Don’t Repeat Yourself: UI тесты для Web, iOS и Android одновременно

Игорь Балагуров

igor.balagurov@uptick.com

 ibalagurov 

Обо мне

- Занимаюсь автоматизацией тестирования с 2015г.
- ERP для гидрологических расчетов и прогнозирования
  (E2E тесты на Ruby + Capybara)
- Новые Облачные Технологии: File manager, Text editor
  (E2E тесты: Web и API на Ruby + Watir + Cucumber)
- Uptick: CRM
  (E2E тесты: Mobile, Web, API на Python интеграционные и
  компонентные: C#
  white box, configuration, machine learning testing)

О чём это я

- Никого не удивишь, что у продукта есть web-версия и мобильное приложение
- Автотесты чаще всего пишутся отдельно для desktop Web и iOS \ Android приложений
- Информации о единых решениях мало (automated-testing.info/t/obshhij-testovyj-frejmvork-dlya-mobile-i-web-appium-selenium/)

Зачем это вообще надо?

- Вспомогательный код — сложнее переиспользовать
- Обновление в поведении приложения — надо менять одно и то же в нескольких местах
- Несколько решений — это чаще всего несколько наборов технологий/подходов

Зачем это вообще надо?

- Избежать дубликатов тестовых сценариев
- Обойти проблемы синхронизации нескольких решений

Зачем это вообще надо?

Существенно снизить затраты на разработку и поддержку

Чего дальше НЕ будет?

- Красивых отчетов
- Запуска 100500 тестов в 150 потоков
- Устройство Appium
- Как тестировать конфигурацию, machine learning и т.д.

План

- Предыстория: проект и приложение
- Идея
- Реализация и решение проблем
- Демо (запуск тестов)
- Подводные камни
- Вопросы

Предыстория проекта

- Проект недавно начался и быстро развивается
- Backend (C#.NET Core), Machine Learning (Python), Frontend (JS + React + ReactNative)
- Есть unit, integration, component тесты (зелёные)
- Частые крэши в приложении

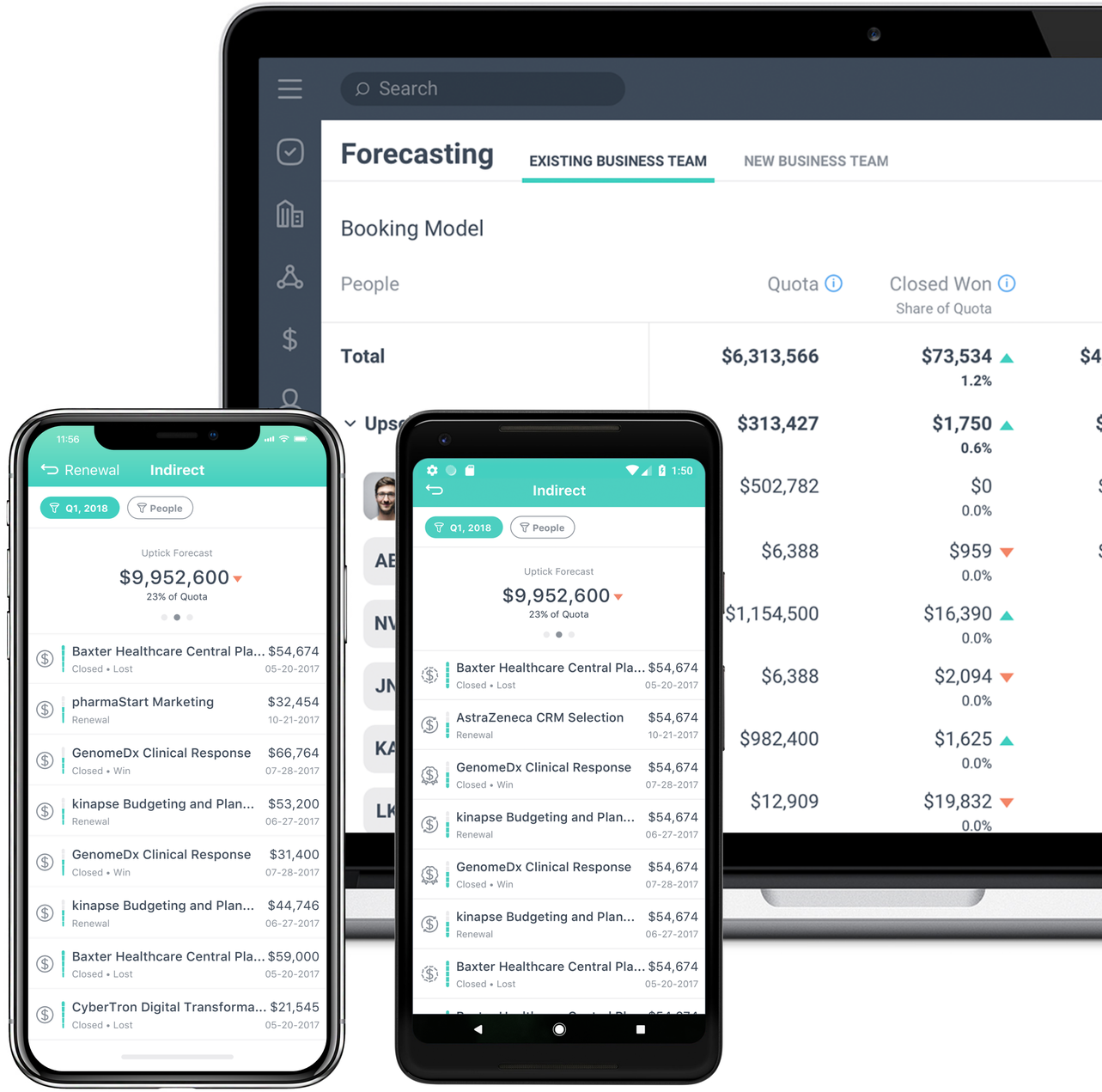
Приложение глазами тестировщика

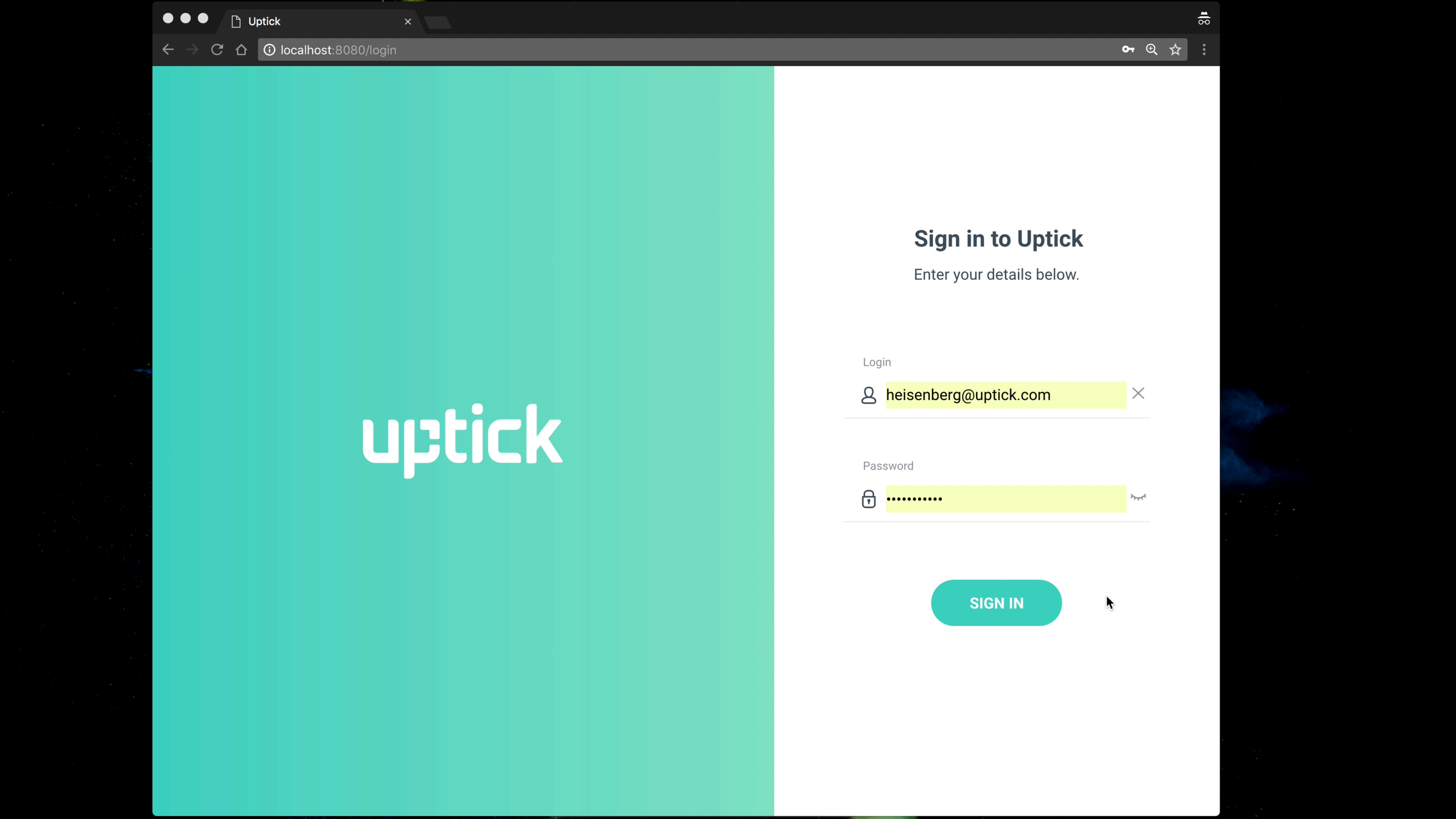
Приложение глазами тестировщика

- iOS, Android, Web
- Микросервисы
- API: REST и GraphQL
- Конфигурация, меняющая приложение «на лету»
- Machine learning
- Нагрузка, безопасность, интеграции со сторонними сервисами…

Приложение глазами тестировщика

iOS, Android, Web




Uptick
localhost:8080/login
uptick
Sign in to Uptick
Enter your details below.
Login
heisenberg@uptick.com
Password
SIGN IN

Идея

- Appium реализует API Selenium
- Можно взять Selenium wrapper
- Единые тесты(сценарии)
- Steps(решаем проблему с разным UX)
- Page objects(одни actions, разные локаторы)

Идея

Можно переиспользовать

- Верхнеуровневую логику тестов
- Actions для page objects
- Steps для группировки Actions

Идея

Что разное в любом случае

- Драйвер для web и mobile
- Локаторы
- Специфичные для платформы действия

Идея

Что иногда разное

UX

Опасения

- Selenium wrapper развалится от Appium driver
- Переходы между экранами
- Много отдельного кода для каждой платформы
- Кто так делал?
- Что-то пойдёт не так
- Сложно
- Долго

От идеи к реализации

Надо делать прототип

На чём?

- C# + NSelene / Atata
- JavaScript + test cafe / webdriver.io / …
- Ruby + Capybara / Watir
- Golang + Agouti
- Python + …

Почему не…

- JS, C# — не самые простые языки
- Golang — мало инструментов для тестирования
- Ruby — зачем тащить ещё один язык?

Python

- Не понравилось всё остальное
- Экспертиза в компании (Data Science)
- Есть инструменты для тестирования
- KISS test automation (можно без ООП)
- Скорость разработки
- Простой и понятный (даже без опыта)
- Выразительный (без boilerplate)

Python testing tools

- Selene — простой, понятный, модный подход (со всеми особенностями Selenide)
- Pytest — механизм фикстур, есть встроенные matchers, очень много плагинов

Python testing tools альтернативы

- JDI — Python версия «сырая», исходники немного испугали
- Webium — только page objects
- Elementium — был следующий на очереди после Selene
- Behave, RobotFramework — посчитали человекочитаемый слой излишним в нашем случае

Реализация

Вперёд, по граблям!

