Федеральное государственное автономное образовательное учреждение

высшего образования

"Российский университет транспорта" (МИИТ)

Кафедра «Цифровые технологии управления транспортными процессами»

**ОТЧЁТ**

**ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ**

«Разработка программного обеспечения для тестирования Web-приложений по заданной архитектуре»

**Тип практики:** ПРЕДДИПЛОМНАЯ ПРАКТИКА

Выполнил ст. г. УВА-411 Д. Н. Романов

Принял руководитель практики

доцент кафедры ЦТУТП Е.Я. Соймина

Москва  
2022

|  |
| --- |
| УТВЕРЖДАЮ  Профессор кафедры ЦТУТП, д.т.н.  \_\_\_\_\_\_\_\_\_\_\_ Э. К. Лецкий  « » 2022 год |

**ЗАДАНИЕ НА ПРАКТИКУ**

**(преддипломная)**

студента группы УВА-411 Романова Даниил Николаевича

**Наименование темы:**

Разработка программного обеспечения для тестирования Web-приложений по заданной архитектуре.

**Необходимо изучить:**

1. Паттерны проектирования Java приложений
2. Архитектурные решения для разработки автоматизированных тестов
3. Allure. Динамические отчёты о тестировании.

**Необходимо самостоятельно выполнить:**

1. Разработку программного обеспечения для тестирования Web-приложений

2. Обеспечение запуска тестов в многопоточном режиме

3. Демонстрацию контрольного примера

4. Генерацию динамических отчётов о проведённом тестировании

**Основные требования к проделываемой работе:**

1. Представить описание шагов тестовых сценариев.
2. Использовать ГОСТ Р 7.32-2001 – для оформления пояснительной записки.

**Источники информации:**

1. Роман Савин «Тестирование Дот Ком». – Москва: «Roman Savenkov», 2017. – 312 с.
2. Джейсон Арбон, Джефф Каролло, Уиттакер Джеймс «Как тестируют в Google» – СПб.:«Питер», 2014 – 320. с.

**Материалы к защите:**

1. Заполненный, подписанный и утверждённый лист индивидуального задания на технологическую практику.
2. Пояснительная записка (отчёт).

|  |  |
| --- | --- |
| **Студент группы** **УВА-411** | Романов Д.Н. |
| **Руководитель практики** | Соймина Е.Я. |

Оглавление

[1. Введение 4](#_Toc105266251)

[2. Тестирование как элемент цикла разработки ПО 5](#_Toc105266252)

[3. Разработка фреймворка для тестирования Web-приложений 7](#_Toc105266253)

[3.1 Модуль с тестовыми сценариями 10](#_Toc105266254)

[3.2 Связующий модуль 13](#_Toc105266255)

[3.3 Функциональный модуль 13](#_Toc105266256)

[4. Обеспечение запуска тестов в многопоточном режиме 15](#_Toc105266257)

[5. Генерация динамических отчётов о проведённом тестировании 17](#_Toc105266258)

[6. Демонстрация контрольного примера 20](#_Toc105266259)

[7. Заключение 26](#_Toc105266260)

### Введение

В современном мире внедрение информационных технологий становится неотъемлемым требованием в различных областях человеческой деятельности.

Крупным IT компаниям приходится постоянно решать новые задачи, гибко подстраиваясь под требования рынка. Постоянно растёт потребность в разработке нового программного обеспечения. Решения находят быстро, но зачастую они не всегда оказываются наиболее эффективными.

Каждая новая программа (система), проходит несколько жизненно важных этапов разработки. Одним из этапов является тестирование программного обеспечения. Задачи организации тестирования программного обеспечения каждая компания решает по-своему, но проблема остаётся на сегодняшний день открытой.

Для проведения тестирования ПО требуется постоянно растущий штат специалистов и серьезные временные затраты, что оборачивается для компании большими убытками.

Проблема является наиболее актуальной для крупных компаний, в которых тестированию подлежит большой объем создаваемого программного обеспечения.

Цель исследования – сделать процесс тестирования программного обеспечения гибким, не ресурсоемким и удобным.

Задачи практики:

* разработку программного обеспечения для тестирования web-приложений
* обеспечение запуска тестов в многопоточном режиме
* подготовку контрольного примера
* генерацию динамических отчётов о проведённом тестировании

### Тестирование как элемент цикла разработки ПО



Рис. 2 - Схема взаимодействия разработчика, тестировщика и заказчика

Разработчик постоянно реализует новый функционал для приложения и исправляет выявленные ошибки. Тестировщик занимается проверкой работы приложения и передаёт его заказчику. Заказчик в свою очередь получает приложение и предлагает идеи по его улучшению, исправлению не выявленных при тестировании ошибок и т.д.

Ручное тестирование – трудоёмкий процесс, который можно автоматизировать. На рынке существует множество решений, которые позволяют работать с браузером, базами данных и API приложений программным путём. Используя автоматизированные тесты, можно отказаться от специалистов по ручному тестированию.

Ручное тестирование проводится человеком, он играет роль конечного пользователя, на которого ориентированно разрабатываемое приложение. Рассмотрим ручное тестирование web приложений. В его рамках обычно предусматривают прямое открытие стенда с приложением и его непосредственной эксплуатации в рамках написанных сценариев. Это долгая и очень дорогая процедура, так как приложение всегда имеет достаточного широкий функционал, который необходимо проверить. И делать это нужно постоянно, так как приложение всё время находится в разработке, и любая ошибка разработчика может оказать воздействие на уже протестированный функционал.

Итого мы имеем множество однообразных действий, которые выполняются человеком в ручном режиме. Такие процессы в реальной жизни принято автоматизировать, тестирование не стало исключением.

Автоматизированные тесты (Автотесты) – это программа, которая способна эмулировать поведение конечного пользователя в системе, тем самым проверяя приложение, генерировать отчёты об ошибках, проделанных действиях и полученных результатах. Возможности «автотестов» очень широки и применяются повсеместно.

### Разработка фреймворка для тестирования Web-приложений

Предлагается разработать приложение из трёх компонентов:

* Функциональный модуль. Содержит основные функциональные возможности автоматизированного теста.
* Модуль с тестовыми сценариями. Содержит тестовые сценарии.
* Связующий модуль. Преобразует тестовые сценарии в полноценный автоматизированный тест.

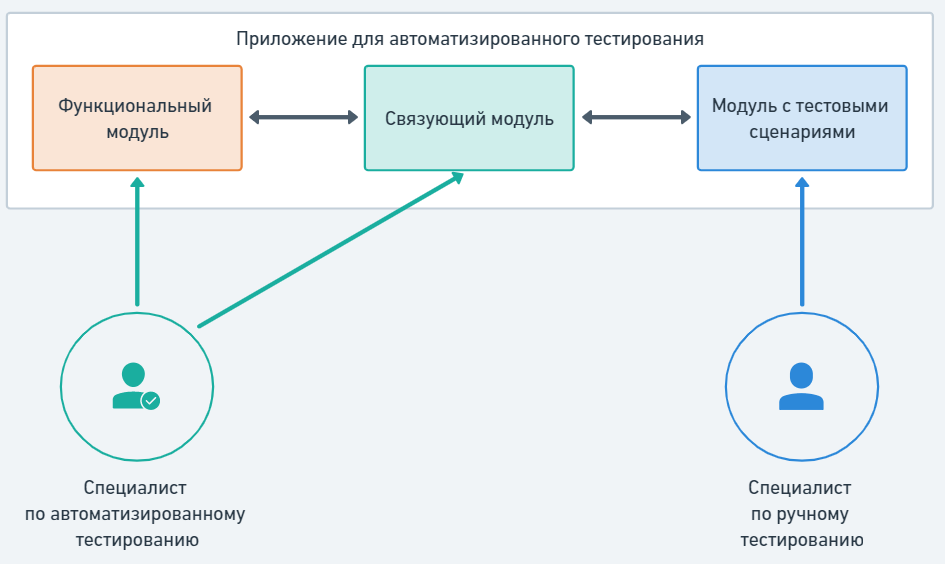


Рис. 3 - Схема взаимодействия тестировщика и автотестера №1

Специалист по автоматизированному тестированию реализует функциональный и связующий модули, которые предоставят готовый конструктор по внедрению новых автоматизированных тестов. Обеспечит стабильную работу этих модулей и займётся их поддержкой.

