Тестирование: Авто/Нагрузка/Трекинг

ФИТ Лекция №11 - 12. 25’

Поговорим о том, что такое автоматизация тестирования и багтрекинг

Автотесты

- Полезность для проекта и руководителя проекта
- Как сформировать команду автотестеров
- Инструментарий работы с автотестами

Нагрузочное тестирование:

- Нагрузка/ производительность - отдельный вид автотестов со своим инструментарием
- Как помогает и что в действительности обнаруживает
- Какими инструментами делать

Багтрекинг и планирование работ:

- Откуда появилась такая необходимость.
- В чем особенность работы и планирования работы команды с багтрекингом
- В каких инструментах работать

Автотесты - зачем нужны в принципе

Ключевые преимущества автотестов:

Экономия времени: автоматизация позволяет проводить тесты быстрее, чем вручную и в не рабочее время

Повышение точности: исключение человеческого фактора (замыливания глаз) минимизирует ошибки.

Повторяемость: возможность многократного использования тестов на разных этапах разработки.

Сферы применения автотестов:

Регрессионное тестирование для проверки, что новые изменения не ломают существующий функционал.

Тестирование производительности для оценки стабильности под нагрузкой.

Интеграционное тестирование для проверки взаимодействия различных модулей.

Автотесты - как помогают ПМ-у

Обеспечение предсказуемости сроков:

- Автоматизация позволяет сократить время на поиск ошибок, так как тесты **проходят быстрее**, чем это может сделать вручную тестировщик.
- Регулярное выполнение автотестов **в рамках CI/CD** помогает команде быстро получать обратную связь и реагировать на возможные проблемы. Снижает накопление ошибок.
- Регрессионное тестирование с использованием автотестов **защищает стабильность** существующего функционала при внедрении новых изменений.
- Раннее обнаружение критических ошибок

Прогнозирование ресурсов:

- Сокращение времени на выполнение рутинных проверок освобождает специалистов для работы над сложными задачами, такими как анализ архитектуры или исследовательское тестирование.
- Автотесты помогают распределять усилия команды более эффективно, так как позволяют оценить нагрузку и распределить задачи между разработчиками и тестировщиками.

Автотесты - выбираем стек типовых технологи и фреймоворков

Для корректного выбора стека технологий требуется оценить следующие параметры

- Масштаб проекта
- Основной язык разработки в проекте
- Тип проекта (Web, Desktop, МП)
- Уровень подготовки команды тестирования

К выбору стека стоит подойти ответственно, так как сменить стек после написания достаточного количества тестов становится слишком дорогостоящим мероприятием, включающим следующие шаги:

Обучение тестировщиков новому стеку (дорого)

Переписывание текущих тестов на новый стек (очень дорого в плане ресурсов)

Изменение CI/CD процессов под новый стек (ещё одни дополнительные затраты на DevOps)

Языки – краткий обзор

Автотесты - Python

Популярный, большая вероятность встретить на работающих проектах.

Преимущества:

- Простота изучения и использования.
- Большой выбор библиотек для работы с автоматизированными тестами (например, Selenium, Pytest, unittest).
- Высокая скорость написания тестов.
- Хорошая поддержка тестирования как веб-приложений (с использованием Selenium и других инструментов), так и API.

Фреймворки и библиотеки:

- **Pytest**: Один из самых популярных фреймворков для написания юнит-тестов. Очень гибкий и легко расширяемый.
- **Selenium**: Используется для автоматизации браузеров. Подходит для тестирования пользовательского интерфейса.
- **unittest**: Встроенная библиотека для написания тестов в Python, аналогична JUnit в Java.
- **Robot Framework**: Используется для создания автоматизированных тестов с использованием ключевых слов, хорошо проверяет интеграции.

Пример автотеста для проверки успешного логина на Python

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time

# Инициализация браузера
browser = webdriver.Chrome()

try:
    # Открытие страницы логина
    browser.get("https://example.com/login")

    # Ввод имени пользователя
    username_input = browser.find_element(By.NAME, "username")
    username_input.send_keys("test_user")

    # Ввод пароля
    password_input = browser.find_element(By.NAME, "password")
    password_input.send_keys("password123")

    # Нажатие кнопки "Войти"
    login_button = browser.find_element(By.NAME, "login")
    login_button.click()

    # Ожидание загрузки страницы
    time.sleep(2)

    # Проверка успешной авторизации
    welcome_text = browser.find_element(By.XPATH, "//h1[contains(text(), 'Добро пожалс
    assert welcome_text is not None, "Текст приветствия не найден, авторизация не удал

finally:
    # Закрытие браузера
    browser.quit()
```

Автотесты - JavaScript

Очень востребован для тестирования фронтенд-приложений, так как JavaScript активно используется для разработки веб-сайтов.

Преимущества:

- Интеграция с браузером. Тесты можно писать прямо для JavaScript-коде, что упрощает тестирование фронтенда.
- Большая экосистема фреймворков и библиотек.
- Быстрая обратная связь с клиентом, что важно для веб-разработки.

Фреймворки и библиотеки:

- **Jest:** Популярный фреймворк для тестирования JavaScript, в том числе для React-приложений.
- **Mocha:** Гибкий фреймворк для юнит-тестирования с возможностью интеграции с другими библиотеками (например, Chai для утверждений).
- **Cypress:** Современный инструмент для тестирования веб-приложений, предоставляющий доступ к разработке и отладке тестов в реальном времени.
- **Puppeteer:** Используется для тестирования приложений, взаимодействующих с браузером (с помощью API для Chrome).

Автотесты - Java

Один из самых популярных языков для автоматизации тестирования в крупных проектах и enterprise-системах.

Преимущества:

- Отличная поддержка многозадачности и параллельного выполнения тестов.
- Богатая экосистема фреймворков для тестирования.
- Легко интегрируется с системами контроля версий, CI/CD и облачными решениями.

Фреймворки и библиотеки:

- **JUnit:** Основной фреймворк для юнит-тестирования в Java, широко используется для модульных тестов.
- **TestNG:** Подобен JUnit, но предлагает больше возможностей для настройки тестов, таких как параллельное выполнение.
- **Selenium:** Для автоматизации UI-тестирования.
- **RestAssured:** Для тестирования REST API, поддерживает удобное создание запросов и проверку ответов.
- **Appium:** Инструмент для автоматизированного тестирования мобильных приложений (iOS и Android).

Автотесты - Kotlin

Получил популярность благодаря своей совместимости с Java и использованию для разработки мобильных приложений на Android.

Преимущества:

- Сильная интеграция с Java, возможность использовать все фреймворки и инструменты, созданные для Java.
- Легкость в использовании, с улучшенным синтаксисом по сравнению с Java.

Фреймворки и библиотеки:

- **JUnit и TestNG** (совместим с Java фреймворками).
- **KotlinTest:** Специализированный фреймворк для тестирования на Kotlin.

Пример на Java

```java
import com.codeborne.selenide.Condition;
import com.codeborne.selenide.Selenide;
import org.junit.jupiter.api.Test;

import static com.codeborne.selenide.Selenide.*;

public class LoginTest {

    @Test
    public void testLogin() {
        // Открываем страницу
        open("https://example.com/login");

        // Вводим логин и пароль
        $("#username").setValue("testuser");
        $("#password").setValue("password123");

        // Нажимаем кнопку "Войти"
        $("#login-button").click();

        // Проверяем, что после входа отображается сообщение "Добро пожаловать"
        $(".welcome-message").shouldHave(Condition.text("Добро пожаловать!"));
    }
}
```

Автотесты - C#

Часто используется в экосистемах Microsoft для автоматизации тестирования на платформе .NET

Преимущества:

- Полная интеграция с Visual Studio и другими инструментами Microsoft.
- Высокая производительность.
- Прекрасная поддержка для разработки Windows-приложений и веб-приложений.

Фреймворки и библиотеки:

- **NUnit:** Основной фреймворк для юнит-тестирования, аналог JUnit.
- **xUnit:** Большие возможности для интеграции.
- **SpecFlow:** Для поведенческого тестирования (BDD).
- **Selenium:** Используется для автоматизации UI-тестов в браузерах.
- **Appium:** Для мобильных приложений.

Пример на C#

```
namespace WebAutomationTests
{
    public class LoginTest
    {
        private IWebDriver driver;
        private LoginPage loginPage;

        [SetUp]
        public void Setup()
        {
            driver = new ChromeDriver();
            driver.Manage().Window.Maximize();
            driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
            loginPage = new LoginPage(driver);
        }

        [Test]
        public void TestLogin()
        {
            driver.Navigate().GoToUrl("https://example.com/login");
            loginPage.EnterUsername("testuser");
            loginPage.EnterPassword("password123");
            loginPage.ClickLoginButton();
            Assert.AreEqual("Добро пожаловать!", loginPage.GetWelcomeMessage());
        }

