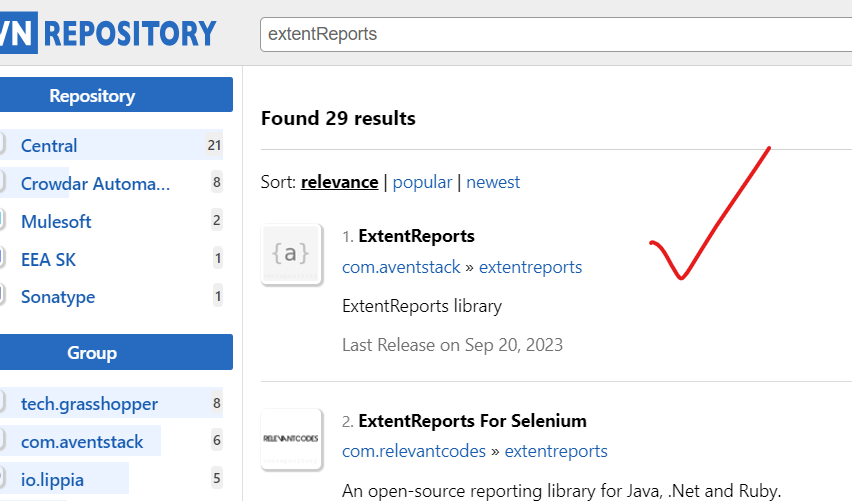
Extent Report - не принадлежит ни TestNG ни Selenium. Нам нужны отдельно разработанные кем то классы для этого репорта.

1- добавить зависимость

2- создать тест кейс (класс)

3)- создать реализацию ITesTListener

Для него нужна зависимость



мой класс - это набор разных мной предопределённых сценариев, методов (pass, failed, skipped).

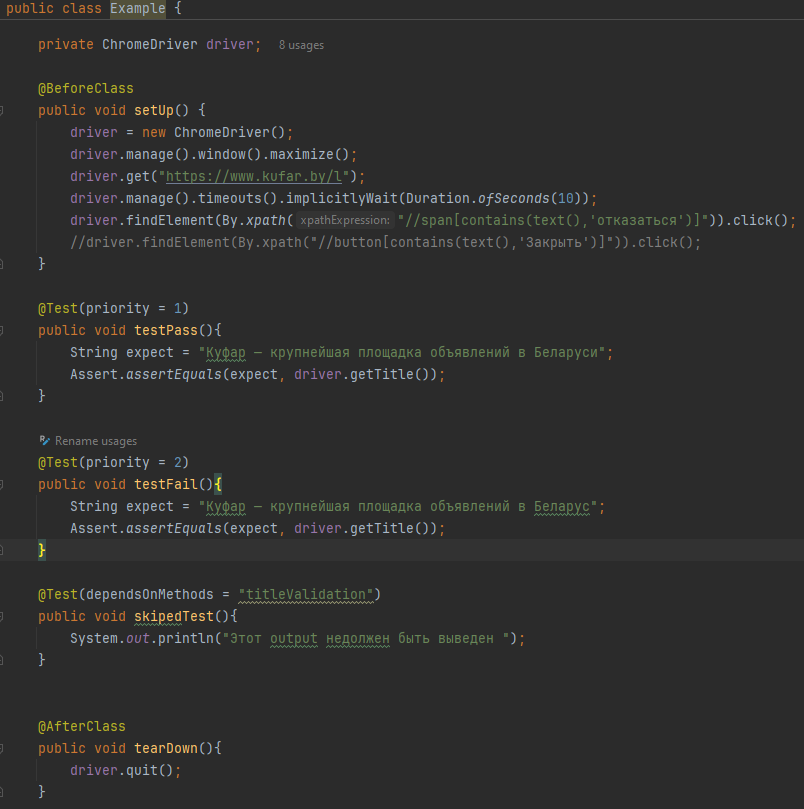
setUp - сетит драйвер + переходит по урл

testPass - должен пройти

testFail- чёт- там проверяет и должен упасть

skippedTest- должен пропустится так как dependent на testFail

tearDown-закрывает драйвер.



