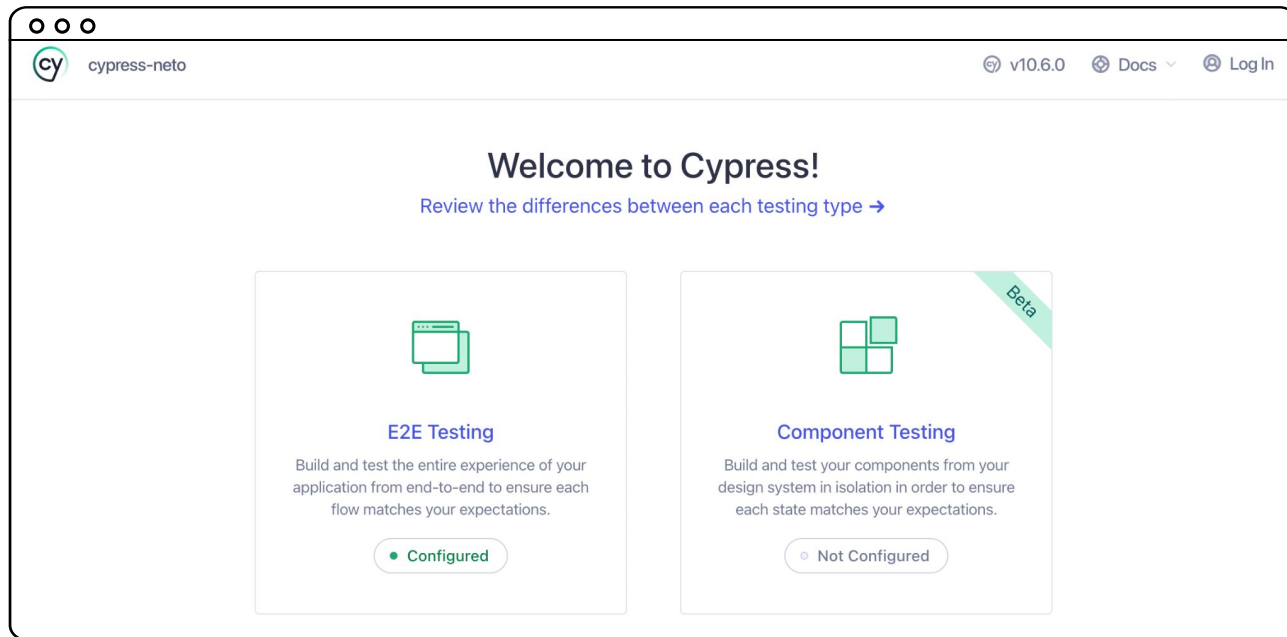
Запуск Cypress



2

Запуск Cypress

После первого запуска Cypress открывает приложение:



Выберем работу с E2E тестами и Cypress сгенерирует конфигурационный файл `cypress.config.json` и добавит его в наш проект

