

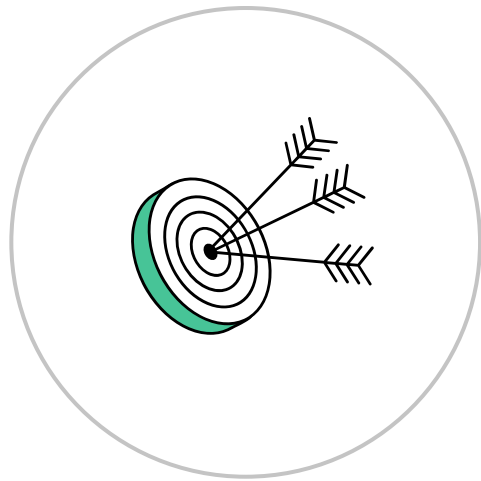
# Организация тестирования

Александр Шлейко  
Разработчик интерфейсов  
в Яндексе



# Цели занятия

- Познакомимся с разными видами тестирования
- Вспомним, как писать Unit-тесты
- Узнаем об E2E-тестировании
- Внедрим тесты компонентов
- Научимся работать с JSDOM



# План занятия

1 [Пирамида тестирования](#)

2 [Unit - тестирование](#)

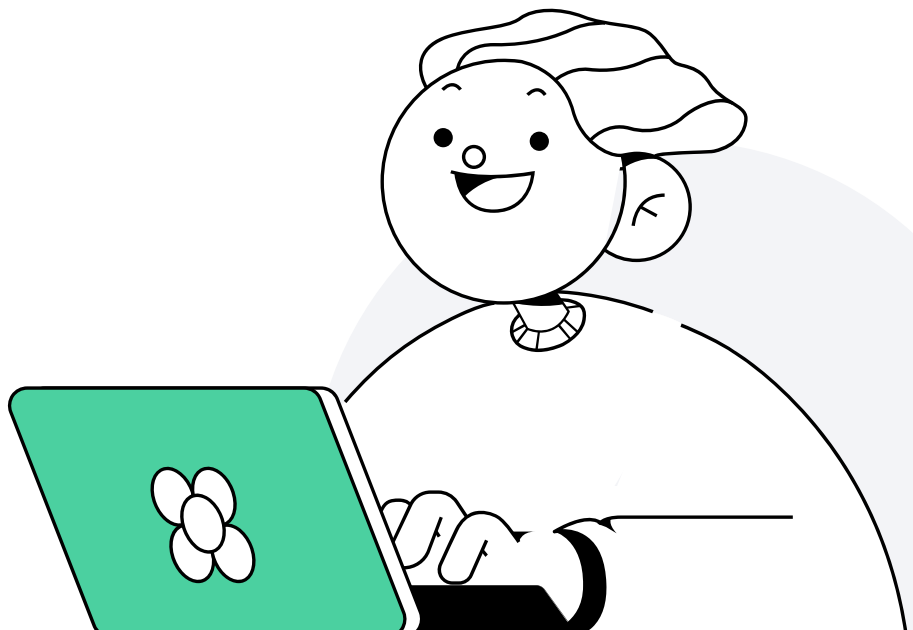
3 [E2E - тестирование](#)

4 [Компоенты](#)

5 [JSDOM](#)

6 [Итоги](#)

