Пловдивски университет „Паисий Хилендарски“

Факултет по математика и информатика

Катедра „Компютърна информатика"

Дипломна работа

за получаване на образователно-квалификационна степен „Бакалавър“

Тема: „Разработка на Ptbfw“

Дипломант: Ангел Коилов

Специалност: информатика, редовно

Фак. № 0801261017

Научен ръководител:

гл. ас. Павлина Иванова

Пловдив, 2012

Съдържание

Увод

Програмистите сме умни хора. Поне си мислим че сме уми. Забелязвали ли сте, че в края на разработката на дадена система пишем доста по-добър код, отколкото в началото ? Това не е защото сме станали по-уми, а защото сме станали по-добри разработчици. Първото нещо което трябва да се научим е да разчитаме на методите си.

Случвало ни се е да оправяме проблем в код, писан от някой друг. В такива ситуации е много вероятно да се счупим нещо, което е далече като функционалност и като код.

Даже когато задачата е да променим дадена функционалност, всъщност системата се изкарва от нормалното и поведение, създаваме бъг.

За да се предпазим от подобни проблеми пишем тестове. Може и без тях и да разчитаме на тестерите, но не и за големи проекти. Едва ли помним миналата година на 7 Май след обедната почивка бъга който е оправихме? Но ако бяхме написали тест за него, ще е сигурно че няма да се прояви отново.

Колко често ни се случва да напишем код, да го качим на продукшън сървъра, да се стискаме палци и да се молим всичко да работи ?

TDD (Test-driven development) помага да се тества кода преди да е написан. Трябва да се запитаме какво всъщност правим. Да разберем отделните стъпки. Да напишем сценарии за тези стъпки. Преправяме написания код – подобряваме и се отърваваме от ненужния. Тестваме, преправяме кода и така отново и отново.

Ако TDD изглежда сложен процес, има и друг вариант. Сядаме и пишем текст. Разрешаваме проблема с който се борите в момента. Трябва да си дадем лимит – около тридесет минути. Разглеждаме какво сме измислили. Изхвърляме направеното. Започваме отначало. Вече знаем как работи и ще ни е много по-лесно да го направим отново. Но този път ще стане по-хубаво.

Не е лесно. Повечето разработчици пишем скапани тестове. Тестове които сякаш са точно отражение на кода ни. Единственият начин да се подобрим с това е да напишем много скапани тестове, много много много тестове и да изчетем много код. Хубаво е да оглеждаме тестовете на колегите си. По-добри ли са от нашите или по-лоши ? С какво ? Може ли да научим нещо от сравнението ?

Ако имаме TDD тестове не значи че приложението ни е непоклатимо. Трябват ни и unit тестове.

Точно тук е най-големият проблем. Разработчиците не знаем кои инструменти за какво са създадени. Често се опитваме да използваме един инструмент за всички тестове, понеже сме го опознали.

Главната задача на дипломната работа е да създадем инструмент, с който да не се налага да губим часове в инсталиране и конфигуриране. Трябва за няколко минути да можем да започнем писането на тестове. Инструментът трябва да ни ограничава да не правим изключително грешни неща.

В момента има два големи проекта Spec + Cucumber и Behat. Трябва да изберем кой ще надграждаме. Първият е написан на ruby, а вторият на php. Трябва ни driver за контрол на

браузъра. Driver-ът трябва да работи поне с най-известните браузъри. Ще трябва да се направим помощни модули за връщане на промените след отделните тестове в познатото за нас начално състояние на системата. Най-често това го пазим в база данни. Ще изградим модули за най-известните бази данни. Всяко голямо приложение използва кеширане. След всеки тест всякаква кеширана информация трябва да бъде премахвана. Тестовете написани от един отдел трябва лесно да бъдат преизползвани от друг отдел.

Настоящата документация се състои от въведение, изложение в седем глави, заключение и използвана литература.

- Първа глава - разглежда основните средства и технологии за реализиране.
- Втора глава - предоставя информация за:
    - инсталиране и конфигуриране на използвани и помощни приложения.
    - основните възможности на behat
    - нови и разширени модули чрез ptbfw
- Трета глава - съдържа съвети за писане на добри тестови сценарии
- Четвърта глава – предоставя информация за направата на добавка за behat
- Пета глава – предоставя информация как да направим нов driver за Mink
- Шеста глава – представя информация за формата YML
- Седма глава – съдържа информация за проблеми и поправки по използваните приложения

Използвани технологии

Почти всичко което се използва е написано на PHP.

За основа използваме behat – framework за BDD тестове. Behat използва много библиотеки, за да можем лесно и бързо да следим за промени по тях се ползва composer.

Тестовете които ще правим са съсредоточени върху уеб приложения. Трябва ни браузър. За целта използваме Selenium2 – единственият компонент който не е написан на php.

За свръзка със Selenium2 използваме наша библиотека, която се изгражда върху behat/mink-selenium2-driver, който пък използва instaclick/php-webdriver.