Разные драйверы

Положить нужный для платформы драйвер

```
def get():
    driver = appium if config.test_run.IS_MOBILE else selenium
```

Разные драйверы

Положить нужный для платформы драйвер

```
def get():
    driver = appium if config.test_run.IS_MOBILE else selenium
    return driver.Remote(desired_capabilities=_capabilities)
```

Разные драйверы

Положить нужный для платформы драйвер

```
def get():
    driver = appium if config.test_run.IS_MOBILE else selenium
    return driver.Remote(desired_capabilities=_capabilities)
```

Закрыть должным образом

```
def close(driver):
    driver.close_app() if config.test_run.IS_MOBILE else driver.quit()
```

Разные драйверы

Вариант для локальной разработки

```
def get():
    if config.test_run.IS_MOBILE:
        return appium_driver.Remote(desired_capabilities=_capabilities)

    return selenium_driver.Chrome(
        executable_path=ChromeDriverManager().install(),
        desired_capabilities=_capabilities
    )
```

Опасения

- Selenium wrapper развалится от Appium driver
- Переходы между экранами
- Много отдельного кода для каждой платформы
- Кто так делал?
- Что-то пойдёт не так
- Сложно
- Долго

Опасения

- ~~**Selenium wrapper развалится от Appium driver**~~
- Переходы между экранами
- Много отдельного кода для каждой платформы
- Кто так делал?
- Что-то пойдёт не так
- Сложно
- Долго

Selene

Заворачиваем в селен драйвер

```
from selene import factory
```

Selene

Заворачиваем в селен драйвер

```
from selene import factory

driver = ui_driver.get()
```

Selene

Заворачиваем в селен драйвер

```
from selene import factory

driver = ui_driver.get()

factory.set_shared_driver(driver)
```

Selene

Заворачиваем в селен драйвер

```
from selene import factory

driver = ui_driver.get()

factory.set_shared_driver(driver)
```

На выходе надо «правильно» закрыть

```
ui_driver.close(driver)
```

Selene

Ожидания элементов

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- Переходы между экранами
- Много отдельного кода для каждой платформы
- Кто так делал?
- Что-то пойдёт не так
- Сложно
- Долго

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~**Переходы между экранами**~~
- Много отдельного кода для каждой платформы
- Кто так делал?
- Что-то пойдёт не так
- Сложно
- Долго

Разные локаторы

Добавляем специфичные типы локаторов

```
from appium.webdriver.common.mobileby import MobileBy


def accesibility_id(_id):
    return MobileBy.ACCESIBILITY_ID, _id
```

Разные локаторы

Учим Selene «понимать» локаторы для разных платформ

```
def element(selectors):
    return s(locator_for_platform(selectors))
```

Разные локаторы

Page Element

```
user_email = ui.element({
    'IOS': by.accessibility_id('emailInput'),
    'WEB': by.css('[type=login]')
})
```

Разные локаторы

Page Element

```
user_email = ui.element({
    'IOS': by.accessibility_id('emailInput'),
    'WEB': by.css('[type=login]')
})
```

Actions

```
def login_with(email, password):
    user_email.set(email)
    user_password.set(password)
    submit_button.click()
```

Page Objects

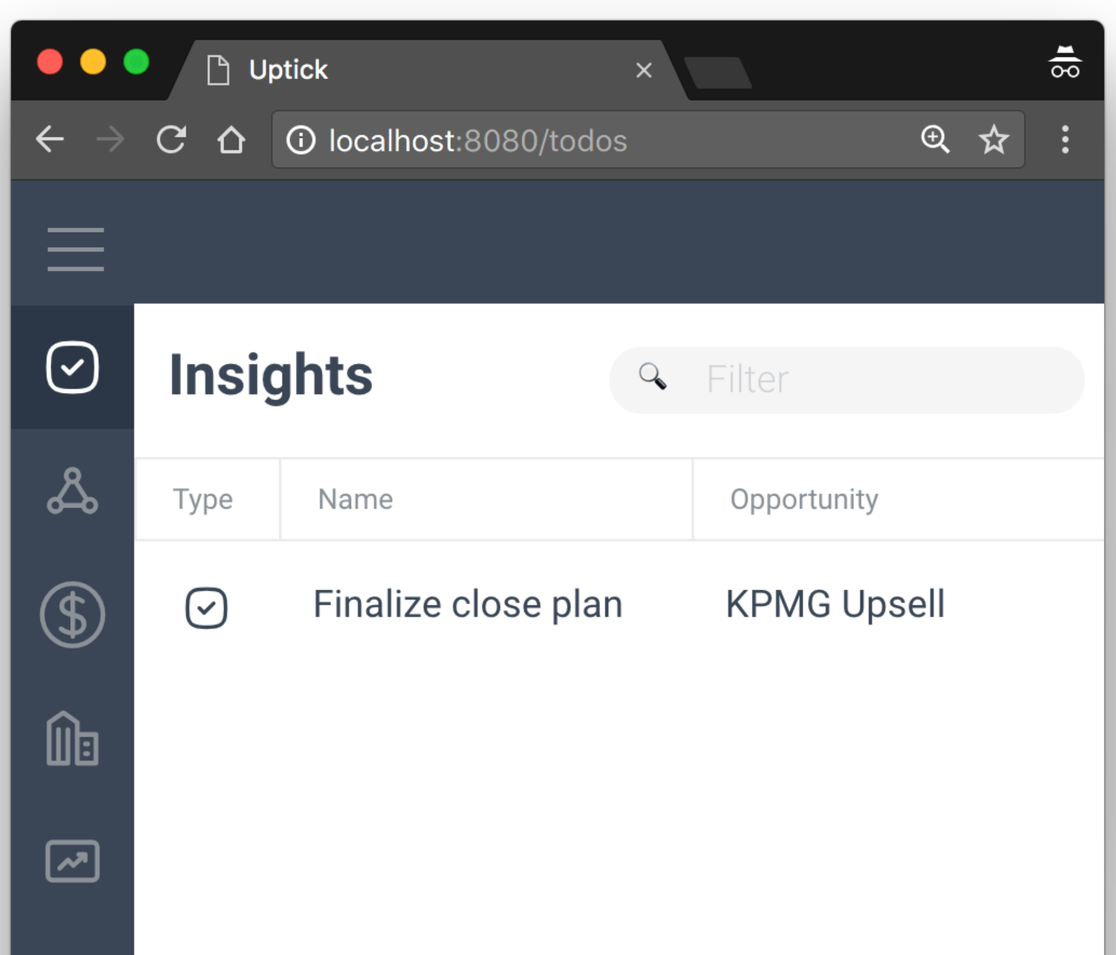
- Что из себя представляет?
- Как разбить?

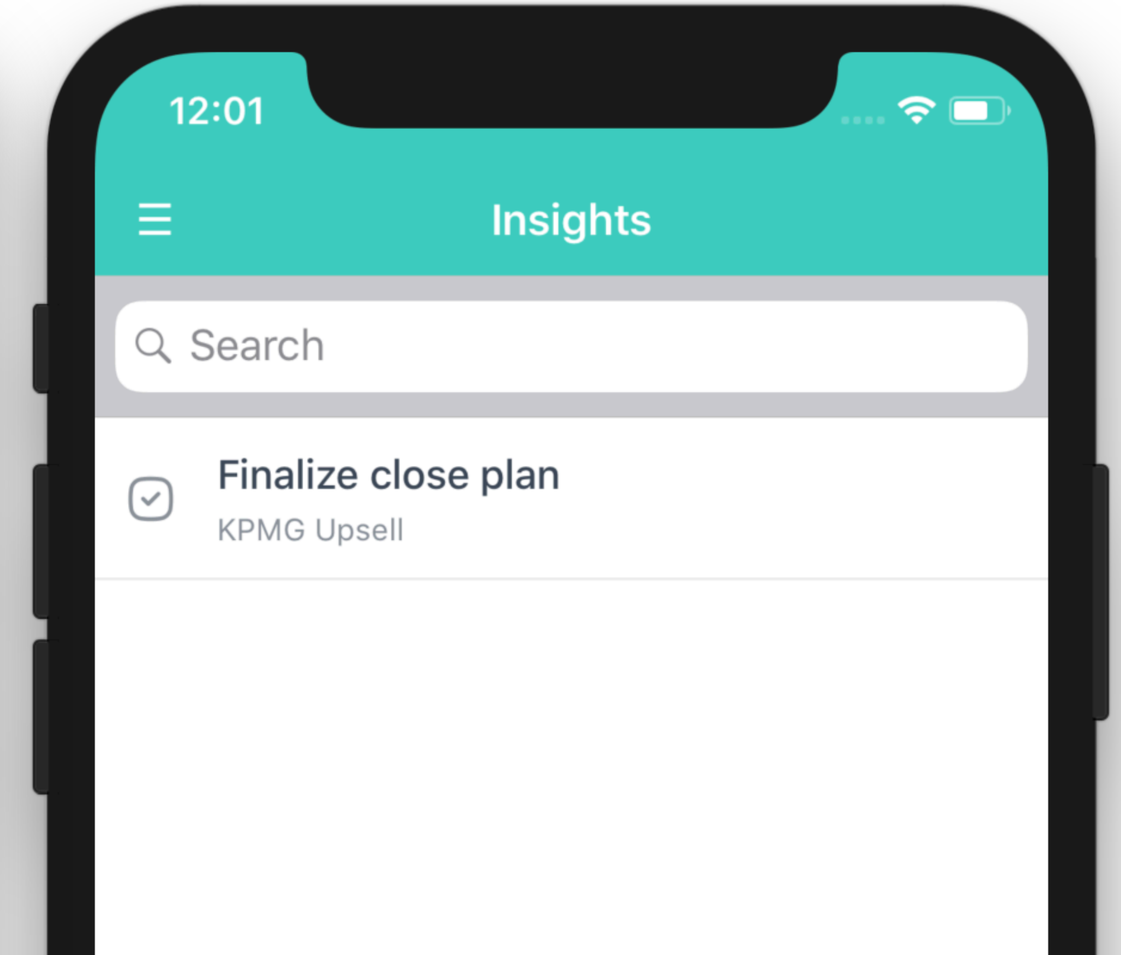
Page Objects

- Набор неделимых элементов в контексте приложения
- Чаще всего — это часть экрана мобильного приложения

