УДК 004.75

АВТОМАТИЗАЦИЯ ПРОЦЕССА ТЕСТИРОВАНИЯ И ОТЧЕТНОСТИ НА ПРОЕКТЕ WEB-INSYNC В ЗАО «АЛЬФА-БАНК»

У. М. Козак
Студент БГУИР, специалист по обеспечению качества ПО, ЗАО «Альфа-Банк»
ulyana.kozak.new@gmail.com

В. Г. Лунин
менеджер по обеспечению качества ПО ЗАО «Альфа-Банк»

У. М. Козак
Студентка Белорусского государственного университета информатики и радиоэлектроники. Работает в ЗАО «Альфа-Банк» в должности специалиста по обеспечению качества ПО.

В. Г. Лунин
Менеджер по обеспечению качества ПО, ЗАО «Альфа-Банк», координирующий направление тестирования на проектах физических лиц.

Аннотация. На крупных проектах покрытие автотестов растет с каждым днем и когда тестовых сценариев становится очень много, вести документацию и отчетность, а также следить за запусками автотестов из разных мест становится непросто, для решения данной проблемы на проекте *Web-Insyn*c ЗАО «*Альфа-Банк*» применяется специальная методология *TestOPS*, которая реализуется с использованием инструмента *Allure TestOPS*, что позволяет хранить всю документацию, запуски тестов и отчеты в одном месте и на базе этих данных получать графики, анализировать работу фреймворка и создавать динамическую документацию на базе автотестов.

Ключевые слова: *Allure TestOPS*, автоматизация тестирования, *Playwright*, динамическая документация

Введение. В быстро изменяющемся мире незамедлительная реакция на критические события может сэкономить десятки тысяч долларов или спасти чью-то жизнь. Огромное количество данных генерируется разными источниками ежесекундно. Анализируя эти данные, можно предсказывать проблемы и реагировать на них в реальном времени, избегая и предупреждая серьезные последствия. До недавнего времени еженедельные и ежемесячные аналитические отчеты полностью покрывали запросы бизнеса. В ЗАО «Альфа-Банк» специалисты по обеспечению качества занимаются не только ручным, но и автоматизированным тестированием. Зона ответственности у каждого специалиста зависит от платформы, которую от тестирует.

Web QA – занимается тестированием сайтов и браузерных приложений, используя разнообразное ПО, методологии и технологии. В зону ответственности входит проверка фронта (*UI*) и интеграции на уровне (*API*).

Mobile QA – занимается тестированием мобильных приложений, используя разнообразное ПО, методологии и технологии. В зону ответственности входит проверка фронта (*UI*) для *Android* и *iOS* платформ.

Зоны ответственности за тестирование на проектах *INSNC* показаны на рисунке 1.

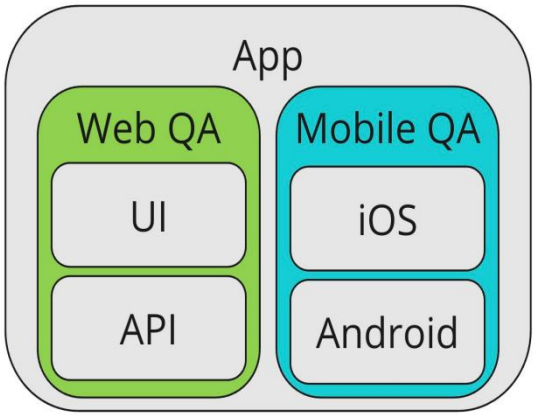

Рисунок 1. Зоны ответственности

Все тестирование проводится в соответствии с определенными этапами, начиная с тестирования бизнес-требований, далее тестирование макетов в *Figma*, тестирование *API* на ДЕВ, ТЕСТ стендах и написание автотестов на новый функционал, после тестирование интеграции бэка с фронтом и тестирование на пре-проде конечной реализации, оформление *UI* автотестов. Примерный жизненный цикл процесса тестирования показан на рисунке 2.