За да можем да симулираме въвеждане на текст в текстови полета, влачене на елементи и други подобни използваме javascript библиотеката Syn.

Behat използва предимно компоненти от Sympfony. При работа с файлове също използваме symphony компонент - Symfony\Component\Finder.

Всяко голямо приложение използва база данни, за уеб приложенията това най-често е MySQL, за връзка към нея ще ползваме PHP Data Objects.

За да може лесно да се правят нови компоненти за behat, почти всичко използва dependency injection, вчастност The Dependency Injection Component от Symphony. За да отделим лесно конфигурацията в кофигурационен файл използваме компоненти на symphony като Symfony\Component\Config\FileLocator и Symfony\Component\DependencyInjection\Loader\XmlFileLoader.

За assertion използваме PhpUnit.

За контрол на версиите се използва git,с хранилище в github.

Техническо ръководство

Инсталация

Php

Изисква се php версия 5.3 или по-нова, може да бъде изтеглен от http://www.php.net/. За **debian** базираните дистрибуции е най-добре да ползваме хранилищата, инсталираме чрез

```
$ apt-get install php5
```

Версията на php може да проверим чрез командата php-v

```
$ php -v
```

PHP **5.4.0-3** (cli) (built: Mar 21 2012 20:33:26)

Copyright (c) 1997-2012 The PHP Group

Zend Engine v2.4.0, Copyright (c) 1998-2012 Zend Technologies

в конкретния случай имаме версия **5.4.0-3**

Ако имаме по-стара версия ще получим грешка при инсталацията на модули при composer (виж Composer).

Ако не сте доволни от пакета, предоставен от хранилището на дистрибуцията може да ползвате dotdeb - http://www.dotdeb.org/.

виж http://www.php.net/manual/en/install.php

Git

Кодът на проекта се намира в хранилищата на github. За да изтеглим кода ще ни трябва git. Може да бъде изтеглен от http://git-scm.com/ . За debian базираните дистрибуции е най-добре да ползваме хранилищата, инсталираме чрез

```
$ apt-get install git
```

Ptbfw

Изтегляме кода на проекта чрез git

```
$ git clone git@github.com:ptbfw/ptbfw.git
```

За обновления проверяваме чрез командата

```
$ git pull
```

Сега имаме ядрото на проекта, но не и зависимите пакети. За разрешаване на зависимостите използваме Composer

Composer

Изтегляме изпълним файл чрез командата

```
$ wget http://getcomposer.org/composer.phar
```

за да инсталираме нужните разширения, трябва да сме в директорията на проекта(ptbfw) и да изпълним следната команда

```
$ php composer.phar install
```

за да следим за обновления по зависимите пакети изпълняваме

```
$ php composer.phar update
```

Вече имаме инсталирани

- Behat
- Behat\MinkExtension
- Beehat\Gherkin
- Behat\Mink
- Behat\MinkSelenium2Driver
- Ptbfw\Initializer
- Ptbfw\Selenium2-driver
- плюс всички необходими подпакет и всички използвани библиотеки

> Ако използвате **suhosein**, ще се наложи да whitelist-нете phar. В php.ini за команден ред (/etc/php5/**cli**/php.ini) добавете

```
suhosin.executor.include.whitelist="phar"
```

Виж Още за Composer.

Виж http://getcomposer.org/

PhpUnit

Behat използва assert функциите на phpUnit, поне аз не знам как да инсталираме phpUnit чрез composer, затова ще използваме **pear**

```
$ pear channel-discover pear.phpunit.de

$ pear channel-discover pear.symfony-project.com

$ pear install pear.phpunit.de/PHPUnit
```

виж http://pear.php.net/

Selenium 2 driver

Java

За да стартираме Selenium може да ползваме Java, C#, Ruby или Python. Ще дам пример с Java. Препоръчително е да използваме java-та на sun http://www.oracle.com/technetwork/java/javase/downloads/index.html ще използваме архивираната версия.

Selenium

Изтегляме Selenium Server (the Selenium RC Server) от http://seleniumhq.org/download/.

Стартираме сървъра чрез командата

```
$ sunJava/bin/java -jar selenium/selenium-server-standalone.jar
```

FirePath

Виж Защо ни е нужно да знаем това ?.

FirePath https://addons.mozilla.org/en-US/firefox/addon/firepath/ е добавка за **Firefox**, с нея можем да тестваме селектори по **xpath**, изисква инсталиран **firebug** - https://addons.mozilla.org/en-US/firefox/addon/firebug/

Използване на FirePath

1. Стартираме FireBug
2. Избираме таба на FirePath
3. Избираме селекция по xpath
4. Въвеждаме xpath селектор
5. Намерения елемент се очертава

Стартиране

Можем да стартираме тестовете чрез

```
$ ptbfw/vendor/behat/behat/bin/behat
```

или

```
$ ptbfw/vendor/bin/behat
```

Още за Composer

Подробна информация за composer може да се намери на сайта му http://getcomposer.org/.

