**Содержание**

[1. Утилита “TestAutomationFrameworkCreate” (“TAFC”) 2](#_Toc41435302)

[1.1 Описание программы “TAFC” 2](#_Toc41435303)

[1.2 Структура программы “TAFC” 4](#_Toc41435304)

[2. Разработка автотестов 5](#_Toc41435305)

[2.1 UI тесты 5](#_Toc41435306)

[2.2 API тесты 6](#_Toc41435307)

[2.3 Тесты общего назначения 6](#_Toc41435308)

[2.4 Логирование 6](#_Toc41435309)

[2.5 Описание автотеста с использованием reporter 7](#_Toc41435310)

[3. Создание item в Jenkins. 7](#_Toc41435311)

[4. Allure отчеты 11](#_Toc41435312)

[5. Ссылки на документацию 12](#_Toc41435313)

# 1. Утилита “TestAutomationFrameworkCreate” (“TAFC”)

Создание проекта происходит с помощью утилиты “TestautomationFramework”.   
Данная утилита представляет собой фреймворк с графическим интерфейсом для разворачивания структуры директорий UI, API автотестов. C использованием связок библиотек: для UI тестов – Puppeteer + Jest , для API тестов - Axios + Cheerio.

## 1.1 Описание программы “TAFC”

Данная программа, после предварительной конфигурации, создает необходимую директорию “./имя\_проекта/test”, с поддиректориями “api”, “e2e”, “common\_tests”, в зависимости от выбранного варианта.

Также в директории “./имя\_проекта” создается стандартный npm проект (команда “npm init”) c файлом package.json, директорией “node\_modules” и конфигурационными файлами “.babelrc”, ”.gitignore”, описанные в файлe templates.py. В зависимости от конфигурации пользователем, в проект устанавливаются необходимые для разработки node модули, хранящиеся в файле npm\_libraries.py.

Для использования утилиты необходимо запустить bash-терминал в директории тестового фреймворка “./TestAutomationFramework” и выполнить команду “python main.pyw” , в центре экрана появиться окно программы, показанное на рисунке 1.1.

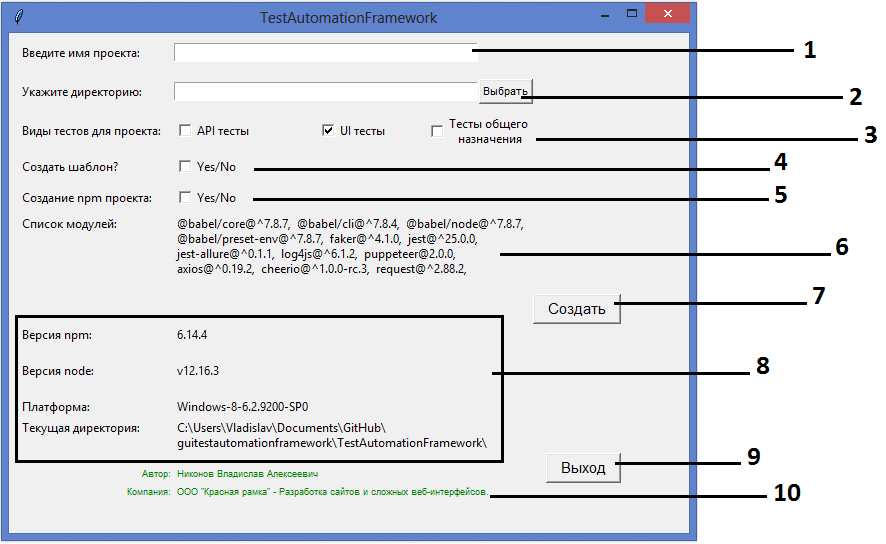


Рисунок 1.1 – Окно утилиты “TestAutomationFrameworkCreate”

Как показано на рисунке 1.1, программа имеет следующие элементы интерфейса:

1. Поле ввода названия проекта. Введенное значение используется в названии создаваемой под проект директории и, как правило, должно соответствовать названию тестируемого проекта.

2. Кнопка выбора директории «Выбрать». При нажатии кнопки, появляется системное окно выбора директории, как показано на рисунке 1.2, в которой будет создана папка под проект.