Page Objects

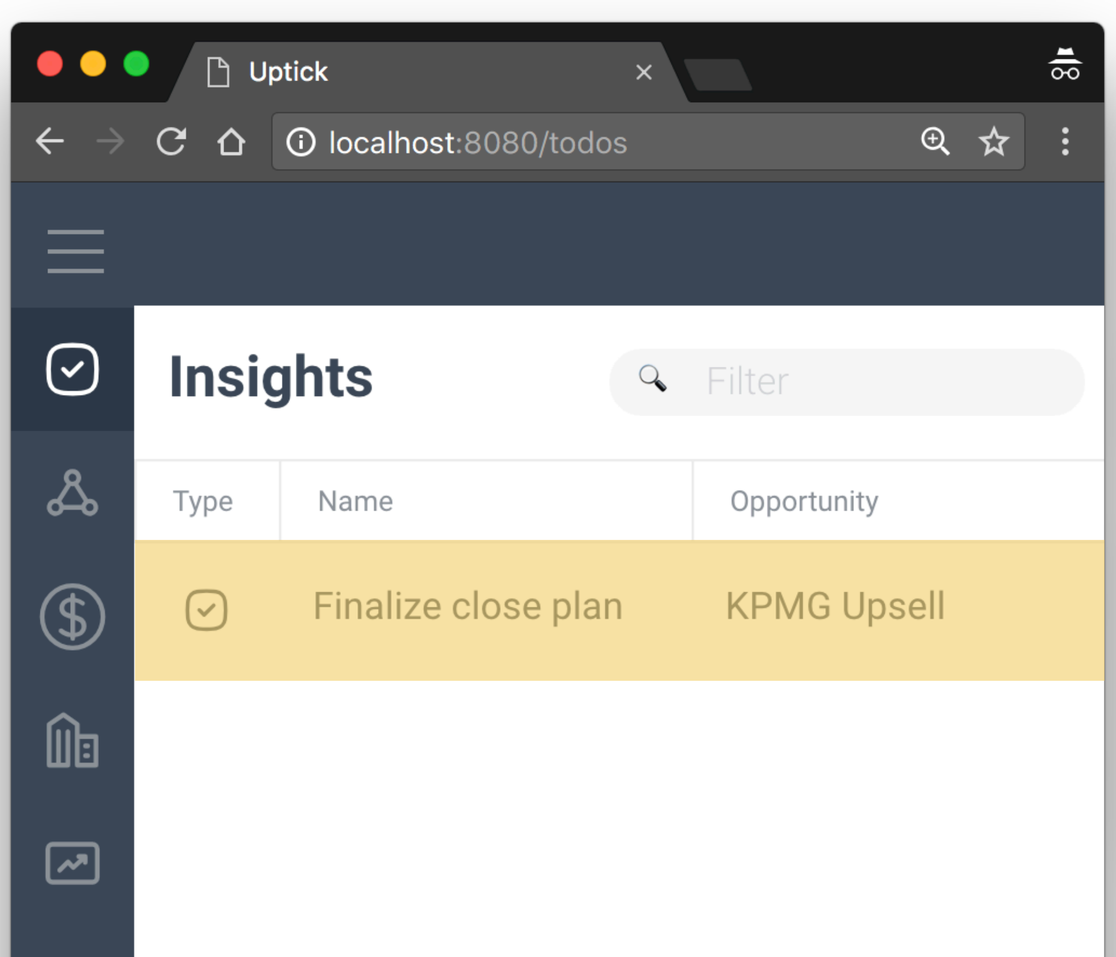
Выделяем одинаковые элементы

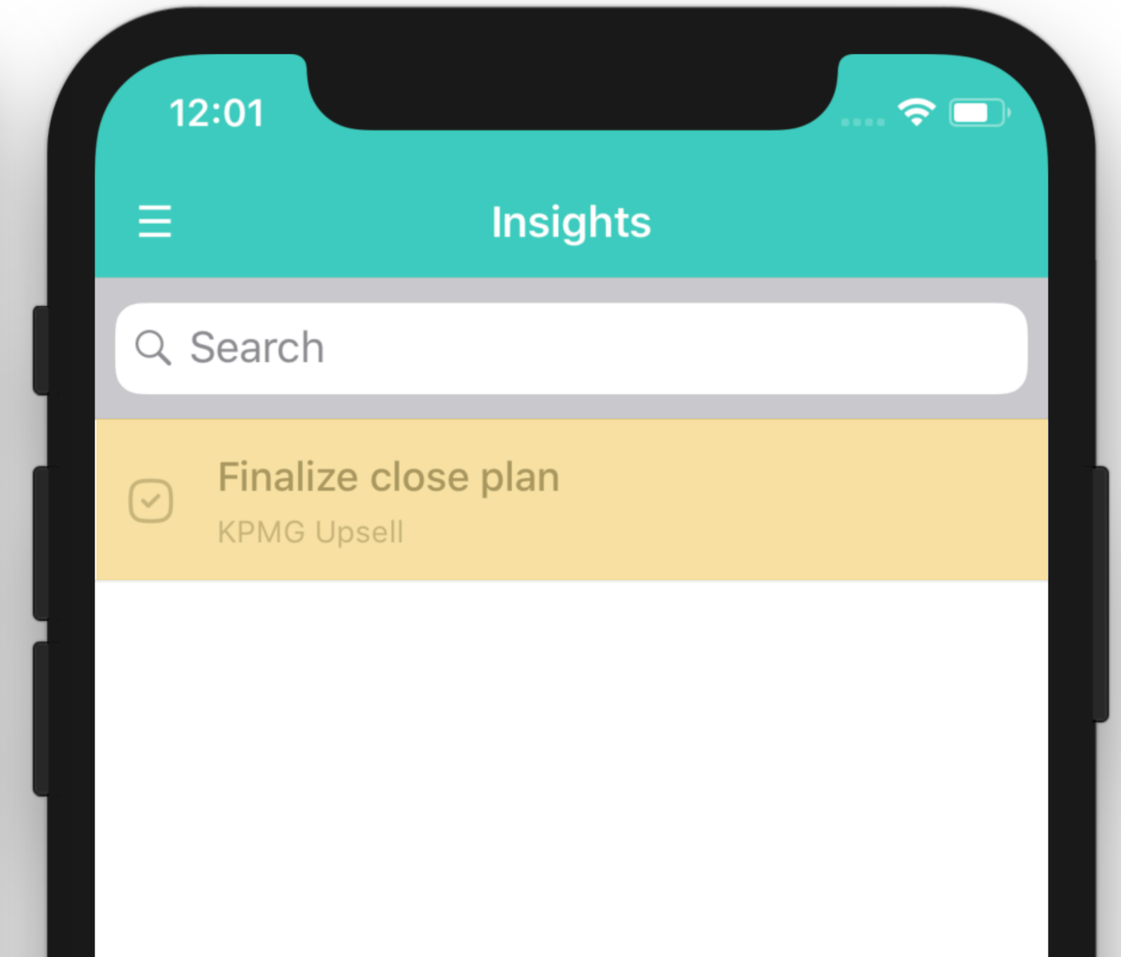




Page Objects

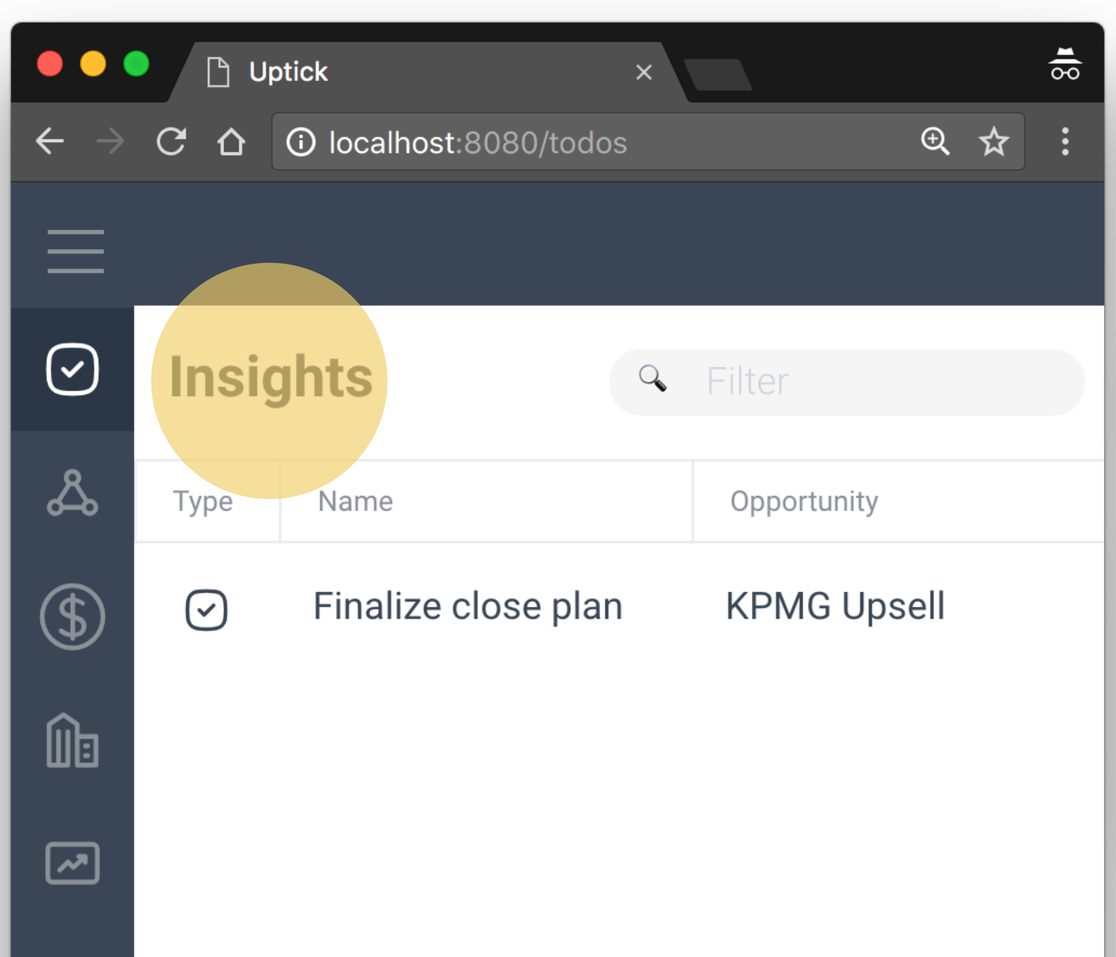
Выделяем одинаковые элементы

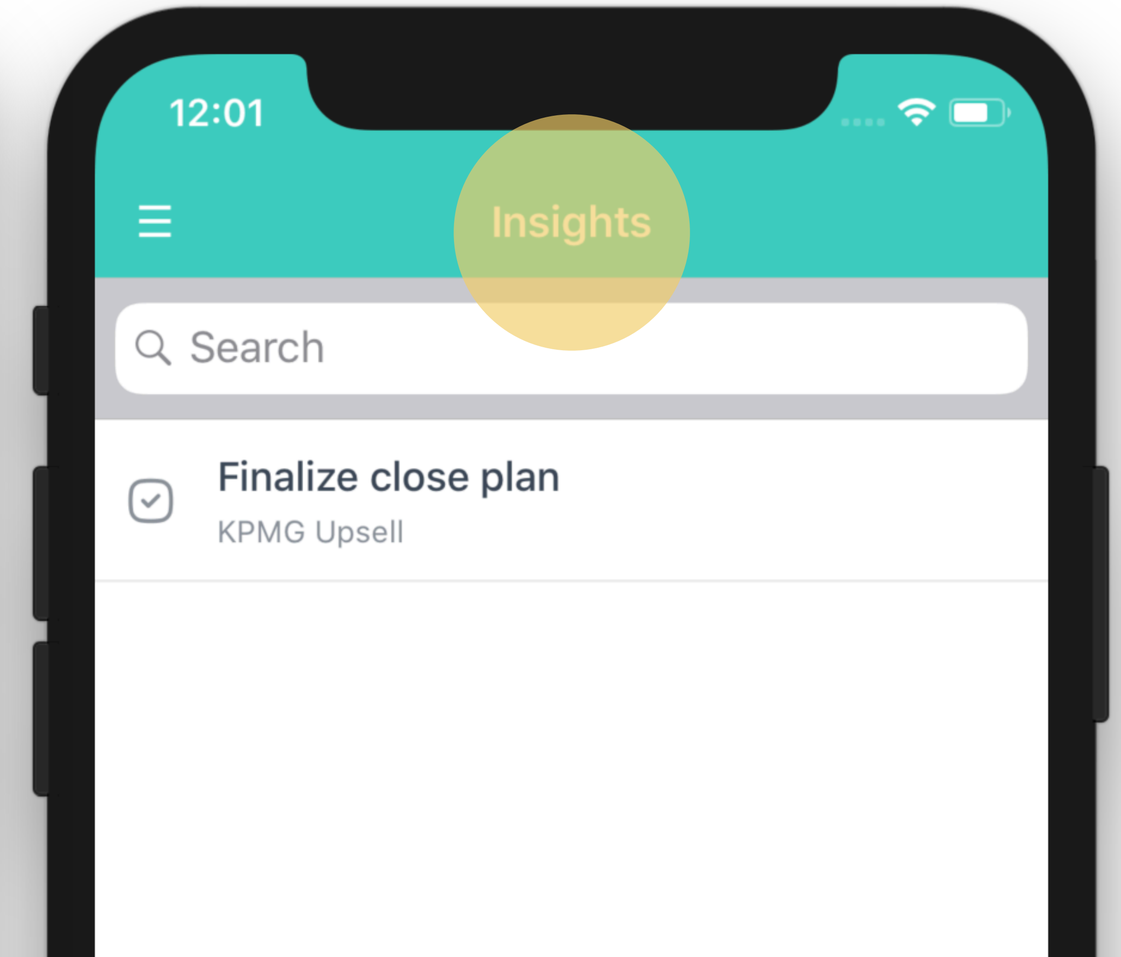




Page Objects

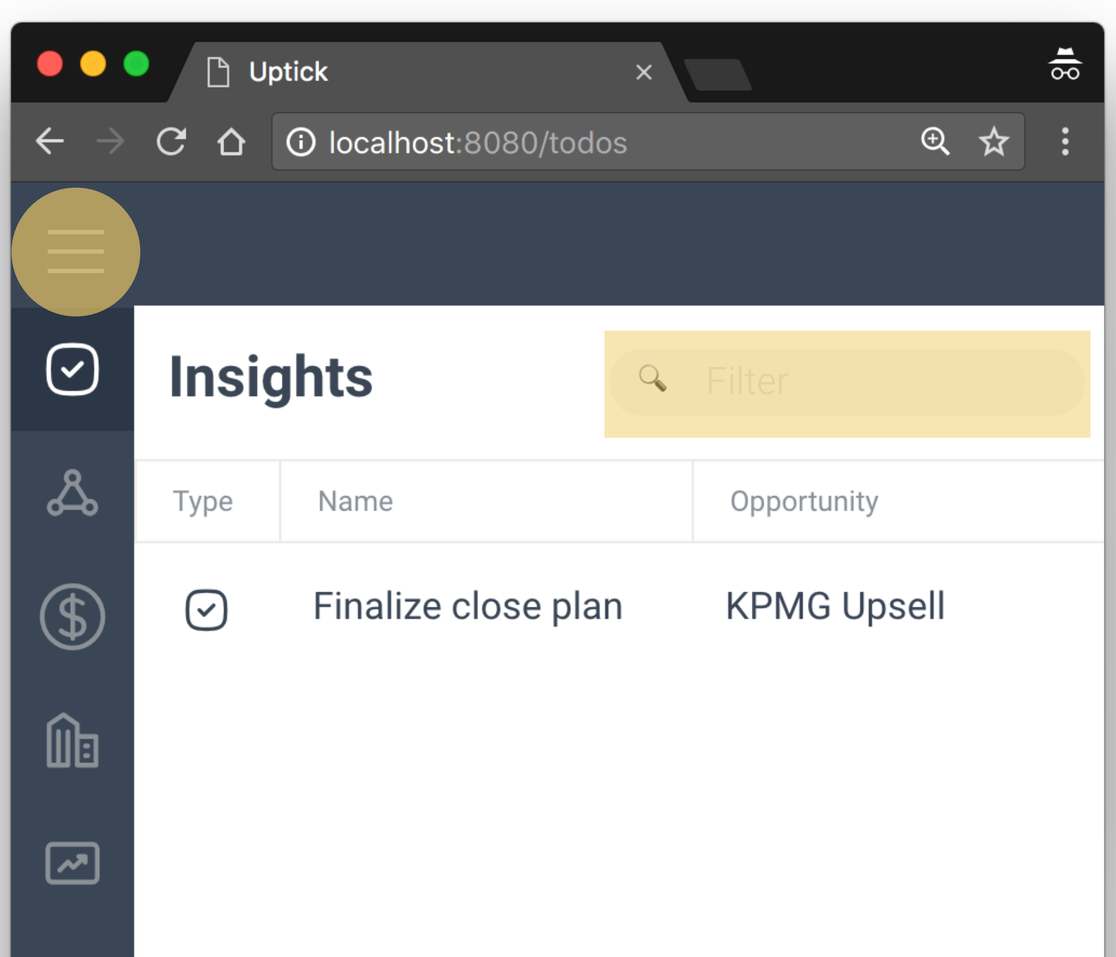