За обновления по composer следим чрез командата

```
$ php composer.phar self-update
```

за по-удобно можем да добавим символичен линк към composer в /bin

```
$ ln -s /path_to_composer/composer.phar /bin/composer
```

така вместо

```
$ php composer.phar [command]
```

ще пишем само

```
$ composer [command]
```

Какво всъщност е composer?

Composer е инструмент които ни доставя нужните за проекта библиотеки и нужните на използваните библиотеки библиотеки. Разрешава и възникнали конфликти. Създава и classloader за всички библиотеки инсталирани през него.

Виж http://packagist.org/

gitHub – кратко описание

За контрол на версиите използваме Git. Всяка система за контрол версиите се нуждае от сървър. GitHub ни предоставя repository на което да си съхраняваме кода. GitHub е най-популярното repository за проекти с отворен код.

Виж https://github.com/

*Behat**

Behat

Behat е работна рамка(framework) за тестване на софтуер чрез BDD (виж http://en.wikipedia.org/wiki/Behavior_Driven_Development). Идеята на BDD е да се пишат истории, лесно четими от човек. Истории които описват поведението на разработваната система. Тези истории могат да се изпълнят като истински тестове върху системата.

Нека приемем че трябва да напишем тестове за популярната команда **`ls`** за UNIX система. За основа можем да ползваме:

```
Feature: ls

  In order to see the directory structure

  As a UNIX user

  I need to be able to list the current directory's contents



  Scenario: List 2 files in a directory

    Given I am in a directory "test"

    And I have a file named "foo"

    And I have a file named "bar"

    When I run "ls"

    Then I should get "bar foo"
```

Това е. Вече може да изпълним теста.

Behat може да се използва за всичко, дори и за уеб, предоставя библиотека наречена Mink.

Behat е вдъхновение от проекта Cucumber написан на ruby. Behat е написан на php.

Gherkin

Gherkin е парсер за историите описани в тестовете на behat.

Поддържа над 40 езика. Преводите се взимат автоматично от проекта **cucumber/gherkin**.

Независим е от Behat и може да се ползва и без него.

Виж http://cukes.info/

https://github.com/cucumber/gherkin

Mink

Една от най-важните части в уеб пространството е уеб браузъра. Браузърът е прозореца през който потребителят борави с уеб приложенията. Потребителите почти винаги използват уеб браузъри. За да тестваме дали даден браузър се държи правилно трябва да симулираме дейността между браузъра и приложението. Нуждаем се от **Mink**.

Виж http://mink.behat.org/

MinkExtension

MinkExtension е връзката между Behat и Mink.

Behat command line

За нас Behat се явява само програма, стартираща се от командния ред, която изпълнява тестове написани на Gherkin. Разполагаме с множество опции и команди, можем да ги видим като изпълним

```
$ behat -h
```

За да видим версията на behat

```
$ behat -V
```

Инициализиращи опции

Смяна на конфигурационен файл

По подразбиране behat опитва да зареди behat.yml и config/behat.yml, но ако искаме да използваме друг можем да го постигнем с параметъра –config

```
$ behat --config custom-config-file.yml
```

Смяна на използван profile

В конфигурационния файл можем да имаме различни профили. Използваният профил се задава с параметъра –profile

```
$ behat --config behat.yml --profile ios
```

Форматиращи опции

Behat поддържа различни начини на извеждане на резултатите от тестовете. Смяната на профилът използван за изходния текст се постига с параметъра –format

Има шест вградени форматъри:

1. pretty – отпечатва сценариите както са си.
2. progress – отпечатва по една буква на стъпка
3. html – същия като pretty, но изходът е форматиран в html
4. junit – генерира репорт близък на този на Junit
5. failed – отпечатва пътят до стъпките с грешки
6. snippets – отпечатва само подсказките на недефинираните стъпки.

Можем да зададем изходен файл в който да се записва резултатът

```
$ behat --format html --out report.html
```

Можем да използваме няколко форматъра наведнъж

```
$ behat -f pretty,progress
```

Може да кажем и различните форматъри да изпращат резултатите на различни места. Това се постига със запетая, като всеки форматър съответства на поредния параметър подаден към –out

```
$ behat -f pretty,progress,junit --out ,progress.out,xml
```

Понякога е трудно да се ориентираме къде точно е проблемният тест, тогава можем да ползваме –expand, за по-детайлна информация за всяка стъпка

```
$ behat –expand
```

Помощни опции

В началото е трудно да запомним целия синтаксис. Ако забравим как се започваше писане на сценарии можем да си помогнем с командата

```
$ behat –story-syntax
```

която ни дава примерна история.

Можем да пишем сценариите на различни езици, например ако искаме да използваме френски го постигаме с параметъра –lang със стойност fr

```
$ behat --story-syntax --lang fr
```

В случая ще получим примерен сценарии на френски.