Специалисты по ручному тестированию напишут тестовые сценарии для проверки работоспособности приложения. Сценарии станут основой для вызова функционального модуля, который и обеспечит работу теста.

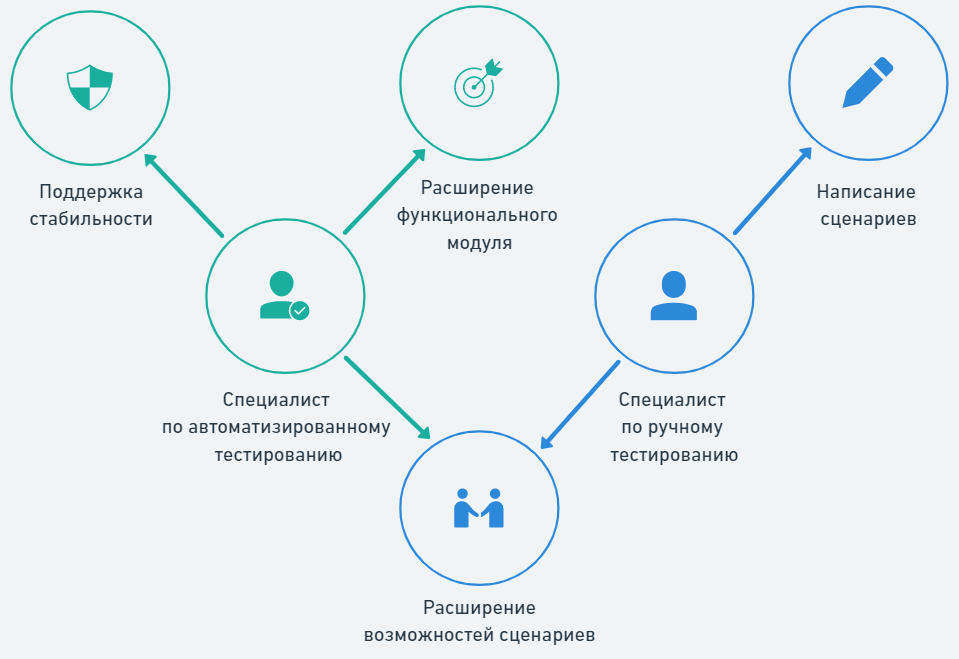


Рис. 4 - Схема взаимодействия тестировщика и автотестера №2

Такой вариант решения проблемы тестирования ПО позволит специалистам по ручному тестированию функционально конструировать автоматизированные тесты.

Целевое решение имеет ряд преимуществ, таких как:

* Тестовые сценарии, написанные на русском языке
* Высокая скорость тестирования.
* Низкая стоимость тестирования.
* Повышение эффективности работы специалистов по ручному тестированию.
* Повышение эффективности работы специалистов по автоматизации тестирования.
* Конструктор для создания автоматизированных тестов.

Была разработана следующая архитектура.

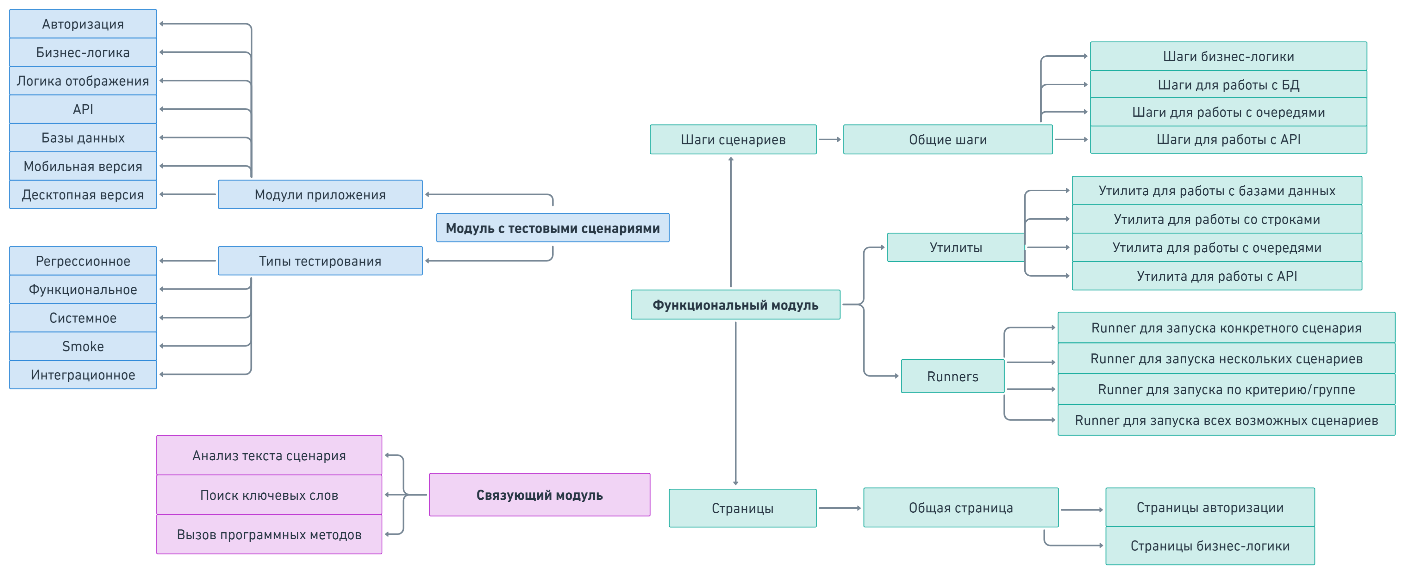


Рис. 5 – Архитектура гибридного фреймворка.

Каждый модуль является самостоятельной единицей и способен встраиваться в любые приложения как в комбинации с другими, так и индивидуально.

Модуль с тестовыми сценариями подразумевает под собой набор тестов, написанных для всех предыдущих релизов, образующих тесты типа регрессионные и набор для нового релиза.

Функциональный модуль представляет из себя полноценное приложение для создания автоматизированных тестов, со своей архитектурой, паттернами и программной реализацией.

Связующий модуль представляет собой жесткую связь слово – метод, в которой словом является шаг в тестовом сценарии, а методом фрагмент реального программного кода.

### Модуль с тестовыми сценариями

Модуль тестовых сценариев предлагается реализовать через структуру папок и файлов внутри проекта с исходным кодом. Архитектура построения папок будет подразумевать разбиение сценариев по логическим частям тестируемого приложения. Каждая папка будет содержать в себе сценарии исключительно определённой части функционала приложения.