Выделяем одинаковые элементы

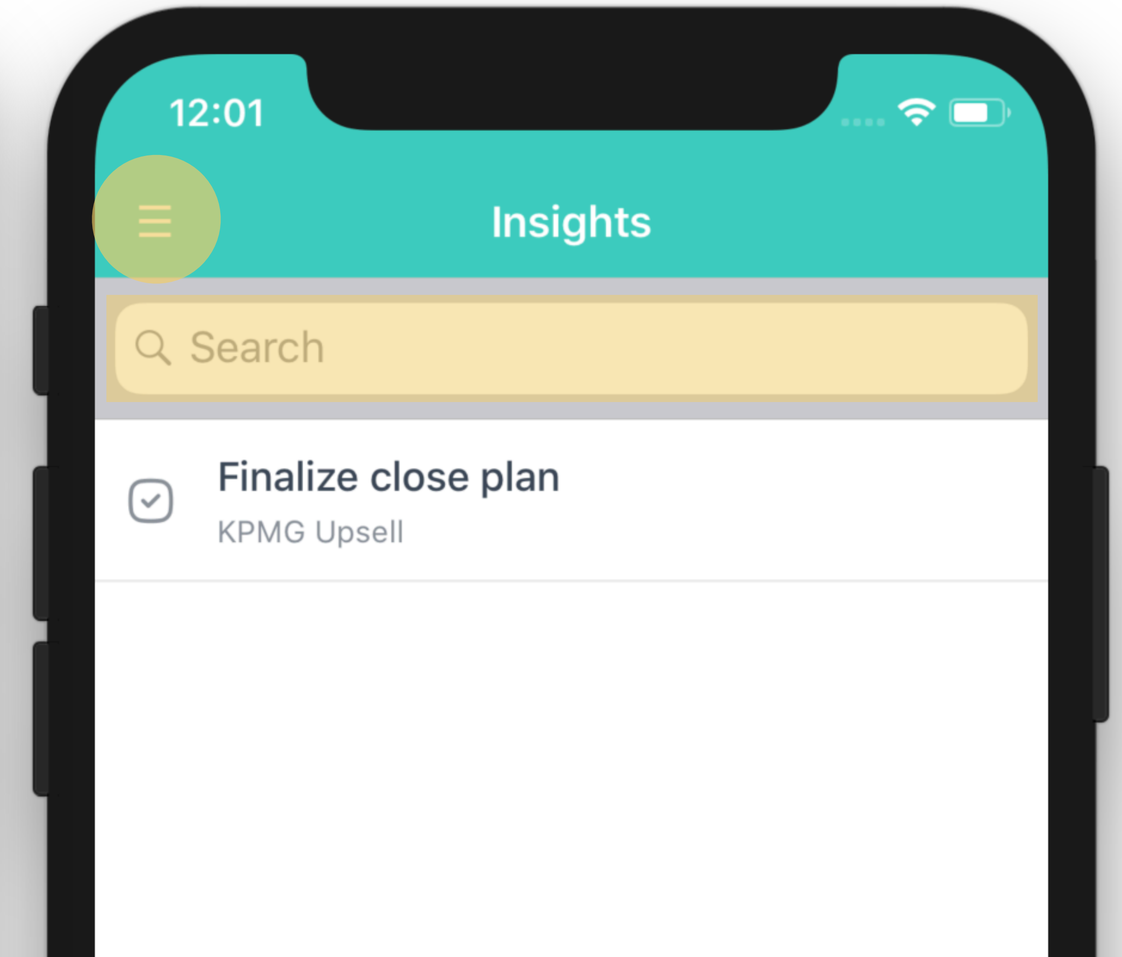




Page Objects

Выделяем одинаковые элементы

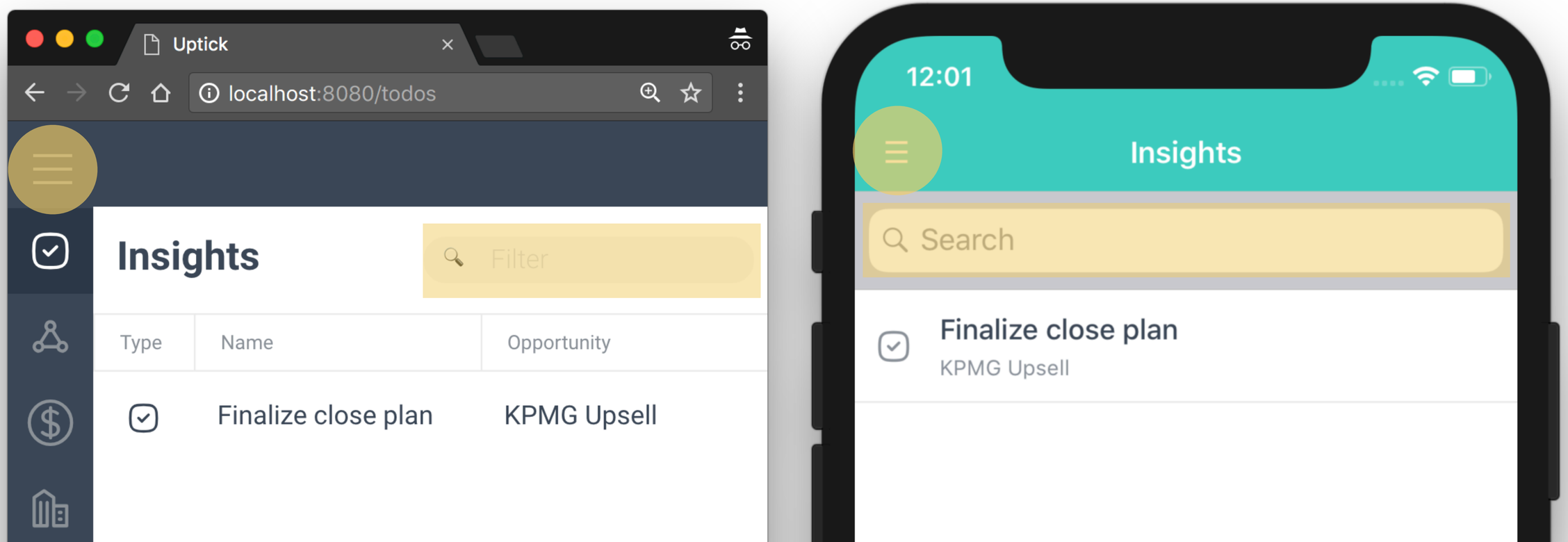




Page Objects

Выделяем одинаковые элементы

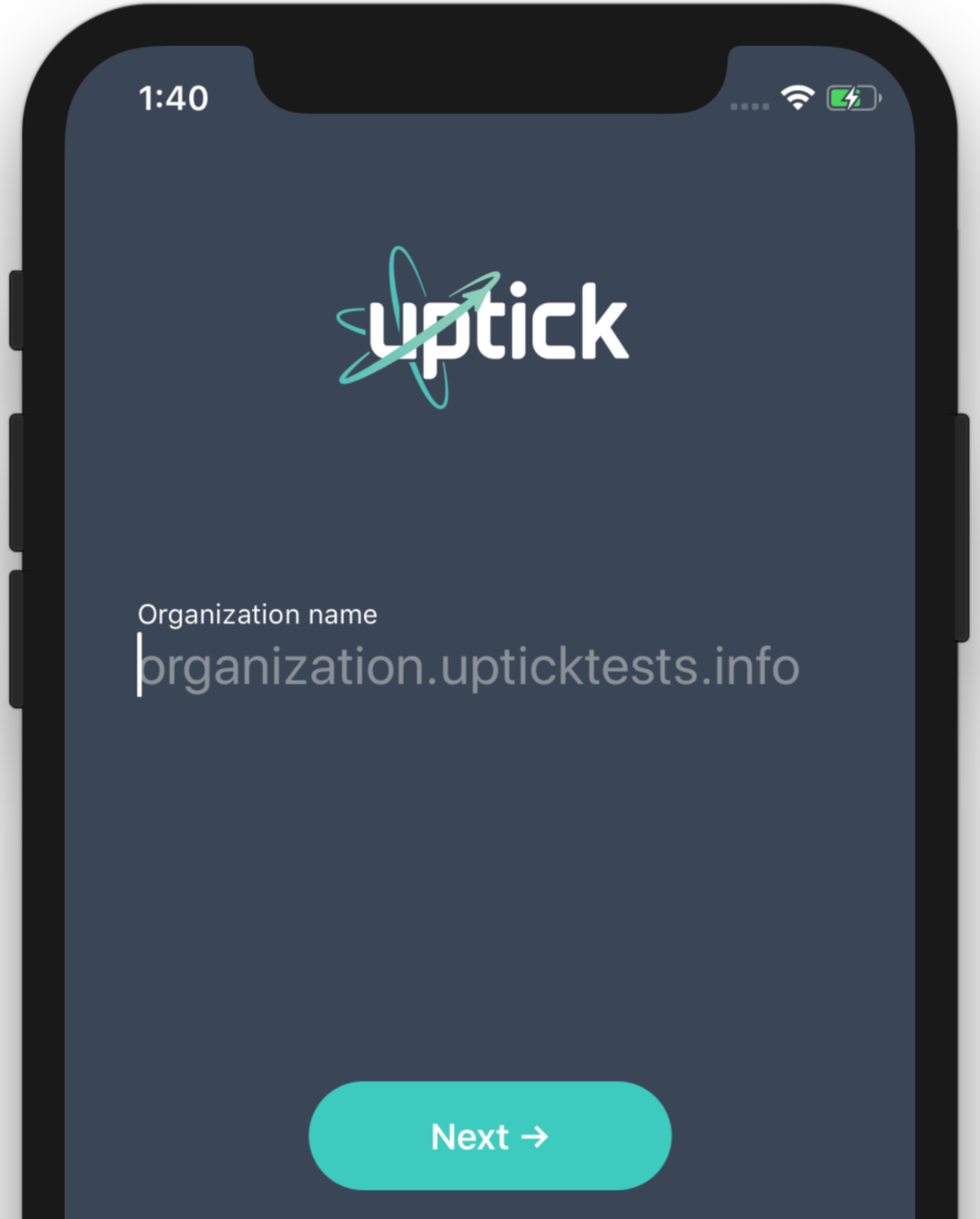
- Отличаются только локаторы
- Pages и actions - одни и те же