Случва се да забравим какви стъпки сме описали. Можем да ги потърсим с командата -dl

```
$ behat -dl
```

Ако искаме повече информация, можем да ползваме

```
$ behat -di
```

Можем да търсим определена дефиниция

```
$ behat -d 'search string'
```

Филтри за Gherkin

Някой път искаме да пуснем само някои тестове. Един от начините за постигане на това е чрез филтриране по тагове.

```
$ behat --tags '@orm,@database'

$ behat --tags 'ticket,723'

$ behat --tags '@orm&&@fixtures'

$ behat --tags '~@javascript'
```

Първата команда ще стартира само сценарии които имат таг **@orm или** таг **@database**.

Втората команда ще стартира само сценарии които имат таг **@ticket или** таг **@723**

Третата команда ще стартира само сценарии. които имат **едновременно** таговете **@orm** и **@fixtures**.

Четвъртата команда ще стартира само сценариите които **нямат** таг **@javascript**.

```
$ behat --name 'number has'
```

Така ще стартира само тестовете които **съдържат number has** в **заглавието** си.

Виж http://docs.behat.org/guides/6.cli.html

Структура на Ptbfw

Разлики между Behat и Ptbfw

При behat структурата е следната

```
.:
.  ..  features


./features:
.  ..  bootstrap


./features/bootstrap:
.  ..  FeatureContext.php
```

При Ptbfw структурата е променена на

```
.:
behat.yml  composer.json  features


./features:
bootstrap  features


./features/bootstrap:
```

```
_classLoader.php  FeatureContext.php  _lib



./features/bootstrap/_lib:

ExampleSiteTwo  ExmapleSite  Ptbf  PtbfCommonContext



./features/bootstrap/_lib/ExampleSiteTwo:

Context
```

Features

Features е изместена във features/features/[project]. Идеята е да можем да използваме сценарии от други проекти.

Feature Context

Feature Context е изместено в features/bootstrap/_lib/[projects]. Като трябва да се спазва структурата папките да показват използвания namespace.

__classloader

- Регистрираме classloader-и за
    - Feature Context
    - Библиотеките които не използват composer:
        - PhpUnit

FeatureContext

Единствената разлика е че класа няма да наследява Behat\Behat\Context\BehatContext а ще наследи Ptbf\Context\FeatureContext. Целта е да запазим ядрото на behat максимално непроменено, за да може лесно да се добавят други разширения.

behat.yml

Behat.yml е конфигурационен файл за нашето приложение. Виж Какво е .YML.

виж http://docs.behat.org/guides/7.config.html

Допълнителни модули

Initializer

Целта на initializer е да доведе състоянието на проекта до познат етап. За целта предоставяме interface за добавяне на модули за “рестартиране” на състоянието на проекта. Рестартирането се прави преди всеки сценарии (scenario, scenario outline).

Налични service-и за Initializer

MySql

Всеки голям проект използва база данни. В уеб проектите една от най-предпочитаните бази данни е MySQL. Модулът MySQL за initializer предоставя да изпълняваме SQL заявки за довеждане на базата до познато за нас състояние.

Параметрите за модула се подават в конфигурационния файл на проекта behat.yml

```
default:

    extensions:

      Ptbfw\Initializer\Extension:

          myDataBaseInit:

                type: 'mysql'

                host: 'localhost'

                user: 'behat'

                password: '1'

                database: 'behat'

                directory: 'local'
```

Ptbfw\Initializer\Extension приема като параметър опции за service. Който ще извършва промени по базата.

- MyDataBaseInit – име използвани само от нас, информативно, за какво е service-а. Името трябва да е уникално.
- type: е видът на използвания модул за Initializer
- host: host към който се извършва връзката с базата данни
- user: потребител използван за връзка към базата данни
- password: парола използвана за връзка с базата данни
- database: име използвана база данни
- directory: директория в която се намират файлове с разширение .sql. Взимат се всички файлове по азбучен ред. Съдържанието им се сглобява в една SQL заявка която се изпълнява преди всеки сценарии.

Можем да използваме няколко service-а наведнъж, например

```
default:

    extensions:

     Ptbfw\Initializer\Extension:

         local_service:

               type: 'mysql'

               host: 'localhost'

               user: 'behat'

               password: '1'

               database: 'behat'

               directory: 'local'

         full_path_test:

               type: 'mysql'
```

```
                host: 'localhost'

                user: 'behat2'

                password: '1'

                database: 'behatTwo'

                directory:
'/var/www/ptbfw/features/bootstrap/database/test_full_path/'
```

Незадължителни параметри:

- port – порт използван за връзка към базата данни. По подразбиране е 3306.
- init_command: инициализираща команда използвана при връзка с базата данни, например

```
init_command: 'SET NAMES "UTF8"'
```

пример:

```
default:

    extensions:

          full_path_test:

                type: 'mysql'

                host: 'localhost'

                user: 'behat2'

                password: '1'

                database: 'behatTwo'

                port: 3306

                directory:
'/var/www/ptbfw/features/bootstrap/database/test_full_path/'
```

```
init_command: 'SET NAMES "UTF8"'
```

Executer

Executer изпълнява зададени му команди.