Теперь класс который будет **реализовывать** принцип Test Listeners но логически составлен для генерации репорта. Я его не писал. В нём много сторонних классов с помощью которых можно детально настраивать репорт.

Про принцип тест листенеров читать здесь - [TestNG Listeners](https://docs.google.com/document/u/0/d/18Ldo-YNhcLz5bVs7VniounrUvNfThz2S4MPdHUAxrPk/edit)

public ExtentSparkReporter sparkReporter; // UI of the report

public ExtentReports extent; //populate common info on the report

public ExtentTest test; // creating test case entries in the report and update status of the test methods

**public void onStart(ITestContext context) {**

sparkReporter**=new ExtentSparkReporter(“**location of the report”**)**; //specify location of the report

sparkReporter.config().setDocumentTitle("Automation Report"); // TiTle of report

sparkReporter.config().setReportName("Functional Testing"); // name of the report

sparkReporter.config().setTheme(Theme.STANDARD);

extent=new ExtentReports();

extent.attachReporter(sparkReporter);

extent.setSystemInfo("Computer Name","localhost");

extent.setSystemInfo("Environment","QA");

extent.setSystemInfo("Tester Name","Pavan");

extent.setSystemInfo("os","Windows10");

extent.setSystemInfo("Browser name","Chrome,Firefox,Edge");

}

**public void onTestSuccess(ITestResult result) {**

**test** = extent.createTest(result.getName()); // create a new entry in the report

**test**.log(Status.PASS, "Test case PASSED is:" + result.getName()); // update status p/f/s

}

**public void onTestFailure(ITestResult result) {**

**test** = extent.createTest(result.getName()); // create a new entry in the report

**test**.log(Status.FAIL, "Test case FAILED is:" + result.getName());

**test**.log(Status.FAIL, "Test Case FAILED cause is: " + result.getThrowable());

}

**public void onTestSkipped(ITestResult result) {**

**test** = extent.createTest(result.getName());

**test**.log(Status.SKIP, "Test case SKIPPED is:" + result.getName());

}

**public void onFinish(ITestContext context) {**

extent.flush();

}

Технически этот класс является тест листенерем а по своей логике он генерирует репорт. Т.е это как одно из применений тест листенер концепта. Так как мы собираемся генерить рипорт то и методы мы имплементируем соответственно.

**onStart** - технически этот метод начнёт выполнятся только один раз как только начнётся выполнение тест-методов. Внутри метода используя детализированные методы сторонних классов мы создаём UI репорта. Это логично, а чего ждать.