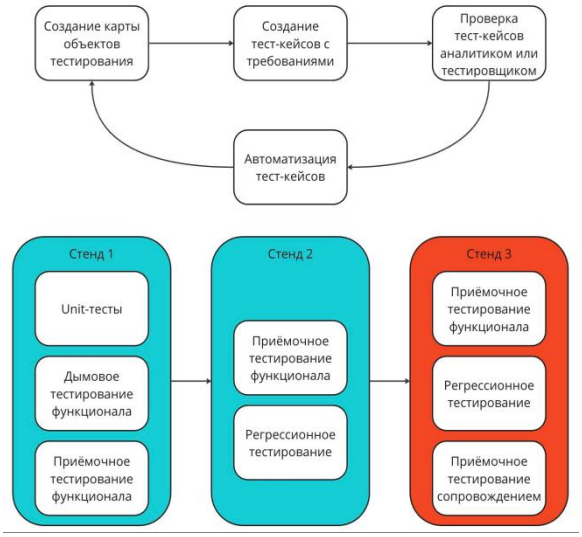

Рисунок 2. Примерный жизненный цикл процесса тестирования

В качестве основных инструментов для автоматизации тестирования используется следующий перечень приложений, показанный на рисунке 3. В центре всей схемы инструментария находится связующее звено, инструмент который позволяет сделать подход к тестированию унифицированным и организовать порядок в жизненном цикле разработки автотестов – *Allure TestOPS* [1].

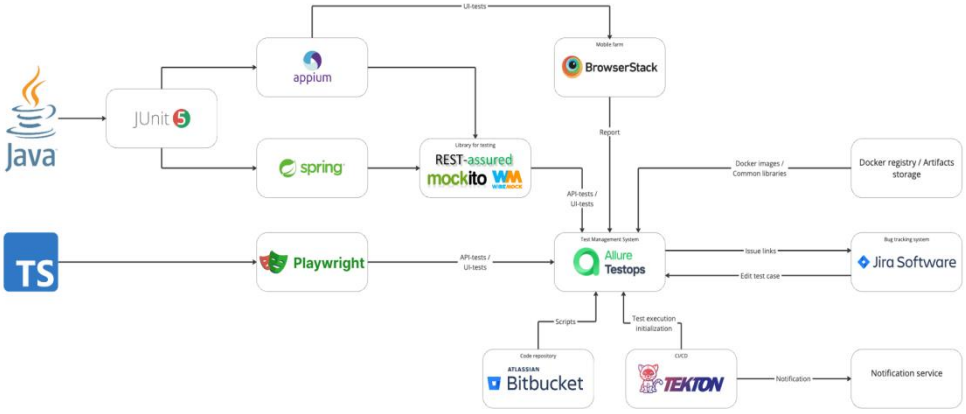

Рисунок 3. Инструментарий тестирования

В данной статье более подробно рассмотрим как использование в работе с большим количеством автоматизированных тестов и масштабными проектами, такими как *Web Insync* и *INSNC* мобильный использование *Allure TestOPS* помогает ускорить процессы работы не только с отчетностью но и с созданием документации.

Allure TestOps – это *Test Management System* (*TMS*), которая изначально строилась с фокусом на автоматизацию и подход «тест-кейсы как код». Конечно же, это не случайно, т.к большинство автоматизаторов используют *Allure Report* и в коде автотестов уже используются механизмы разметки отчетов – эти же механизмы используются и для интеграции с *Allure TestOps* [2].

Allure TestOps стоит внедрять, если:

1 Написание и сопровождение подробной тестовой документации требует много ресурсов. В реальной жизни практически невозможно найти на это время. Например, сейчас в нашем тестовом проекте более 1000 *Playwright*-тестов и их число растет с каждым спринтом.

2 Тестовая документация плохо структурирована. *Allure TestOPS* помогает во первых отказаться от ручного написания тест-кейсов, а заменить этот процесс на автоматическую генерацию тест-кейсов на базе прогона, во вторых само приложение организовано так, что при добавлении в код авто-тестов специальных тегов можно указывать какая платформа, разработчик, протокол и т.п. соответствует данному кейсу, что в дальнейшем делает возможность поиска и фильтрации кейсов намного проще, а сами файлы – структурированными.

3 Отчеты хранятся в разных местах. После того, как тесты автоматизированы и готовы, мы запускаем их в *CI* (в Альфа Банке мы используем *TektonCI*). К каждой сборке прикреплен отчет *Allure*. Отчеты *Allure* это здорово, но по разным техническим причинам у нас много сборок, в которых прогоняются *Playwright*-тесты. Получалось так, что отчеты хранились не в одном месте, и проанализировать конкретный тест было сложно.

4 Проблемы со стабильностью тестов. Как и многие другие инженеры по автоматизации, мы мучились с нестабильными тестами. Но не все неудачные тесты являются нестабильными некоторые не проходят из-за ошибки в тестовой среде или просто требуют обновления. Выполнение этих тестов не только отнимало время у людей, просматривающих отчет, но и увеличивало время выполнения тестовой сборки. Было бы неплохо вообще не запускать эти тесты и исправлять их, когда придет время.

5 Не очевидно кто отвечает за тест. Благодаря специальным возможностям *Allure TestOPS* также можно в тегах указывать разработчика и тестировщика ответственного за данный тест-кейс, что помогает избежать недопониманий и конфликтов.