пример:

```
default:

    extensions:

      Ptbfw\Initializer\Extension:

          test:

            type: 'Executer'

            commands:

              - "ls"

              - "whoami"
```

commands се задават като списък в конфигурационния файл behat.yml.

Препоръчително е да не се използват кратки пътища за достъп до файлове или директории.

Главно модулът се ползва за стартиране на скриптове за изчистване на кеширана информация. Командите се изпълняват под ред – отгоре надолу.

При възникване на грешка при изпълнение на командата, сценарият се счита за провален. За грешки се приемат:

- върната грешка при изпълнение
- липса на какъвто и да е изходен текст от командата.

Memcached

Memcached изчиства кеширана информация от зададени му сървъри.

Пример:

```
default:
```

```
context:

extensions:

  Ptbfw\Initializer\Extension:

       memcached:

        type: Memcached

        servers:

          host: localhost
```

Има два незадължителни параметъра

host, по подразбиране е 'localhost'

port, по подразбиране е 11211

Може да приема за параметри различен брой сървъри.

Виж http://memcached.org/

Създаване на нов service за Initializer

За да създадем свой service за initializer трябва класът ни да имплементира interface-а Ptbfw\Initializer\Initers\Init. Най-важен е методъ init(), той се извиква преди всеки тестови сценарии.

Добавяме модула в конфигурационния файл behat.yml

```
extensions:

  Ptbfw\Initializer\Extension:

      моят_нов_як_модул:

           type: '[тук пишем пълното име на класа]'

           параметър1: 'стойност1'

           параметър2: 'стойност2'
```

```
параметър3: 'стойност2'
```

Параметрите ще бъдат автоматично предадени като масив към конструктора на класа.

Ако има някакви грешки init() трябва да хвърля exception. Не се смята за грешка ако върне false.

Как да използваме Initializer без Ptbfw

Първо трябва естествено да се сдобиете с кода. initializer е разработен като самостоятелен extension за behat. Кодът се намира в github, ptbfw\Initializer https://github.com/ptbfw/Initializer

За предпочитане е да ползвате composer. Initializer се разпознава от composer с името

```
ptbfw/initializer
```

В packegist Inializer може да намерите под името ptbfw/initializer, http://packagist.org/packages/ptbfw/initializer

Примерен composer.json:

```
{
    "require": {
        "behat/behat": "2.4@stable",
        "ptbfw/initializer": "*@dev"
    }
}
```

Добавете Initializer като extension в конфигурационния файл behat.yml

```
default:
  extensions:
    Ptbfw\Initializer\Extension:
        local_service:
            type: 'mysql'
```

```
                host: 'localhost'

                user: 'behat'

                password: '1'

                database: 'behat'

                directory: 'local'
```

Как е реализиран initer

Моля първо прочети Как да направим добавка за behat.

Initializer добавя два service-а

- ptbfw.initializer
- ptbfw.initializer.context.initializer

Инжектира код в context-файл чрез @BeforeScenario hook. За да се задейства extension-а трябва класът да имплементира

```
\Ptbfw\Initializer\InitializerAwareInterface
```

За намирането на файловете се ползва **The Finder Component** от Symphony.

Виж http://symfony.com/doc/2.0/components/finder.html

Ptbfw Selenium driver

Защо се налага да ползваме собствен driver?

Честно казано изобщо не вярвах че ще се стигне до това да ползваме собствен driver. Mink ни предоставя предостатъчно driver-и нека обясня накратко за всеки един

`GoutteDriver` – позволява ни да ползваме Goutte headless браузъра. Goutte е класически чист php браузър, написан е от създателя на Symfony framework - Fabien Potencier.

Поради факта че goutte е headless браузър не ни върши работа, не можем да изпълняваме лесно JavaScript.

`SahiDriver` – позволява ни да ползваме Sahi. Sahi нов контролер за браузъри. Доста бързо измества стария Selenium. Лесно се инсталира и конфигурира. Работи за почти всички браузъри. За контрол на браузъра използва proxy сървър.

Това изглежда точно за нас. Нека го пробваме.