**onTestSuccess** - создаётся новая запись в репорте, а в ней устанавливается passed.

**onTestFailure** - создаётся новая запись в репорте, а в ней устанавливается failed.

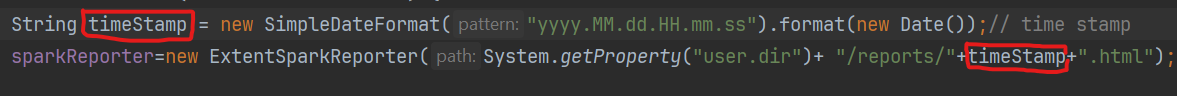
**onTestSkipped** - ну ты понял

**onFinish** - ну ты понял

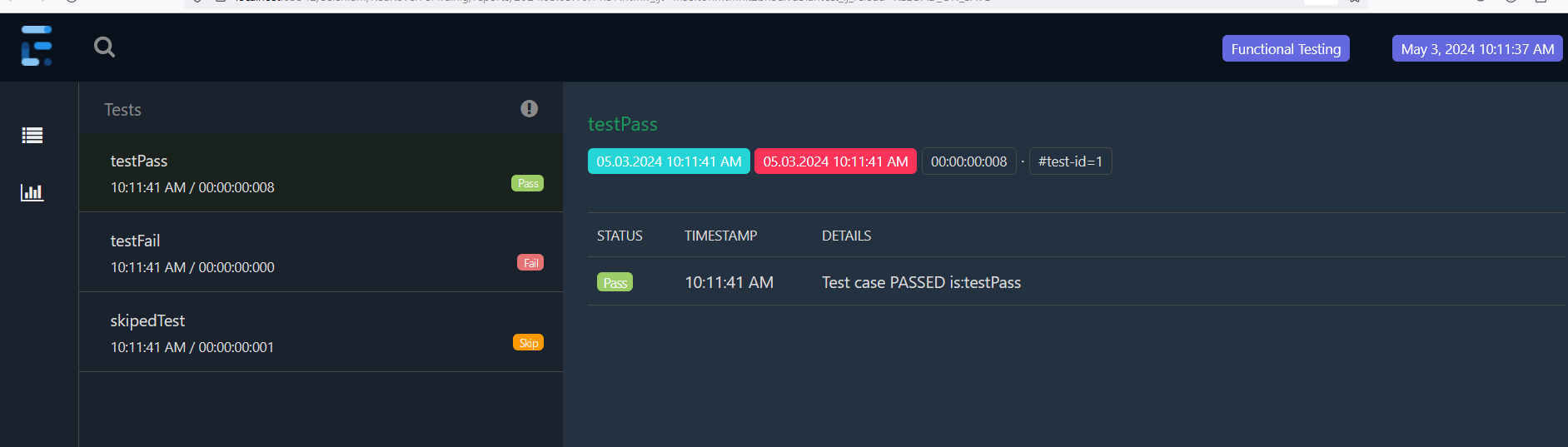
Используя этот класс как генератор репорта мы не можем обеспечить историю репортов. Каждый раз репорт будет создаваться новый, на месте старого, (переписыватся) это из за имени которое мы предоставляем в коде. Решением этого может быть привязка времени от системы, в этом случае мы получаем разное имя и соответственно старый(предыдущий) репорт не будет переписан. Как это сделать технически пока не знаю.

Теперь знаю:

Надо зделать Стрингу которая бы менялась каждую секунду, как вариант можно создать время и дату и залить в стринг, а потом зделать эту стринг частью имени репорта. Смотри скрин ниже



Сам репорт будет выглядеть вот так



С лева названия которые генерятся после прохождения тест методов

**ОКАЗЫВАЕТСЯ** можно привязать тест кейс(класс) и листенер класс и без xml. Для этого в тест кейсе (класс отвечающий за тест-методы, **не** за реализацию листенера) над сигнатурой класса указывается

**@Listeners(расположение листенер класса)** аннотация с локейшенам где расположен тест листенер класс.



Но у этого подхода есть минус, если у тебя только один тест кейс(класс) то ок, но если классов несколько то предётся писать эту херню в каждом классе.

Для создания xml файла - [TestNG Listeners](https://docs.google.com/document/u/0/d/18Ldo-YNhcLz5bVs7VniounrUvNfThz2S4MPdHUAxrPk/edit)

Ещё один пример Репорт Менеджера класса который отвечает за генерацию репортов.

public class ExtentReportManager implements ITestListener {

public ExtentSparkReporter sparkReporter;

public ExtentReports extent;

public ExtentTest test;

String repName;

**public void onStart(ITestContext testContext) {**

1 String **timeStamp** = new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss").format(new Date());// time stamp

2 repName = "Test-Report-" + **timeStamp** + ".html";

3 sparkReporter = new ExtentSparkReporter("./reports/" + repName);// specify location of the report

4 sparkReporter.config().setDocumentTitle("opencart Automation Report"); // Title of report

5 sparkReporter.config().setReportName("opencart Functional Testing"); // name of the report

6 sparkReporter.config().setTheme(Theme.DARK);

7 extent = new ExtentReports();

8 extent.attachReporter(sparkReporter);