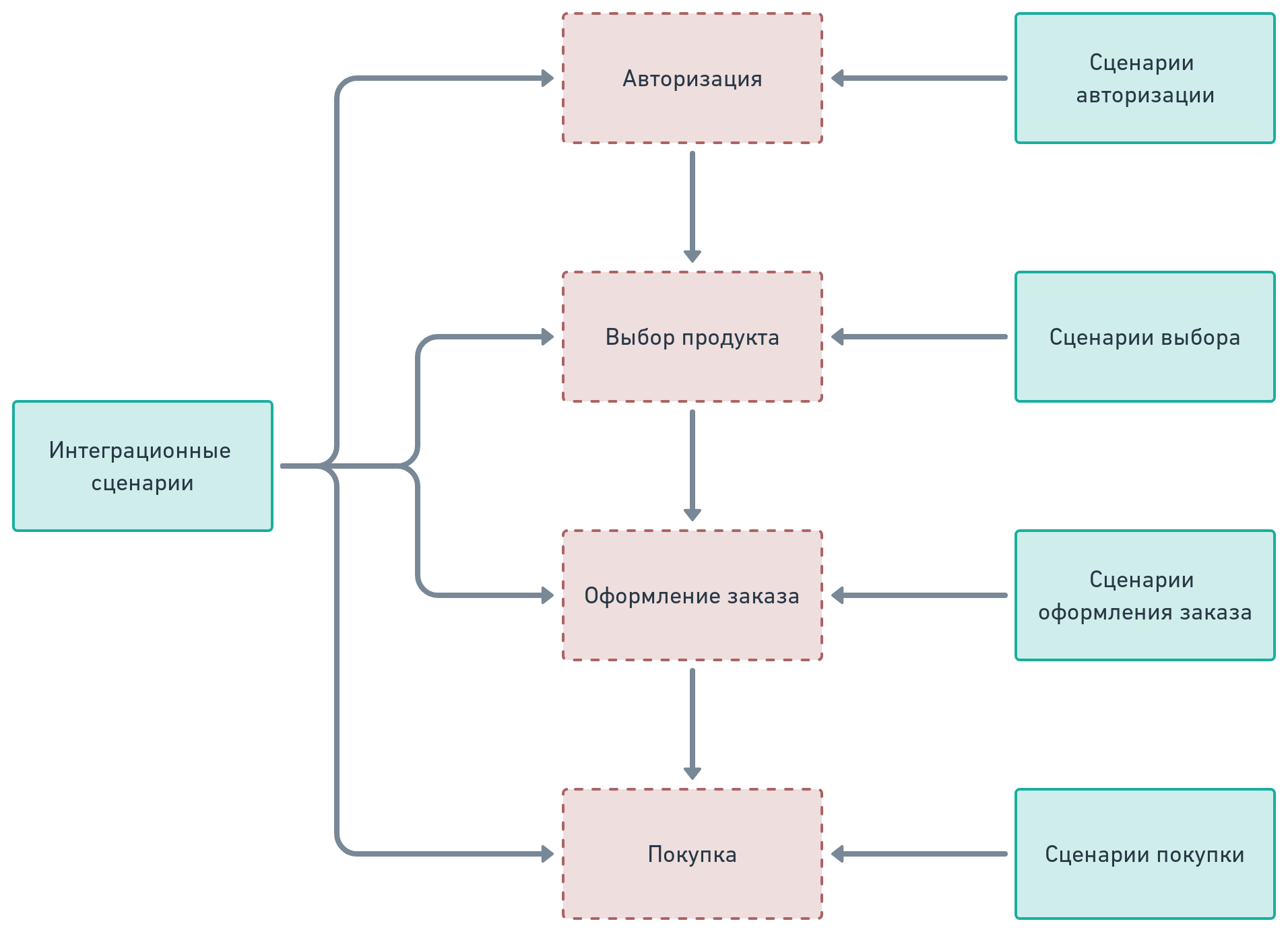


Рис. 5 – Структура сценариев для интернет-магазина

Из предложенной схемы видно, что на каждое возможное действие пользователя у нас имеются, как и отдельные сценарии, так и полные маршруты по всему функционалу.

Для этого создадим специальные директории, в которые разместим файлы с тестовыми сценариями.

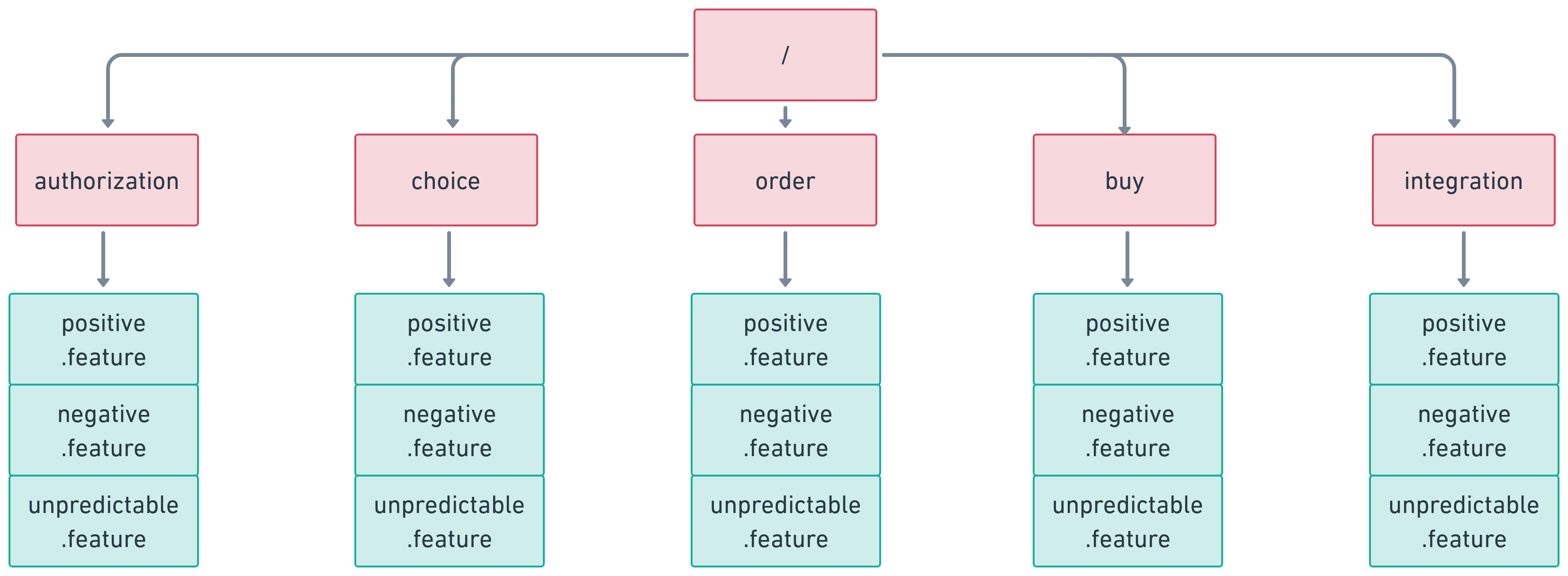


Рис. 6 – Структура папок и файлов

В корне схемы находится исходный каталог, может быть любым. Красным цветом обозначены директории, зеленым цветом файлы в формате feature.

Такое разбиение позволит быстро масштабировать сценарии при их огромном количестве и легко ориентироваться в нужных элементах тестируемого приложения. Наглядно можно увидеть, что уже покрыто тестами, а что находится в разработке. Разделение тестов на папки является хорошей практикой в проектировании архитектуры тестовых сценариев.



Рис. 7 – Файл с примером тестового сценария

Специальное слово «Feature» служит для объявления набора тестов, у него есть описание и название.

Слово «Scenario» означает объявление самого теста, имеет имя.

Слово «Then» означает шаг внутри теста, например, «Нажать на кнопку подтверждения» значит найти на странице кнопку типа «submit» и кликнуть на неё.

Ключевые слова позволяют однозначно определить, что является шагом теста, а что комментарием или описанием. Они служат для связи сценариев с функциональным модулем.

### Связующий модуль

Связующий модуль реализован при помощи библиотек компании «Cucumber», в нём задействован функционал связи ключевых слов с кодом проекта. Реализация основана на имплементации ключевых слов в файлах feature и их связи с реальными тестовыми методами через аннотации @Then.

Cucumber ищет среди классов проекта методы помеченные аннотацией @Then и исполняет их, передавая необходимые аргументы или без таковых.

Стоит дополнить, что связующий модуль является лишь удобным решением для связи текста с программным кодом, никакой другой логики он не подразумевает, потому реализация этого модуля не было изменена в рамка выпускной квалификационной работы.

Таким образом, связующий модуль используется исключительно как сторонняя библиотека для реализации связи между описанием тестового сценария и исполнением реального кода проекта.