Page Objects

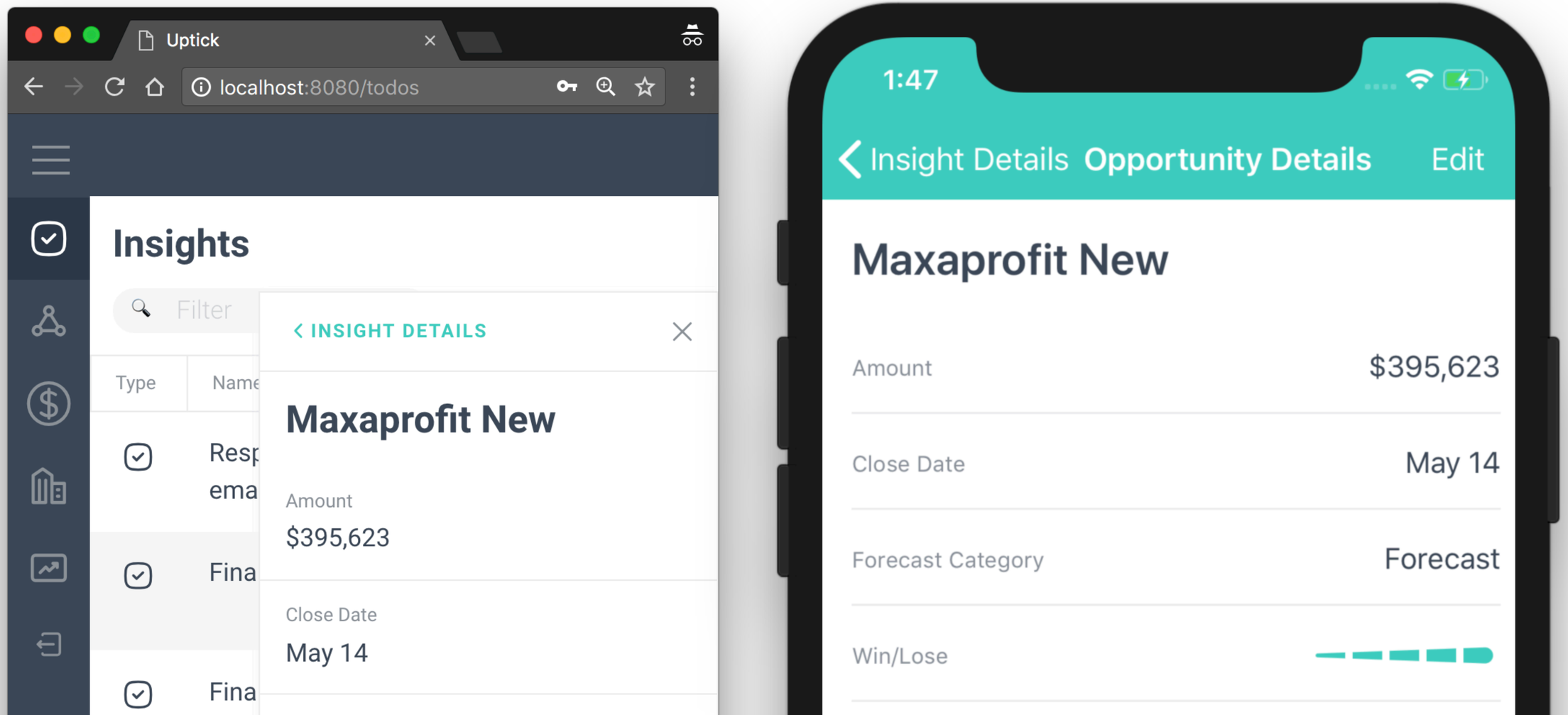
Специфичные экраны - отдельно

- Вызов напрямую в тесте
- На тест - tag/mark



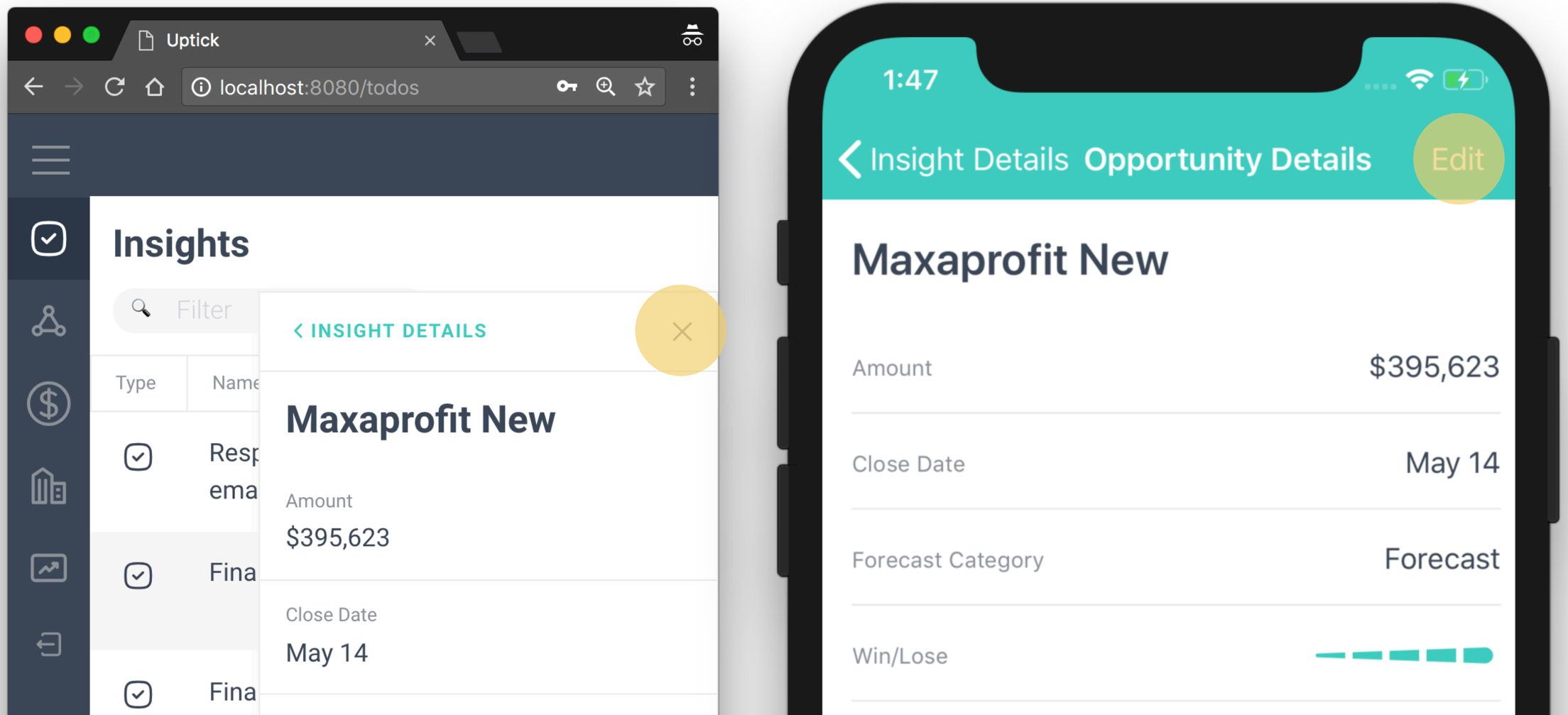
Page Objects

Функциональность - в тот же PO



Page Objects

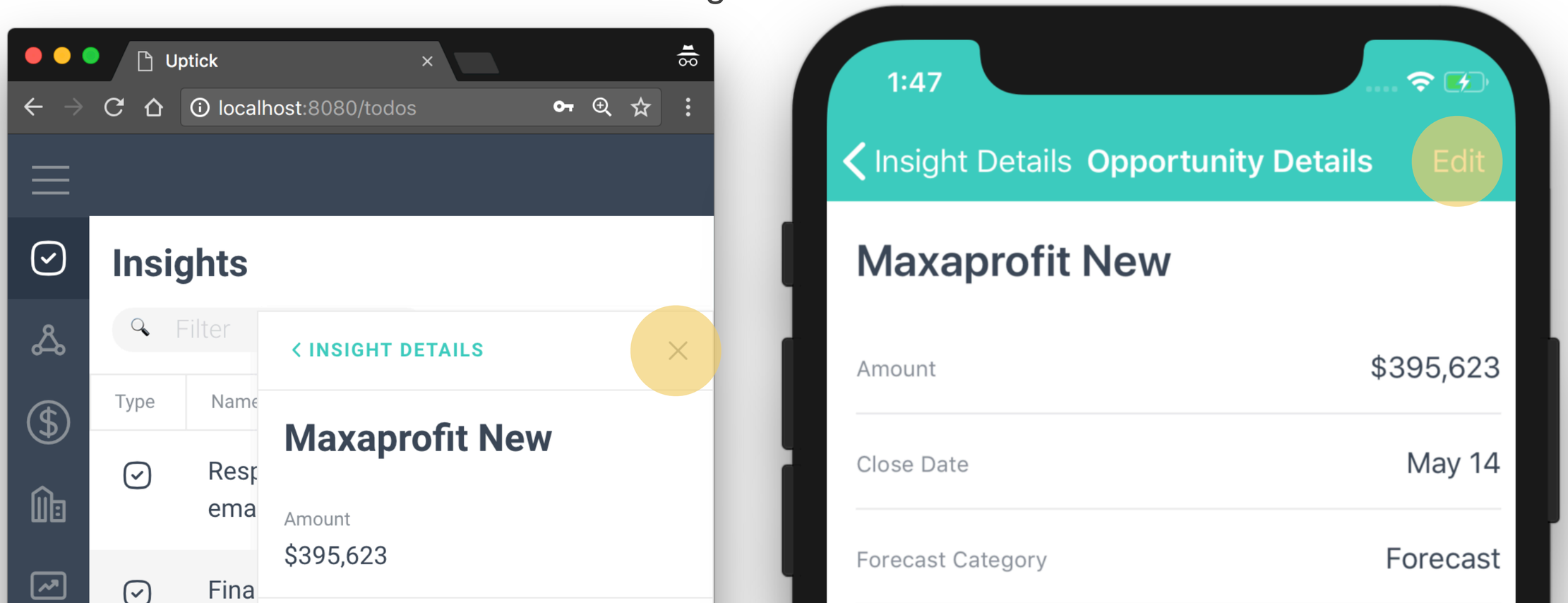
Функциональность - в тот же PO



Page Objects

Функциональность - в тот же PO

- Вызов напрямую в тесте
- На тест - tag/mark



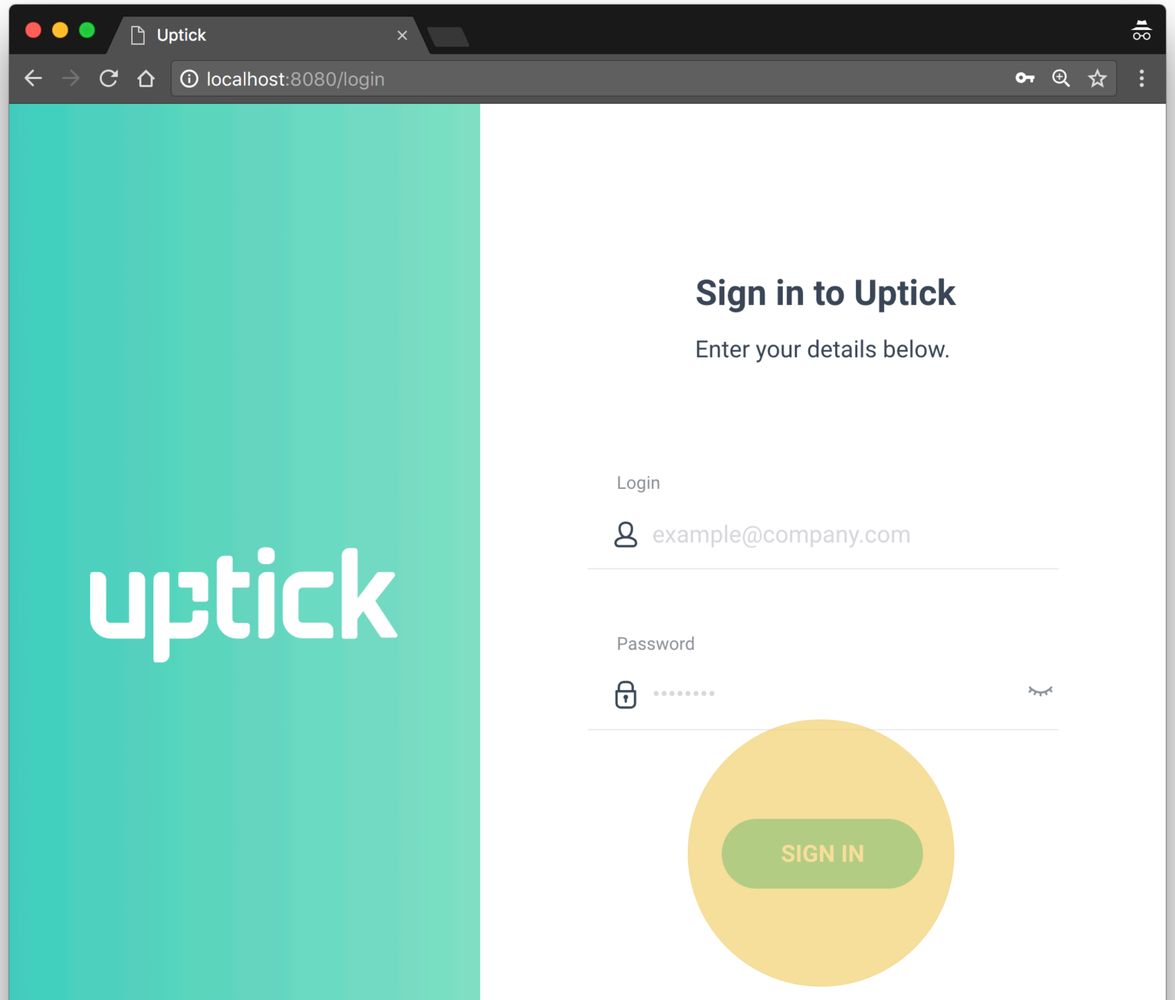
Steps

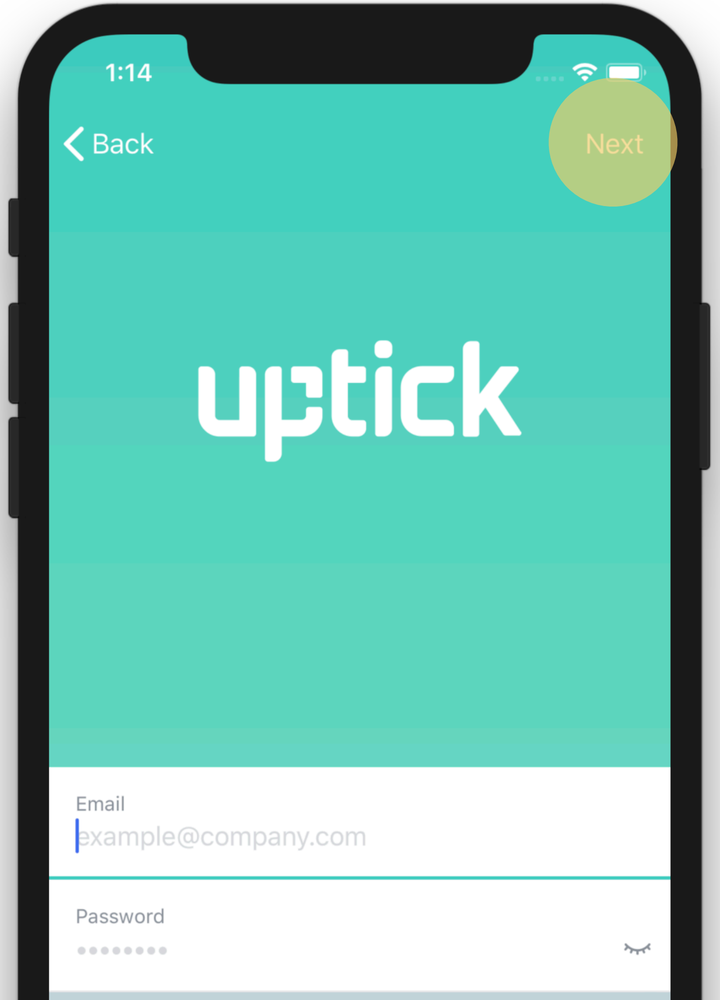
Группировка Actions

```
def login_as(as_user):
    user = config.auth.get_user(as_user)
    login_form.login_with(
        email=user['email'], password=user['password']
    )
    submit_login_form()
```