Проблеми

- Стартира нов браузър или таб, вместо да използва стария. Това не е проблем за малки проекти, но за големи с хиляди сценарии се стартират хиляди браузъри и рано или късно рам паметта на сървъра свършва. Този проблем можем да го решим като на 100-200 сценария убиваме всички активни браузъри, но пък така ако използваме например firefox за разработване и firefox за тестване, ще си спрем и процеса на разработващия браузър. Също стартирането и убиването на браузър е изключително бавна операция.
- Sahi е че изчаква страницата да се зареди напълно, включително и ajax заявките след зареждане на страница. Това някой път не е необходимо и изисква изключително много време.
- Изчаква и ajax заявки предизвикани от потребителя – смяна на select бутон, писане на текст в input и други. Проблемът е че колкото и интелигентно да е направен скрипта за изчакване той си има timeout и ако нашите заявки го надвишат теста ще се провали. Също не всеки път може да се определи кои заявки да се изчакат и кои не, това трудно се определя даже и от човек. При някакво събитие по страницата sahi трябва да определи дали има заявки за изпълнение, да прецени дали да ги изчака, всичко това е бавно и в редки случай неправилно.
- Проблеми с alert() и prompt()

За прости тестове sahi е перфектен, но за по-сложни страници създава повече проблеми.

`ZombieDriver` – ни позволява да ползваме Zombie.js браузър емулатора. Zombie.js е headless браузър, написан е върху node.js. Поддържа всичко което и Sahi и работи почти толкова бързо колкото Goutte. Най-доброто от двата свята :), в момента работи само под Chromium. Изисква node.js и npm да са инсталирани.

Освен по причините по които не ползваме goutte, тук се добавя и лимитация само до един браузър, на нас ни трябват истински браузъри с техните бъгове :)

`SeleniumDriver` – ни позволява да ползваме известния Selenium.

Тук ще бъда кратък защо не го използваме – старо е и има версия 2.

`Selenium2Driver` – позволява ни да ползваме Selenium2 (webdriver).

За разлика от Sahi, Webdriver на изчаква ajax и не извиква javascript събитията при промяна на елемент от страницата.

Автоматично потвърждава alert() и prompt() предизвикани до зареждане на страницата. Бърз е. Разработчиците на Selenium са се съсредоточили в разработката на Selenium2.

Избрах да използваме за основа Webdriver.

Виж http://seleniumhq.org/

http://sahi.co.in/

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

какво прави Ptbfw Selenium driver

Накратко ще обясня какво ни предоставя Selenium2 и как е променено.

```
retry()
```

Пробваме да изпълним команда, ако не успеем пробваме отново.

```
getContent()
```

Взима съдържанието на страницата.

Промяна: минава през retry()

```
find($xpath)
```

Връща ни обект, отговарящ на xpath селектор.

Промяна: минава през retry()

```
executeJsOnXpath($xpath, $script, $sync = true, $requireReturn =
false)
```

Изпълнява javascript код за елемент намерен от xpath селектор.

Промяна: минава през retry()

```
getTagName($xpath)
```

Връща името на обект отговарящ на xpath. Например Select, Input, Textarea.

Промяна: минава през retry()

```
getText($xpath)
```

Връща текста, отговарящ на xpath селектор.

Промяна: минава през retry()

```
getHtml($xpath)
```

Връща html кода на страницата в момента.

Промяна: минава през retry()

```
getAttribute($xpath, $attr)
```

Връща атрибут на елемент отговарящ на xpath селектор.

Промяна: минава през retry()

```
getValue($xpath)
```

Връща стойността на елемент, отговарящ на xpath селектор.

Промяна: минава през retry()

```
setValue($xpath, $value)
```

Задава стойност на елемент, отговарящ на xpath селектор. Използва се за текстови полета – input и Textarea.

Промяна:

Съхраняваме намерения елемент в JavaScript.

Изчистваме стойността на елемента.

Ако въвеждаме текст става винаги със Syn.

```
check($xpath) {
```

Маркира чекбокс елемент, отговарящ на xpath селектор.

Промяна:

Съхраняваме намерения елемент в JavaScript.

Викаме retry() за parent метода.

Оповестяваме слушателите(listeners) за предизвикано събитие за

- клик от мишката
- смяна на стойността

```
uncheck($xpath)
```

същото като check(), но размаркира.

```
isChecked($xpath)
```

Проверява дали елемент отговаряща на xpath е чекнат.

Промяна: минава през retry()

```
selectOption($xpath, $value, $multiple = false)
```

Избира стойност от елемент от тип Select, отговарящ на xpath селектор. Ако елемента може позволява избиране на няколко опции, $multiple определя дали да добавим елемента към селекцията.

Промени:

Съхраняваме намерения елемент в JavaScript.

Викаме parent::selectOption() през retry()

Оповестяваме слушателите за

- действие с мишката
- промяна на стойността

```
click($xpath)
```

Клика върху елемент отговарящ на xpath селектор.

Промяна: минава през retry()

```
executeScript($script)
```

Промяна: минава през retry()

```
evaluateScript($script)
```

Изпълнява javascipt.

Промяна: минава през retry()

```
wait($time, $condition = 'false')
```

Изчаква $time секунди или докато $condition се оцени на true.

Промяна: по подразбиране $condition е 'false'.

Защо е направено така, грозно ?

Оригиналният driver е в процес на разработка, това което сме направили са бързи и грозни фиксове. Когато се намерят елегантни решения ще изтрием предефинирания метод и всичко ще

работи. В идеалния случай нашият клас само ще наследява оригиналния driver без да предефинира нищо.