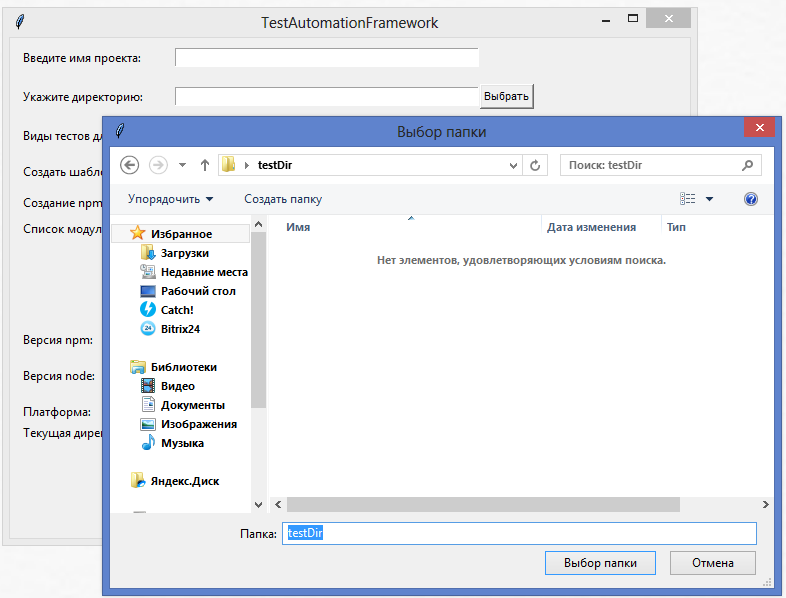


Рисунок 1.2 – Окно выбора директории после нажатия «Выбрать»

3. Чекбоксы для выбора видов тестов. Позволяют выбрать вид тестов, которые, предположительно, будут разрабатываться в проекте. На выбор предоставляются три вида: «api», «ui», «общего назначения». Будут созданы соответствующие выбору подкаталоги и модули.

4. Чекбокс выбора создания шаблонов.

При выборе данного чекбокса, в каталогах под разные виды автотестов, будут созданы шаблоны автотестов. Шаблон содержит необходимую структуру тестов, соответствующую Jest фреймворку, примеры правильного описания автотеста с помощью глобальной переменной «reporter», а также необходимые пояснения в виде комментариев, как показано на рисунке 1.3.

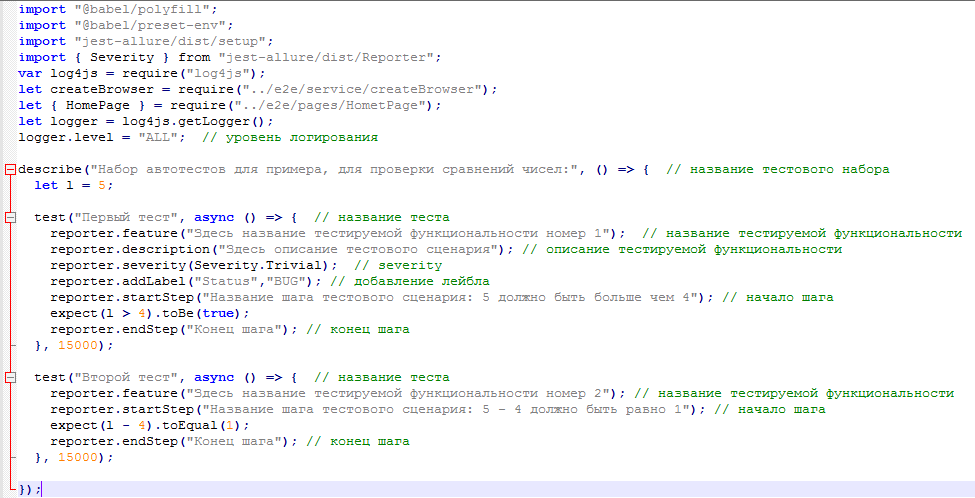


Рисунок 1.3 – Пример шаблона автотеста

5. Чекбокс выбора создания npm проекта.

При выборе данного чекбокса, в директории проекта создается файл package.json, который содержит описание проекта по дефолту (соответствует команде «npm init --yes»). Также, в соответствии с выбором вида тестов, устанавливаются необходимые node-модули (соответствует команде «npm install названия\_библиотек»), которые хранятся в файле npm\_libraries.py.

6. Список стандартных модулей.

В данном блоке отображаются все node.js библиотеки, которые используются в проекте, как стандартные. Данные список храниться в файле npm\_libraries.py.

7. Кнопка «Создать».

По нажатию данной кнопки, происходит создание проекта в соответствии с выбранной заранее конфигурацией.

8. Информация о системе.

В данном блоке отображается информация о версиях node.js и npm, установленных на текущей платформе. В случае проблем с вышесказанными программами, в данном блоке будет отображена ошибка, что позволит разработчику устранить проблемы до работы, непосредственно, с функционалом утилиты “TAFC”. Также, для удобства, отображена версия текущей платформы и текущая директория, в которой запущена утилита.

9. Кнопка «Выход». С помощью данной кнопки происходит выход из программы.

10. Информация о разработчике.

## **1.2 Структура программы “TAFC”**

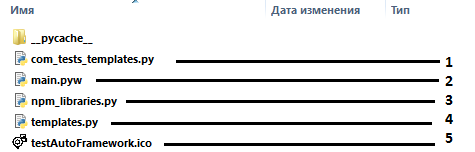


Рисунок 1.4 – Структура программы “TAFC”

Утилита “TestAutomationFrameworkCreate” состоит из нескольких модулей, показанных на рисунке 1.4:

1. Модуль “com\_test\_templates.py” содержит в себе шаблоны тестов общего назначения. Данные тесты разрабатываются независимыми от проекта и проверяют общую для большинства проектов функциональность.