Чтобы понять, как нам удалось справиться со всеми проблемами, давайте посмотрим на код. В Альфа-Банке для написания автоматических тестов мы используем *JavaScript*, поэтому примеры кода также будут на нем.

Рассмотрим пример одного из наших простейших автоматических тестов:

Листинг 1: Пример *UI* автотеста с использованием *Playwright*

```
test.describe('Список истории операций', () => {
 test.beforeEach(async ({ authorizationSteps, pagesManager,
responseUtils }) => {
   allure.owner('ukozak');
   allure.feature('История');
   allure.epic('История операций');
   allure.suite('Список истории операций');
   allure.labels(
     { name: 'Platform', value: 'Mobile' },
     { name: 'Platform', value: 'Desktop' },
     { name: 'layer', value: 'UI Tests' }
   );
   allure.tag('UI');
    });
 test('Отображение суммы зачисленных А-бонусов в истории
@allure.id=23492', async ({
   pagesManager,
   historyVerificationSteps,
   textUtils
 }) => {
await authorizationSteps.authorize(baseCredentialsModel);
   const [res] = await Promise.all([
     responseUtils.getResponseData(API_ROUTES.history.items),

authorizationSteps.clickElement(pagesManager.headerPage.historyButton)
   ]);
   historyItems = res as HistoryItems;
const { bonusSum, isBadgeRequired } = historyItems.items[29];

  //1 Проверить внешний вид суммы зачисленных бонусов bonusSum
   await historyVerificationSteps.checkElementText(
     pagesManager.historyPage.historyListBonusAmount,

`+${textUtils.formatNumberThousands(bonusSum.amount)}${bonusSum.postfix}
`
   );
   await historyVerificationSteps.checkCSS(
     pagesManager.historyPage.historyListBonusAmount,
     'color',
     COLORS_BONUS.positive
   );
   //2 Проверить отображение галочки на фронте для операции с параметром
isBadgeRequired=true
   await historyVerificationSteps.checkElementPresence(
     pagesManager.historyPage.historyListBonusIcon,
     isBadgeRequired
   );
 });
});
```

*Листинг 2: П*ример *API* автотеста с использованием *Playwright*

```
import {  test, *** } from '../../dependencies';
test.describe('Детали операции', () => {
 let user: TestUserWithOperationModel, authHeaders: { [key: string]:
string };
 test.beforeEach(async ({ usersManager, authorizationSteps }) => {
   allure.owner('UKozak');
   allure.feature('Детали операции');
   allure.labels(
     { name: 'Platform', value: 'Web' },
     { name: 'Microservice', value: 'history-external-web' },
     { name: 'Developer', value: 'UKozak' },
     { name: 'layer', value: 'API Tests' },
   );
   allure.tag('RETENECC2C');
   allure.link('https://wiki.by/page', 'Протокол');
   user = (await usersManager.getUsersWithOperation(CurrencyType.BYN,
TypeOfOperation.TPD))[0];
   authHeaders = await authorizationSteps.firstLogin(user);
 });
 test('Оторбражение кнопки "Добавить в шаблон" в детальке операции
@allure.id=23239', async ({
   apiMethodsSteps,
   verificationSteps,
   historySteps
 }) => {
   allure.tag('GET');
   allure.labels({ name: 'Endpoint', value: 'history/details' });
   const response = await apiMethodsSteps.getRequest(
     `${API_ROUTES.history.details}?id=${user.operationId}`,
     authHeaders
   );
   await verificationSteps.checkResponseStatus(response, StatusCode.OK);
   await verificationSteps.checkResponseSchema(response,
HISTORY_DETAILS_SCHEMA);
   await verificationSteps.checkResponseFieldValue(response, 'id',
user.operationId);
   await historySteps.checkButtonFromShowButtons(response, 'RECEIPT');
   await historySteps.checkButtonFromShowButtons(response,
'ADD_TO_TEMPLATE');
 });
});
```

Для того что бы на проекте можно было использовать *Allure TestOps* для отображения отчетов при использовании *Playwright*, необходимо предварительно установить *allure-playwright*.

1 Пример присвоения аннотаций ко всем тестам при помощи *beforeAll*

Листинг 3: Присвоения аннотаций *beforeAll*

```
test.beforeAll(async () => {
   allure.owner(ukozak);
   allure.feature('Шаблоны: создание, редактирование');
   allure.label({'name':'UI','value':'UI Tests'});
```

```
    allure.link({ url: "***", name: "***" });
  });
```

2 Название *suite*

По умолчанию репортер использует путь к тестовому файлу в качестве имени пакета. Если тесты используют *allure.suite*() и его значение должно использоваться в пользовательских полях *Allure TestOps*, установите опцию *suiteTitle*: *false*.