### Функциональный модуль

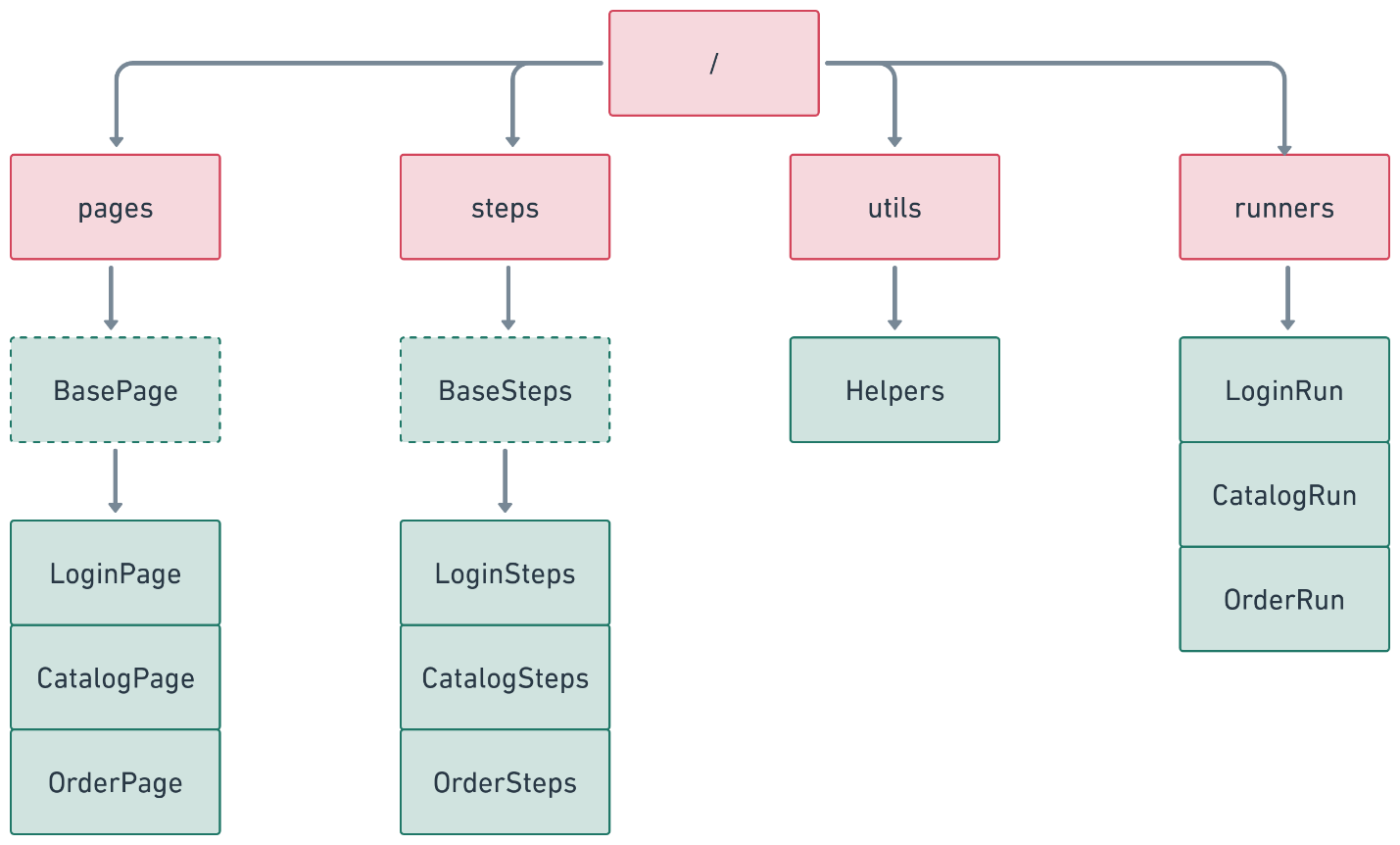


Рис. 8 – Структура классов и директорий

В корне схемы находится исходный каталог, может быть любым. Красным цветом обозначены директории, зеленым цветом программные классы. Абстракции изображены пунктиром.

Директория pages содержит в себе классы, описывающие элементы web страниц и действия, которые можно производить над ними. Она служит для предоставления программисту набора методов и функций для работы с конкретной страницей тестируемого приложения

Директория steps содержит классы реализующие тестовые шаги, описанные в конкретных сценариях. Шаги являются конечной абстракцией в тестах, именно в них находится самая низкоуровневая логика теста.

Директория runners содержит классы, отвечающие за запуск тестов и сканирование определённого сценария из выбранного файла feature.

Директория utils содержит вспомогательные классы - утилиты, которые упрощают работу со специфичными инструментами.

### Обеспечение запуска тестов в многопоточном режиме

Автоматизированные тесты в больших количествах при последовательном запуске могут выполняться достаточно долго. На практике их выполнение происходит непосредственно перед развёртыванием тестируемого приложения, что может приводить к замедлению процессов релиза конечного приложения.

Целесообразно будет обеспечить запуск автоматизированных тестов в многопоточном режиме, тем самым ускорив процесс тестирования в несколько раз. При этом всё также оставляя перед специалистами по ручному тестированию простой инструмент для написания автоматизированных тестов, не обременяя их работой со сложными механизмами многопоточности.

На рынке уже существуют решения, которые обеспечат работу тестов в многопоточном режиме, например, TestNG.

TestNG — это тестовый раннер, созданный, для упрощения работы с тестами. Он широко используется вместе с Selenium (Selenide), более мощный, чем некоторые современные аналоги, такие как JUnit 4 и JUnit 5.

Обладает существенными преимуществами в управлении потоком выполнения программы. Его архитектура позволяет сделать автоматизированные тесты более структурированными и запускать тесты в параллельном режиме.

Для формирования тестового набора достаточно объявить конфигурационный файл в формате XML.

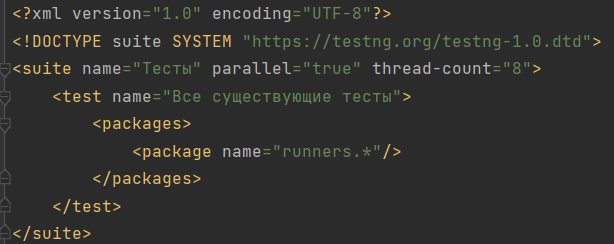


Рис. 9 – Конфигурационный файл TestNG в формате XML

Ключевые свойства конфигурационного файла для запуска автоматизированных тестов это parallel и thread-count, которые обеспечат запуск в многопоточном режиме, используя 8 потоков.

Если тестов будет больше, чем доступных потоков, то остальные тесты будут ждать исполнения в очереди и далее занимать освободившийся поток. Таким образом, можно ускорить выполнение больших групп тестов в несколько раз.

Свойство packages указывает на директорию с тестами, относительно папки с исходным кодом. Именно там будут необходимые тесты.

Так как в проекте используется иная форма оформления тестов, необходимо будет написать простой адаптер для запуска сценариев. Адаптеры называются runner.



Рис. 10 – Runner класс для запуска тестов

В свойствах указывается папка в которой содержатся файлы в формате «feature», далее они сканируются и осуществляется запуск тестов в многопоточном режиме.

Демонстрация запуска приведена далее.

### Генерация динамических отчётов о проведённом тестировании

Ключевая задача любого тестирования – это выявление ошибок в конечном тестируемом продукте и в дальнейшем предоставление шагов к её воспроизведению. Для таких целей в ручном тестировании применяется написание отчёта о тестировании в документах Microsoft Office Word.

В автоматизированном тестировании отчёты, как правило, генерируются автоматически. Есть множество готовых инструментов для генерации отчётов, например, sure-fire и testng-reports. Но самыми наглядными и эффективными на фоне остальных выглядят отчёты Allure2.

Allure2 – это инструмент с открытым исходным кодом, разработанный компанией Яндекс. Он позволяет практически на любом из популярных тестовых раннеров генерировать красивые отчёты о проведённом тестировании.