2. Модуль “main.py” содержит основной код программы: функции создания необходимых директорий и графический интерфейс.

3. Модуль “npm\_libraries.py” содержит списки node.js библиотек.

4. Модуль “templates.py” содержит шаблоны конфигурационных файлов, а также шаблоны структур автотестов с описанием и подсказками.

5. Иконка программы.

# 2. Разработка автотестов

Неопытным разработчикам следует выбрать чекбокс «Создать шаблон?», как показано на рисунке 2.1. Далее, при создании директорий видов тестов, будут созданы файлы примеров автотестов, со структурой , описанием и подсказками в виде комментариев, как показано на рисунке 1.3. Это позволит новичку быстрее разобраться и понять условия разработки автотестов.

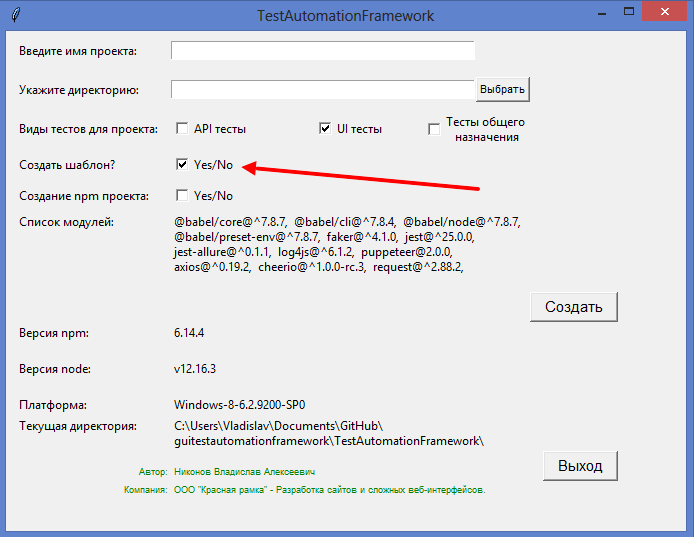


Рисунок 2.1 – Выбор создания шаблонов автотестов

## 2.1 UI тесты

При разработке UI тестов используется инструмент Puppeteer. Создаётся директория «е2е» и поддиректории «pages» и «service», как показано на рисунке 2.2.

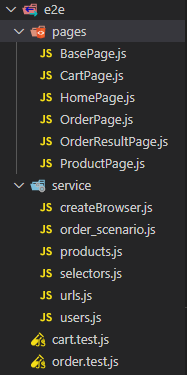


Рисунок 2.2 – Структура директории «е2е»

В директории «service» содержаться: модуль «urls.js» - данный модуль содержит ссылки применимые для определенного проекта, «selectors.js» - данный модуль содержит список селекторов, разделенных на логические блоки, для взаимодействия с элементами страницы, а также модуль «createBrowser.js» - модуль, содержащий код создания экземпляра браузера в Puppeteer.

В директории «pages» содержаться модули содержащие классы, логически разбитые по функционалу на веб-страницы, где данный функционал применим. Это объясняется применением при разработке паттерна PageObject.

Сами тесты находятся в самой директории «е2е» и взаимодействуют с дополнительными модулями.

## 2.2 API тесты

При разработке API тестов используются библиотеки для http/https запросов, например Axios и библиотеки парсинга веб-страниц, к примеру Cheerio. Создаётся директория «api» и поддиректория «service», как показано на рисунке 2.3.

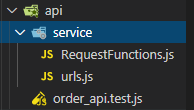


Рисунок 2.3 – Структура директории «api»

В директории «service» содержаться: модуль «urls.js» - данный модуль содержит ссылки применимые для определенного проекта, и дополнительные модули, которые содержат классы с функционалом для соответствующих API тестов.

## 2.3 Тесты общего назначения

При разработке тестов общего назначения могут использоваться различные библиотеки, но обычно используются библиотеки для http/https запросов ( Axios ) и библиотеки парсинга веб-страниц ( Cheerio ). Создаётся директория «common\_tests» и поддиректория «service», как показано на рисунке 2.4.

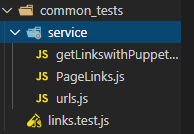


Рисунок 2.4 – Структура директории «common\_tests»

В директории «service» содержаться: модуль «urls.js» - данный модуль содержит ссылки применимые для определенного проекта, и дополнительные модули, которые содержат классы с функционалом для соответствующих тестов.

## 2.4 Логирование

Логирование применяется для вывода в консоль информации о состоянии выполнения процессов в автотесте. В фреймворке используется библиотека «log4js».  
Ссылка на документацию: <https://www.npmjs.com/package/log4js>

## 2.5 Описание автотеста с использованием reporter

Глобальная переменная reporter позволяет описывать автотест, что делает его более читаемым при составлении отчетов в Allure. Пример использования показан на рисунке 3.  
Ссылки на документацию:

1. <https://jestjs.io/docs/en/configuration.html#reporters-arraymodulename--modulename-options>

2. <https://devцeloper.aliyun.com/mirror/npm/package/jest-puppeteer-allure>

# 3. Создание item в Jenkins

Необходимо перейти на страницу создания сборки, где будет предложено ввести имя сборки и выбрать вариант конфигурации. Для текущих проектов выбрана конфигурация «Создать задачу со свободной конфигурацией», как показано на рисунке 3.1.