Steps

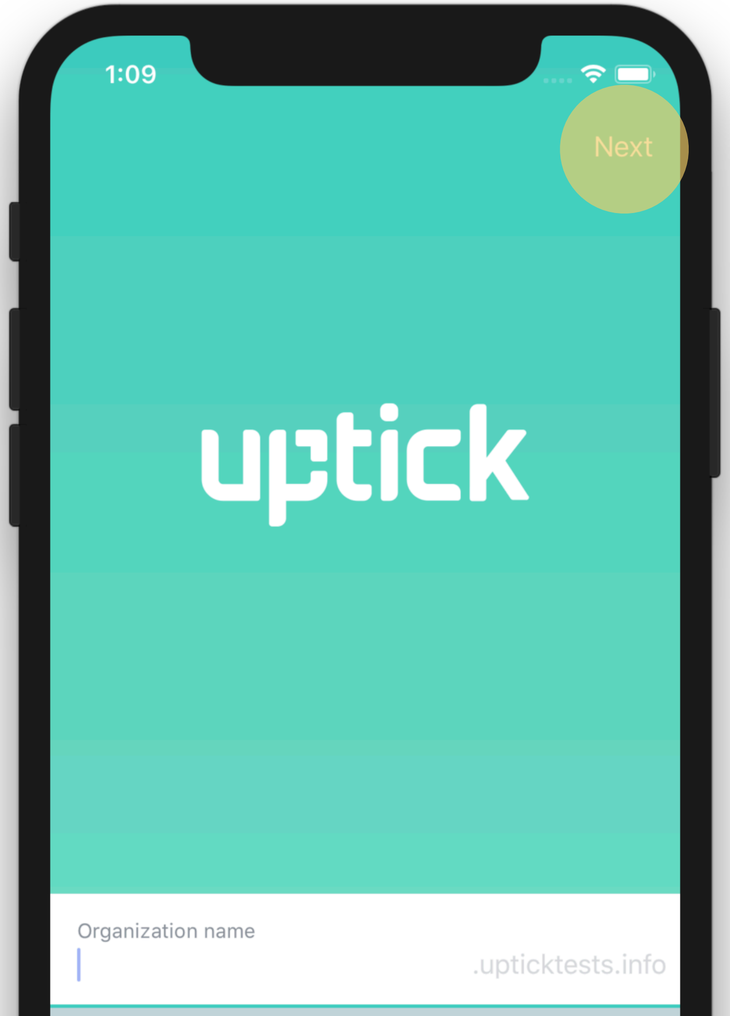
Разный UX

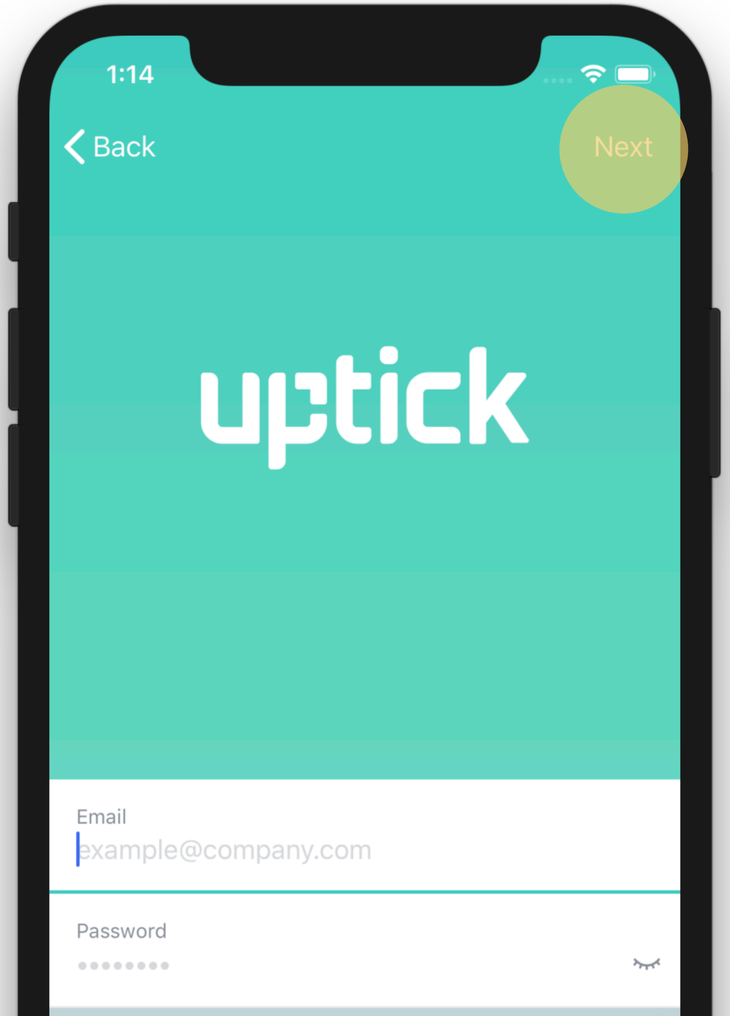




Steps

Разный UX





Steps

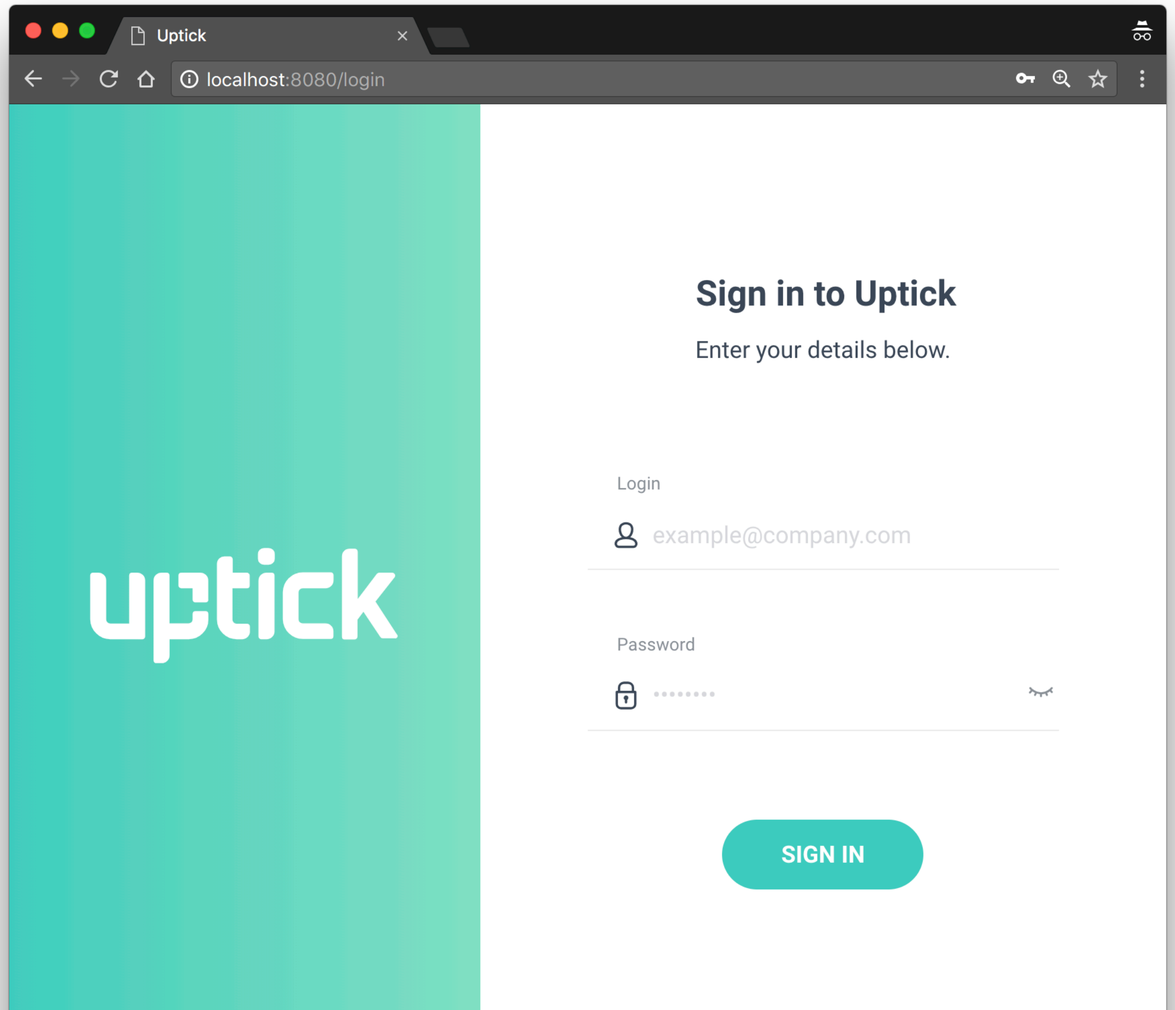
Разный UX

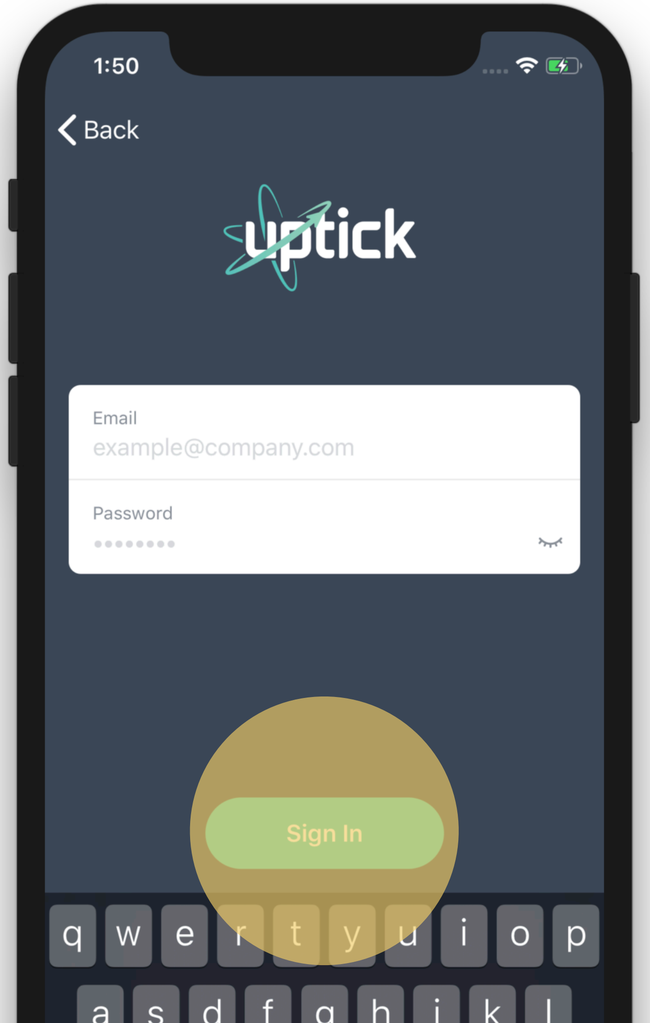
Обычно для iOS и Android разные только локаторы

```
def submit_login_form():
    if config.test_run.IS_MOBILE:
        next_button.click()
    else:
        login_form.submit_button.click()
```

Steps

Разный UX





Steps

Так проще

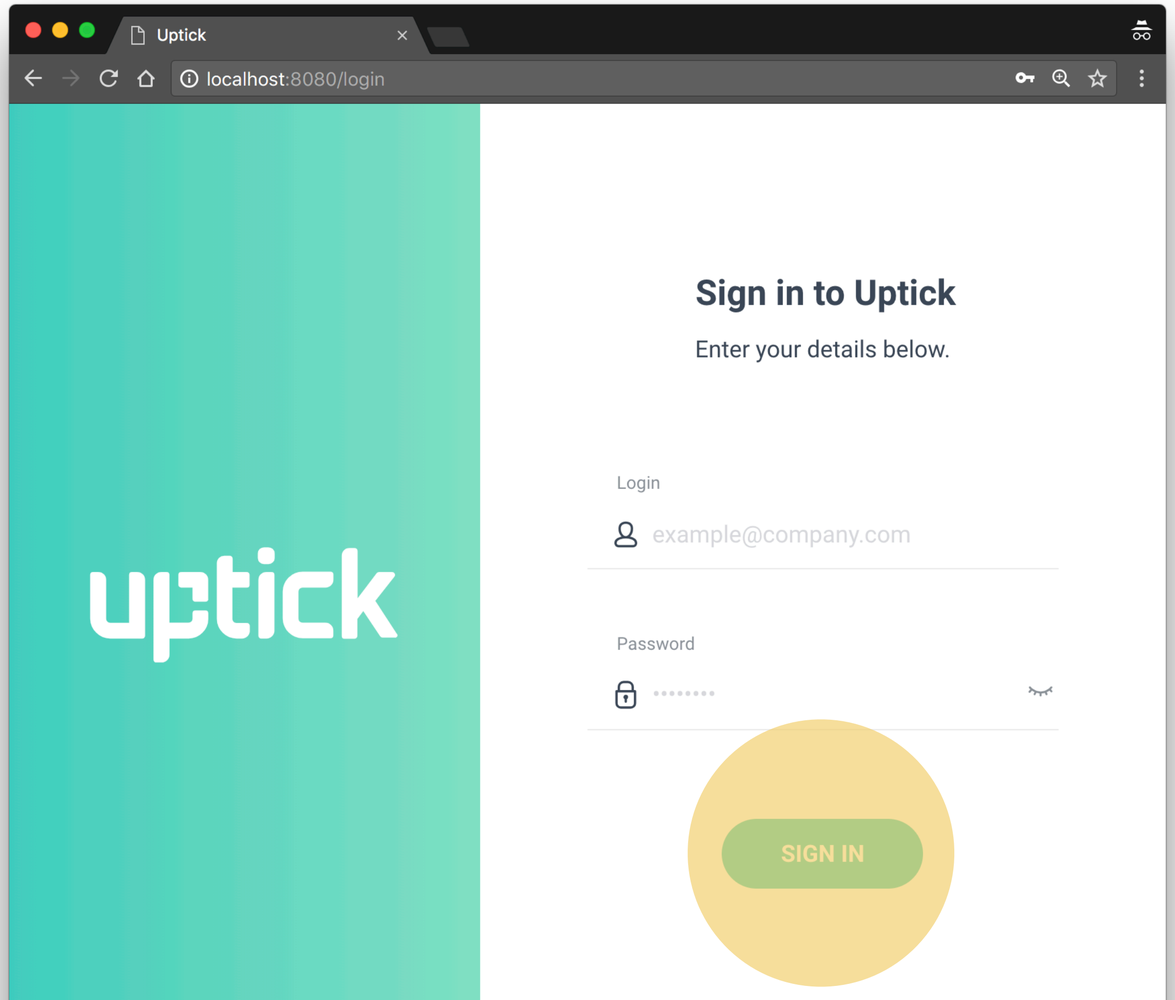
```
def login_as(as_user):
    user = config.auth.get_user(as_user)
    login_form.login_with(
        email=user['email'], password=user['password']
    )
```

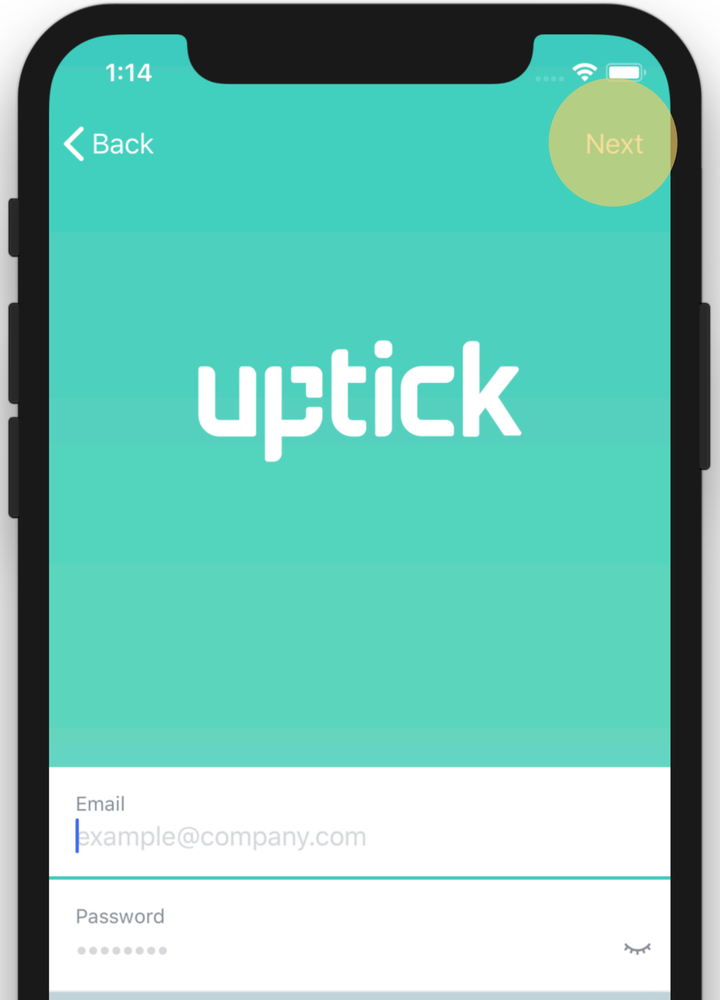
DRY в умеренных количествах

```
def login_with(email, password):
    user_email.set(email)
    user_password.set(password)
    submit_button.click()
```

Steps

Разный UX





Единая точка для работы с приложением

Через application модуль пробросим steps и page objects

```
from pages import login_form
from pages import next_button

login_form = login_form

def submit_login_form(): …
```

Selene

Заворачиваем в селен драйвер

```
from selene import factory

driver = ui_driver.get()

factory.set_shared_driver(driver)
```

На выходе надо «правильно» закрыть

```
ui_driver.close(driver)
```

Добавляем Pytest

```
from testlib.ui import application
```

Добавляем Pytest

```
from testlib.ui import application

@pytest.fixture
def app():
```

Добавляем Pytest

```
from testlib.ui import application

@pytest.fixture
def app():
    driver = ui_driver.get()
    factory.set_shared_driver(driver)

    yield application
```

Добавляем Pytest

```
from testlib.ui import application

@pytest.fixture
def app():
    driver = ui_driver.get()
    factory.set_shared_driver(driver)

    yield application

    ui_driver.close(driver)
```

Сами тесты — одинаковые

```
def test_related_entity_details_list(app, new_user, arrange):
```

Сами тесты — одинаковые

```
def test_related_entity_details_list(app, new_user, arrange):
    opportunity = arrange(entity='Opportunity', ownerId=new_user['userId'])

    insight = arrange(
        entity='Insight', ownerId=new_user['userId'],
        opportunityId=opportunity['id']
    )
```