9 extent.setSystemInfo("Application", "opencart");

10 extent.setSystemInfo("Module", "Admin");

11 extent.setSystemInfo("Sub Module", "Customers");

12 extent.setSystemInfo("User Name", System.getProperty("user.name"));

13 extent.setSystemInfo("Environemnt", "QA");

14 String os = testContext.getCurrentXmlTest().getParameter("os");

15 extent.setSystemInfo("Operating System", os);

16 String browser = testContext.getCurrentXmlTest().getParameter("browser");

17 extent.setSystemInfo("Browser", browser);

18 List<String> includedGroups = testContext.getCurrentXmlTest().getIncludedGroups();

19 if(!includedGroups.isEmpty()) {

20 extent.setSystemInfo("Groups", includedGroups.toString());

}

}

**public void onTestSuccess(ITestResult result) {**

1 test = extent.createTest(result.getTestClass().getName());

2 test.assignCategory(result.getMethod().getGroups()); // to display groups in report

3 test.log(Status.PASS,result.getName()+" got successfully executed");

}

**public void onTestFailure(ITestResult result) {**

1 test = extent.createTest(result.getTestClass().getName());

2 test.assignCategory(result.getMethod().getGroups());

3 test.log(Status.FAIL,result.getName()+" got failed");

4 test.log(Status.INFO, result.getThrowable().getMessage());

try {

5 String imgPath = new BaseClass().captureScreen(result.getName());

6 test.addScreenCaptureFromPath(imgPath);

} catch (IOException e1) {

e1.printStackTrace();

}

}

**public void onTestSkipped(ITestResult result) {**

test = extent.createTest(result.getTestClass().getName());

test.assignCategory(result.getMethod().getGroups());

test.log(Status.SKIP, result.getName()+" got skipped");

test.log(Status.INFO, result.getThrowable().getMessage());

}

**public void onFinish(ITestContext testContext) {**

extent.flush();

String pathOfExtentReport = System.getProperty("user.dir")+"\\reports\\"+repName;

File extentReport = new File(pathOfExtentReport);

try {

Desktop.getDesktop().browse(extentReport.toURI());

} catch (IOException e) {

e.printStackTrace();

}

/\*

\* try { URL url = new

\* URL("file:///"+System.getProperty("user.dir")+"\\reports\\"+repName);

\*

\* // Create the email message

\* ImageHtmlEmail email = new ImageHtmlEmail();

\* email.setDataSourceResolver(new DataSourceUrlResolver(url));

\* email.setHostName("smtp.googlemail.com");

\* email.setSmtpPort(465);

\* email.setAuthenticator(new DefaultAuthenticator("pavanoltraining@gmail.com","password"));

\* email.setSSLOnConnect(true);

\* email.setFrom("pavanoltraining@gmail.com"); //Sender

\* email.setSubject("Test Results");

\* email.setMsg("Please find Attached Report....");

\* email.addTo("pavankumar.busyqa@gmail.com"); //Receiver

\* email.attach(url, "extent report", "please check report...");