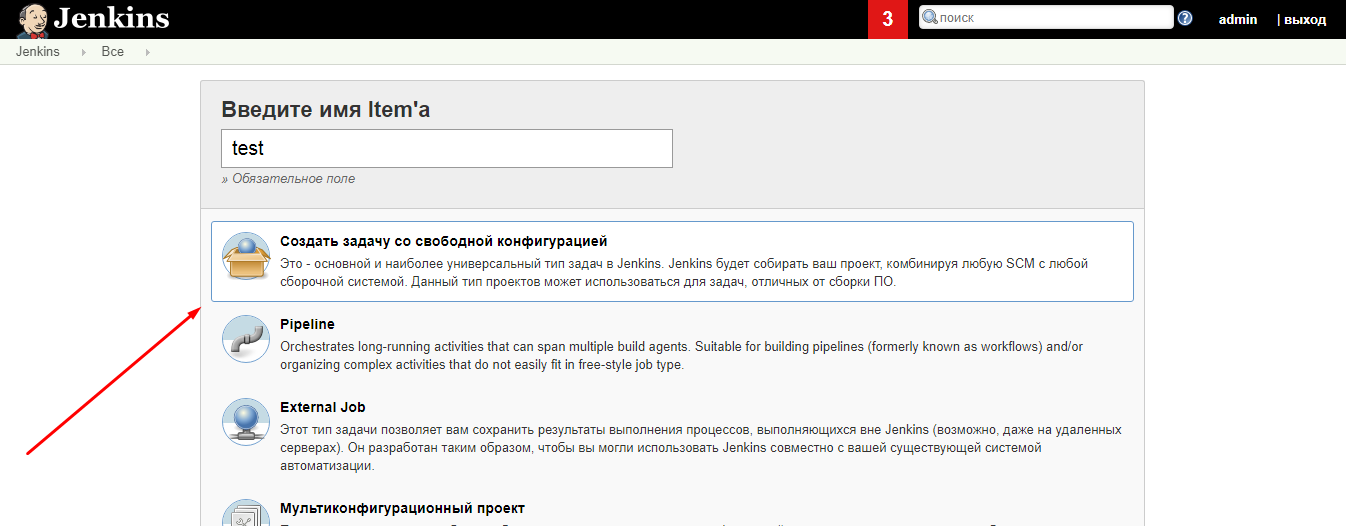


Рисунок 3.1 – Меню создания сборки в Jenkins

Сама конфигурация сборки делится на несколько частей: «Общая конфигурация», «Управление исходным кодом», «Триггеры сборки», «Среда сборки», «Сборка», «Послесборочные операции», как показано на рисунке 3.2.