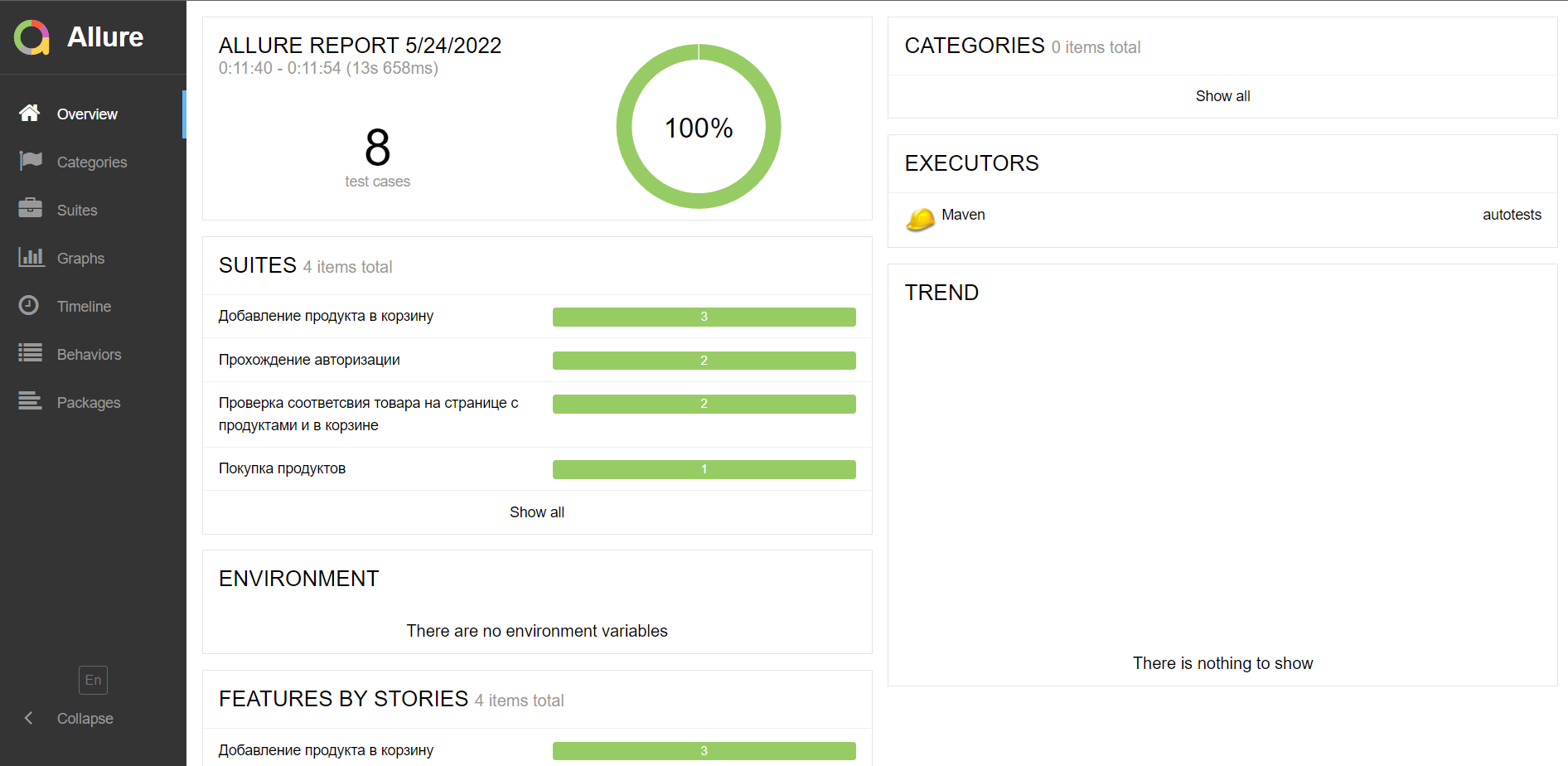


Рис. 11 – Allure2 главная страница

Allure2 по окончании тестирования запускает http сервер на котором расположены отчёты, разбитые на категории, с подробной информацией о запуске тестов.

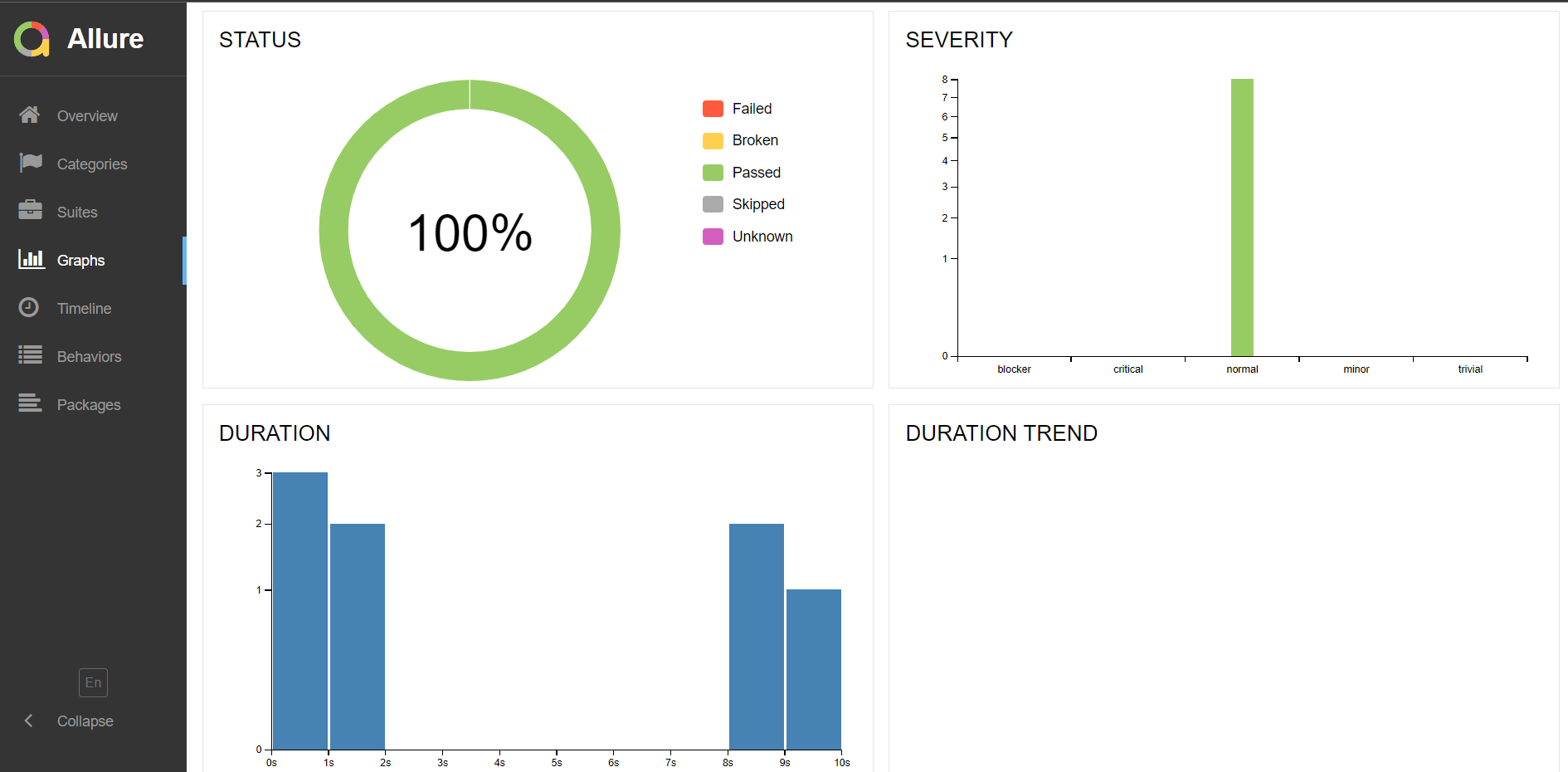


Рис. 12 – Allure2 построение графиков на основе тестовых данных

Так же отчёты содержат информативные графики результатах тестов, их количестве и пропорции по категориям.