Листинг 4: Название *suite*

```
const config = {
  reporter: [['allure-playwright', {
    detail: true,
    outputFolder: 'my-allure-results',
    suiteTitle: false
  }]],
};
```

3 Пример использования *allure.label*

Аннотация *label* используется при работе кастовыми полями. *Allure Labels* – это набор пар ключ-значение, которые используются для хранения метаданных в результатах *Allure*. Поддерживаемые типы атрибутов показаны в этой статье

Листинг 5: Пример использования *allure.label*

```
//Если одно значение:
allure.label({'name':'Platform','value':'Mobile'});
//Если два и более:
allure.label([
    {'name':'Platform','value':'Mobile'},
    {'name':'layer','value':'UI Tests'}
]);
```

Чтобы увидеть методы в отчете, необходимо использовать аннотацию *@Step*. Эта аннотация также может работать с параметрами, устанавливая фактическое значение переменной вместо заглушки.

Шаблон оформления тест-кейса в *Allure TestOps*, пример тестового сценарий представлен на рисунке 4, где цифрами обозначены пункты шаблона:

- название тест-кейса на русском языке и из него должно быть понятно, в чем суть тест-кейса;
- в поле Link указываем ссылки на спецификацию (Альфа-Знания) и структуру *API* (*Swagger*, *Google Docs* и/или др.);
- в поле *Description* записываем: для *API Tests* – Метод и *Endpoint* запроса, для *UI Tests* – Описание тест-кейса (необязательно);
- в поле *Precondition* записываем предусловия которые предшествуют данному запросу или другу важную информацию необходимую для выполнения данного запроса (Например: для *UI Tests* можно указать необходимые данные или шаги которые нужно выполнить заранее, чтобы выполнить тест-кейс (например создание предчека, выбор/создание необходимого пользователя и т.д.));
- в *Scenario* пишем проверки, которые необходимо провести. По необходимости можно писать в подшагах ожидаемый результат с ключевым словом [Ожидаю] либо [ОР];

– в поле *Issues links* указываем ссылки на задачи в Альфа-Проекты связанные с данным запросом указываем *Test layer* указываем *API Tests* либо *UI Tests*.

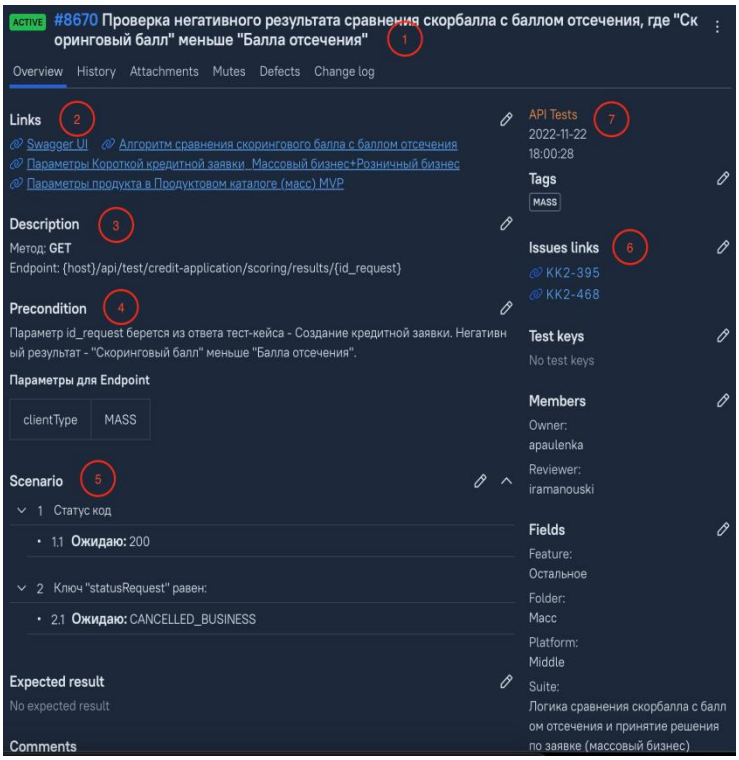

Рисунок 4. шаблон тест-кейса в *Allure TestOPS*

Концепции которых придерживаются на проекте *INSNC*.

1 Отчетность.

Отчет *Allure* отображает шаги и ошибки в тесте четко и понятно, что делает процесс тестирования более информативным и продуктивным. Все действия теста должны быть отражены и легко читаемы.

2 Структурированность.