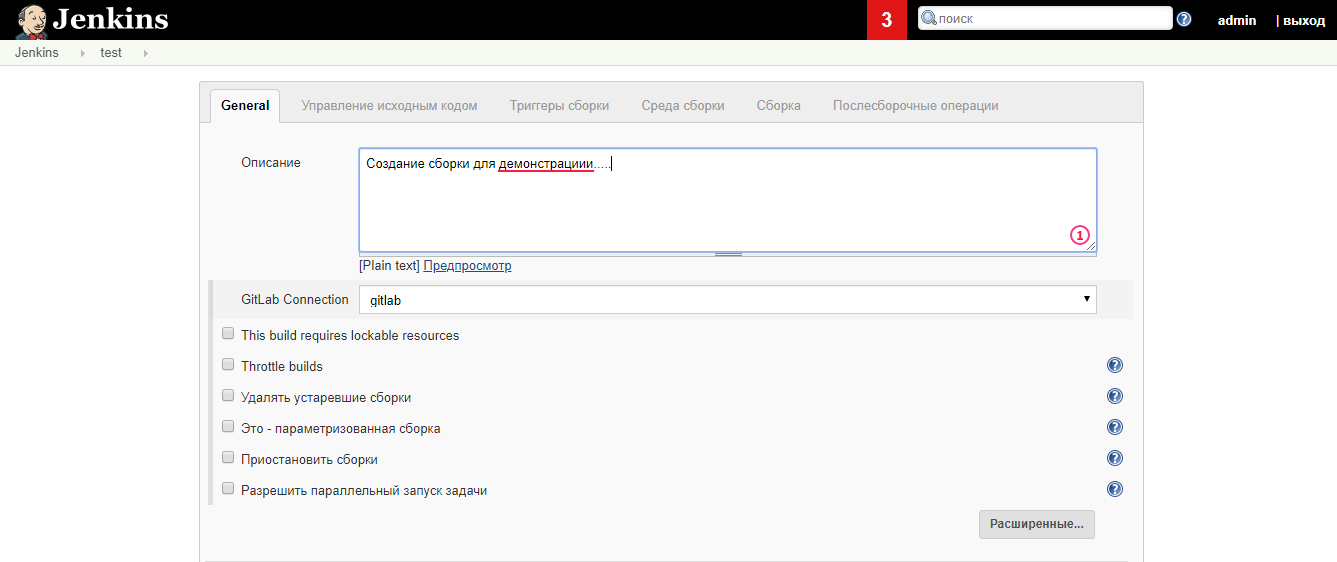


Рисунок 3.2 – Форма общей конфигурации сборки

В «Общей конфигурации» следует указать описание сборки, а также настроить удаление устаревших сборок, чтобы не копить их и не занимать место на жестком диске сервера.

Далее следует «Управление исходным кодом». В данной вкладке указывается название удаленного репозитория, а также, если данный репозиторий приватный, следует указать данные для доступа в поле «Credentials». Как правило, проект имеет несколько веток в репозитории, необходимо указать, какая именно ветка будет стянута и впоследствии протестирована, данную информацию указывают в поле «Branch Specifier», как показано на рисунке 3.3.

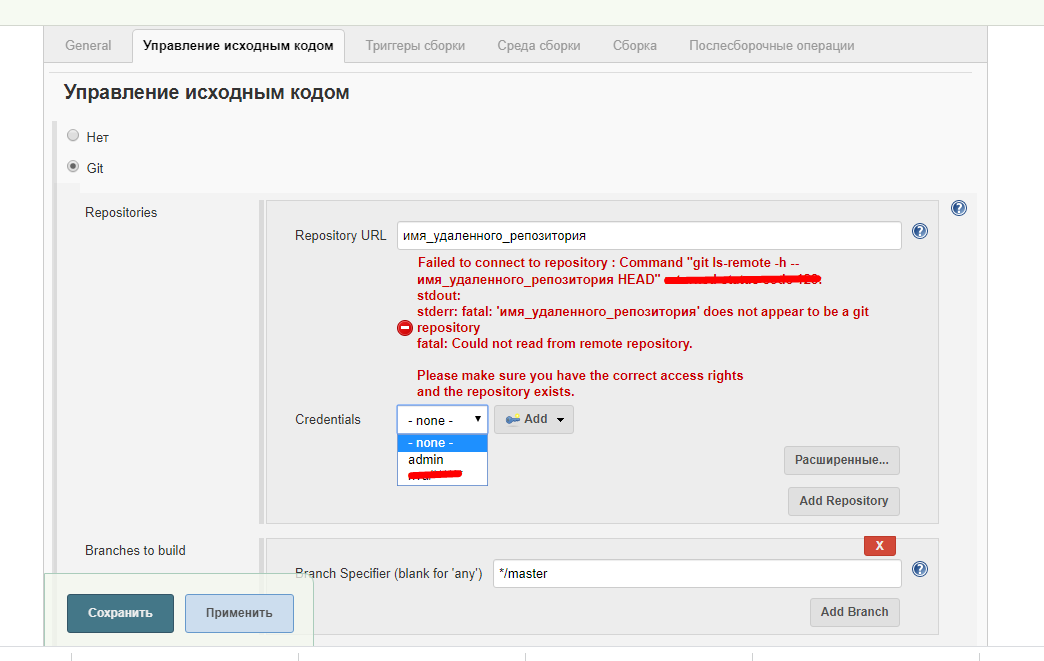


Рисунок 3.3 – Форма конфигурации управления исходным кодом

Далее следует вкладка «Триггеры сборки». Как было сказано выше сборка будет начинаться при изменении проекта в удаленных репозиториях Gitlab. Для обеспечения связи используется технология «Webhooks». Как показано на рисунке 3.4, следует скопировать предложенный Jenkins webhook.