        [TearDown]
        public void Teardown()
        {
            driver.Quit();
        }
    }
```

Автотесты - Ruby

Используется в основном в стартапах и для тестирования веб-приложений.

Преимущества:

- Является частью экосистемы Ruby on Rails, что делает его подходящим для тестирования приложений, написанных на этом фреймворке.
- Простой синтаксис, легко осваиваемый.

Фреймворки и библиотеки:

- **RSpec:** Чаще всего используемый фреймворк для написания тестов на Ruby.
- **Capybara:** Для тестирования веб-приложений с использованием взаимодействий с браузером.
- **Cucumber:** Используется для BDD-тестирования, поддерживает написание тестов на обычном языке (в стиле "Given, When, Then").
- **Selenium:** Для автоматизации браузеров.

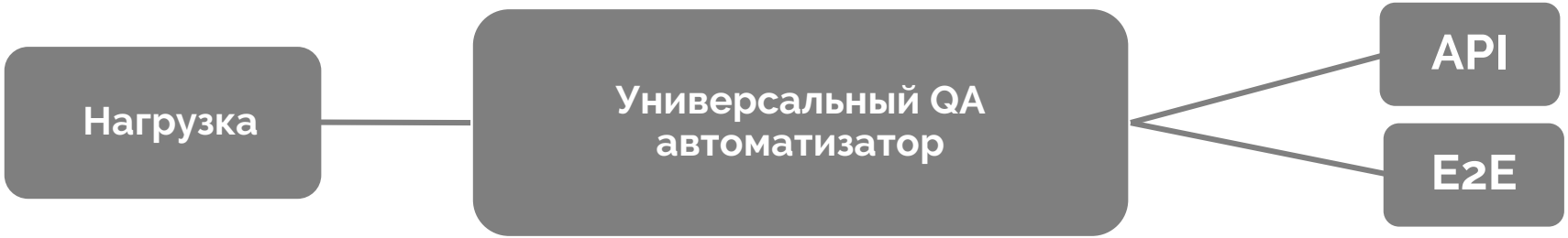
Команда автотестировщиков

Формируем команду автотестировщиков

Формирование команды тестирования — это важный процесс, который требует учета различных факторов, включая тип проекта, масштаб задач и опыт сотрудников. Для успешной автоматизации тестирования необходимо собрать команду с разнообразными навыками и компетенциями.

Структура команды

Команда автотестировщиков может быть организована в разных структурах в зависимости от размера компании и проекта. В малых командах несколько ролей могут быть объединены, в крупных — каждый тестировщик может отвечать за узкую область (например, только тестирование API или только тестирование UI).



Команда

Тестировщик-автоматизатор (Automation Tester)

Основные обязанности: Написание, поддержка и выполнение автоматизированных тестов.

Необходимые навыки:

- Знание языков программирования (из рассмотренных ранее; например, Python, Java, JavaScript).
- Опыт работы с фреймворками для автоматизации тестирования (Selenium, Cypress, Appium и другие).
- Умение работать с системами контроля версий (Git).
- Опыт интеграции тестов в CI/CD пайплайны.
- Знание принципов TDD (Test-Driven Development) или BDD (Behavior-Driven Development).

Команда

Инженер по тестированию API (API Tester)

Основные обязанности: Опыт работы с инструментами для тестирования API (Postman, Insomnia, RestAssured, SoapUI, можно использовать и Cypres). Знания HTTP, JSON, REST и SOAP. Опыт в автоматизации тестирования API.

Необходимые навыки:

- Опыт работы с инструментами для тестирования API (Postman, Insomnia, RestAssured, SoapUI, можно использовать и Cypres).
- Знания HTTP, JSON, REST и SOAP.
- Опыт в автоматизации тестирования API.

Инженер по производительности (Performance Tester)

Основные обязанности: Тестирование производительности приложения, включая нагрузочные тесты, стресс-тесты, тесты на отказоустойчивость.

Необходимые навыки:

- Опыт работы с инструментами для нагрузочного тестирования (JMeter, Gatling, LoadRunner).
- Знания принципов работы с большими объемами данных и виртуализацией.
- Опыт анализа метрик производительности и поиска узких мест.

Команда

Архитектор автоматизации (Test Automation Architect)

Основные обязанности: проектирование архитектуры тестирования, выбор инструментов для автоматизации, руководство командой автоматизаторов.

Необходимые навыки:

- Знание различных фреймворков и подходов к автоматизации.
- Опыт проектирования гибких и масштабируемых тестовых систем.
- Опыт интеграции тестов в CI/CD пайплайны.
- Опыт работы с различными типами тестов (функциональные, нагрузочные, UI и т.д.)

Мультиинструменталист

Основные обязанности: выполнение функций всех перечисленных ролей. На практике это вполне выполнимая задача, если проект небольшой.

Пример из жизни:

Один QA выполняет ручное тестирование, следит за стратегией тестирования, параллельно настраивает фреймворки, знает JS и пишет автотесты.

За 8 месяцев работы небольшой проект обзавёлся автоматизированным регрессом и встройкой в CI/CD автотестов + сохранено мануальное тестирование для проверки новой функциональности.

Автотесты – основные технологии и фреймворки

Инструменты для автоматизации функциональных тестов и E2E

Selenium

Один из самых популярных инструментов для автоматизации тестирования веб-приложений. Он поддерживает множество языков программирования, включая Java, Python, C#, JavaScript, Ruby.

Cypress

Инструмент для тестирования фронтенда, ориентированный на JavaScript и предназначенный для быстрой и надежной автоматизации тестов с удобной отладкой по ходу выполнения тестов

Playwright

Инструмент от Microsoft для тестирования веб-приложений, похожий на Cypress, но с рядом улучшений

TestCafe

Инструмент для тестирования веб-приложений на основе JavaScript. В отличие от Selenium, TestCafe не требует установки дополнительных драйверов для браузеров.

Инструменты для автоматизации тестирования API

Postman

Популярный инструмент для тестирования REST API. Он позволяет отправлять HTTP-запросы и проверять ответы, имеет возможность автоматизации тестирования с помощью коллекций и скриптов

RestAssured

Java-библиотека для автоматизированного тестирования REST API, поддерживает BDD-стиль, работает с json и xml.

SoapUI

Инструмент для тестирования как REST, так и SOAP API, так же есть возможность тестирования производительности и нагрузки

Инструменты для автоматизации тестирования **мобильных приложений**

Appium

фреймворк с открытым исходным кодом для тестирования мобильных приложений на Android и iOS, может работать с нативными, гибридными и мобильными веб-приложениями

Espresso

Фреймворк для тестирования Android-приложений, имеет интеграцию с Android studio

Инструменты для автоматизации тестирования **десктопных проектов**

WinAppDriver

Инструмент для тестирования десктопных приложений на Windows, разработанный Microsoft.

Winium

Расширение Selenium для автоматизации Windows-приложений.

Pywinauto

Библиотека для автоматизации GUI-тестирования приложений на Windows

SikuliX

Инструмент для автоматизации через распознавание изображений, подходит для тестирования интерфейсов.

Инструменты для тестирования **производительности**

Apache JMeter

Один из самых популярных инструментов для тестирования производительности, нагрузки и стресс-тестирования.

Gatling

Инструмент для тестирования производительности, известен своей высокой производительностью, основан на языке Scala, что позволяет эффективно управлять тестами

Инструменты для CI/CD

Jenkins

Один из самых популярных инструментов для автоматизации процессов сборки и тестирования в CI/CD.

Особенности:

Поддержка множества плагинов для интеграции с различными системами и инструментами.

Возможности для параллельного выполнения тестов.

GitLab CI

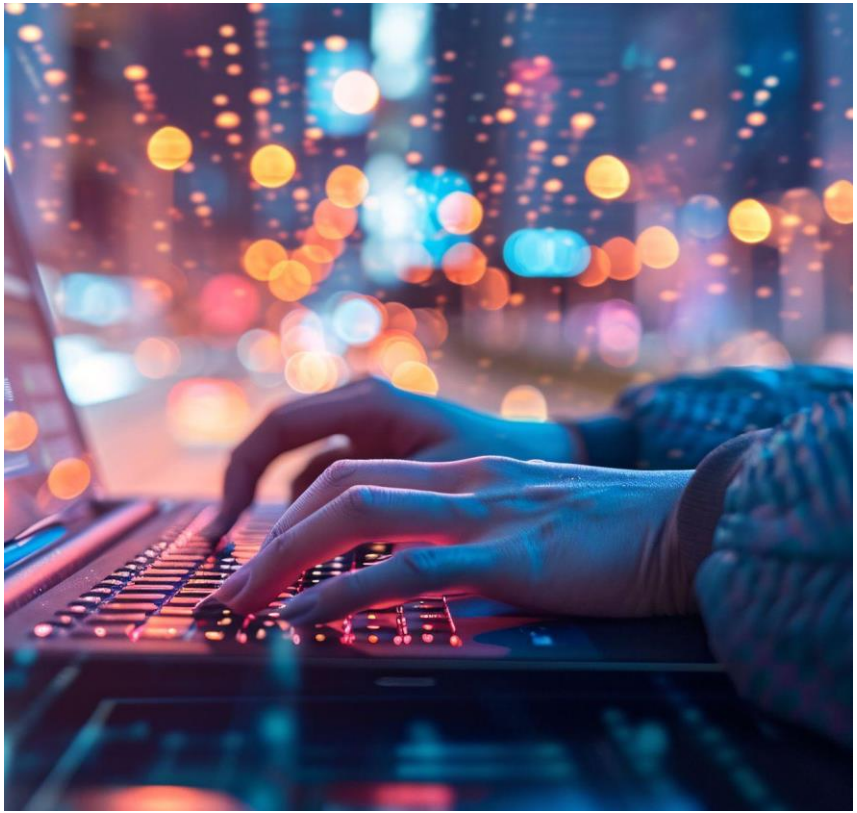
Встроенная в GitLab система для CI/CD.

Особенности:

Поддержка конфигурации пайплайнов через YAML.

Интеграция с репозиториями GitLab и другими инструментами.

Когда использовать: Для автоматизации процессов сборки, тестирования и деплоя в проектах, использующих GitLab.



Автотесты на примере WEB-технологий

Автотесты на примере веб-технологий

Почему WEB-приложения нуждаются в автотестах?

- **Высокая динамика изменений**: регулярные обновления интерфейса и функционала.
- **Разнообразие устройств и браузеров**: необходимость кросс-браузерного и кросс-платформенного тестирования.
- **Интеграции с внешними сервисами**: API, платежные системы, базы данных.

Пример автотеста для проверки успешного логина на Python

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time

# Инициализация браузера
browser = webdriver.Chrome()

try:
    # Открытие страницы логина
    browser.get("https://example.com/login")

    # Ввод имени пользователя
    username_input = browser.find_element(By.NAME, "username")
    username_input.send_keys("test_user")

    # Ввод пароля
    password_input = browser.find_element(By.NAME, "password")
    password_input.send_keys("password123")

    # Нажатие кнопки "Войти"
    login_button = browser.find_element(By.NAME, "login")
    login_button.click()

    # Ожидание загрузки страницы
    time.sleep(2)

    # Проверка успешной авторизации
    welcome_text = browser.find_element(By.XPATH, "//h1[contains(text(), 'Добро пожалс
    assert welcome_text is not None, "Текст приветствия не найден, авторизация не удал

finally:
    # Закрытие браузера
    browser.quit()
```

Зачем нужна PageObjectModel

Проблема:

При больших объемах кода автотестирования требуется много времени на поддержку, код не читаем, и много повторяющихся элементов.