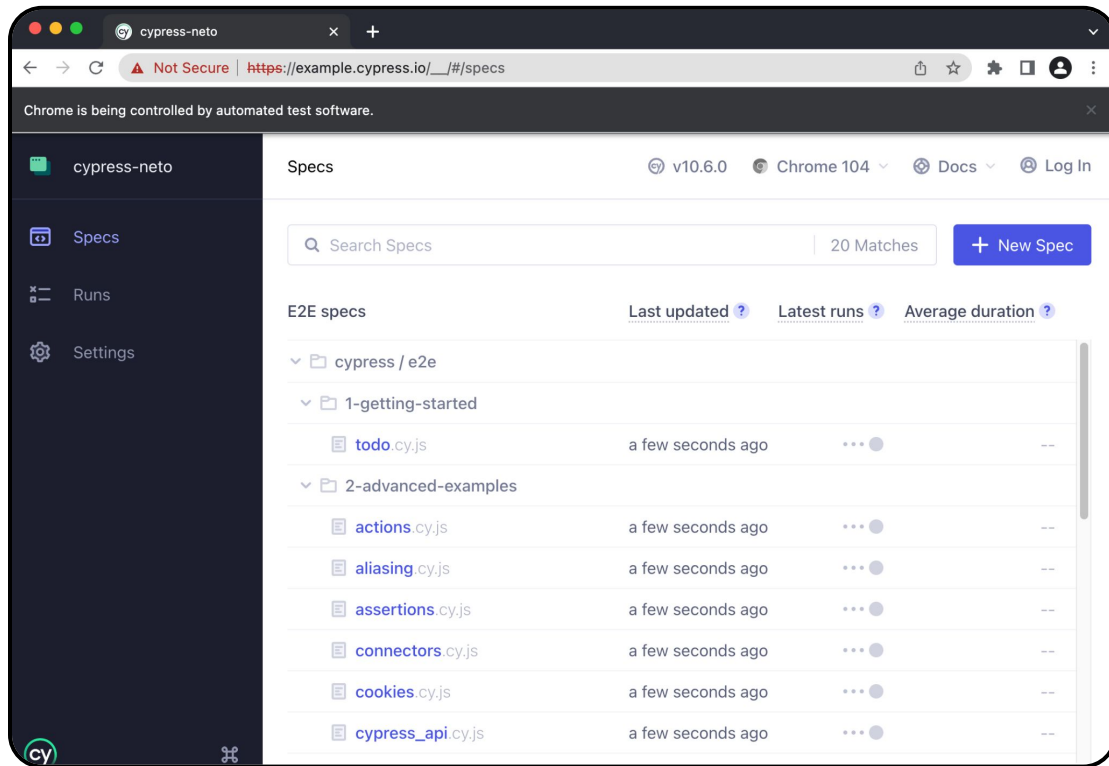
Запуск Cypress

Демонстрация функций:

- окно с тестами
- браузеры
- настройки, конфигурации
- документация

Запуск Cypress

Для примера Cypress предоставил уже готовые тесты с основными функциями. Попробуем их запустить:



Запуск Cypress

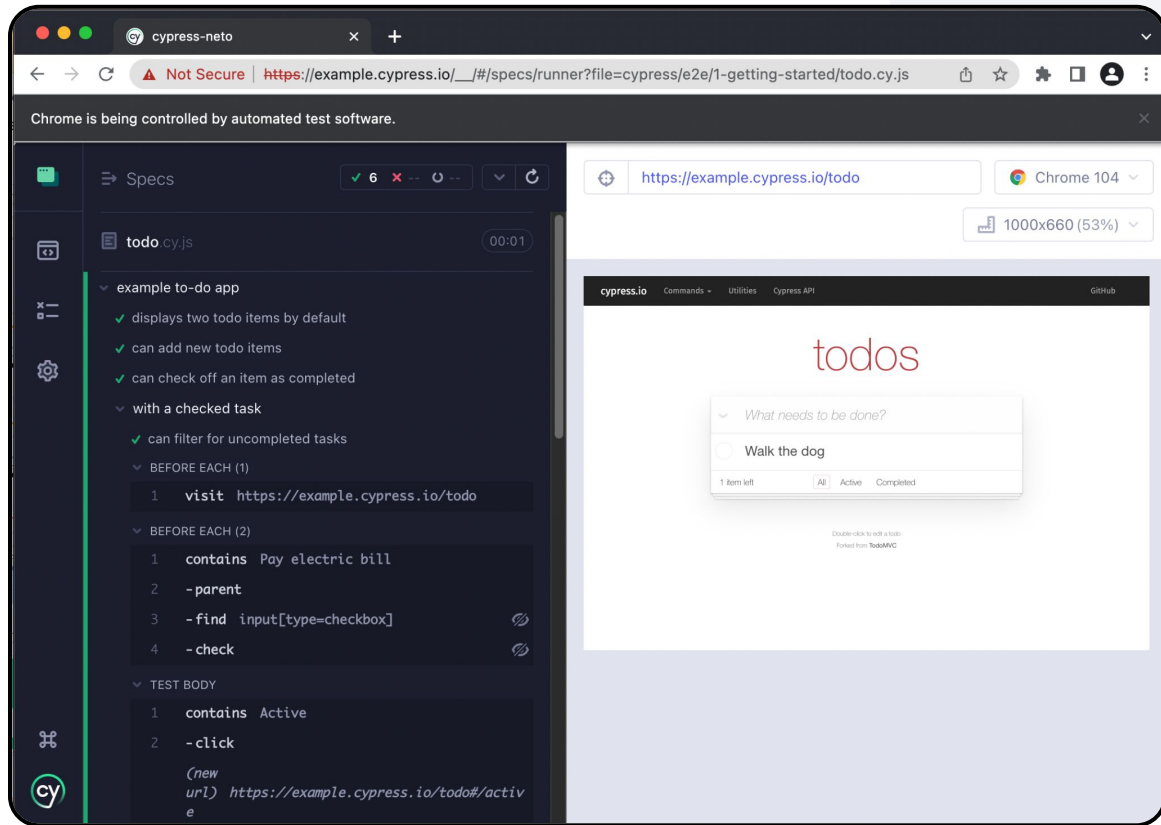
Демонстрируем запуск тестов и функции приложения Cypress:

- управление запуском тестов
- «путешествие» по исполнению тестов
- исследование страницы

Посмотрим подробнее

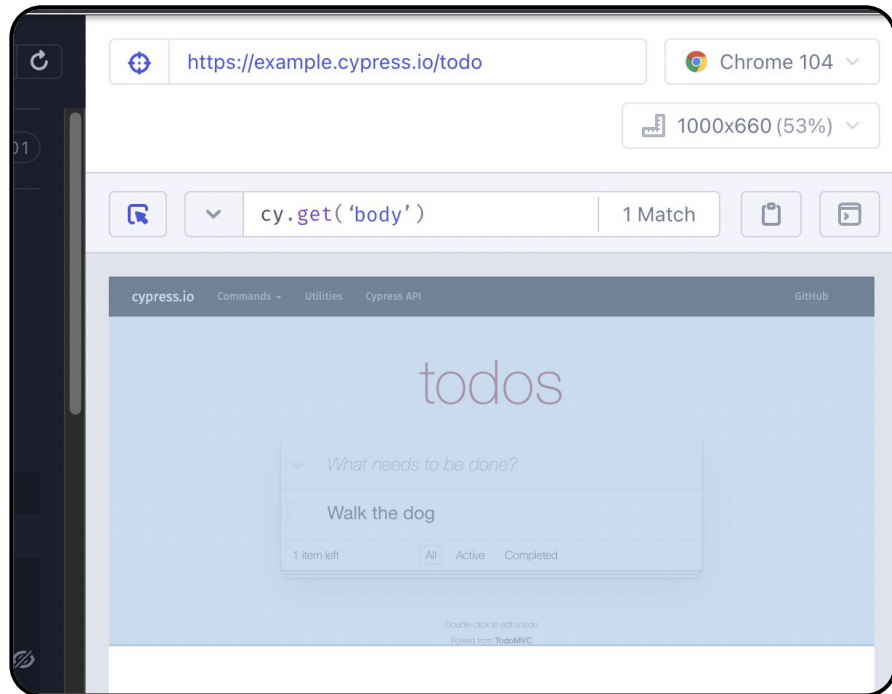
Запуск Cypress

«Путешествие» по исполнению тестов. Исследуем процесс теста пошагово. Изучаем состояние до и после выполнения команды:



Запуск Cypress

Cypress позволяет исследовать страницу и строить нужные селекторы для использования в тестах:



Структура проекта



3

Структура проекта Cypress

После первого запуска Cypress создаёт структуру проекта:

Разберём её подробнее

✓ CYPRESS-NETO

✓ cypress

> downloads

> e2e

> fixtures

> support

> videos

> node_modules

JS cypress.config.js

{} package-lock.json

{} package.json

Структура проекта Cypress

- Папка **fixtures** будет хранить файлы с данными, которые мы собираемся использовать в тестах
- Папка **e2e** предназначена для хранения тестовых файлов — спеков. Тестовые файлы должны иметь окончание **.cy.js**. Для удобства внутри можно добавлять больше папок, формируя структуру
- В папке **support** мы будем организовывать служебные функции, например, напомним собственные кастомные Cypress-команды

Структура проекта Cypress

Файл `cypress.config.js` содержит настройки проекта.