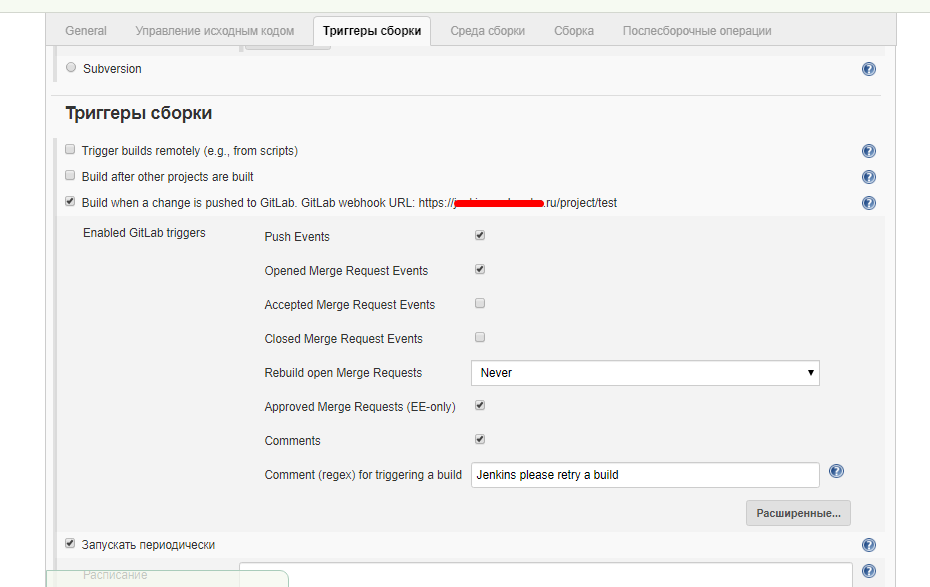


Рисунок 3.4 – Форма конфигурации триггеров сборки

Далее уже в самом сервисе Gitlab, на странице «Integrations», создается webhook в поле «url» которого вставляется, скопированное из Jenkins значение, как показано на рисунке 3.5. Также следует выбрать событие, которое будет выступать в качестве триггера для Jenkins. Для данного проекта выбран «Push events», это значит, что сборка в Jenkins запустится, как только разработчик выполнит команду push для того, чтобы сохранить сделанные им изменения в репозиторий проекта.

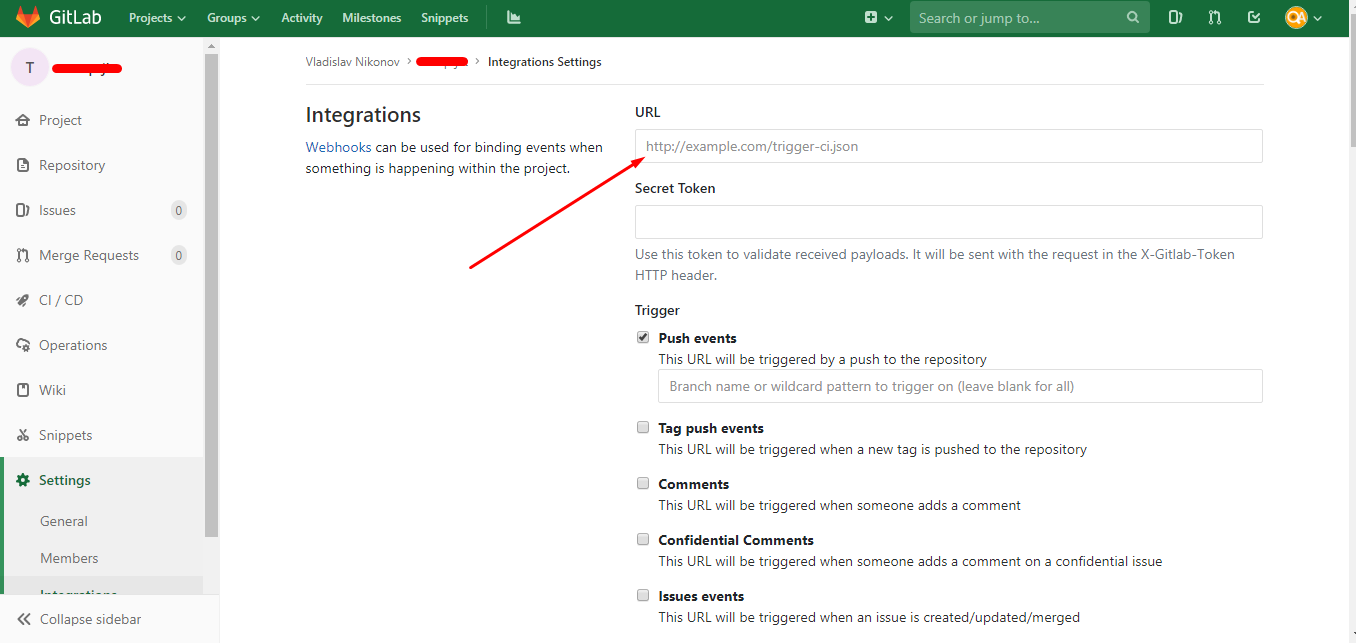


Рисунок 3.5 – Страница создания webhook в сервисе Gitlab

Следующий шаг это «Среда сборки». Так как был установлен плагин «NodeJS Plugin», в списке доступных сред для сборки появился пункт «Provide Node & npm bin/folder to PATH». Далее выбирается подходящая под текущий проект версия node, как показано на рисунке 3.6.