\* email.send(); // send the email

\* }

\* catch(Exception e) { e.printStackTrace(); }

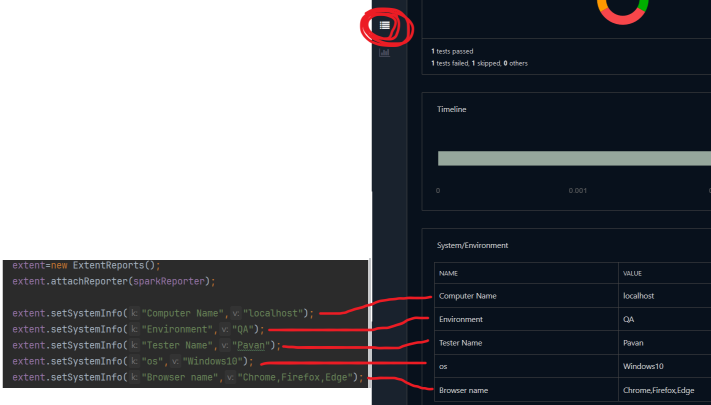
\*/

}

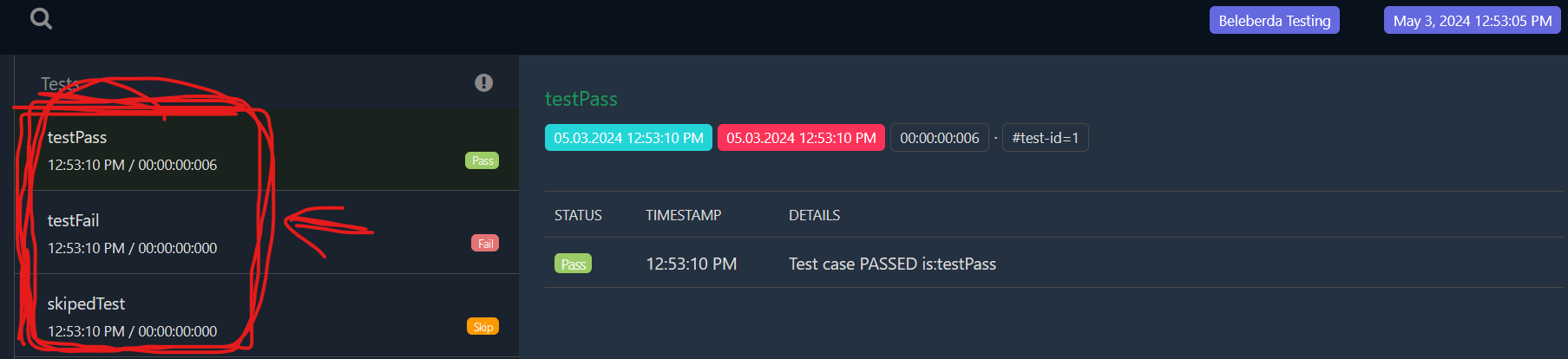
**Что за поля класса?**

**ExtentSparkReporter sparkReporter** - отвечает за общий UI репорта

**ExtentReports extent** - этот объект отвечает за информативную часть



**ExtentTest test** ответственен за создание тестов в левой части UI



Разбор класса по строкам

Метод **onStart**

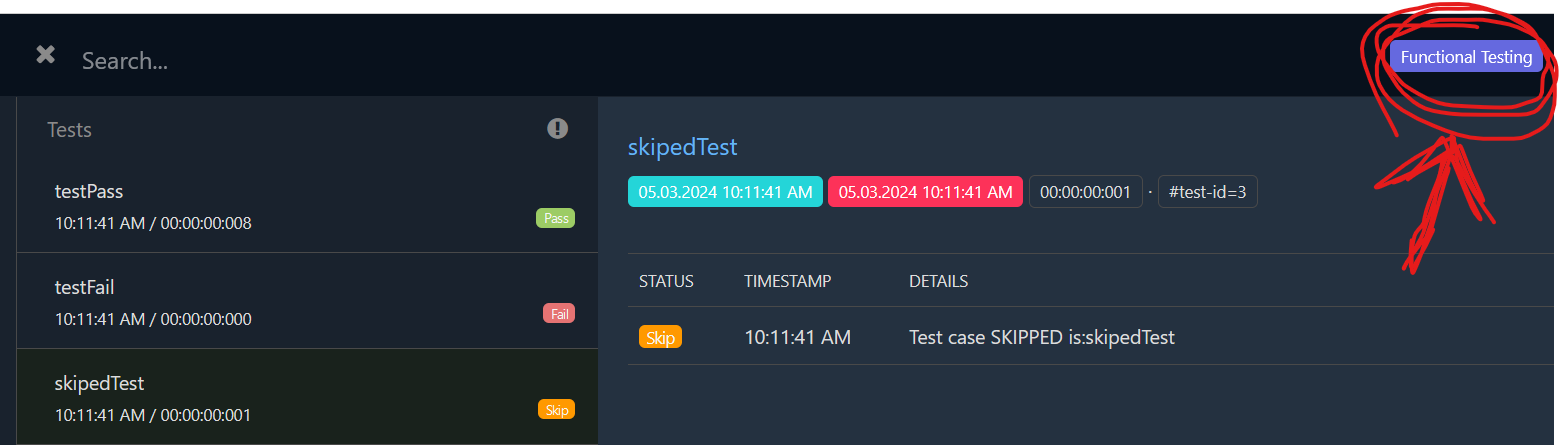
**1** Создаём стринг из текущей даты и времени (с секундами) - для того чтобы стринг получилась уникальной