Решение 1: забросить автотесты

Решение 2: каждый элемент страницы или блок на странице представить отдельным объектом, и, при изменении UI, менять код в одном месте, внутри объекта блока страницы.

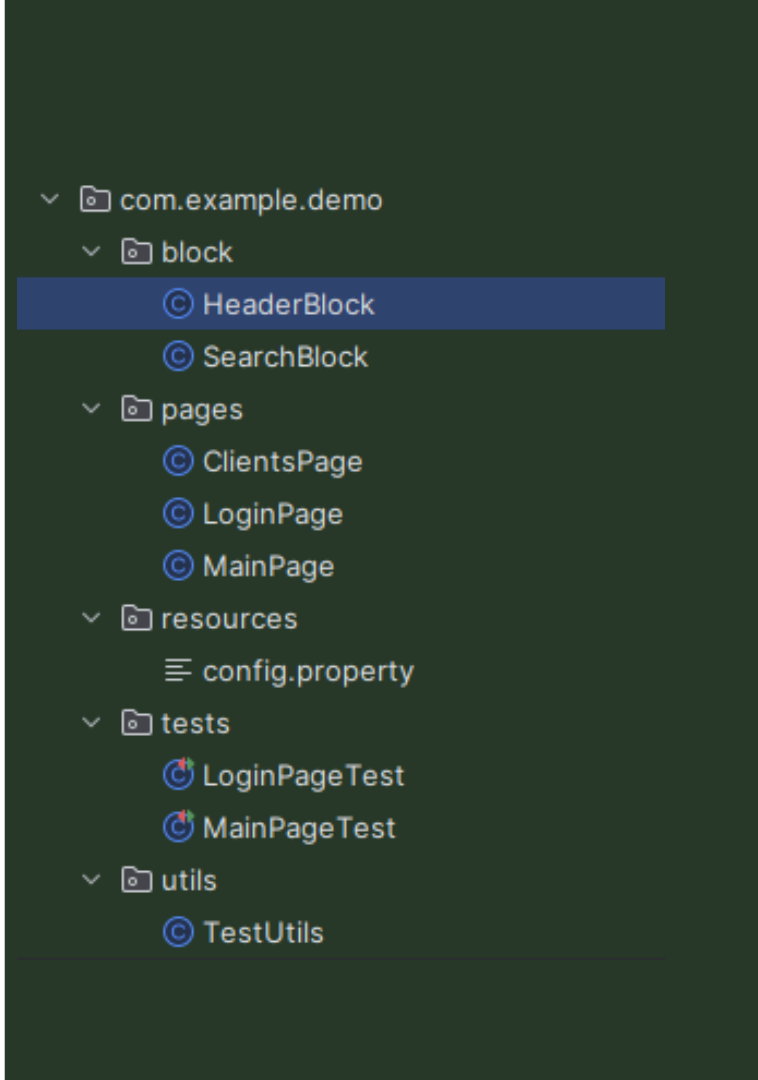
Зачем нужна PageObjectModel

Повторное использование кода
Тесты могут использовать одни и те же классы для взаимодействия с элементами страниц.

Чистота и читаемость тестов: Тесты становятся более читаемыми и абстрагированными от деталей

Легкость в изменении тестов: достаточно обновить общий блок или страницу.

Удобство тестирования различных страниц: Каждая новая страница может быть собрана из имеющихся элементов



E2E (End-To-End) - автотесты

Это тип тестирования программного обеспечения, при котором **проверяется работа приложения в целом**, начиная от пользовательского интерфейса (фронтенда) и заканчивая базой данных и бэкендом.

Цель E2E тестов — убедиться, что все части системы взаимодействуют друг с другом корректно и что пользователь может выполнить свои задачи от начала до конца, как в реальной ситуации.

Например чтобы покрыть все части системы (интерфейс, логику аутентификации, сервер и базу данных) E2E тесты будут включать следующие шаги:

1. **Открыть страницу логина**
2. **Ввести правильные данные (логин и пароль)**
3. **Нажать кнопку "Войти".**
4. **Проверить, что после успешного входа пользователь перенаправляется на главную страницу.**
5. **Проверить, что на главной странице отображается имя пользователя.**
6. **И негативный сценарий с корректной обработкой ошибки**

Пример позитивного и негативного E2E

```
describe( title: 'Login Page',  fn: () : void  => {
    beforeEach( fn: () : void  => {
        // Открытие страницы логина перед каждым тестом
        cy.visit( url: 'https://example.com/login');
    });

    it( title: 'should log in successfully with correct credentials',  config: () : void  => {
        // Ввод правильных данных
        cy.get( selector: 'input[name="username"]').type( text: 'correctUsername');
        cy.get( selector: 'input[name="password"]').type( text: 'correctPassword');

        // Клик по кнопке входа
        cy.get( selector: 'button[type="submit"]').click();

        // Проверка, что после входа пользователь перенаправлен на главную страницу
        cy.url().should( chainer: 'eq',  value: 'https://example.com/home');
        cy.get( selector: 'h1').should( chainer: 'contain.text',  value: 'Welcome, correctUsername');
    });

    it( title: 'should show an error with incorrect credentials',  config: () : void  => {
        // Ввод неправильных данных
        cy.get( selector: 'input[name="username"]').type( text: 'incorrectUsername');
        cy.get( selector: 'input[name="password"]').type( text: 'incorrectPassword');

        // Клик по кнопке входа
        cy.get( selector: 'button[type="submit"]').click();

        // Проверка, что на экране отображается сообщение об ошибке
        cy.get( selector: '.error-message').should( chainer: 'be.visible').and( chainer: 'contain.text',  value: 'Invalid username or password');
    });
});
```

UI - автотесты

UI-тестирование (или тестирование пользовательского интерфейса) фокусируется только на проверке компонентов и элементов пользовательского интерфейса (кнопки, формы, поля ввода, текстовые блоки, стили, элементы управления и т.д.). **UI тесты проверяют, правильно ли отображаются и работают эти элементы в рамках интерфейса.**

Например:
UI тесты могут проверять, правильно ли отправляется форма, показывается ли кнопка, активна ли она, или выполняются ли анимации.

Принципы UI тестирования:

Действия пользователя:

Автоматические тесты должны имитировать реальные действия пользователя: клики, ввод данных, прокрутка страницы, наведение курсора и другие действия.

Тестирование в разных браузерах:

UI-тесты часто проверяют, как приложение выглядит и функционирует в разных браузерах, включая Chrome, Firefox, Safari, Internet Explorer и другие.

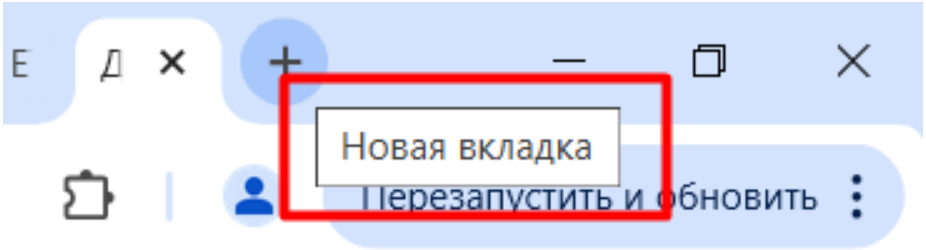
Проверка на различных разрешениях экрана:

Некоторые инструменты могут тестировать интерфейс на различных разрешениях, что особенно важно для адаптивных дизайнов.

Проверка визуальных изменений:

Проверка того, что элементы UI правильно отображаются, без ошибок в верстке или других визуальных дефектов.