Сейчас он содержит базовую настройку, но мы будем его наполнять по ходу обучения.

[Документация по использованию](#)

Создаём первый тест

4

Удаляем заготовки

Нам не нужны заготовки, предоставленные Cypress — мы будем учиться писать тесты самостоятельно.

Удалим их из папки **e2e**

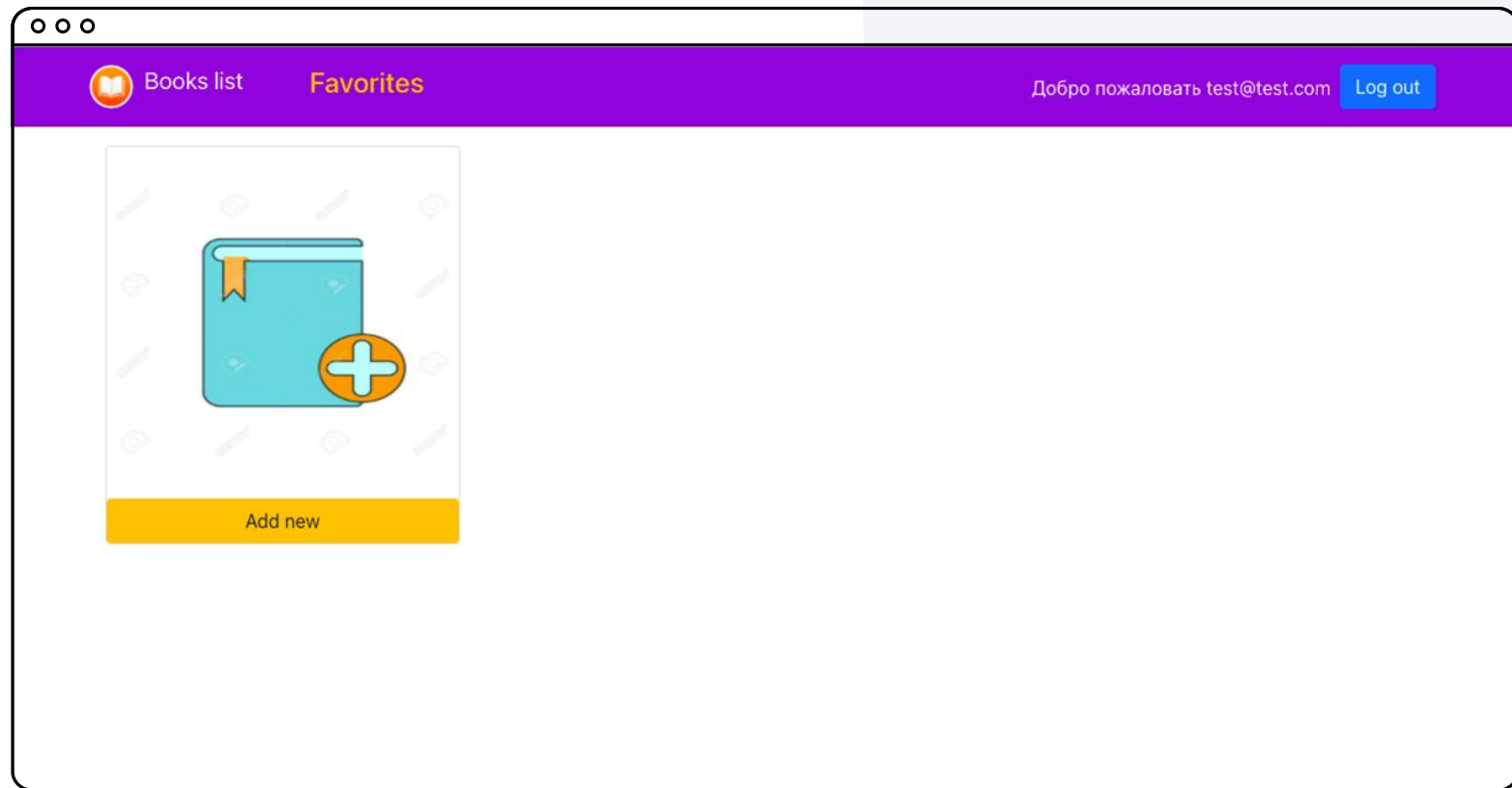
Задача

Студенты Нетологии пишут приложение для хранения и скачивания книг.