**2** Создаём имя (Стринг) для репорта с участием уникальной стринг из предыдущего шага - это обеспечит то что имя будет уникальным при каждом запуске (так как секунды будут отличатся) соответственно репорт не будет перезаписываться из-за одного и того же имени, а будет создаваться новый, тем самым у нас будет определённая история репортов.

**3** sparkReporter - это обьект ответственен за UI будущего репорта. его надо инициализировать передав в конструктор место расположения будущего репорта и его имя, которое мы создавали в предыдущих 2-х шагах

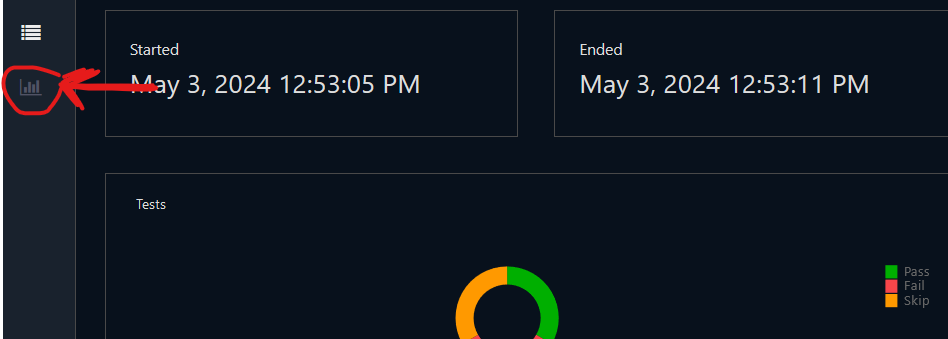
**4** Сетит title репорта, но где этот тайтл в репорте я так и не нашел.(

**5** Сетит название репорта

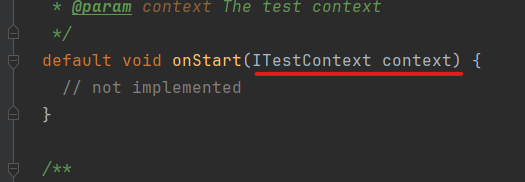


**6** Сетит тему репорта, здесь она тёмная

**7 - 13** информативные настройки отображаются в этой вкладке



**14** С помощью **testContext** который передаётся в параметрах, а откуда его брать то? и зачем он здесь в нашем методе этот **testContext** ?



потомучто (как видно из картинки выше) таким образом интерфейс требует реализовать данный метод, а наш класс реализует ITestListener, но по факту мы сами этот объект (**testContext )**не используем, т.е. не передаём так как этот класс который мы сейчас создали мы нигде не вызываем, все методы которые мы здесь создаём будут вызваны где то самой программой, ведь мы используем тест листенер концепт, и все методы будут тригерится не нами, соответственно и объект **testContext** также где то программой будет создан и передан в наш метод. Через этот объект доступна различная инфа из нашего теста, в том числе и всё что есть в .xml, и мы можем, записать в переменную ту инфу которую хотим, а затем с помощью этой переменной прочитать что-то в репорте, чтобы эта инфа отображалась там. Конкретно здесь Достаём значение параметра ‘os’ из xml и записываем в стрингу

**15** Сетим/создаём/добовляем\_параметр в репорт "Operating System" и его значение, стрингу которую достали с предыдущего шага.