UI - автотесты



Например появление подсказки «Новая вкладка» можно сделать одним из шагов проверки UI браузера

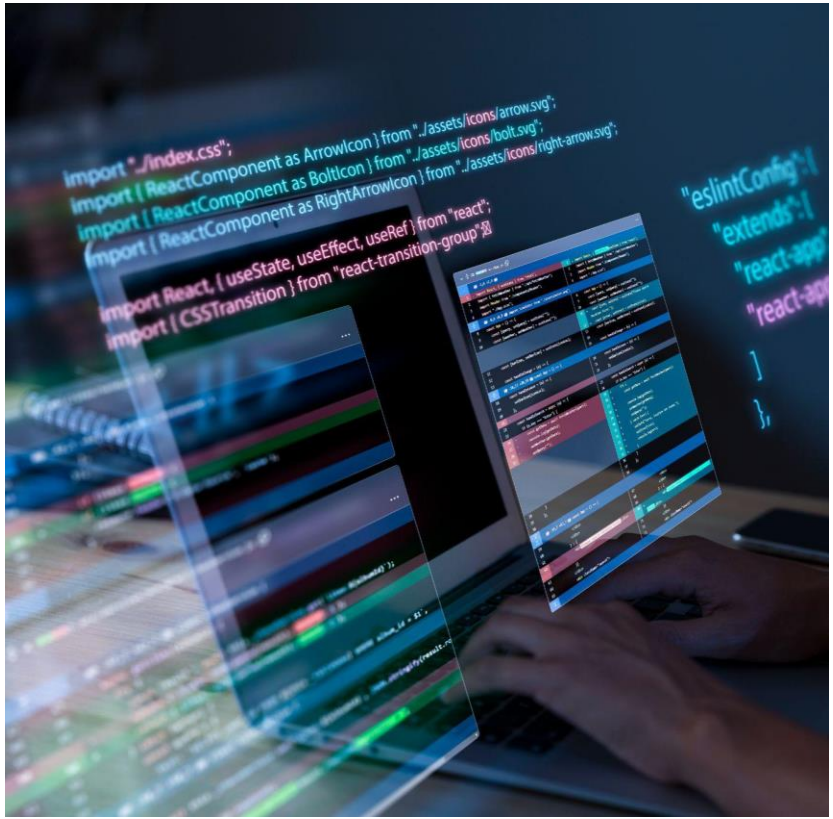
UI-тестирование можно как совмещать с E2E, добавляя требуемые ожидания определенных анимаций, или параллельное навешивание ожиданий на внешний вид кнопок, как и сделать отдельными тестами, для быстрого прогона посреди цикла разработки

UI - автотесты

Попиксельное сравнение UI (или **Pixel-Perfect Testing**) — это метод тестирования, при котором проверяется точное соответствие визуальных компонентов веб-приложения или мобильного приложения заданному дизайну.

Цель этого подхода — убедиться, что каждый элемент интерфейса отображается на экране точно так, как задуман дизайнером, с учетом каждого пикселя.

Существуют неплохие решения с открытым кодом, которые можно использовать.



Фреймворки для E2E тестирования WEB-приложений

Selenium

Один из самых популярных фреймворков для E2E тестирования.

Он поддерживает множество браузеров (Chrome, Firefox, Internet Explorer) и языков программирования (Java, Python, C#, Ruby, JavaScript).

Особенности:

- Открытый исходный код.
- Поддержка разных браузеров.
- Широкая экосистема плагинов и инструментов.

Cypress

Современный фреймворк для E2E тестирования, который работает непосредственно в браузере.

Отличается высокой производительностью и удобством для разработчиков.

Особенности:

- Работает только с браузером Chrome, но поддерживает тестирование в других браузерах.
- Встроенная возможность отладки.
- Простой API и быстрая интеграция с CI/CD.

Playwright

Современный фреймворк от Microsoft для автоматизации тестирования, который поддерживает несколько браузеров, включая Chrome, Firefox и WebKit.

Особенности:

- Поддерживает тестирование на мобильных устройствах.
- Простой в использовании и мощный API.
- Встроенная поддержка параллельного выполнения тестов.

TestCafe

Фреймворк для E2E тестирования, который не требует установки дополнительных браузерных драйверов и позволяет запускать тесты на разных устройствах и браузерах.

Особенности:

- Нет необходимости в браузерных драйверах.
- Поддержка параллельного тестирования.
- Легкость в настройке и использовании.

WebDriverIO

Мощный и гибкий фреймворк для автоматизации тестирования веб-приложений. Он может работать с браузерами через WebDriver или с использованием DevTools.

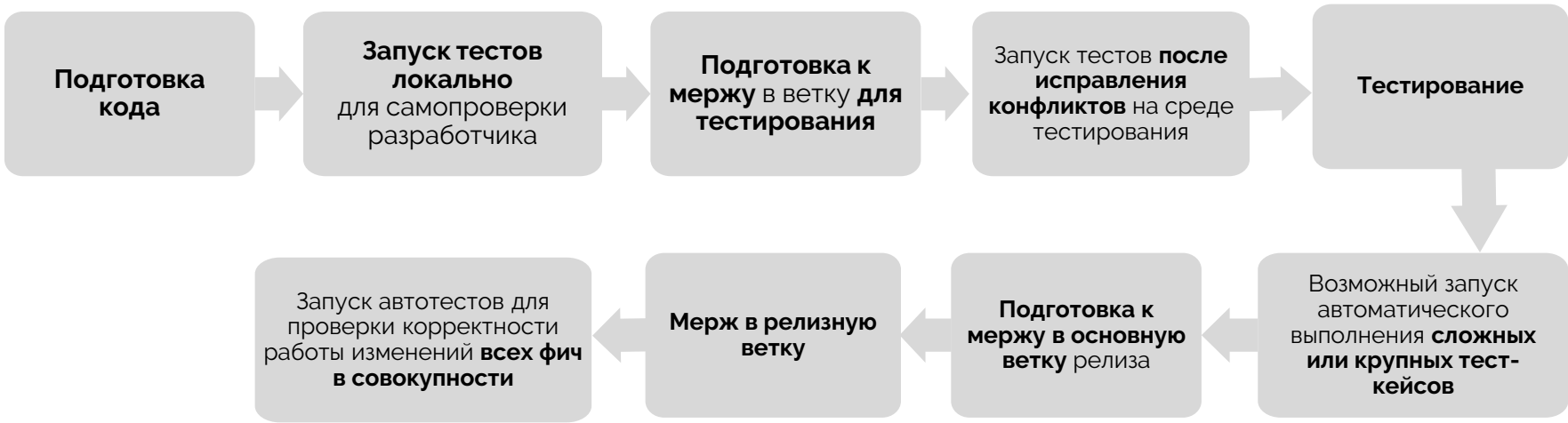
Особенности:

- Поддержка различных браузеров и мобильных устройств.
- Легкая настройка и интеграция.
- Поддержка тестирования с использованием различных фреймворков (Mocha, Jasmine, Cucumber).

Когда запускать автотесты

Автотесты могут быть запущены как отдельно, так и быть интегрированы в CI/CD процесс

Этапы разработки в которые могут быть запущены автотесты



Встраиваем в CI/CD для Stage, Prod

Для встраивания автотестов в процесс непрерывной интеграции в основном используются тесты трех видов:

1. **модульные** (unit tests)
2. **интеграционные** (integration tests)
3. **end-to-end** (E2E)

Существует два этапа :

CI (Continuous Integration) — непрерывная интеграция

На этом этапе автотесты проверяют каждый новый commit или pull request. (в данном тестировании участвуют в основном модульные и интеграционные, чтобы не замедлять процесс разработки и доставки)

CD (Continuous Delivery) — непрерывная доставка

После прохождения тестов приложение автоматически развертывается в staging или production.

Доступное в России ПО для WEB-тестов

Пример стека технологий для автотестов в России

Язык программирования: Java, Python, JavaScript.

Фреймворк: Selenide, Playwright, Cypress.

Управление тестами: Test IT, Allure.

CI/CD: GitLab CI/CD, Jenkins.

Отчеты: Allure Framework.

Автоматизация нефункционального тестирования: производительность и нагрузка

Тестирование производительности и нагрузки

Тестирование производительности — это важнейший аспект жизненного цикла разработки программного обеспечения, направленный на оценку характеристик производительности приложения в различных условиях нагрузки. Оно включает оценку того, насколько хорошо приложение работает с точки зрения скорости, стабильности, масштабируемости и использования ресурсов.

Нагрузочное тестирование — это тип тестирования производительности**,** который позволяет понять, как система работает в условиях высокого трафика и интенсивного использования, и определить, что нужно сделать для оптимизации производительности.

Типы тестирования производительности

Подробно мы остановимся
на нагрузочном тестировании.

1. **Нагрузочное**
2. **Пиков нагрузки**
3. **Расширяемости**
4. **Стрессовое**
5. **Объемное**
6. **Стабильности**

Что эти тесты дают руководителю проекта?

> *Почти **70% потребителей признают, что скорость работы страницы влияет на их готовность совершить покупку** в интернет-магазине. – **Unbounce, 2019***

Оценка времени отклика: Важно, чтобы приложение работало быстро, особенно в условиях высокого трафика. Низкое время отклика — это ключ к удовлетворенности пользователей и повышению конверсии (например, в интернет-магазинах).

Избежание сбоев: Тесты производительности помогают выявить участки кода или архитектуры, которые могут стать узким местом, если нагрузка на систему увеличится.

Прогнозирование поведения: Производительность приложения можно спрогнозировать на основе результатов тестов, что помогает в планировании инфраструктуры и масштабировании.

Что эти тесты дают руководителю проекта?

Минимизация рисков отказов: Нагрузочные и стресс-тесты выявляют пределы системы и позволяют заранее выявить слабые места, чтобы избежать сбоев в продакшн-среде.

Оптимизация пользовательского опыта: Понимание того, как система работает при пиковых нагрузках, помогает оптимизировать интерфейсы и серверную инфраструктуру для обеспечения стабильной работы.

Обнаружение утечек памяти и других проблем: Длительное тестирование системы под нагрузкой помогает выявить утечки памяти или другие ошибки, которые могут привести к долгосрочным сбоям.

Как часто запускать тесты?

- Нагрузочные тесты должны проводиться регулярно в качестве проактивной меры, но особую важность они приобретают в преддверии событий с высоким трафиком или после внесения изменений в архитектуру приложения, системные зависимости или алгоритмы кода.

- 57% организаций проводят тесты производительности и/или нагрузки во время каждого спринта, а 95% обязуются проводить тестирование ежегодно.
- Только 5% ИТ-специалистов *“никогда”* не проводят тесты производительности (в т.ч. нагрузочные).

Когда проводить тесты?