Мы с вами уже знаем, что тестирование нужно начинать как можно раньше. Приложение уже имеет интерфейс, с которым мы можем взаимодействовать и писать тесты.

Мы будем работать с приложением, которое развернём локально. Инструкции по настройке вы найдёте в репозитории с кодом к лекции

Демонстрация приложения



Задача

Для начала проверим сценарий сохранения книги в избранном

Сценарий:

- 1 Логинимся
- 2 Добавляем книгу в избранное
- 3 Проверяем наличие книги в избранном

Добавляем первый тест

Создадим файл login.cu.js в директории e2e.

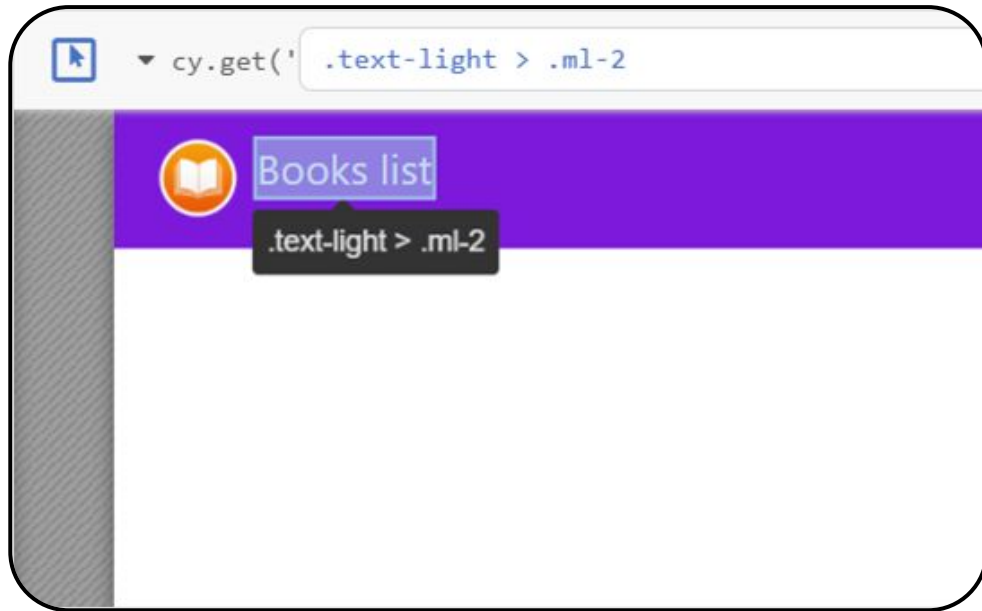
Добавим тест, используя следующую структуру:

```
it("Should open the main page", () => {  
  cy.visit("localhost:3000");  
})
```


Добавляем первый тест

Добавим assertion, используя Cypress для исследования страницы:

```
it("Should open the main page", () => {  
  cy.visit("localhost:3000");  
  cy.get('.text-light > .ml-2');  
})
```



Добавляем первый тест

Создадим файл login.cu.js в директории e2e.

Добавим тест, используя следующую структуру:

```
it("Should open the main page", () => {  
  cy.visit("localhost:3000");  
  cy.get('.text-light > .ml-2');  
})
```

Как вы думаете, насколько удобен этот селектор?

Добавляем первый тест

```
it("Should open the main page", () => {  
  cy.visit("localhost:3000");  
  cy.get('.text-light > .ml-2');  
})
```

Как вы думаете, насколько удобен этот селектор?

Абсолютно неудобен, ненадёжен и непонятен для чтения.

Cypress предлагает нам ещё одну опцию работы с селекторами:

```
it("Should open the main page", () => {  
  cy.visit("localhost:3000");  
  cy.contains('Books list');  
})
```

Роняем тест

```
it("Should open the main page", () => {  
  cy.visit("localhost:3000");  
  cy.contains('Bks list');  
})
```

Добавляем количество попыток запуска

Иногда могут встречаться нестабильно соединение или другие внезапные проблемы с приложением, которые вызывают ложное падение тестов.