**16** - **17**  тоже самое что предыдущие 2 шага (14-15) только с другим значением

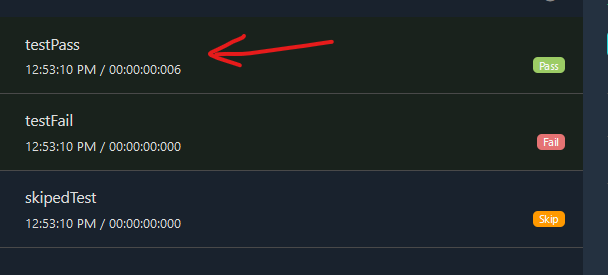
**18** Создаём лист в нём будем хранить перечень всех групп доступных/имеющихся в конфигурационном xml файле

**19 - 20** Если лист не пустой значит у нас действительно есть группы в конфигурационном файле и мы их записываем как параметр “Groups” в наш репорт со значениями - названия групп которые мы достали в 18 шаге.

**Назначение** метода - записать в UI репорта всё то что у нас есть в коде, всю инфу. Тест будет выполнен как только начнётся выполнение фреймворка. onStart по умолчанию всегда будет выполнятся первым, эта его суть в интерфейсе, отсюда и название соответствующее, какую бы логику ты бы туда не вложил, в нашем случае мы генерим репорт (UI +инфа)

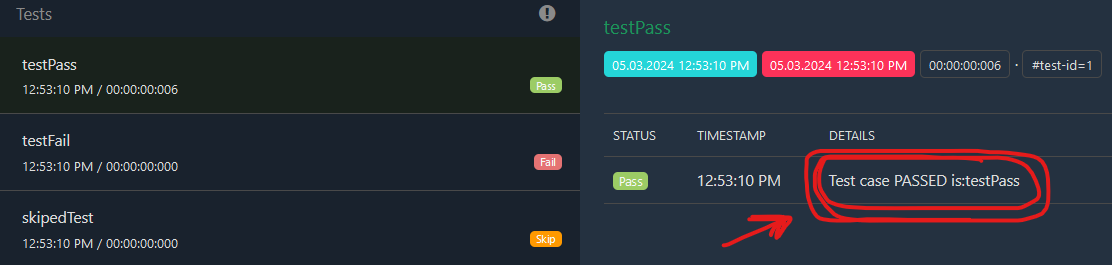
Метод **onTestSuccess**()

1 создаём новую запись, наверно вот такую



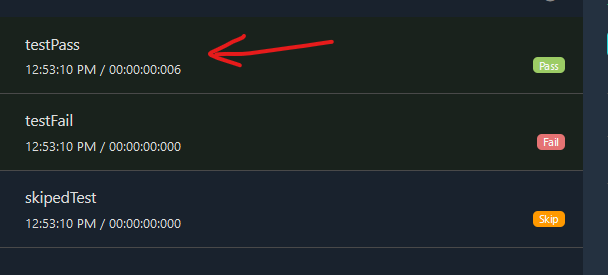
2 отображаем группы в этом методе если они есть

3 Добовляем log к этому тесту, используем константу из класса Status + название лога которое хотим добавить. будет отображон вот здесь



Метод **onTestFailure**

1 создаём новую запись, наверно вот такую



2 отображаем группы в этом методе если они есть

3 Добовляем лог, что метод зафейлился

4 Добовляем лог, как какоето внутреннее (дежурное) сообщение об ошибке

5 Записываем путь в Стринг, метод который вернёт эту стринг расположен в БэйсКлассе и принимает 1 параметр-result.getName(), но, чтобы не создавать обьект класса БэйсКласс пишем

new BaseClass().captureScreen(result.getName())

6 Добавляем скриншот к нашему репорту, с помощью метода test.addScreenCaptureFromPath(imgPath);