

# Введение в CodeceptJS

CodeceptJS - это инструмент для end-to-end тестирования, который использует простой и понятный синтаксис, упрощая написание тестов. Он поддерживает BDD (Behavior Driven Development), что позволяет описывать поведение приложения на естественном языке.

Преимущества:

- **Читаемость:** Тесты записываются в описательной и понятной форме.
  - **Гибкость:** Поддерживает различные тестовые фреймворки и драйверы (в нашем случае - Playwright и Appium).
  - **Модульность:** Позволяет переиспользовать код благодаря хелперам и паттерну Page Object.
  - **Кроссплатформенность:** Один тест можно запустить на разных платформах (веб, Android, iOS), что уменьшает дублирование кода и облегчает поддержку.
- 

## Написание тестов

Тесты пишутся с точки зрения пользователя. В CodeceptJS есть объект `I`, который представляет пользователя и выполняет действия из хелперов. Пример простого e2e теста:

```
Feature('Логин');

Scenario('Логин с валидными учетными данными', ({ I }) => {
  I.amOnPage('/');
  I.fillField('emailField', 'user@test.com');
  I.fillField('passwordField', 'password');
  I.click('Submit');
  I.see('Welcome, user!');
});
```

- **Feature:** Описывает функциональность, которую мы тестируем.
- **Scenario:** Конкретный тест-кейс, моделирующий пользовательское действие.
- **I Object:** Представляет актера (пользователя), выполняющего действия.

- Assertions: Проверки, например, `I.see()`, чтобы убедиться в корректности результата.
- 

## Локаторы элементов

- CSS-селекторы: `I.click('.btn')` - элемент с CSS-классом `btn`
- XPath: `I.click('//button[text()='Submit'])` ([Полезные шпаргалки по XPath](#))
- По тексту: `I.see('Welcome')`
- По accessibility ID в мобильных тестах: `I.seeElement('~username')`;
- По test-id (лучший практический подход):  
`I.click('[data-test-id=submit_button])`;

Пример элемента `searchBox` в Page Object:

```
searchBox: {  
  web: '[aria-label="Search Box"]',  
  android: '//android.widget.EditText[@content-desc="Search Box"]',  
  ios: '//XCUIElementTypeTextField[@name="Search Box"]',  
},
```

---

## Assertions (Проверки)

Для проверки ожидаемого поведения используются утверждения (assertions).

CodeceptJS поддерживает Chai assertions через `codeceptjs-chai`. Chai — это библиотека утверждений, аналогичная встроенному `assert` в Node.js, но с более удобным API.

Подробнее: [Документация Chai](#)

---

## Паттерн Page Object

Для улучшения организации и поддержки тестов используется паттерн Page Object.

Преимущества:

- Инкапсулирует функциональность и локаторы в отдельных классах.
- Улучшает читаемость и переиспользуемость кода.

Подробнее: [Официальное руководство по Page Object](#)

---

## Работа с Appium Inspector

Appium Inspector помогает находить локаторы для Android и iOS (ID, labels, accessibility IDs и т. д.).

Настройка Appium Inspector:

1. Установите [Appium Inspector](#).
2. Запустите сервер Appium: `npx appium --base-path=/wd/hub`
3. Укажите `Remote Port` и `Remote Path`, добавьте `desired capabilities`.

📌 Как узнать UDID для iOS-симулятора:

- Откройте Xcode → Window → Devices and Simulators → Simulators → Выберите симулятор → Найдите Identifier (это UDID).

Подробнее: [Официальная документация Appium Inspector](#)

---

## Запуск и отладка тестов

CodeceptJS поддерживает несколько режимов запуска тестов:

Базовый запуск

```
npx codeceptjs run
```

*Для чего:* Выполняет тесты и выводит краткое резюме (успех/неудача, время выполнения).

Пошаговый вывод (steps)

```
npx codeceptjs run --steps
```

*Для чего:* Показывает шаги теста в реальном времени, полезно для анализа сценария.

Полный режим отладки (debug)

```
npx codeceptjs run --debug
```

*Для чего:* Подробно показывает логи, сетевые запросы, локаторы и все шаги теста.

Интерактивная отладка с pause()

Можно вставить `pause()` в код теста, чтобы временно остановить выполнение и вручную взаимодействовать с приложением.

```
I.fillField('email', 'test@example.com');  
pause(); // здесь тест остановится  
I.click('Submit');
```

Продолжить выполнение можно командой `.continue()` в консоли.

Подробнее: [Отладка в CodeceptJS](#)

---

## Генерация отчетов

По умолчанию CodeceptJS показывает результаты в консоли. Для HTML-отчетов можно использовать Mochawesome reporter:

```
npx codeceptjs run --reporter mochawesome
```

Подробнее: [Настройка отчетов](#)

---

## Добавление тегов в тесты

Можно добавлять теги для удобного запуска тестов:

Способ 1: Использование `.tag()`

```
Scenario('Тест с тегом', ({ I }) => {  
  I.amOnPage('/');  
}).tag('@critical');
```

Способ 2: Включение тега в название

```
Scenario('@regression Проверка входа', ({ I }) => {  
  I.amOnPage('/');  
});
```



Запуск тестов с тегами:

```
npx codeceptjs run --grep @critical
```

Подробнее: [Фильтрация тестов по тегам](#)

---

 Полезные ссылки:

-  Официальная документация CodeceptJS: <https://codecept.io>
  -  Полезные плагины и инструменты: [CodeceptJS Plugins](#)
- 

Это руководство поможет быстрее освоить CodeceptJS и эффективно использовать его для автоматизации тестирования. 