Компании обычно проводят нагрузочные тесты перед такими событиями:

- **Запуск нового продукта** или услуги
- **Крупная распродажа**, например “Черная пятница”
- **Продажа билетов на популярный концерт** или мероприятие
- **Крупное событие** на порталах государственных услуг, например, голосование на выборах
- Крупная **маркетинговая кампания или PR-акция**

Особенности создания окружения

Изоляция тестового окружения

Тестовое окружение должно быть изолировано от основного продакшн-окружения для предотвращения воздействия тестов на реальные данные и пользователей.

Подготовка данных

Для нагрузки и стресс-тестирования важно использовать реальные или близкие к реальным данные, чтобы симулировать реальные условия работы

Особенности создания окружения

Тестирование различных типов нагрузки и трафика

Важно настроить тестовое окружение таким образом, чтобы оно могло имитировать реальные условия эксплуатации — с различной нагрузкой, пиковыми запросами и высокой интенсивностью взаимодействий.

Сетевые настройки и ограничения

Для тестирования поведения системы в условиях слабого или нестабильного интернета нужно настроить сеть так, чтобы она имитировала реальные условия: задержки, потеря пакетов, пропускная способность.

Метрики нагрузочного тестирования

- Метрики нагрузочного тестирования варьируются в зависимости от среды пользователя и заинтересованной стороны.
- Важно понять какая метрика (метрики) поддерживает цель проекта.

- Существует такое понятие, как «слишком много метрик». Каждая отслеживаемая метрика требует времени для сбора, анализа и будет использоваться в той или иной форме отчета.
  После получения результатов может быть хорошей идеей объединить данные, для облегчения восприятия.

Метрики нагрузочного тестирования

1. Техническая среда (мобильная, веб-приложение и т.д.)

- Время отклика
- Пропускная способность
- Время обработки
- Использование ресурсов

2. Операционная среда

- Время запуска/выключения
- Оповещения
- Резервное копирование/восстановление

3. Бизнес среда

- Соглашение об уровне обслуживания (Service Level Agreements)
- Пропускная способность
- Бизнес-процессы (эффективность)

Инструментарий нагрузочного тестирования

На рынке масса инструментов, в основном они двух категорий:

1. С открытым исходным кодом, их можно использовать для написания и выполнения тестов (например, **Gatling** и **JMeter**).

2. Облачные SaaS-инструменты для тестирования в облаке и предоставляющие более широкую поддержку и возможности (например, **LoadNinja** и **BlazeMeter**)

Даже если вы используете для написания тестов инструмент с открытым исходным кодом, например JMeter, для проведения нагрузочного тестирования в облаке, вам потребуется развернуть тесты на нескольких серверах, выполнять их одновременно, а также визуализировать и хранить результаты тестирования. SaaS-продукт будет в этом отношении проще, но дороже.

Топ инструментов нагрузочного тестирования с открытым исходным кодом

- **JMeter**. Java-приложение, разработанное специально для нагрузочного тестирования, позволяет тестировать веб-приложения и время отклика. Очень популярное, можно использовать как opensource в условиях санкций.
- **Gatling**. Мощный инструмент, разработанный для непрерывного нагрузочного тестирования. Нагрузочные тесты Gatling пишутся на языке Scala, выполняются из консоли, а результаты генерируются в HTML.
- **The Grinder**. Java-фреймворк для нагрузочного тестирования, позволяющий выполнять распределенное тестирование с использованием множества генераторов нагрузки. Grinder работает на любой системе с Java API.
- **Locust**. Инструмент распределенного нагрузочного тестирования на основе языка Python, позволяющий “запустить” систему с миллионами одновременно работающих пользователей.

Топ SaaS-инструментов для нагрузочного тестирования

- **k6 Cloud**. SaaS-версия инструмента нагрузочного тестирования k6 с открытым исходным кодом. k6 Cloud позволяет создавать сценарии, записывая действия пользователей в браузере, проводить тестирование из 21 различных географических точек и запускать тесты в облаке с одновременным присутствием до 1 млн. виртуальных пользователей.

- **LoadNinja**. Позволяет проводить нагрузочное тестирование с использованием реальных браузеров. Это приближает тест к воспроизведению реальных ситуаций с высоким трафиком.

Топ SaaS-инструментов для нагрузочного тестирования

- **BlazeMeter**. Один из немногих сервисов нагрузочного тестирования, созданный специально для Apache JMeter. Он позволяет быстро настраивать тесты, создавать тесты с количеством одновременных пользователей до 1 млн. человек, имитировать мобильное тестирование на реальных устройствах и запускать тесты из нескольких географических точек.

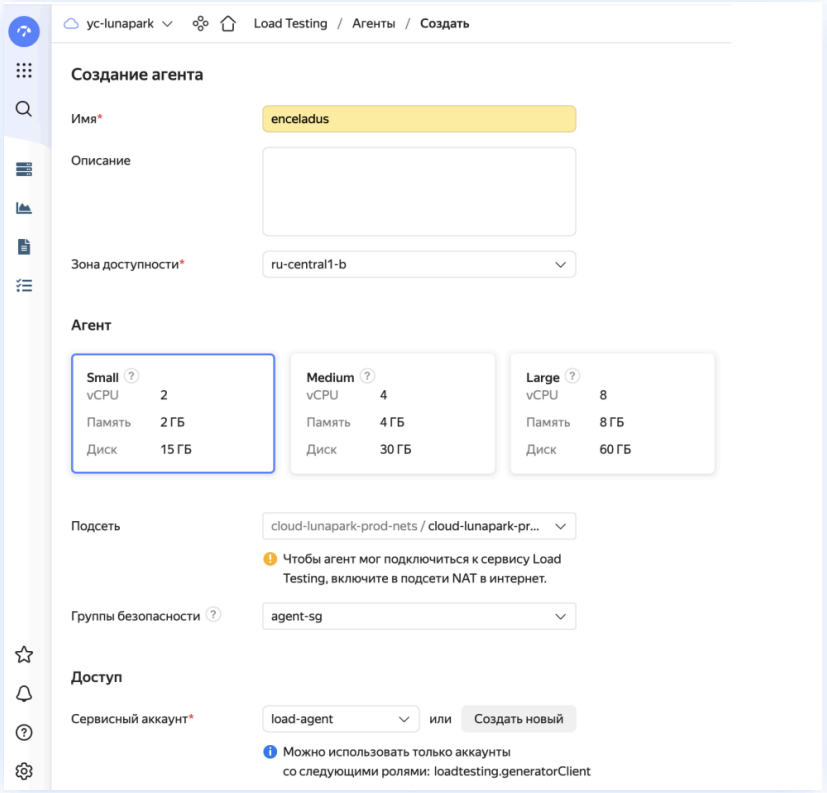
- **LoadRunner**. Это сложный инструмент нагрузочного тестирования, позволяющий обнаружить проблемы с производительностью веб-приложений, ERP-программ и унаследованных систем. Он специально разработан для выявления “узких мест” еще на этапе внедрения приложения, позволяя оценить его производительность до запуска в эксплуатацию. Запатентованный механизм автокорреляции позволяет точно определять узкие места на уровне системы, конечного пользователя и кода.

Российское ПО - Yandex Load Testing

Yandex Load Testing - это российский облачный сервис для нагрузочного тестирования.

Для проведения нагрузочного теста вам понадобятся:

- сервис, который вы собираетесь тестировать;
- агент — ВМ с инструментом для тестирования и генераторами нагрузки - например, JMeter;
- конфигурация теста;
- нагрузка.

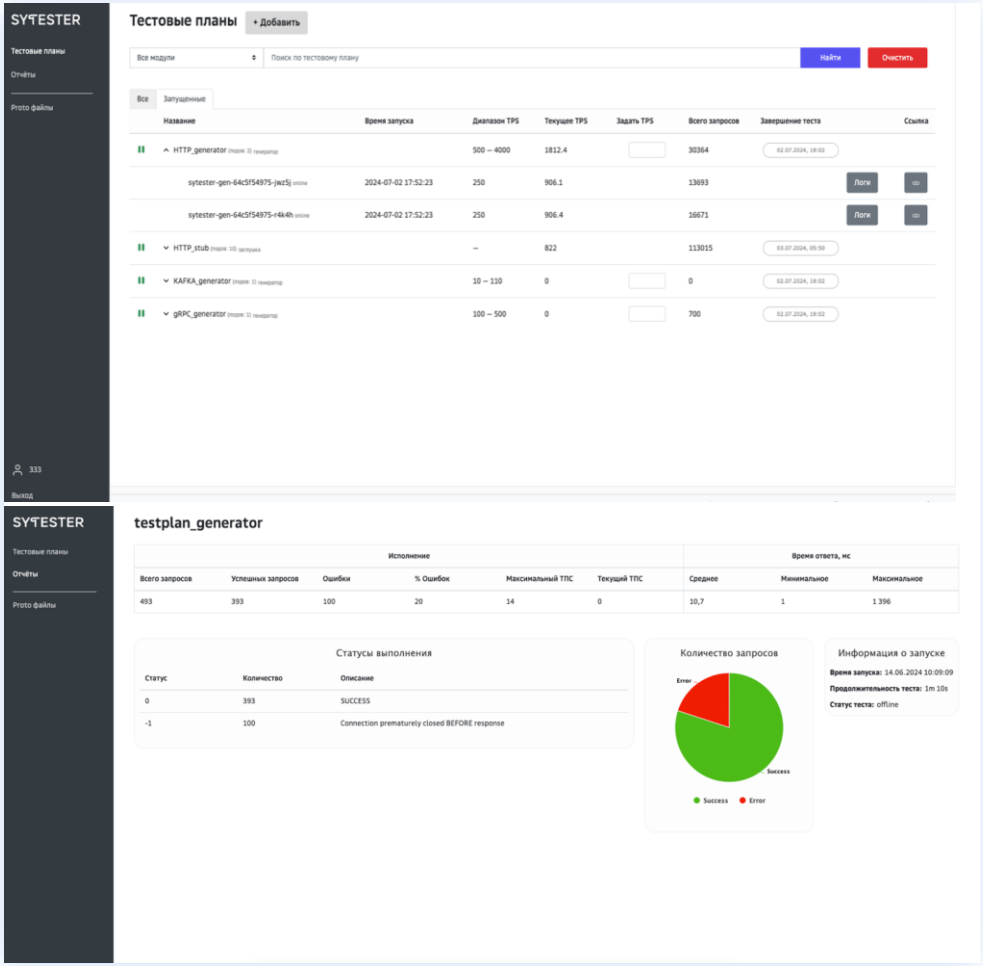


Возможности Yandex Load Testing

- Позволяет проводить нагрузочное тестирование в Яндекс. Облаке или внутри своей инфраструктуры,
- Автоматически разворачивает всю необходимую инфраструктуру,
- Позволяет удобно запускать нагрузочные тесты и анализировать их результаты в виде графиков и таблиц,
- Дает возможность проводить и анализировать регрессионное нагрузочное тестирование.