Каждая папка проекта содержит свой логический слой данных или файлов, точно также каждый файл, следуя принципу единственной ответственности, должен содержать только связанный логикой набор функций и переменных.

3 *Page Object*.

Один из наиболее полезных и используемых архитектурных решений в автоматизации. Данный шаблон проектирования помогает инкапсулировать работу с отдельными элементами страницы, что позволяет уменьшить количество кода и его поддержку. Если, к примеру, дизайн одной из страниц изменен, то нам нужно будет переписать только соответствующий класс, описывающий эту страницу.

4 Уникальность проверок.

Идея заключается в том, что если функциональность проверена одним тестом, то нет смысла проверять ее же следующим тестом. Таким образом это помогает избавиться от лишнего кода, сократить время на потенциально дублирующие друг друга проверки и делает тесты более стабильными, поскольку выполняется меньше лишних действий [3].

5 Шаги.

Все действия направленные на создание каких-либо сущностей или на переход к какой-либо странице приложения выделяются в отдельную прослойку «*steps*», т.к. потенциально могут быть использованы многократно. Такой подход позволяет:

– избавиться от дублирования кода, в частности не требуется каждый раз писать *title* шага;

– отчет содержит подробную инструкцию о том, что конкретно выполнялось в рамках теста.

6 Предсказание ошибок.

В *Allure TestOPS* можно оформить категории вероятных ошибок, что помогает реализовать концепцию *Error Guessing* («Предсказание ошибок»). Суть данного подхода заключается в том, чтобы при анализе и разборе упавших тестов ускорить процесс разбора причин возникновения упавших тестов: *Allure* разбивает кейсы запуска по определенным категориям, которые помечаются лейбами, на рисунке 5 приведены те категории которые на проекте *Web-Insync* используются для разбора упавших *UI* автотестов.

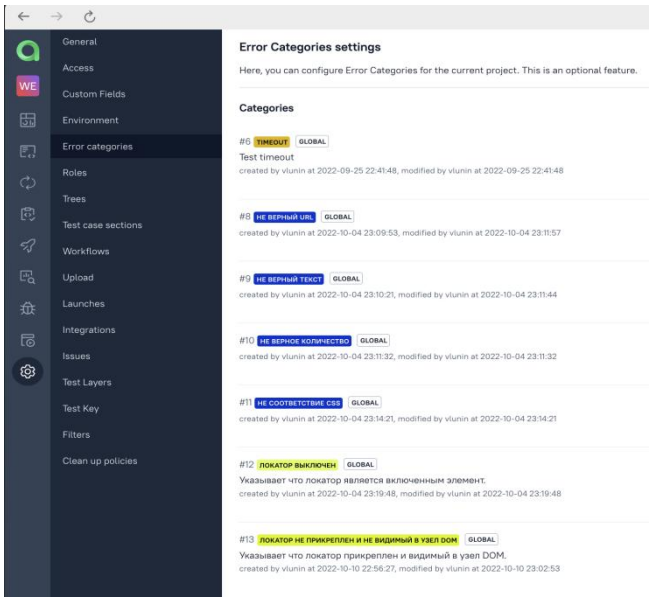

Рисунок 5. Категории ошибок на проекте *Web-Insync*

7 Деревья.

Allure TestOps представляет тест-кейсы с помощью древовидной структуры. Вы можете создавать различные деревья в своем проекте в зависимости от информации, представленной в пользовательских полях ваших тестов, и ваших потребностей. На проекте Insync разделение производится по *Feature, Platform, Suites*. На рисунке 6 приведен пример дерева фильтрации тестовых сценариев на проекте *Web-Insync*.

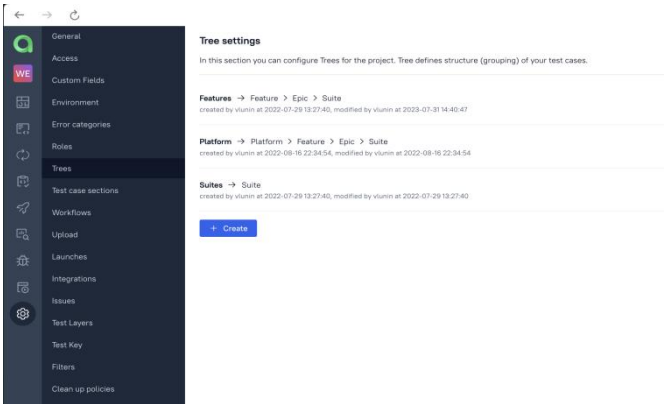

Рисунок 6. Деревья сценариев на проекте *Web-Insync*

8 Дешборды (Графики)

На проекте применяется 2 основных типа дешбордов - стандартные, которые по умолчанию присутствуют в разделе *Dashboard* в *Allure TestOPS*, и пользовательские, которые создаются по нуждам проекта в дополнение к стандартным [4].