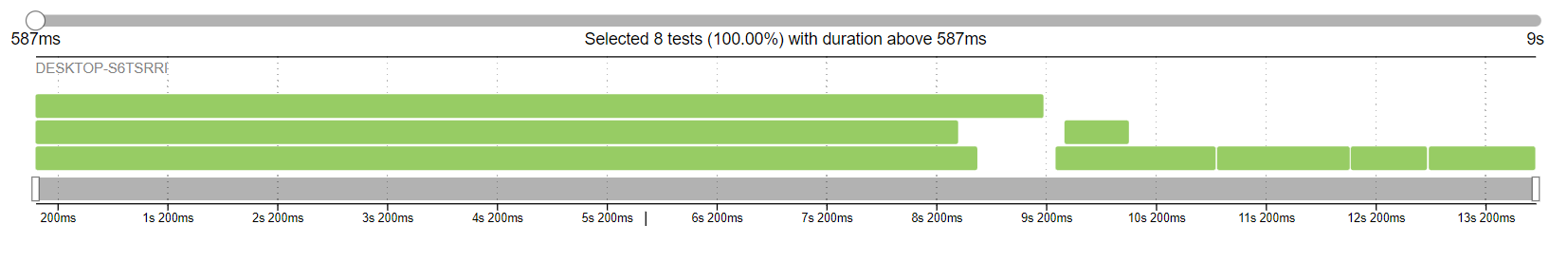


Рис. 13 – Allure2 время выполнения тестов в 3 потока

Allure способен фиксировать время выполнения тестов от начала и до конца, отображая и длительность. На рисунке 13 было запущено три теста в параллельном режиме, потому в начале графика видно три параллельные зелёные полосы.

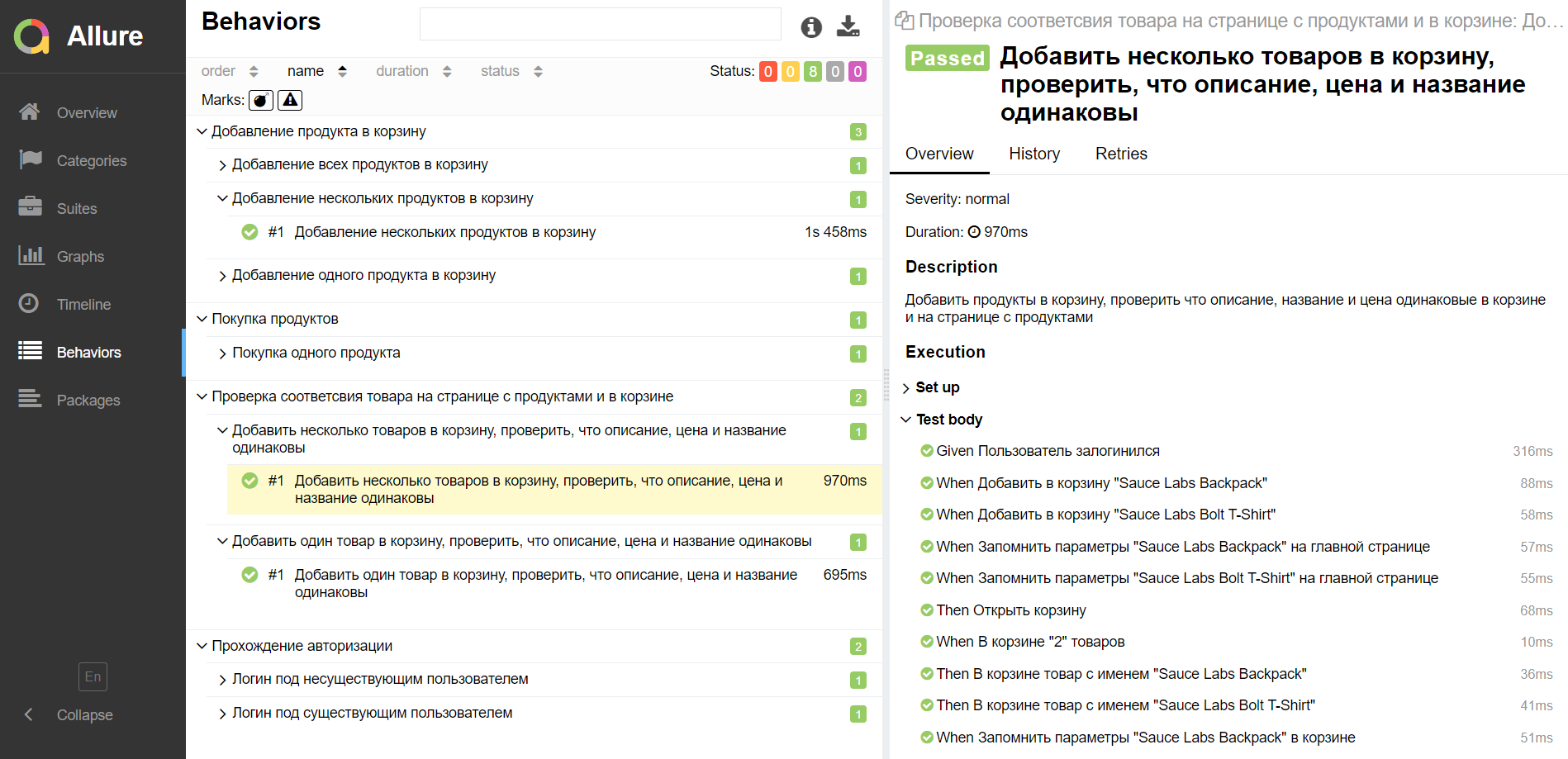


Рис. 14 – Allure2 иерархия тестов

На рисунке 14 изображена самая важная часть отчёта, на ней можно легко увидеть какие именно тесты, из каких категорий и с каким статусом были завершены. Можно просмотреть каждый тест отдельно и увидеть какие шаги были выполнены.

### Демонстрация контрольного примера

В качестве контрольного примера предлагается рассмотреть простое web приложение (интернет-магазин), на которое будут написаны тесты с использованием архитектуры описанной в предыдущих главах.

Тестируемое приложение имеет следующий функционал:

* авторизация пользователя,
* просмотр продукта,
* выбор товара из каталога,
* наполнение корзины с продуктами,
* оформление заказа.

Для начала необходимо написать тестовый сценарий, который далее будет реализован специалистами по автоматизированным тестам.

Составим простой сценарий для проверки возможности оформления заказа с одним товаром:

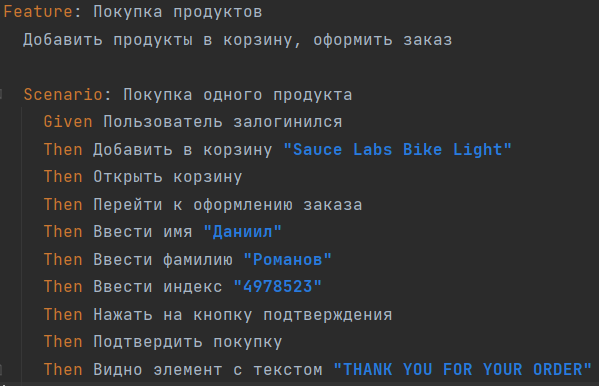


Рис. 15 – Сценарий

Данный сценарий содержит предусловие к выполнению теста, пользователь должен пройти авторизацию перед совершением покупки.

Далее происходит добавление товара в корзину, заполнение данных по доставке, оплата и оформление.

Реализация сценария подразумевает под собой реализацию каждого шага, описанного в сценарии. Например, реализуем авторизацию на данной странице.