SyTester

- SyTester - инструмент для нагрузочного тестирования, разработанный Сбером.
- Имеет две редакции: Enterprise Edition (EE) и
- Community Edition (CE) - бесплатная версия
- Поддерживаемые протоколы и технологии из коробки:
- HTTP; gRPC; Kafka; IBM MQ;
- Active MQ; Artemis MQ.
- Для всех протоколов есть возможность настройки SSL.



Особенности инструмента SyTester

- **Поддержка API**, с помощью которого можно встроить процесс тестирования в свой CI/CD (примеры команд: загрузить тест, запустить тест, получить результаты и т. п.);
- **Генератор и заглушка доступны из коробки для всех поддерживаемых протоколов**;
- **Простой графической интерфейс**, из которого можно создать тест и мониторить тестирование;
- **Доступны как встроенные отчёты, так и расширенные**;
- **Метрики из коробки**: имя сервиса, длительность транзакции, статус транзакции;
- **Возможность изменять ТПС и время завершения теста онлайн во время его исполнения (без перезапуска)**;
- **Журналы доступны как во время теста, так и после его остановки**;
- **Поддержка** виртуальных пользователей.

Фиксация багов и трекинг работы

Для чего нужно фиксировать баги и что такое баг-репорты?

- Баг нужно фиксировать, **чтобы его увидели и исправили.**
- О дефекте можно забыть - фиксируем, **чтобы не потерять**
- Чтобы отследить историю бага - **жизненный цикл дефекта**
- Чтобы зафиксировать **исполнителя** для исправления бага
- Для **сбора статистики** багов на проекте и анализа качества

Баг-репорт — это инструмент, который тестировщик использует для передачи информации о найденных проблемах в программе. Эффективный баг-репорт повышает шансы на быстрое исправление проблемы, улучшает коммуникацию в команде и экономит время разработчиков.

Общепринятый жизненный цикл бага

1. **Обнаружен (submitted)** - начальное состояние, иногда называется "Новый" (new), в котором находится баг сразу после создания.
2. **Назначен (assigned)** - в это состояние баг переходит с момента, когда кто-то из проектной команды назначается ответственным за исправление данного дефекта.
3. **Исправлен (fixed)** - в это состояние баг переводит ответственный за исправление после выполнения соответствующих работ по устранению дефекта.
4. **Проверен (verified)** - в это состояние баг переводит тестировщик, удостоверившись, что дефект был устранен.
5. **Закрыт (closed)** - состояние бага, означающее, что по данному дефекту больше не планируются работы.
6. **Открыт заново (reopened)** - в это состояние тестировщик переводит дефект, удостоверившись, что он все еще воспроизводится в билде, в котором он должен был быть исправлен.
7. **Рекомендован к отклонению (to be declined)** - состояние, при котором дефект выносится на рассмотрение для отклонения по каким-либо причинам.
8. **Отклонен (declined)** - в это состояние отчет переводится, когда баг не является дефектом.
9. **Отложен (deferred) -** в это состояние баг-репорт переводится, если в данный момент его исправление является нерациональным или не представляется возможным, однако есть понимание, что он будет исправлен, если ситуация изменится.

Жизненный цикл бага

В некоторых трекинговых системах баг в состоянии «Закрыт» или «Отклонён» может быть переведён из множества предшествующих состояний с резолюциями наподобие:

- **"Не является дефектом"** — функционал так и должен работать, описанное поведение не является аномальным.
- **"Дубликат"** - данный баг уже был заведен ранее
- **"Не удалось воспроизвести"** - при попытке разработчиком повторить шаги, описанные при постановке бага, фактический результат не удалось воспроизвести.
- "**Не будет исправлено"** - баг зафиксирован, но по каким-то причинам принято решение его не исправлять.
- **"Невозможно исправить"** — непреодолимая причина дефекта находится вне области полномочий команды разработчиков,

Что требуется от тестировщика при баг-репортинге

Без токсичности 😂 Текст отчёта должен быть простым и понятным. Отчёт должен быть ясным, четким и полным. Никого не обвиняйте. Ваша цель – сообщить о баге, как пользователь видит проблему. Вместо «Я увидел сообщение об ошибке», лучше написать: «Появилось сообщение об ошибке».

Общие рекомендации

- Будьте объективны. Не давайте оценок коду, сосредоточьтесь на проблеме.
- Проверяйте свои баг-репорты. Ошибки или пропущенные шаги могут затруднить работу.
- Уточняйте приоритет и серьезность. Например, баг, из-за которого падает приложение, будет критическим.

Структура баг-репорта

1. Заголовок — кратко описывает проблему. Например, "Ошибка ввода данных в поле 'Имя' — принимаются цифры".

2. Краткое описание (summary) — должно быть коротким и ёмким. Когда кто-то первый раз читает саммари, он должен понимать, о чем этот баг-репорт. Это идеальное саммари, к которому вы должны стремиться.

3. Шаги для воспроизведения — пошаговая инструкция, которая помогает воспроизвести проблему.

5. Фактический результат — что происходит на самом деле.

6. Приложения — скриншоты, видео или логи, которые иллюстрируют проблему.

Приоритет, критичность, влияние на решение ПМ-а

Критичность и приоритет дефекта: в чем разница?

- **Приоритет** связан с планированием, а **критичность** — со стандартами.
- Скорость исправлений основывается на приоритетах проекта и критичности дефекта.
- Степень критичности проблемы определяется в соответствии с оценкой рисков заказчика и фиксируется в выбранном им средстве отслеживания.
- Ошибки в программном обеспечении могут “серьезно” повлиять на график работ, что, в свою очередь, может привести к переоценке и пересмотру “приоритетов”.

Что такое критичность и приоритет бага?

- **Критичность (severity)** показывает степень ущерба, который наносится проекту существованием дефекта.
- **Приоритет (priority)** показывает, как быстро дефект должен быть устранён.

QA инженер классифицирует дефекты по степени серьезности в зависимости от сложности и критичности дефектов.

Приоритет дефектов определяют все заинтересованные стороны, включая руководителя проекта, бизнес-аналитиков, владельца продукта.

Для повышения прозрачности работы с бэклогом, руководитель проекта должен помочь QA-инженерам легко понимать приоритет в проекте. В этом омгут помчоь матрицы приоритета функционала, проводимые ретроспективы с командой и др.

Градация приоритета дефекта (Priority):

Градация может быть изменена в соответствии с требованиями проекта или компании

P1 Критический (Critical) - Критическая для проекта ошибка. Должна быть исправлена как можно быстрее.

P2 Высокий (High) - Дефект с приоритетом P2 идет следующим в очереди на устранение после критических дефектов. Это проблема, без решения которой выпуск нового релиза невозможен.

P3 Средний (Medium) Не критичная для проекта ошибка, однако требует обязательного решения.

P4 Низкий (Low) Наличие данной ошибки не является критичным и не требует срочного решения. Может быть исправлена, когда у команды появится время на ее устранение.

Градация критичности дефекта (Severity):

Градация может быть изменена в соответствии с требованиями проекта или компании

- **Блокирующий (S1 – Blocker) -** тестирование значительной части функциональности вообще недоступно. Блокирующая ошибка, приводящая приложение в нерабочее состояние.

- **Критический (S2 – Critical) -** критическая ошибка, неправильно работающая ключевая бизнес-логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, но имеется workaround (обходной путь), позволяющий продолжить тестирование.

- **Значительный (S3 – Major/Medium) -** не работает важная часть одной какой-либо функции/бизнес-логики, но при выполнении специфических условий, либо есть workaround, позволяющий продолжить ее тестирование либо не работает не очень значительная часть какой-либо функции. **Незначительный (S4 – Minor)** часто ошибки GUI, которые не влияют на функциональность, но портят юзабилити или внешний вид. Также незначительные функциональные дефекты, либо которые воспроизводятся на определенном устройстве.

- **Тривиальный (S5 – Trivial) -** почти всегда дефекты на GUI — опечатки в тексте, несоответствие шрифта и оттенка и т.п.

Как они влияют на решение ПМ-а о включении бага в спринт?

Пирамида приоритезации

Мы знаем, что по-хорошему, в текущий спринт не рекомендуется добавлять новые задачи, но критические дефекты должны исправляться чем быстрее, тем лучше.