Виж https://developer.mozilla.org/

Syn

Syn е javascript библиотека. Използва се за тестване на javascript код. Ние я използваме главно за да симулираме въвеждане на текст в дадено поле.

Тя се грижи прати известия на всички слушатели за всяко действие

- натискане на клавиша
- натиснат клавиш
- вдигане на клавиша

Използва се и за влачене на елементи.

виж http://bitovi.com/blog/2010/07/syn-a-standalone-synthetic-event-library.html

Лимити и бъгове

- В момента driver-а работи стабилно само под firefox.
- От Clément Herreman има предложен вариант за изпращане на съобщения за дадено събитие, работещ под всички браузъри **Fixing the fact that JS 'change' event isn't always fired** https://github.com/Behat/MinkSelenium2Driver/pull/2. Може би е добре да заменим нашият вариант с педложения.

Добри практики при писане на тестове

Използване на аргументи

Нека имаме примерен тест

```
/**
 * @When /^I open file test.txt$/
 */
public function iOpenFileTest()
{
    // код
}
```

Грешно е да се ограничаваме само до един файл, место това можем да ползваме аргументи

```
/**
 * @When /^I open file (.*)$/
 */
public function iOpenFileTest($filename)
{
    // код
}
```

Така можем да напишем по-общ сценарии за всякакъв файл.

Използване на по-стриктни ограничения за аргументите

Нека имаме примерен тест

```
/**
 * @When /^I should have (.*) dollars$/
 */
public function iShouldHave($dollars)
{
    // код
}
```

Проблемът тук е че **(.*)** ще хване доста неща, както исканите

- 1
- 2
- 5

така и неща като

- някакъв тест
- две коли

Можем да го ограничим до

```
/**
 * @When /^I should have ([0-9]+) dollars$/
 */
```

По този начин имаме тест само за варианти с цели числа. При евентуална грешка при писане на сценарии ще ни е много по-лесно да я намерим и отстраним.

Не се ограничавайте само до един аргумент

Нека имаме примерен тест за влизане в система

```
/**
 * @Given /^I log in using “([^”])” with password “([^”])”$/
 */
public function iLogInUsing($username, $password)
{
    // код
}
```

Понякога е по-добре да обединим две стъпки в една, за по-голяма прегледност.

Проблем с Redundant стъпки в големи проекти

Тази част е най-трудна, починете си малко преди да започнете да четете.

Имаме за задача да направим тестове на сайт, който симулира работа с телефон. На екрана ни се появява телефон и с мишката можем да цъкаме по менютата.

Нека имаме следния сценарии

```
Feature: Call test


Scenario: Go to Contacts, find John and call him
Given I am in Contacts
And I select John
```

```
Then I press Call
```

Да се съсредоточим само върху последната част **Then I press Call**

Ще изглежда приблизително като

```
/**

 * @Then /^I press Call$/

 */

public function call(){

    $element = $this->getPage()->findById('call');

    $element->Click();

}
```

Работи, няма проблеми.

След няколко месеца се появава телефон, на който **OK** и **CALL** са един и същ бутон.

Вече findById('call') не ни връща нищо.

Можем да се справим с проблема с една проста проверка

```
/**

 * @Then /^I press Call$/

 */

public function call(){

    $element = $this->getPage()->findById('call');

    if ($element === null) {

        $element = $this->getPage()->findById('ok');

    }
```

```
    $element->Click();

}
```

Отново всичко работи.

След няколко месеца се появява телефон с опция за показване на детайли, когато се фокусираш върху call, появяват се домашен телефон, служебен телефон, мобилен телефон и може да си избереш на кой да звъннеш.

Кода ни за такова действие ще е подобно на

```
/**

 * @Then /^I press focus on Call$/

 */

public function focusCall(){

    $element = $this->getPage()->findById('call');

    $element->focus();

}
```

Добре, но вече сме с опит и знаем за ситуацията с OK и CALL от преди няколко месеца, затова добавяме и този случай

```
/**

 * @Then /^I press focus on Call$/

 */

public function focusCall(){

    $element = $this->getPage()->findById('call');

    if ($element === null) {

        $element = $this->getPage()->findById('ok');
```

```
    }

    $element->focus();

}
```

Тук очеводно имаме проблем – повтаряме кода. За помощ ще имаме метод който ни връща елемента **call**.

```
protected function getCallElement() {

    $element = $this->getPage()->findById('call');

    if ($element === null) {

        $element = $this->getPage()->findById('ok');

    }

    return $element;

}
```

Така нашият код ще изглежда доста по-четим и лесно адаптиращ се към новите промени

```
/**

 * @Then /^I press focus on Call$/

 */

public function focusCall(){

    $element = $this-> getCallElement();

    $element->focus();

}

 * @Then /^I press focus on Call$/
```

```
 */

public function focusCall(){

     $element = $this-> getCallElement();

     $element->focus();
```

Добре, но ако имаме дублиране не само на OK и CALL, ами имаме телефон само с един бутон, с два бутона, без никакви бутони. Проверките дали съществува елемент и ако не да пробваме с друг са бавни. Трябва ни нещо по-добро.