Standard dashboard (стандартная панель) предоставляет мгновенное представление о статусе тестов, которые проводятся на проекте, и находится данная панель в разделе *Dashboard*s проекта. Пример стандартной панели на проекте *Web-Insync* приведен на рисунке 7.

Рисунок 7. Стандартная панель *Web-Insync*, покрытие тестами

Как видно из рисунка на проекте достаточно большое количество тестовых сценариев, как мануальных, так и автоматизированный и данная панель помогает наглядно видеть результаты работы над тестированием на проекте.

С помощью стандартной панели можно увидеть следующие сведения:

- количество тестовых случаев и их распределение по состояниям (активные, на рассмотрении, устаревшие и т. д.);
- количество тестовых случаев и распределение между ручными и автоматизированными тестами;
- тренд автоматизации с информацией о количестве тестов за последние 14 дней;
- количество тестов в день и их статус (провален, пройден);
- продолжительность выполнения тестов в день;
- тренд Mutes, т.е. количество тестов, исключенных из статистики из-за постоянного состояния провала или состояния *flaky*.

Целью пользовательских панелей является предоставление дополнительных аналитических данных для лиц, принимающих решения. Конечный пользователь создает пользовательскую панель в разделе панелей проекта (кнопка + *New Dashboard* в правом верхнем углу). Дополнительная аналитическая информация предоставляется с помощью виджетов. Пользовательские панели на проекте *Web-Insync* приведены на рисунках 8-9.

Рисунок 8. Пользовательская панель *Web-Insync*, покрытие тестами

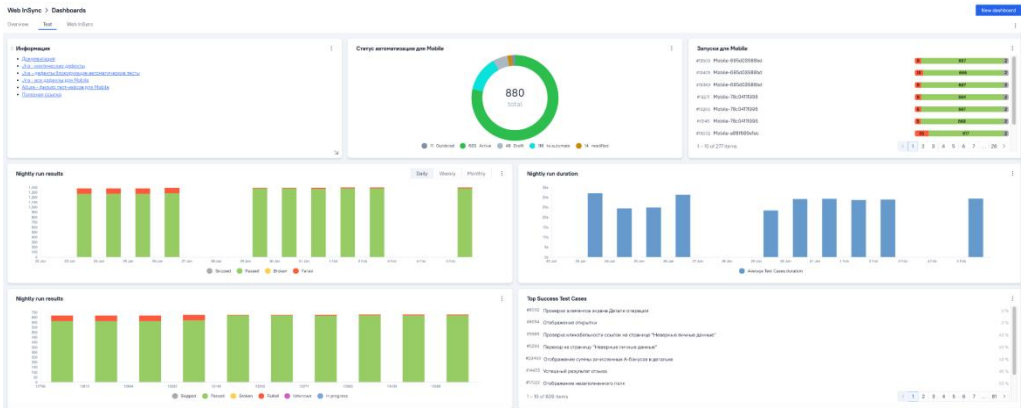

Рисунок 9. Пользовательская панель *Web-Insync*, общая информация

9 Динамическая документация

На проекте *Web-Insync* принятой практикой для создания тест-кейсов является генерация тестовой документации (тест-кейсов) на базе прогона автотестов. На рисунке 10 приведен пример тест-кейса сгенерированного на базе автотеста [5].