Рис. 16 – Страница авторизации

На странице с авторизацией используется всего три элемента, это поле ввода имя пользователя, поле ввода пароля и кнопка логина. Соответственно нам необходимо реализовать Java класс, который опишет все необходимые методы для работы с этой страницей, а именно два поля для ввода логина и пароля.

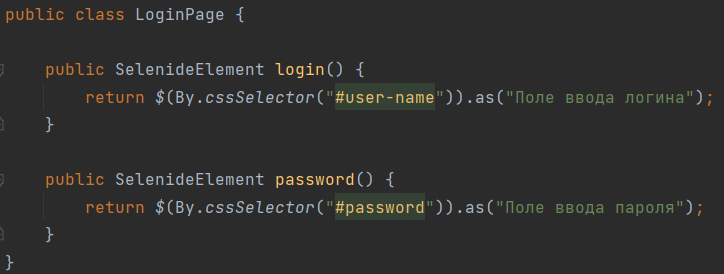


Рис. 17 – Класс LoginPage

В классе LoginPage не реализован метод для поиска кнопки подтверждения из-за того, что она не является уникальной и присутствует на множестве других страниц. Поэтому она реализована в другом классе.

В этом классе содержатся методы, которые возвращают элементы DOM в виде объектов класса SelenideElement, у которых есть множество методов по управлению, таких как click (нажать на элемент), value (ввести значение) и т.д.

Поиск элементов в данном случае завязан на CSS селекторах и реализован через поиск элемента по уникальному идентификатору, что является самым надёжным способом найти элемент на странице.

Реализуем методы для ввода логина и пароля по отдельности, так как скорее всего они нам понадобятся в дальнейших тестах, например, при тестировании страницы с авторизацией. Создадим класс для реализации шагов, внутри которого создадим объект класса LoginPage и с его помощью опишем шаги по вводу логина и пароля.

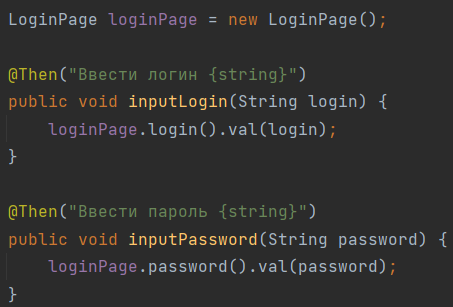


Рис. 18 – Класс LoginSteps

Реализация шагов по вводу логина и пароля осуществляется с помощью метода val, который позволяет вводить текст в элементы DOM типа input. Таким образом, после реализации этих шагов мы можем приступить к реализации предусловия данного сценария.

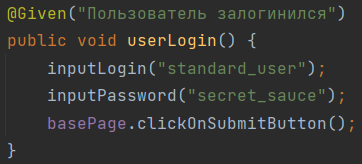


Рис. 19 – Метод userLogin

Здесь добавляется общий метод по нажатию кнопки подтверждения, его реализация содержится в классе BaseSteps.

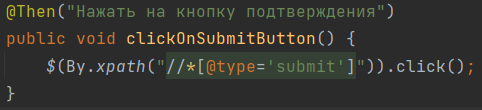


Рис. 20 – Метод clickOnSubmitButton

Данный метод ищет все элементы с типом Submit, реализация происходит средствами XPath, и сразу же вызывается метод click, т.к. никаких других действий с кнопкой подтверждения сделать невозможно.

Таким образом реализуются все оставшиеся шаги сценария. Главное, что необходимо учитывать при реализации – это возможность использования написанного кода в будущем, следуя этим принципам получится покрыть потребности любого возможного сценария.

Время выполнения данного сценария составило 5.744 секунды, по окончании сценария генерируется отчёт о проделанной работе, в котором представлены шаги, время их выполнения, описание каждого шага и т.д.

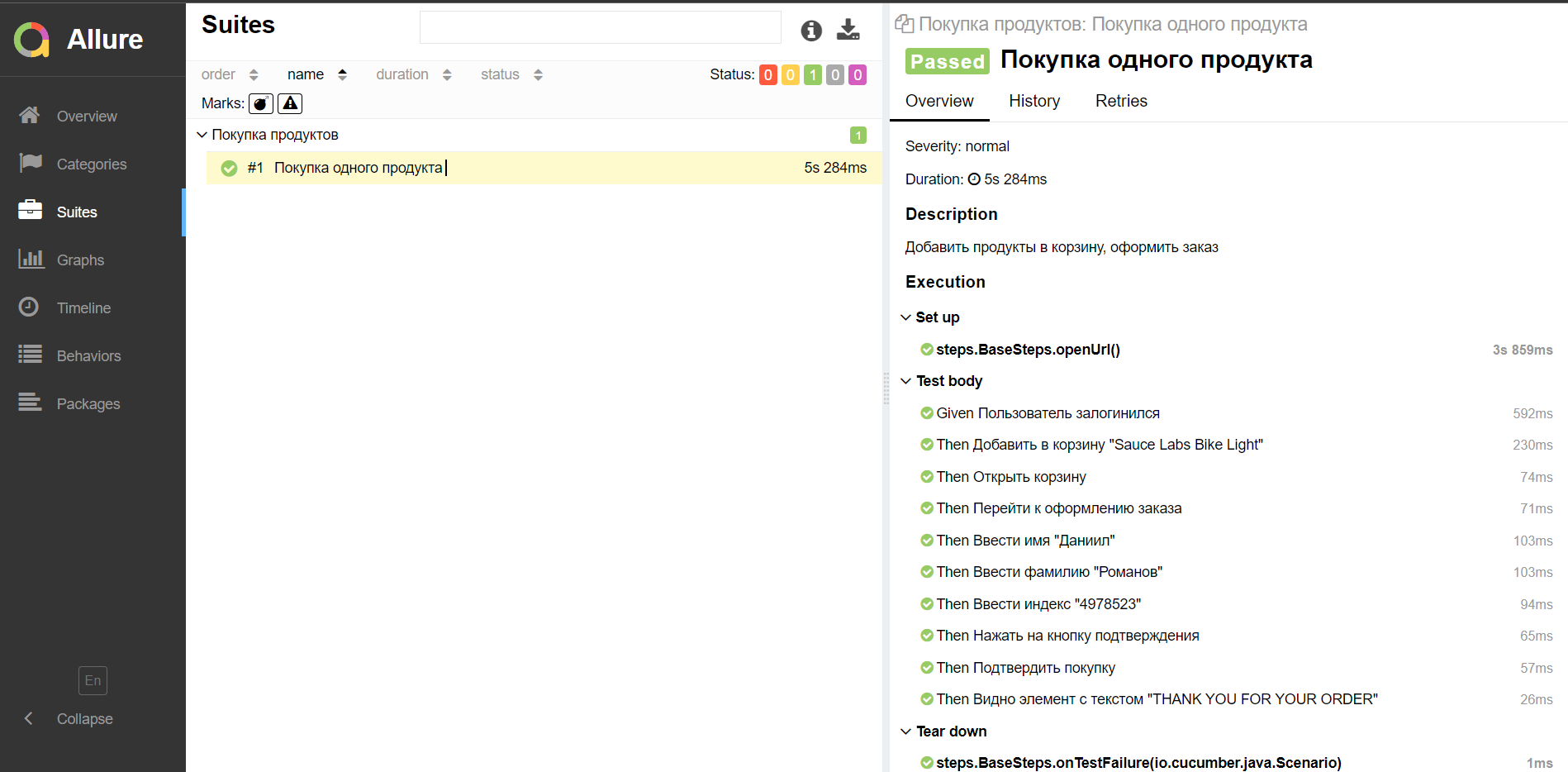


Рис. 21 – Allure2 отчёт об успешном выполнении теста

В отчёте представлен каждый шаг, время его выполнения, предусловия и постусловия выполнения теста. Соответственно при запуске множества тестов мы получаем отчёт о выполнении каждого теста. Если происходит ошибка, то она выводится в отчёт и к нему прикрепляется снимок экрана, чтобы специалист по ручному тестированию мог легко разобраться в причинах ошибочного поведения. Для демонстрации изменим последний шаг и будем ожидать элемент, которого нет на экране.