Баги с высоким приоритетом **P0-P1 могут автоматически добавляться в текущий спринт** и браться в работу любым из участников распределенной команды разработки. А баги с более низким приоритетом **P2-P4 могут попадать в следующие спринты, пропорционально приоритету**. Значение приоритета может означать спринт, в котором баг будет взят в работу.

Как они влияют на решение ПМ-а о включении бага в спринт?

О зависших багах стоит напомнить команде на митапе, а если есть риск того, что баг может переехать в следующий спринт, возможно стоит задуматься о корректировке приоритета на более низкий, а при достижении самого низкого приоритета — принять решение о закрытии бага с резолюцией «не требует исправления».

Если продукт находится на раннем этапе развития, а продуктовых задач слишком много, высока вероятность, что владелец продукта, несмотря на заинтересованность в качестве — будет отдавать предпочтение продуктовым задачам. Это точка внимания руководителя проекта. Возможно стоит обсудить ситуацию с командой, чтобы и качество не пострадало и количество продуктовых задач, доставленных в продакшн было в рамках ожидания заказчика.

Работа со списком багов, бэклогом, планирование спринтов

- Планирование спринта по методологии **Agile**
- Планирование спринта по методологии **Канбан**
- Планирование спринта по методологии **Scrum**

Работа по методологии Agile: планирование спринта

Работа agile-команды разбивается на короткие итерации:
Итерации ограничиваются по времени, т.е. они завершаются вовремя, даже если приходится урезать функциональность. Это означает, что команда последовательно добавляет одну или несколько небольших функций во время каждой итерации, но каждая добавленная функция встроена в продукт, **протестирована и имеет качество**, необходимое для релиза.

Agile-команды участвуют в планировании на трех уровнях:

- **планирование релиза (**обычно от трех до шести месяцев)
- **планирование итерации** (охватывает срок только одной итерации — обычно от двух до четырех недель)
- **дневное планирование** (то результат обязательств членов команды, принимаемых друг перед другом на ежедневных летучках)

Планирование спринта по методологии Agile: работа со списком багов

- Agile-команды **стремятся устранять все ошибки** в той итерации, в которой они обнаруживаются. Это становится возможным с приобретением опыта работы в режиме коротких итераций, в частности в результате применения **автоматического тестирования.**

- Когда программист дает оценку задачи по кодированию чего-либо, он включает в нее время на устранение ошибок, выявленных в процессе реализации, или выделяет этот процесс и оценивает его как отдельную задачу («Устранение ошибок»).

- С багом, обнаруженным позже (или не устраненным во время итерации, в которой он был выявлен), поступают точно так же, как и с пользовательской историей. Устранение дефекта следует приоритизировать в последующей итерации так, как приоритизируют любую другую пользовательскую историю. За пределами итерации идея дефекта в целом начинает сходить на нет.

Работа по методологии Канбан: планирование спринта

- В отличие от Agile подхода, Канбан избавляется от ограниченных во времени итераций и вместо этого рассинхронизирует деятельность по расстановке приоритетов, разработке и поставке.

- Канбан-команды по-прежнему с заданной частотой выдают версии продукта, предпочитая короткие интервалы.

- Метод тоже работает в соответствии с «Принципами манифеста гибкой разработки», однако Канбан старается избегать любых крайностей, связанных с искусственной установкой временных рамок для задач.

Работа по методологии Канбан: работа со списком багов

- При появлении дефектов, они распределяются согласно нагрузке и приоритету.
- Для этого карточка дефекта должна быть подробно описана. Приоритет в Канбане может выставляться любым членом команды, а не только руководителем проекта.
- Благодаря Канбану создается мощный самоорганизующийся механизм управления рисками.
- Кроме того, Канбан, предоставляя членам команды возможность самостоятельно принимать решения о расписании работы и приоритетах, демонстрирует уважение к сотрудникам и доверие к системе (или разработке процесса).

Работа по методологии Scrum: планирование спринта

- **Каждый спринт планируется предварительно на специальных встречах.** Участники оценивают, какой объем работ, на их взгляд, они смогут сделать.
- Из списка задач, расставленных по приоритетам, они **выбирают очередные work items,** предназначенные для выполнения.
- **Спринты всегда имеют фиксированную продолжительность,** которая должна быть меньше месяца. Как правило, выбирают спринты длиной в одну или две недели.
- Группа решает, **сколько единиц работы они в состоянии выполнить** за предстоящий спринт.
- На завершающей стадии спринта участники снова собираются вместе и **показывают друг другу, чего удалось достичь за время совместной работы**. Они смотрят, сколько единиц работы, действительно доведены до конца.
- После ретроспективы **собирается статистика по прошедшему спринту** для формирования следующего.

Работа по методологии Scrum: работа со списком багов

При планировании спринта нужно закладывать время на контроль качества и исправления багов.

Если прилетает **срочная задача во время спринта**, стандартно бывают несколько способов решить проблему:

Вариант 2

Если проект большой и над ним работает несколько продуктовых команд, то выделяют отдельную «пожарную» команду на исправление критичных багов.

Вариант 2

Принимается решение не брать задачу в текущий спринт, а перенести на следующий (важно понимать приоритет бага).

Вариант 3

Задача берется в работу, но на ретроспективе обсуждается, по какой причине она появилась и принимаются решения для устранения таких проблем в будущих спринтах.

Работа со статусами багов

Отслеживание статуса бага лежит на QA-lead тестирования. Если в команде нет QA-lead, то эта задача ложиться на инженера по тестированию. Также к приоритезации багов может подключаться владелец продукта. Рассмотрим далее в примерах.

Следить за статусом бага имеет смысл, если:

- вы считаете, что он критичный, а его долгое время не исправляют, не ставят в версию,
- исправление этого бага влияет на проверку другой задачи
- по багу предвидится большой объем работы.

Планирование времени в проекте на решение багов

1. Владелец продукта выделяет из баг-трекера самые высокоприоритетные баги, выносит их на планирование спринта.

2. Владелец продукта создаёт story points, соответствующие задачам из баг-трекера. Например, «Исправить самые критические ошибки Ticket-124, Ticket-126, и Ticket-180».

3. Работы по исправлению ошибок не включаются в спринт. Затем, вводится предположение, что команда в каждую итерацию будет тратить определённую часть времени на исправление багов

4. Баги заносятся как обычные story points и рассматриваются для каждого спринта с учетом приоритета.

Трекинговые системы

Jira

JIRA до сих пор является наиболее часто используемой системой управления проектами в России

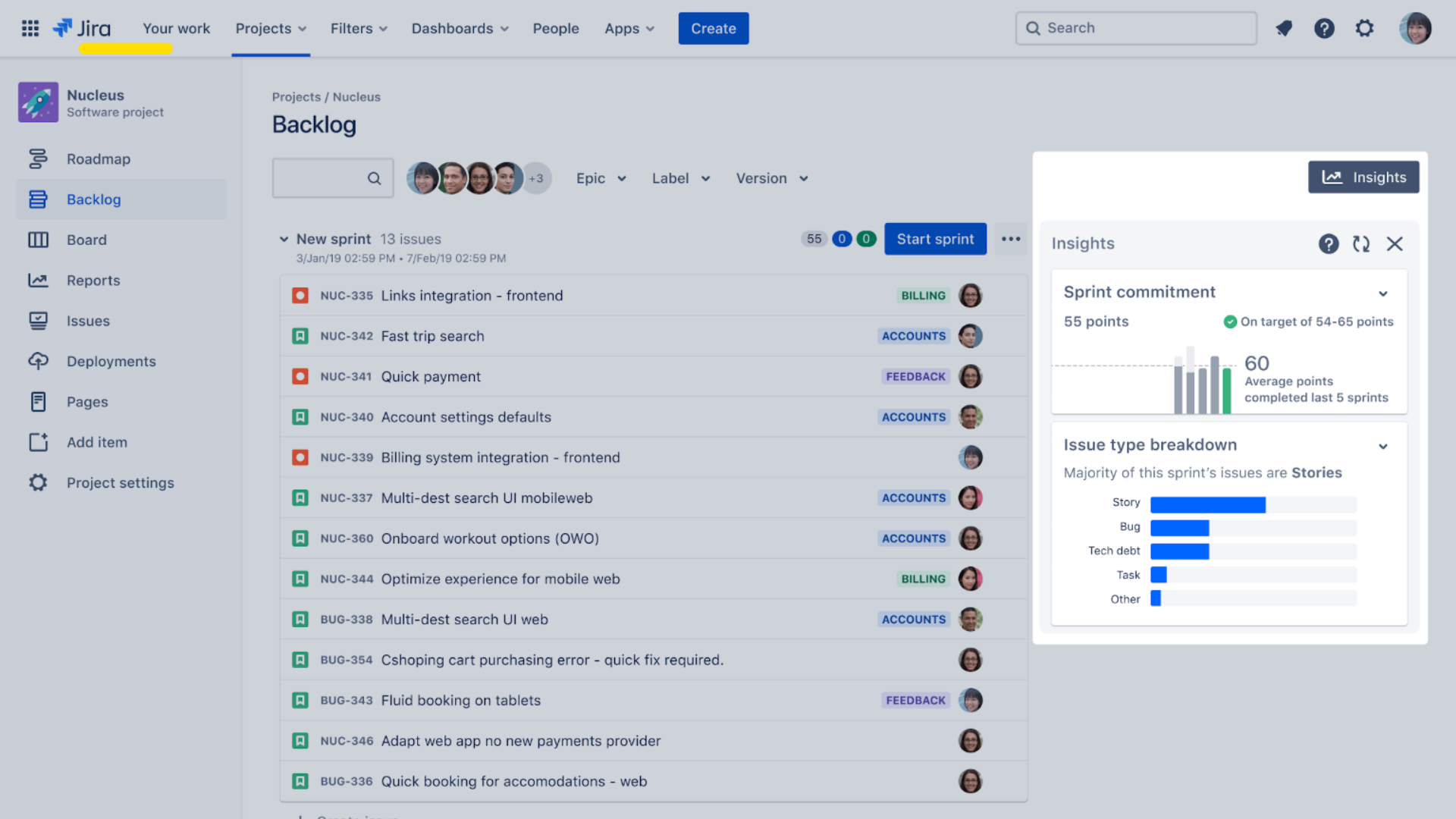
Хотя компания Atlassian больше не работает с российскими пользователям, тем не менее достаточно много ее продуктов по-прежнему установлены у российских компаний.