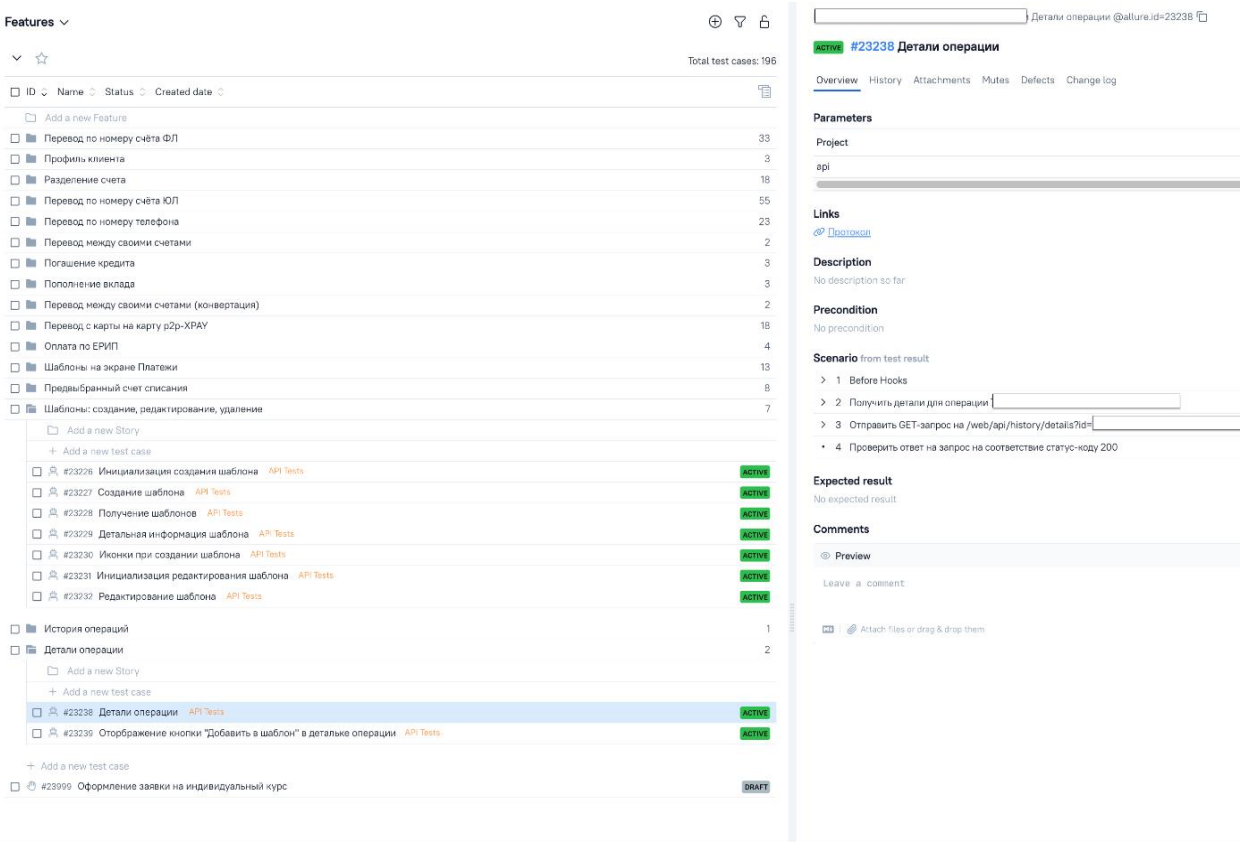

Рисунок 10. Пример тест-кейса, сгенерированного на базе кода автотеста

Как уже было описано ранее, весь код автотестов при написании тестировщик оборачивает в специальные блоки – *Steps*, которые *Allure* считывает и оформляет в понятный человеку набор проверок с ожидаемым результатом.

10 Интеграция *Allure* с *Grafana*

На крупных проектах необходимо управлять тестированием и принимать решения о релизах и понимать, есть ли в данный момент какие-то проблемы или нет. В ходе работы над данным процессом, возникает ряд проблем:

–разработчики пренебрегают изучением отчётов о тестировании;

–мало кто заглядывает в *Allure*, а если заглядывают, то не вполне понимают локаторы и сами сценарии;

–руководителям не хочется погружаться в детали – они хотят иметь картинку перед глазами;

–недостаточное покрытие и как его оценить;

–недостаточная скорость работы тестов.

Чтобы преодолеть эти трудности, нужно научиться измерять работу тестов. В этом помогает *Grafana* – инструмент, которым пользуются разработчики, девопсы, аналитики, продакт-менеджеры. Grafana отображает графики и результаты аналитики [6].

11 Интеграция *Allure* с *Jira*

Настроив интеграцию *Allure TestOps* с *Jira*, создается отдельная страница со всеми запусками (*Launches*) в *Allure TestOps*, пример данной страницы показан на рисунке 11. Страница Launches показывает эту информацию в структурированном виде, позволяет отфильтровать ее и получить нужный срез данных (запущенные тесты, процент успешно выполненных и упавших тестов и другую схожую информацию) по конкретной ветке.

Если провалиться в конкретный *Launch*, можно посмотреть на структуру проекта: какие есть компоненты, какие фичи, сколько было перезапусков, какие именно тесты перезапускались, информацию о дереве тест-кейсов, график, который показывает сколько времени бежал самый долгий тест, график с длительностью тестов, – и в конце концов экспортировать этот прогон в *JIRA*-тикет. Отдельно хочется упомянуть про политики для *Live documentation*, которая в автоматическом режиме обновляет дерево тест-кейсов по правилам и триггерам, например, только из ветки мастер.