Нажмите на раздел для перехода

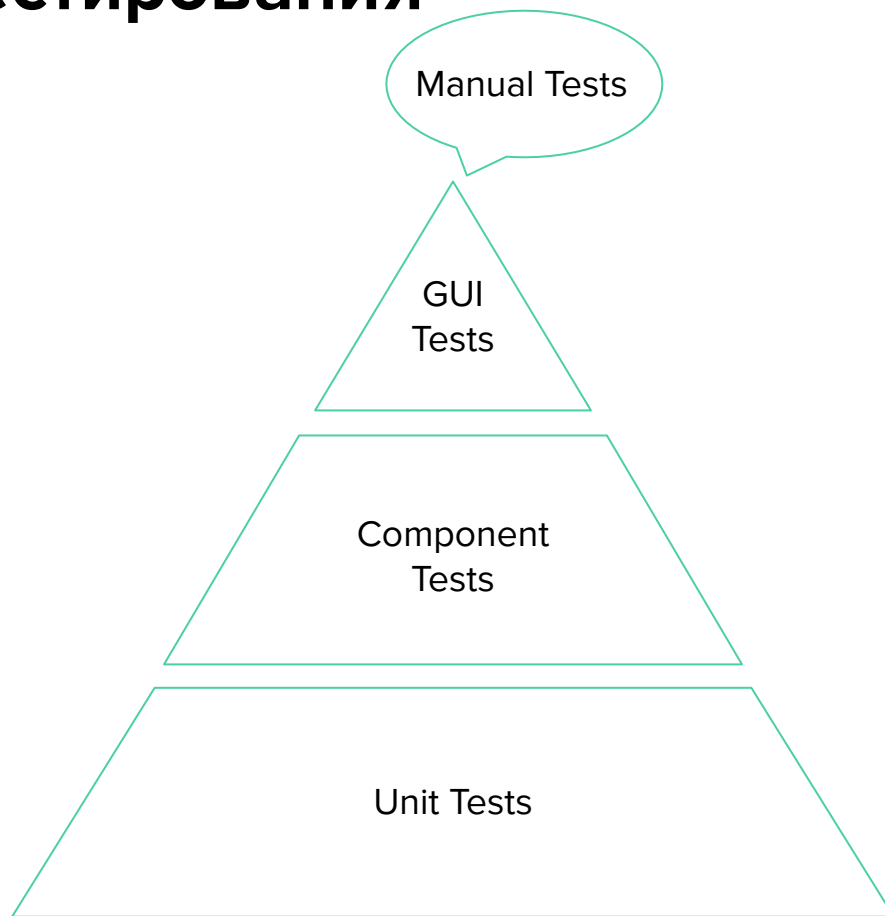


# Пирамида тестирования



1

# Пирамида тестирования



# Разбираемся с терминологией

- ① Unit Tests — изолированное тестирование функциональности
- ② Component Tests — изолированное тестирование отдельных компонентов
- ③ GUI Tests — тестирование с точки зрения конечного пользователя
- ④ Manual Tests — тесты, выполняемые вручную

# Test runner & Assertions library

В качестве инструмента для запуска тестов и проверки результатов мы будем использовать Jest.

## Установка:

```
1 $ npm install --save-dev jest babel-jest @babel/core @babel/cli
2 @babel/preset-env
3 $ npm install core-js@3
```

## В .babelrc:

```
1 {
2   "presets": [[ "@babel/preset-env", {
3     "useBuiltIns": "usage",
4     "corejs": 3
5   } ] ]
6 }
```

# Unit тестирование



2



# Вспомним простые правила

- ① Пишем тест с именем, совпадающим с именем тестируемого файла и добавляем суффикс **test.js**
- ② Тесты кладем в папку tests

# E2E тестирование



2

## **E2E (end-to-end) тестирование**

— тестирование настоящим  
браузером наших  
интерфейсов

# E2E тестирование

В современных браузерах есть встроенная возможность отправлять автоматические команды, то есть управлять браузером извне. Чтобы это делать нам нужна программа, которая умеет отправлять такие команды браузеру.

**puppeteer** — очень популярный пакет для написания автотестов в мире фронтэнда.

# Установка puppeteer

Для установки пакета puppeteer используем `yarn`

# Название файла

В E2E тестировании нет смысла называть тест также, как тестируемый файл, потому что в данном случае тестируется не отдельная функция, а целый сценарий, который затронет разные части приложения.