Россия по популярности JIRA уступает только США (3,1 тыс. серверов) и Германии (2,9 тыс. серверов). Однако для российских пользователей существует проблема установки обновлений, и сейчас, когда в JIRA обнаружили новую уязвимость - нужно как можно быстрее обновить систему до последней версии.

Тем не менее, данный трекер остается в числе самых популярных трекеров в России.

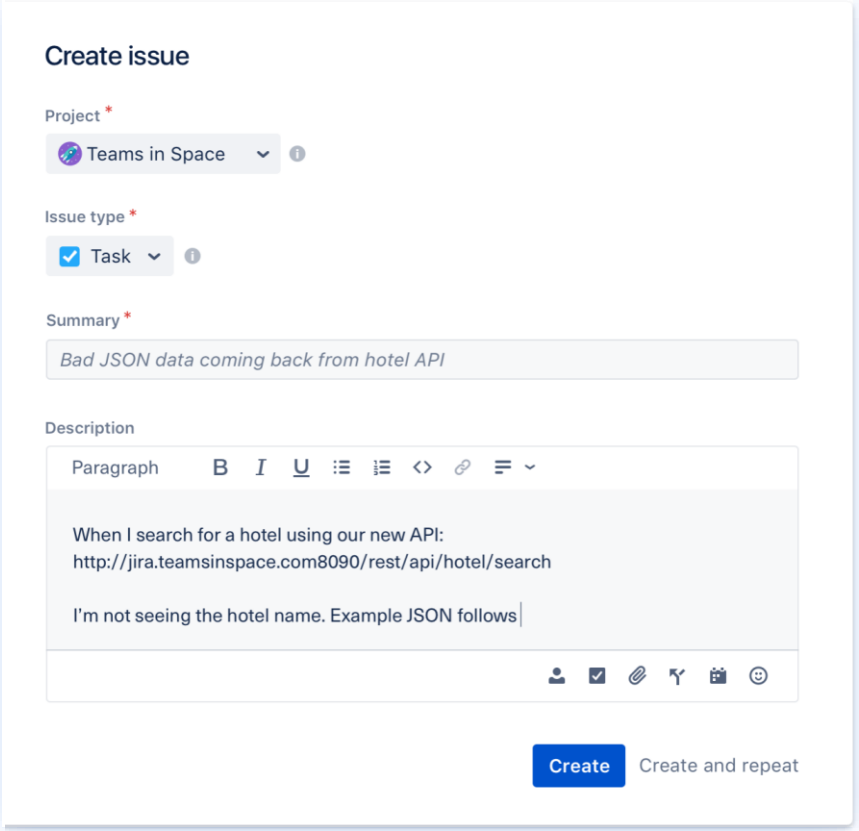
Возможности Jira

- **Jira представляет собой интерактивную доску** (Дашборд), с помощью которой можно следить за выполнением поставленных задач.
- **Все задачи классифицируются различными видами функций, подзадач, багов и т.д.** Они могут редактироваться, назначаться на различных исполнителей или просто изменять статус с «открыт» на «закрыт».
- **Все изменения по задаче записываются в журнал.**


Jira
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
Nucleus
Software project
Roadmap
Backlog
Board
Reports
Issues
Deployments
Pages
Add item
Project settings
Projects / Nucleus
Backlog
+3
Epic
Label
Version
Insights
New sprint 13 issues
3/Jan/19 02:59 PM • 7/Feb/19 02:59 PM
55 0 0
Start sprint
NUC-335 Links integration - frontend BILLING
NUC-342 Fast trip search ACCOUNTS
NUC-341 Quick payment FEEDBACK
NUC-340 Account settings defaults ACCOUNTS
NUC-339 Billing system integration - frontend
NUC-337 Multi-dest search UI mobileweb ACCOUNTS
NUC-360 Onboard workout options (OWO) ACCOUNTS
NUC-344 Optimize experience for mobile web BILLING
BUG-338 Multi-dest search UI web ACCOUNTS
BUG-354 Cshoping cart purchasing error - quick fix required.
BUG-343 Fluid booking on tablets FEEDBACK
NUC-346 Adapt web app no new payments provider
BUG-336 Quick booking for accomodations - web
Insights
Sprint commitment
55 points
On target of 54-65 points
60
Average points completed last 5 sprints
Issue type breakdown
Majority of this sprint's issues are Stories
Story
Bug
Tech debt
Task
Other

Возможности Jira

- **Широкий функционал**, который можно дополнительно расширить с помощью плагинов.
- **Интеграция с различными системами** (Git, Zephyr, Trello, Slack, Google Drive & Docs, draw.io и так далее).
- Есть возможность строить **диаграмму Ганта**.
- Рабочие столы можно **настроить под себя**.
- Позволяет составлять **план работы**.
- Возможность **искать задачи по гибким фильтрам**.



Microsoft TFS (Azure DevOps)

- Azure DevOps — это платформа Microsoft «Программное обеспечение как услуга» (SaaS), которая предлагает командам комплексную цепочку инструментов DevOps, предназначенную для разработки и развертывания качественного программного обеспечения:
- Управление требованиями.
- Управление проектами как для гибких команд разработки программного обеспечения, так и для каскадных команд.
- Контроль версий с помощью Team Foundation Version Control (TFVC) или Git.
- Автоматизированные сборки.
- Отчеты, такие как результаты тестирования, и показатели разработки, такие как количество невыполненных работ и скорость выпуска.
- **Управление тестированием и релизами.**

Azure DevOps - возможности для команды и тестирования

- С помощью Azure DevOps команды могут создавать рабочие элементы, эпики, истории, проектные задачи и многое другое, а также управлять ими, чтобы оставаться на правильном пути от начала до конца.
- Azure DevOps предлагает **комплексные возможности тестирования**, поддерживая исследовательские, ручные, системные и пользовательские приемочные тесты для любого приложения на любом языке.
  Закодированные тесты пользовательского интерфейса могут быть созданы с помощью Visual Studio для тестирования пользовательского интерфейса приложения, а планы тестирования Azure упрощают создание исследовательских тестов.
  Настраиваемые тестовые среды позволяют указать комбинацию аппаратного и программного обеспечения, которая наилучшим образом представляет целевую среду.

Allure TestOps

Allure TestOps – это коробочное решение, позволяющее управлять как ручным, так и автоматизированным тестированием: создавать тест-кейсы и чек-листы, запускать ручные и автоматические прогоны, заводить дефекты и собирать статистику по проделанной работе.

Также у Allure большой набор интеграций с различными инструментами и языками программирования.

Возможности:

- Позволяет работать с ручным и автоматизированным тестированием
- Делать запуски тестов
- Строить отчеты
- Писать тест-планы
- Заводить дефекты в баг-трекере

Яндекс Трекер

Яндекс Трекер

- В Яндекс Трекере есть всё, что нужно для ведения проектов:
- канбан-доска,
- задачи и проекты,
- ответственные и время выполнения задачи,
- теги и напоминания,
- статусы,
- оценка сложности задач.
- Плюс в том, что это полностью российский сервис, которым можно пользоваться бесплатно, если у вас в команде 5 человек или меньше. Работает в браузере, доступен с любого рабочего места, нужна учётная запись Яндекса.

Kaiten

Kaiten — это российский таск-трекер, который помогает в управлении проектами.

Он позволяет ставить задачи и дедлайны по ним, а также отслеживать прогресс их выполнения и анализировать работу команды.

Он хорошо подходит для организации рабочего процесса по методологиям Kanban и Scrum.

В планировщике Kaiten есть такие функции для работы над проектами:

- создание задач;
- назначение ответственных и других участников;
- установка сроков выполнения;
- отслеживание прогресса;
- комментирование задач;
- перенос неактуальных задач в архив с сохранением информации;
- просмотр отчётов о работе команды.

YouGile

- Сервис работает по методикам Agile. Каждое действие имеет аналитику, дедлайны, контакты исполнителя, статус.
- Внешне интерфейс напоминает Трелло, но имеет больше опций.
- Настраивается доступ для каждого человека, отводится список задач.
- На платформе есть иерархия действий, которая позволяет расставлять приоритеты.
- Если в Trello всего три формата участия (админ, наблюдатель, участник), то здесь их 108.

Бесплатный модуль открыт для компании до 10 человек, свыше придется приобрести подписку.

Отличием программы является коробочная версия, когда ПО ставят на внутреннюю сеть фирмы.

Вместо заключения

Для руководителя проекта интеграция всех инструментов снижает риски, экономит в долгосрочной перспективе бюджет, улучшает контроль над проектом.

На крупных проектах лучший эффект достигается при их комплексном внедрении.

Автоматическое тестирование
Обеспечивает стабильность и качество кода
Сокращает время ручных проверок, ускоряет выпуск релизов

Нагрузочные тесты
Выявляют узкие места в производительности системы
Предотвращают сбои при высоких нагрузках и в целом минимизирует простои

Трекинговые системы
Позволяют отслеживать прогресс задач и распределение ресурсов
Повышают прозрачность работы команды

Вопросы

Спасибо
за внимание