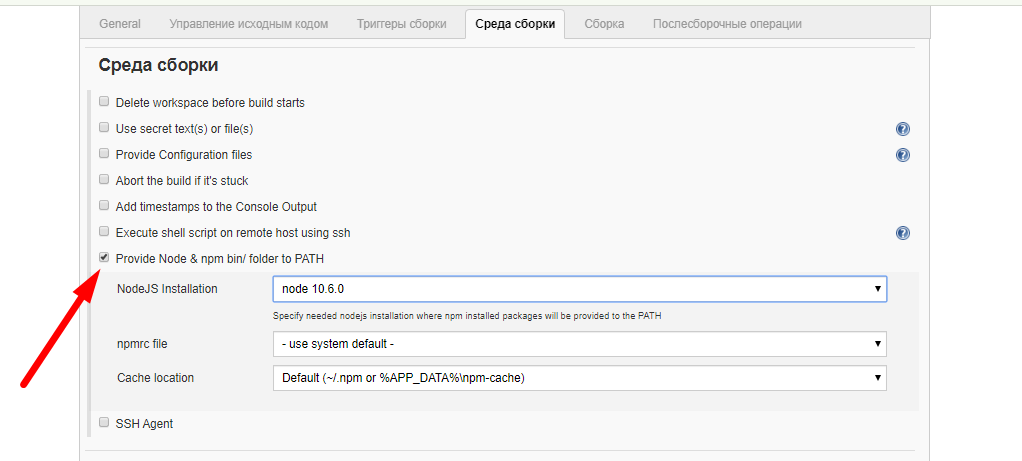


Рисунок 3.6 – Форма конфигурации среды сборки

На этапе «Сборка» прописываются команды, необходимые для работы уже с загруженным с репозитория проектом. Так как в проекте используется пакетный менеджер npm, то для работы с ним необходим терминал shell, и из предложенного списка выбираем именно его. В поле ввода прописываем необходимые команды для загрузки необходимых библиотек, headless браузера и запуска автотестов, как показано на рисунке 3.7.

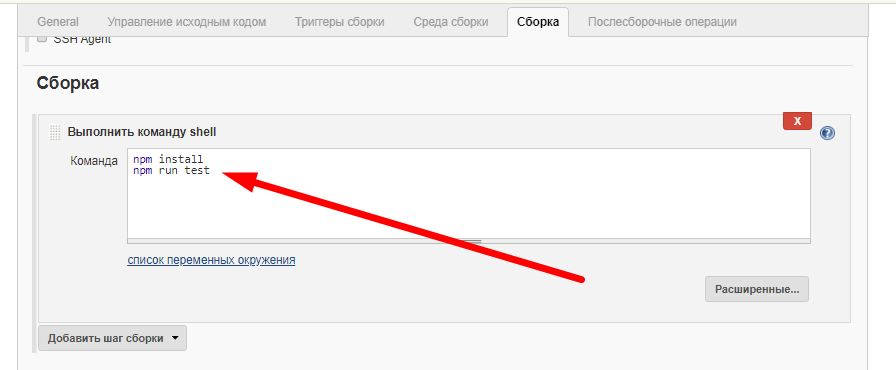


Рисунок 3.7 – Командная строка для сборочных операций проекта

И последним этапом является «Послесборочные операции», где есть возможность выбора различных событий, которые будут происходить если сборка пройдет неудачно. Планируется использование рассылки электронных писем специалистам по качеству, в случае если сборка завершиться с ошибкой. Также можно создавать очередь сборок, где запуск каждой последующих сборок, зависит от предыдущей. Данные функции показаны на рисунке 3.8.

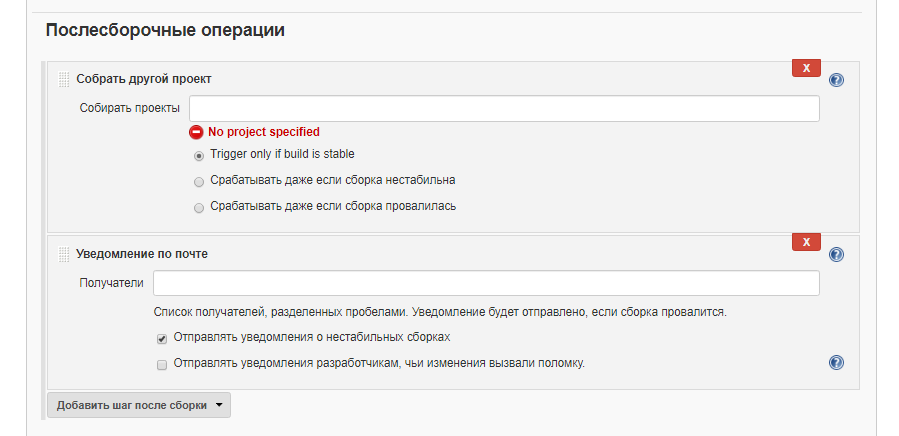


Рисунок 3.8 – Форма настройки послесборочных операций

# 4. Allure отчеты

Для использования Allure в Jenkins необходим плагин «Allure Jenkins plugin». Он не требует глобальных конфигураций, необходимые опции появляются при создании сборки на этапе «Послесборочные операции», как показано на рисунке 4.1.