Для таких случаев Cypress предоставляет возможность изменять количество попыток запуска теста, прежде чем посчитать его упавшим.

Добавим настройку в `cypress.config.js`:

```
{  
  "retries": 2  
}
```


Проверим текущую конфигурацию Cypress



```
experimentalFetchPolyfill: false
experimentalInteractiveRunEvents: false
experimentalSourceRewriting: false
experimentalStudio: false
experimentalSessionSupport: false
fileServerFolder: ""
fixturesFolder: "cypress/fixtures"
hosts: null
ignoreTestFiles: "*.hot-update.js"
includeShadowDom: false
integrationFolder: "cypress/integration"
modifyObstructiveCode: true
nodeVersion: "default"
numTestsKeptInMemory: 50
pageLoadTimeout: 60000
pluginsFile: "cypress/plugins"
port: null
projectId: null
redirectionLimit: 20
reporter: "spec"
reporterOptions: null
requestTimeout: 5000
responseTimeout: 30000
retries: 2
screenshotOnRunFailure: true
screenshotsFolder: "cypress/screenshots"
scrollBehavior: "top"
supportFile: "cypress/support"
taskTimeout: 60000
testFiles: "**/*.js"
trashAssetsBeforeRuns: true
userAgent: null
video: true
videoCompression: 32
videosFolder: "cypress/videos"
videoUploadOnPasses: true
viewportHeight: 660
viewportWidth: 1000
waitForAnimations: true
watchForFileChanges: true
}
```

Проверим количество попыток запуска

ooo

✖ Should open the main page 

Attempt 1 ✖

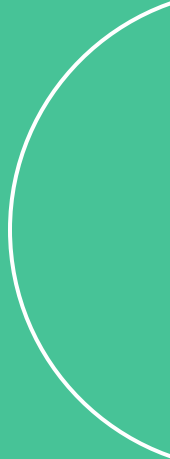
Attempt 2 ✖

Attempt 3 ✖

Добавляем остальные шаги и действия

```
it("Should successfully login", () => {  
  cy.visit("localhost:3000");  
  cy.contains('Log in').click();  
  cy.get("#mail").type("test@test.com");  
  cy.get("#pass").type("test");  
  cy.contains("Submit").click();  
  cy.contains("Добро пожаловать test@test.com").should("be.visible");  
})
```


Перерыв



Добавляем второй тест

5

Первый тест параллельно со вторым

Для удобства на время работы со вторым тестом пропустим первый тест, добавив **.skip** к **it**:

```
it.skip("Should successfully login", () => {  
  cy.visit("localhost:3000");  
  cy.contains('Log in').click();  
  cy.get("#mail").type("test@test.com");  
  cy.get("#pass").type("test");  
  cy.contains("Submit").click();  
  cy.contains("Добро пожаловать test@test.com").should("be.visible");  
})
```

Добавляем второй тест

```
it("Should not login with empty login", () => {  
  cy.visit("localhost:3000");  
  cy.contains('Log in').click();  
  cy.get("#mail").type(" ");  
  cy.get("#pass").type("test");  
  cy.contains("Submit").click();  
  //добавить assertion  
})
```

Какой assertion добавим?

Добавляем второй тест

Ошибка при неверном формате или пустом поле логина не отображается в DOM. Мы не имеем доступа к селектору.

Проверим псевдоклассы, которые можно использовать в HTML:

:invalid — псевдокласс добавляется тем элементам, которые не соответствуют заданным параметрам. Например, наше поле логина без верного формата логина будет иметь такой псевдокласс. С верно введенным форматом будет **:valid**

Проверим в консоли браузера:

```
> $('input:invalid')
< <input placeholder="Enter email" required type="email" id="mail" class="form-control">
> |
```

Добавляем второй тест

Такая конструкция выглядит логичной:

```
cy.get('#mail').should('have.class', 'invalid')
```

Добавляем второй тест

Но она не будет работать в Cypress, так как псевдокласс — это не класс:

```
cy.get('#mail').should('have.class', 'invalid')
```

Добавляем второй тест

Но она не будет работать в Cypress, так как псевдокласс — это не класс:

```
cy.get('#mail').should('have.class', 'invalid')
```

Вновь обратимся к консоли браузера. В Cypress будут работать те же методы, которые мы можем применять в консоли браузера для взаимодействия с элементами. Например:

```
> $('#mail').checkValidity()  
< false  
> |
```


Добавляем второй тест

`cy.get()` работает в другом месте, но так же, как JQuery в консоли браузера.