Сами тесты — одинаковые

```
def test_related_entity_details_list(app, new_user, arrange):
    opportunity = arrange(entity='Opportunity', ownerId=new_user['userId'])

    insight = arrange(
        entity='Insight', ownerId=new_user['userId'],
        opportunityId=opportunity['id']
    )

    app.visit_page(page_name='Insight', as_user=new_user)
```

Сами тесты — одинаковые

```
def test_related_entity_details_list(app, new_user, arrange):
    opportunity = arrange(entity='Opportunity', ownerId=new_user['userId'])

    insight = arrange(
        entity='Insight', ownerId=new_user['userId'],
        opportunityId=opportunity['id']
    )

    app.visit_page(page_name='Insight', as_user=new_user)

    app.list_view.row_with_text(insight['name']).click()
```

Сами тесты — одинаковые

```
def test_related_entity_details_list(app, new_user, arrange):
    opportunity = arrange(entity='Opportunity', ownerId=new_user['userId'])

    insight = arrange(
        entity='Insight', ownerId=new_user['userId'],
        opportunityId=opportunity['id']
    )

    app.visit_page(page_name='Insight', as_user=new_user)

    app.list_view.row_with_text(insight['name']).click()

    app.details_view.related_entity('Opportunity').click()
```

Сами тесты — одинаковые

```
def test_related_entity_details_list(app, new_user, arrange):
    opportunity = arrange(entity='Opportunity', ownerId=new_user['userId'])

    insight = arrange(
        entity='Insight', ownerId=new_user['userId'],
        opportunityId=opportunity['id']
    )

    app.visit_page(page_name='Insight', as_user=new_user)

    app.list_view.row_with_text(insight['name']).click()

    app.details_view.related_entity('Opportunity').click()

    app.details_view.row_value('Amount').should_have(text(opportunity['amount']))
```

Специфичные тесты

Tags

```
@pytest.mark.skipif(config.test_run.IS_WEB, reason=”no team url for web”)
def test_inexistent_team_url(app):
```

Специфичные тесты

Tags

```
@pytest.mark.skipif(config.test_run.IS_WEB, reason=”no team url for web”)
def test_inexistent_team_url(app):
    app.visit_page(page_name=‘Team’)
```

Специфичные тесты

Tags

```
@pytest.mark.skipif(config.test_run.IS_WEB, reason=”no team url for web”)
def test_inexistent_team_url(app):
    app.visit_page(page_name=‘Team’)
    app.team_form.type_url(‘something wrong’)
```

Специфичные тесты

Tags

```
@pytest.mark.skipif(config.test_run.IS_WEB, reason=”no team url for web”)
def test_inexistent_team_url(app):
    app.visit_page(page_name=‘Team’)
    app.team_form.type_url(‘something wrong’)
    app.team_form.error_message.should_have(text(‘Unable to find organisation’))
```

Опасения

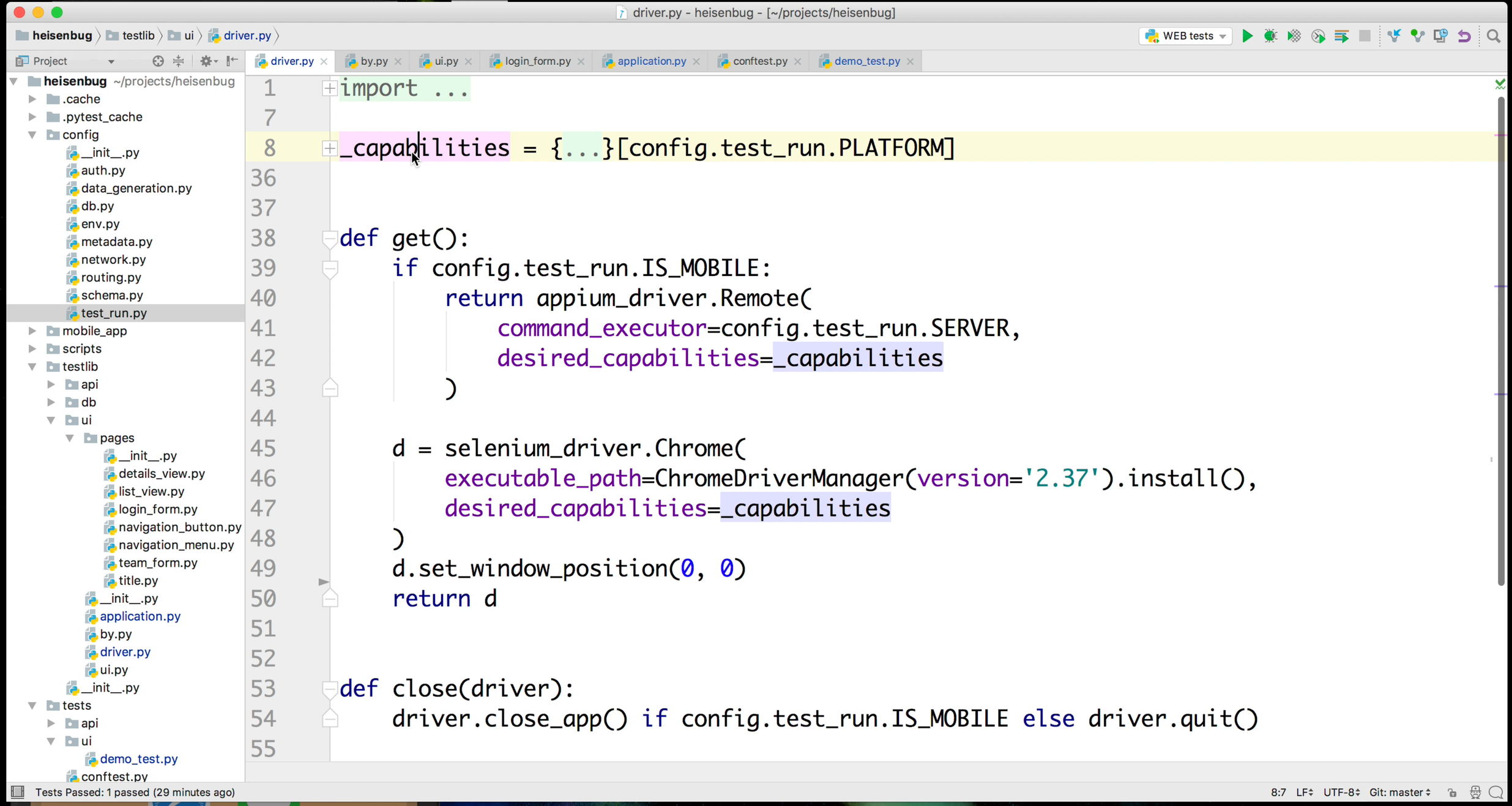
- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- Много отдельного кода для каждой платформы
- Кто так делал?
- Что-то пойдёт не так
- Сложно
- Долго

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~**Много отдельного кода для каждой платформы**~~
- Кто так делал?
- Что-то пойдёт не так
- Сложно
- Долго

Demo Time

- Приложение развёрнуто локально
- API preconditions
- Случайно сгенерированные данные
- Appium, Emulator, Chrome, driver...
- Запускаем тест на iOS / Android / Web
- В тестах есть sleep’ы, чтобы успевать рассказывать


driver.py - heisenbug - [~/projects/heisenbug]
heisenbug
testlib
ui
driver.py
WEB tests
Project
driver.py
by.py
ui.py
login_form.py
application.py
conftest.py
demo_test.py
heisenbug ~/projects/heisenbug
.cache
.pytest_cache
config
__init__.py
auth.py
data_generation.py
db.py
env.py
metadata.py
network.py
routing.py
schema.py
test_run.py
mobile_app
scripts
testlib
api
db
ui
pages
__init__.py
details_view.py
list_view.py
login_form.py
navigation_button.py
navigation_menu.py
team_form.py
title.py
__init__.py
application.py
by.py
driver.py
ui.py
__init__.py
tests
api
ui
demo_test.py
conftest.py
1 import ...
7
8 _capabilities = {...}[config.test_run.PLATFORM]
36
37
38 def get():
39 if config.test_run.IS_MOBILE:
40 return appium_driver.Remote(
41 command_executor=config.test_run.SERVER,
42 desired_capabilities=_capabilities
43)
44
45 d = selenium_driver.Chrome(
46 executable_path=ChromeDriverManager(version='2.37').install(),
47 desired_capabilities=_capabilities
48)
49 d.set_window_position(0, 0)
50 return d
51
52
53 def close(driver):
54 driver.close_app() if config.test_run.IS_MOBILE else driver.quit()
55
Tests Passed: 1 passed (29 minutes ago)
8:7
LF
UTF-8
Git: master

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~Много отдельного кода для каждой платформы~~
- Кто так делал?
- Что-то пойдёт не так
- Сложно
- Долго

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~Много отдельного кода для каждой платформы~~
- ~~**Кто так делал?**~~
- Что-то пойдёт не так
- Сложно
- Долго

Что нужно учесть

- Нужно время чтобы стартануть, выработать практики, поделиться знаниями с командой
- Подумайте о механизме тэгов
- Разбиение на Page Objects требует аккуратности

Грабли мобилок

- React Native + Appium → проблемы с селекторами
- Плохие селекторы → проблемы со скоростью тестов
- Mobile(Appium) медленнее Web’a (Selenium)

Особенности Python

KISS Test Automation

- Почти всегда можно обойтись без классов (структуры данных, named tuples, data classes)
- В 98% случаев в тестовом проекте
- Jack BOS Diederich (Python Core Developer) http://pyvideo.org/pycon-us-2012/stop-writing-classes.html

Особенности Selene

- Есть issues, но не более
- Не страшно смотреть в исходники
- Модульный - можно взять только то, что нужно
- Работает с Appium
- Selenide API

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~Много отдельного кода для каждой платформы~~
- ~~Кто так делал?~~
- Что-то пойдёт не так
- Сложно
- Долго

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~Много отдельного кода для каждой платформы~~
- ~~Кто так делал?~~
- ~~**Что-то пойдёт не так**~~
- Сложно
- Долго

Сложно?

- Берём Selenium и Appium, оборачиваем
- Page Objects → часть экрана в мобильном приложении, разные локаторы
- UX → steps
- Специфичные сценарии → tags

Нужны хитрые инструменты?

Подход не зависит от стэка технологий

- Основное - правильно разбить на слои проект
- Обернуть драйверы (или сделать велосипед)
- Немного специфики добавляет тест раннер

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~Много отдельного кода для каждой платформы~~
- ~~Кто так делал?~~
- ~~Что-то пойдёт не так~~
- Сложно
- Долго

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~Много отдельного кода для каждой платформы~~
- ~~Кто так делал?~~
- ~~Что-то пойдёт не так~~
- ~~**Сложно**~~
- Долго

Какие-то цифры:

- Не было опыта Python и Mobile (был Ruby и Web)
- ≈50 тестов на каждую платформу Web, iOS и Android
- 2 man-months (можно быстрее)
- > 3 тестов в день
- Один человек может развивать и поддерживать*
- В один поток тесты бегут 6 минут для web и 30 минут для iOS, Android

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~Много отдельного кода для каждой платформы~~
- ~~Кто так делал?~~
- ~~Что-то пойдёт не так~~
- ~~Сложно~~
- Долго

Опасения

- ~~Selenium wrapper развалится от Appium driver~~
- ~~Переходы между экранами~~
- ~~Много отдельного кода для каждой платформы~~
- ~~Кто так делал?~~
- ~~Что-то пойдёт не так~~
- ~~Сложно~~
- ~~**Долго**~~

Итоги

Плюсы подхода

- Разработка тестов сразу для всех трёх платформ
- Меньше времени на поддержку, особенно в случае изменения бизнес-сценариев
- Единая точка для анализа разного UX и покрытия сценариев на разных платформах
- Легко переиспользовать «служебный» код

Итоги

Минусы подхода

- Можно задеть поведение на другой платформе
- Необходимо выделить слои
- Есть лишний слой (решение для разного UX)

Когда имеет смысл использовать

Когда имеет смысл использовать

Есть значимая часть сценариев для всех платформ

Don’t Repeat Yourself

Automate them all!

- Экономьте время
- Взгляните на приложение с разных сторон

Спасибо за внимание!

Игорь Балагуров
igor.balagurov@uptick.com
ibalagurov