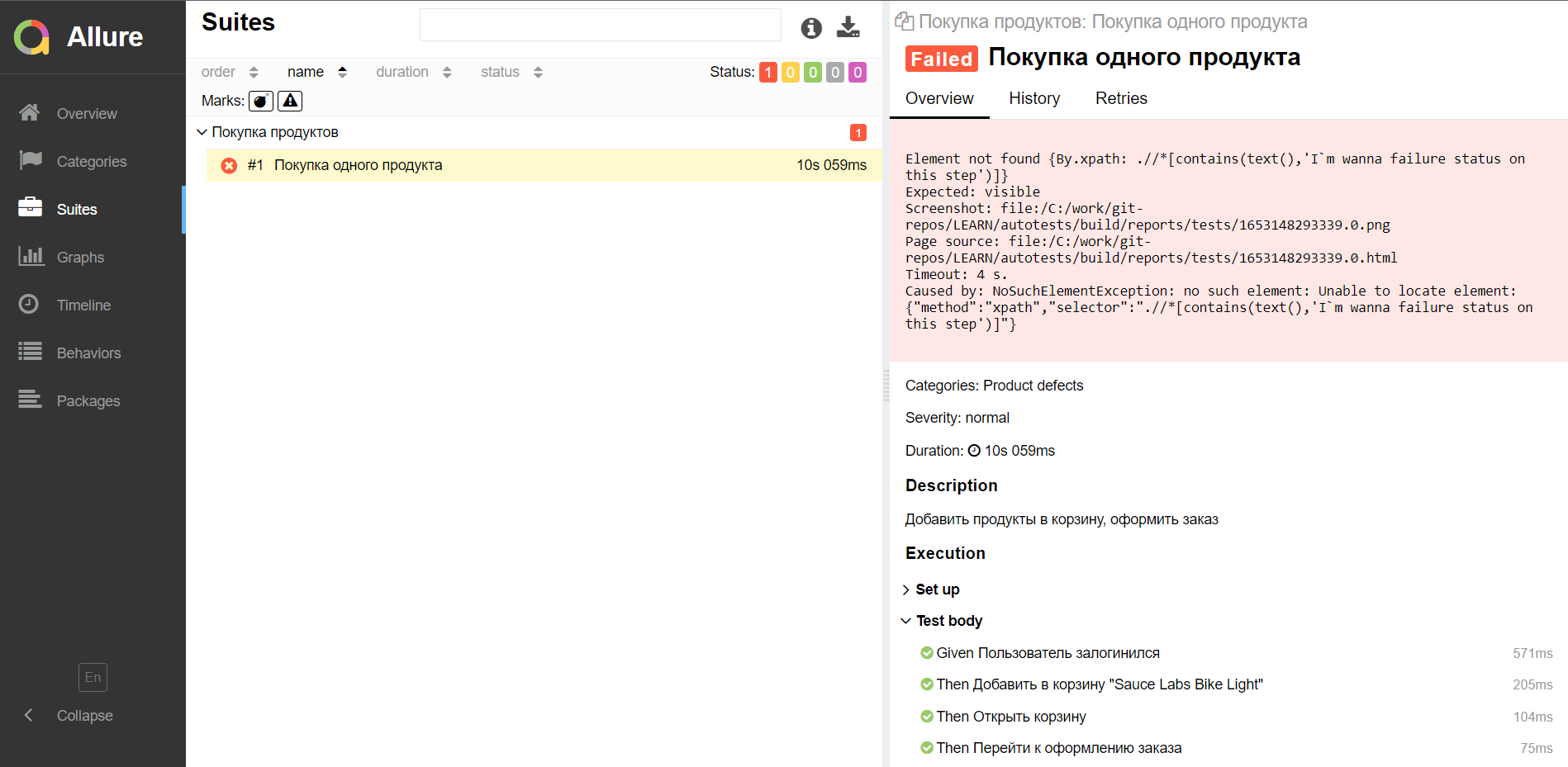


Рис. 22 – Allure2 отчёт об ошибке внутри теста

В шапке отчёта представлены технические ошибки приложения, сам тест перешёл в статус провален, и перекрасился в красный цвет для наглядности.

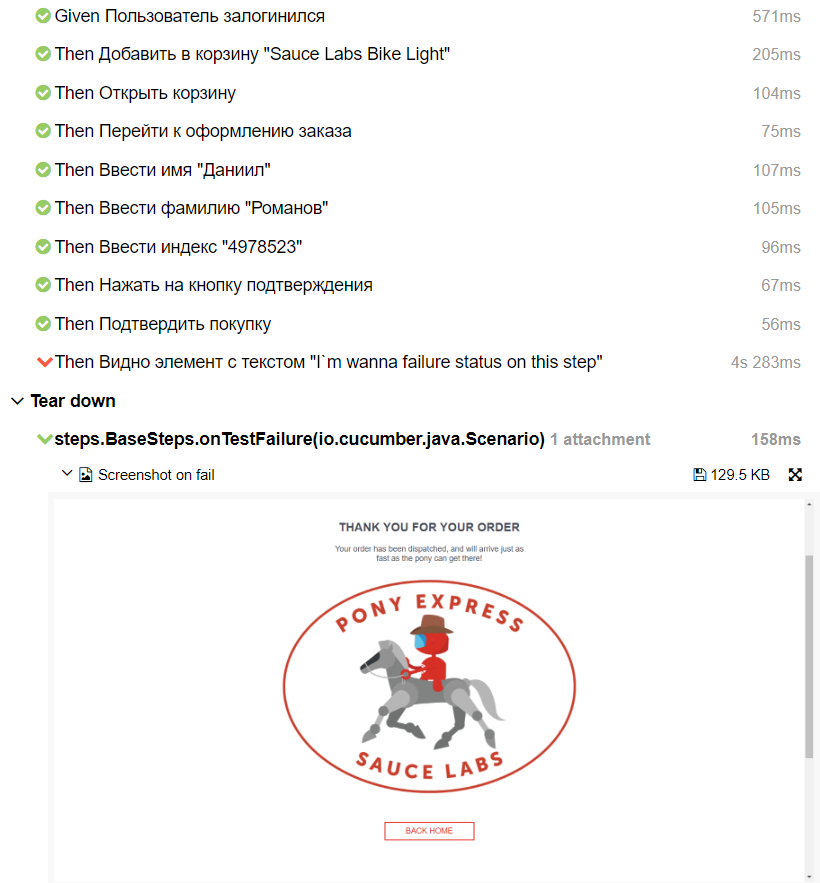


Рис. 23 – Allure2 скриншот шага с ошибкой

Внизу отчёта можно увидеть какой конкретно шаг был провален и прикреплённый к нему снимок экрана.

По отчёту ясно, что ожидался текст «I`m wanna failure status on this step», а на самом деле был изображен текст «THANK YOU FOR YOUR ORDER».

Таким образом, целевое решение для автоматизации заключается в том, что теперь писать новые автоматизированные тесты становится намного проще и быстрее, что существенно ускоряет их внедрение.

### Заключение

В ходе практики был разработан гибридный фреймворк для написания автоматизированных тестов по заданной архитектуре, были разработаны основные модули, написаны сценарии для проведения тестирования, подготовлен и продемонстрирован контрольный пример. Добавлены наглядные и современные отчёты об ошибках, обеспечен запуск автоматизированных тестов в многопоточном режиме.

**Источники информации**

1. Основы тестирования программного обеспечения. Учебное пособие для СПО, 2-е изд., стер.
2. Software Testing Techniques, 2nd Edition, Борис Бейзер
3. Scrum: Революционный метод управления проектами, Джефф Сазерленд
4. Experiences of Test Automation: Case Studies of Software Test Automation 1st Edition, Dorothy Graham