Чтобы работать с этими элементами, необходимо использовать возвращаемое `cy.get()` значение через метод `.then()`:

```
cy.get('#mail').then()
```

Внутри `.then()` мы можем работать с value элемента, через `$el`:

```
cy.get('#mail').then($el => $el[0].checkValidity()).should('be.false')
```

Добавляем третий тест

6

Добавляем третий тест

```
it("Should not login with empty password", () => {  
  cy.visit("localhost:3000");  
  cy.contains('Log in').click();  
  cy.get("#mail").type("test@test.com");  
  cy.contains("Submit").click();  
  cy.get('#pass').then($el => $el[0].checkValidity()).should('be.false')  
})
```

Вопрос аудитории

Как бы вы улучшили эти тесты?



Добавляем общую переменную для всего проекта

В cypress.json настроим основной URL для всего проекта:

```
"baseUrl": "localhost:3000"
```

Теперь Cypress открывает этот URL сразу при запуске.

Мы экономим время на переход между страницами и можем использовать команду открытия новых страниц в формате:

```
cy.visit('/');
```

Обновим это в тестах

Улучшения

Cypress позволяет усовершенствовать тесты и использовать все известные принципы написания чистого кода.

Например, Cypress позволяет помимо основного набора встроенных команд добавлять собственные команды и тем самым не повторять большие куски кода.

Попробуем



Custom commands

7

Рефакторинг логина

Хорошим кандидатом на имплементацию кастомной команды является логин.

Во многих тестах мы будем выполнять логин и повторять однотипные действия, а меняться будут лишь тестовые данные:

```
cy.contains('Log in').click();  
cy.get("#mail").type("test@test.com");  
cy.get("#pass").type("test");  
cy.contains("Submit").click();
```


Добавляем кастомную команду

Кастомные команды хранятся в директории `support`, в файле `commands.js`.

Мы можем добавлять сколько угодно файлов с кастомными командами. Важно заимпортировать их в файл `index.js` в той же директории.

Файл `commands.js` уже заимпортирован там по умолчанию

Добавляем кастомную команду

Добавляем кастомную команду:

```
Cypress.Commands.add("login", (login, password) => {  
  })
```

Добавляем кастомную команду

Добавляем кастомную команду:

```
Cypress.Commands.add("login", (login, password) => {  
  })
```

Переносим код из тестов, заменяя тестовые данные на параметры, подаваемые на вход:

```
Cypress.Commands.add("login", (login, password) => {  
  cy.contains('Log in').click();  
  cy.get("#mail").type(login);  
  cy.get("#pass").type(password);  
  cy.contains("Submit").click();  
})
```

Используем кастомную команду

Теперь наш тест может иметь следующий вид:

```
it("Should successfully login", () => {  
  cy.visit('/');  
  cy.login("test@test.com", "test");  
  cy.contains("Добро пожаловать test@test.com").should("be.visible");  
})
```

Проверим, работает ли тест

Итоги

- Познакомились с Cypress
- Рассмотрели подробнее структуру проекта
- Разобрались с основными функциями Cypress
- Научились добавлять и использовать кастомные команды
- Создали базу для сценария, который закончим в домашней работе

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#)

- Вопросы по домашней работе задавайте в чате группы
- Задачи можно сдавать **по частям**
- Зачёт по домашней работе проставляют после того, как приняты **все задачи**

Дополнительные материалы

- [Подробнее об assertions в Cypress](#)
- [Репозиторий с полезными рецептами Cypress](#)
- [Синтаксис использования кастомных команд](#)

**Задавайте вопросы
и пишите отзыв
о лекции**