**Названия тестам следует давать исходя из того, какую часть функциональности необходимо протестировать, например, basket, login и пр.**

# Асинхронные функции

Команды в puppeteer асинхронные, поэтому и тесты и команды в `beforeEach` необходимо заворачивать в асинхронные функции

# dev server

Вместе с выполнением тестов в новой вкладке нужно запустить **DevServer**



# Завершение тестов

Чтобы завершить тест можно использовать `afterAll` и команду закрытия браузера

```
1  afterAll (async () => {  
2    await broser.close();  
3  });
```

`await` используется для перехода страницы, в противном случае переход не будет осуществлен, и тест не будет ждать пока браузер загрузит страницу и сразу перейдет к `afterAll`

# beforeEach/beforeAll, afterAll/afterEach

При запуске теста вы можете столкнуться с проблемой: количество запущенных браузеров оказывается большим, и один из браузеров так и остается висеть.

Это происходит из-за того, что новый instance браузера запускается в хуке **beforeEach**, то есть перед каждым тестом, а закрывается **afterAll**, то есть после прогона всех тестов. Таким образом один из instance остается незакрытым в том случае когда у нас больше одного теста внутри

# beforeEach/beforeAll, afterAll/afterEach

Есть два варианта действий в этой ситуации:

1. переделать `beforeEach` в `beforeAll`
2. или `afterAll` в `afterEach`

# beforeEach/beforeAll, afterAll/afterEach

Принимайте решение в зависимости от:

- сложности проета,
- количества тестов,
- критичности скорости прогона.

Если вы не хотите думать о том, как тесты могут повлиять друг на друга, тестов много, и вы хотите большей изолированности, то используйте `beforeEach` и `afterEach`

# Тестирование поведения

С точки зрения интерфейса ваша задача проверять тестами именно **поведение** вашего интерфейсного элемента

# Особенности E2E тестирования

- ① Тестируется проект целиком в режиме приближенном на продакшн
- ② Дорого и долго
- ③ Если полностью описать все пользовательские сценарии и пройти по ним автоматическими проверками, то мы точно будем уверены, что приложение работает корректно

# backend

**Важно думать о том, что в момент тестирования будет происходить на backend**


В puppeteer есть специальный инструмент позволяющий мокать ответы бэкэнда  
— `request respond`

# Компоненты



4





**Компонент** — сущность в коде,  
описывающая бизнес-логику,  
внешний вид и поведение  
отдельного конечного блока в  
приложении

# Компоненты

Хорошо, если компонент может быть переиспользован, скопирован и перенесен в любое другое место в проекте, а в идеале и между проектами

# Компоненты

Использование компонентов подразумевает, что в одной папке мы определим:

- верстку виджета
- его стили
- внешний вид
- поведение при действиях пользователя

В эту же папку можно положить документацию, тесты и другие файлы которые в этом компоненте используются

# Тестирование компонентов

Разбивка кода на компоненты помогает в написании более легких тестов в проекте

1. Представим компонент, как класс, внутри которого определим способ отображения компонента и его бизнес-логику
2. Класс на вход в конструктор получит селектор контейнера, в котором наш компонент должен отрисоваться
3. Таким образом, в тесте мы запросто сможем отрисовать компонент в любом контейнере с помощью метода `bindToDOM` и проверить с помощью `purpoteer`, работает ли компонент так, как мы ожидаем

# JSDOM

5



# JSDOM

JavaScript-реализация  
WHATWG-стандарта DOM и  
html для использования

# Использование JSDOM в тестах

При использовании JSDOM в тестах вы сможете писать Document как будто бы вы находитесь в браузере, и все будет работать ровно также, как если бы браузер по настоящему был, но на самом деле его не будет.

В состав JS часто включены все инструменты которые вам нужны и JSDOM не исключение. Все, что вам нужно - это начать писать тесты с использованием Document и Window

# Итоги

Сегодня мы:

1. познакомились с разными видами тестов
2. применили навыки написания Unit-тестов в структуре фронтэнда
3. рассмотрели тестирование интерфейса - E2E-тестирование
4. познакомились с концепцией компонентов и начали мыслить сущностями
5. узнали о структуре JSDOM и научились писать тесты с ее использованием





# Задавайте вопросы и пишите отзыв о лекции

Александр Шлейко  
Разработчик интерфейсов в  
Яндексе