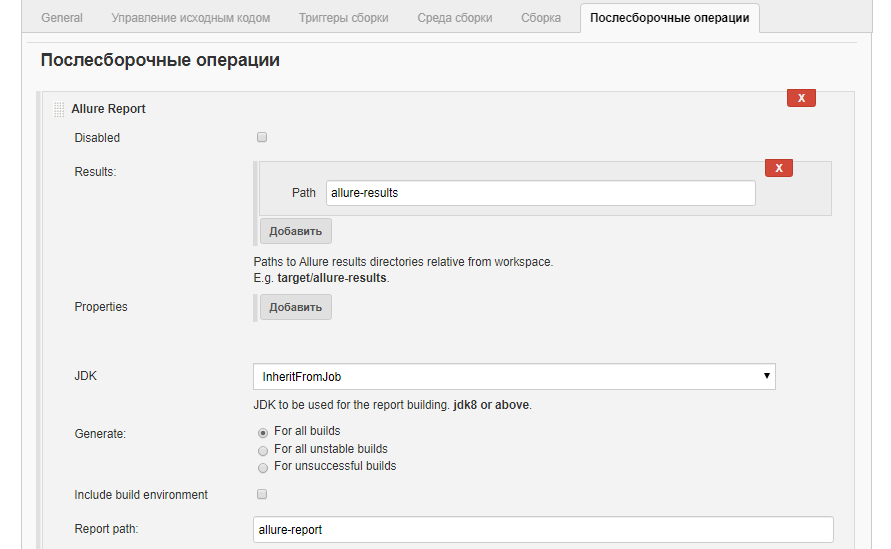


Рисунок 4.1 – Настройка Allure во время конфигурации сборки

Для конфигурации создания отчета требуется указать директорию куда будут создаваться сами отчеты о пройденных автотестах, а также, директорию, где будет сохранятся история отчетов. Пример отчета показан на рисунке 4.2.

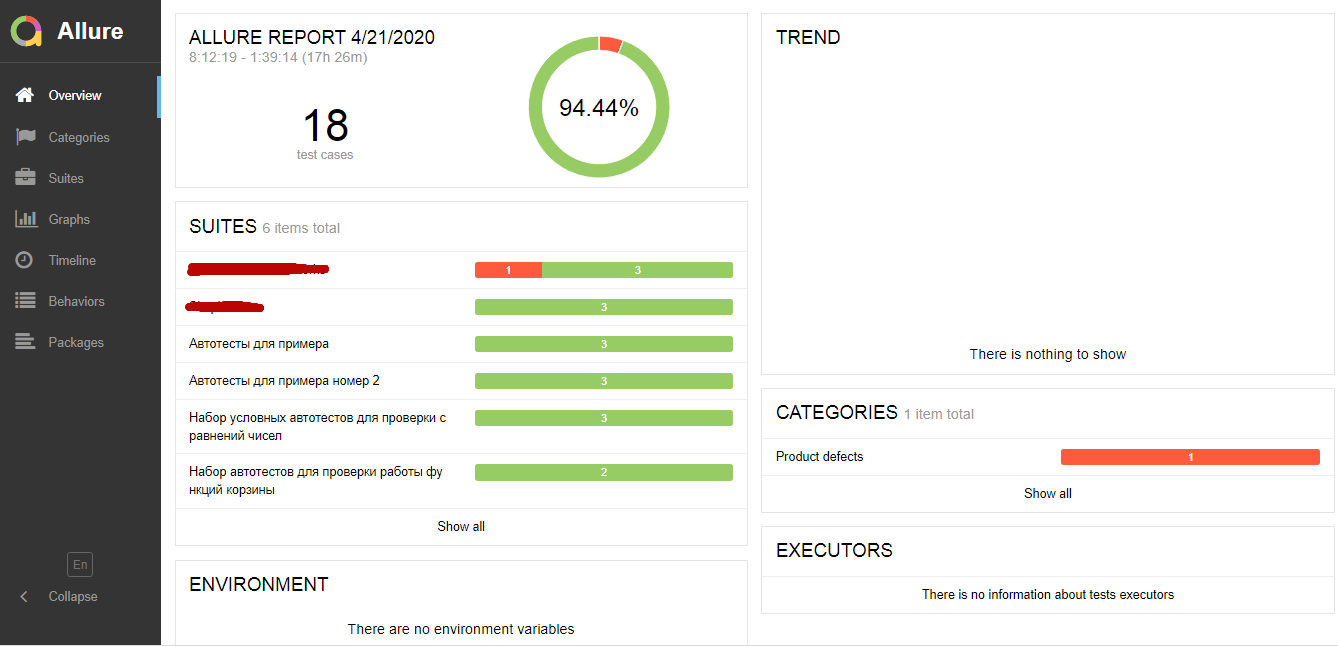


Рисунок 4.2 – Пример сгенерированного отчета Allure

# 5. Ссылки на документацию

Puppeteer:

1. <https://github.com/puppeteer/puppeteer/blob/master/docs/api.md>

Jest:

1. <https://developer.aliyun.com/mirror/npm/package/jest-puppeteer-allure>

2. <https://jestjs.io/docs/en/api>

3. <https://jestjs.io/docs/en/puppeteer>

Log4js:

1. <https://www.npmjs.com/package/log4js>

Axios:

1. <https://www.npmjs.com/package/axios>

Cheerio:

1. <https://github.com/cheeriojs/cheerio>