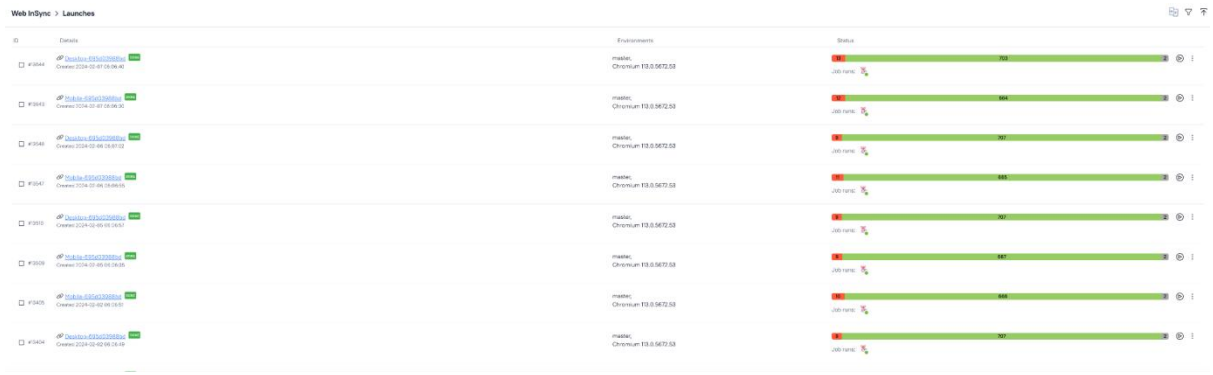

Рисунок 11. Страница запусков *Web-Insync*

Работает это так: у нас есть автотест, в котором изменили пару шагов и поправили аннотации. Далее этот тест запускается и выполняется с набором тестов на мастер-*branch*, после чего можно либо вручную закрыть launch, либо дождаться, когда *launch* будет закрыт по *cron'y* (расписание запусков), который настраивается в интерфейсе *Allure*. При закрытии запуска старый автотест обновляется - получаем актуальный тест-кейс. Это позволяет забыть про актуализацию тестовой документации в интерфейсе [7].

Заключение. Использование в тестировании на больших проектах, с множеством тестовых сценариев автоматизации делает работу быстрее, а анализ данных и метрик намного проще. В ЗАО «Альфа-Банк» тестирование проводится в соответствии с методологией *TestOPS*, где тестировщики выполняют работу как с по мануальному так и

автоматизированному тестированию функционала, а также используют в работе *Allure TestOPS* для автоматизации процесса отчетности и создания тестовой документации. Обработка и анализ тестовых метрик, таких как покрытие авто-тестами и результаты упавших тестов структурируются в *Allure Dashboards*, что позволяет бизнесу более наглядно анализировать результаты тестирования, а тестировщиках оценивать скоуп своей работы и планировать расширение фреймворка на базе имеющихся сведений. Также данный способ организации тестовых сценариев делает процесс регрессионного тестирования в конце спринта проще, что ускоряет работу, а соответственно экономит деньги.

Список литературы

[1] Инструменты автоматизации тестирования [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/768154/

[2]Как мы внедряли Allure TestOps в стриминговом сервисе [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/ru_mts/articles/689330/

[3] Паттерны автоматизации и архитектура автотестов [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/777262/

[4] Dashboards Allure TestOPS [Электронный ресурс]. – Режим доступа: https://docs.qameta.io/allure-testops/briefly/dashboards/

[5] QA Meta blog [Электронный ресурс]. – Режим доступа: https://qameta.io/blog/

[6] Grafana и автотесты: учимся измерять работу тестов [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/ozontech/articles/657933/

[7] Allure. В поисках почти идеальной TMS [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/571476/

Авторский вклад

Авторы внесли равноценный вклад в написании статьи

AUTOMATION OF THE TESTING AND REPORTING PROCESS ON THE WEB-INSYNC PROJECT AT ALFA-BANK

U. M. Kozak
BSUIR student, FullStack QA at AlfaBank, Republic of Belarus

V. G. Lunun
Software Quality Assurance Manager at AlfaBank, Republic of Belarus

Annotation. On large projects the coverage of autotests is growing day by day and when there are a lot of test scenarios, it becomes difficult to keep documentation and reports, as well as to keep track of autotest runs from different places. To solve this problem on the Web-Insync project of CJSC «Alfa-Bank» a special TestOPS methodology is used, which is implemented using the Allure TestOPS tool, which allows to store all documentation, test runs and reports in one place and on the basis of these data to get graphs, analyze the work of the framework and create a dynamic analysis.

Keywords: Allure TestOPS, test automation, Playwright, dynamic documentation