За да решим този проблем ще имаме клас абстрактен клас Phone, който ще бъде наследяван от всеки нов телефон който дойде за тестване. При нужда ще разширим някой методи.

За да вземем правилната инстанция ще използваме шаблона Fantory(препоръчвам да погледнете http://www.apress.com/9781430229254/).

```
abstract class Phone{

     private $page;

     function __construct($page) {

          $this->page = $page;

     }



     public function getCallElement() {

          return $this->page->findById('call');

     }



     public function Call() {

          $this->getCallElement()->click();
```

```
    }


    public static function factory($page) {
        if ($page->hasContent('555')) {
            return new Phone555();
        } elseif ($page->hasContent('6262')) {
            return new Phone6262();
        }
    }
}


class Phone555 extends Phone{
    public function getCallElement() {
        return $this->page->findById('ok');
    }
}


class Phone6262 extends Phone{
    public function getCallElement() {
        return $this->page->findById('call');
```

```
        }

}
```

Този вариант е бърз и ни позволява гъвкавост при поява на нов телефон, можем даже да си направим помощни абстракни класове като Iphone, Android, WindowsMobile и други.

Тествайте бизнес логиката

Много хора не са запознати за какво всъщност са предназначени инструментите които използват при разработката на софтуер. Представете си да използваме поялник за да отвием болт. Поялникът не е лош инструмент, просто не е създаден за да завива и отвива болтове. Каква е целта на инструменти като xUnit и PHPUnit ? Целта им е да тества кодът. За да тества код, трябва да имаме написан код! Предоставят ни се инструменти като code coverage, mocks, fixed amount of calls – всичко това е много полезно при unit тестовете. Целта е да ни информира когато счупим нещо. Изграждаме система която ни информира кога нещо не работи. Имаме изключително лесен за тестване код. Но не от това се нуждаем в TDD. Едни инструменти се грижат да опазят кода ни от счупване и това е главната им цел. Други се грижат да изградим правилна логическа връзка между системата и обектите в системата. Едните се целят в тестването на нещо вече създадено и напълно описано, а другите описват нещо ново. Това е основната разлика между двете. Вероятно в момента не разбирате нищо, това е защото инструментите които ползвате за вас се явяват просто различни по синтаксис, без да се задълбочите каква всъщност е целта им.

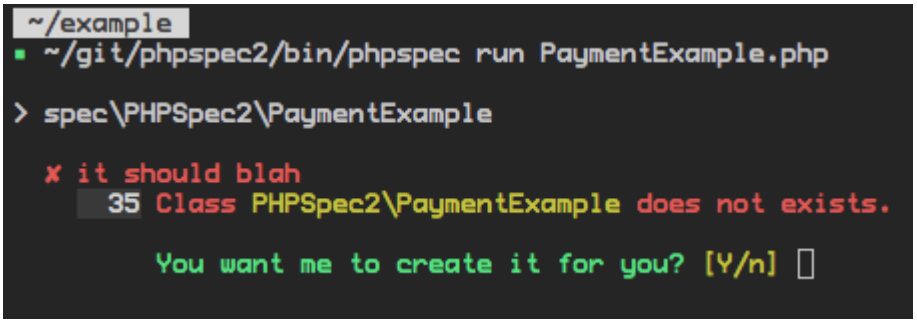
Разликата

Какво ще стане ако пробваме да използваме xUnit за тестване на несъществуващ клас ? Вече смятам че сте наясно, че с BDD можем да опишем код, преди да е написан.

```
~/example
• pu PaymentTest.php
PHPUnit 3.6.12 by Sebastian Bergmann.

PHP Fatal error:  Class 'Payment' not found in /Users/everzet/example/PaymentTest.php on line 7
PHP Stack trace:
PHP   1. {main}() /usr/local/php5-20120823-092307/bin/phpunit:0
PHP   2. PHPUnit_TextUI_Command::main() /usr/local/php5-20120823-092307/bin/phpunit:46
PHP   3. PHPUnit_TextUI_Command->run() /usr/local/php5-20120823-092307/lib/php/PHPUnit/TextUI/Command.php:130
PHP   4. PHPUnit_TextUI_TestRunner->doRun() /usr/local/php5-20120823-092307/lib/php/PHPUnit/TextUI/Command.php:192
PHP   5. PHPUnit_Framework_TestSuite->run() /usr/local/php5-20120823-092307/lib/php/PHPUnit/TextUI/TestRunner.php:325
PHP   6. PHPUnit_Framework_TestSuite->runTest() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestSuite.php:745
PHP   7. PHPUnit_Framework_TestCase->run() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestSuite.php:772
PHP   8. PHPUnit_Framework_TestResult->run() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestCase.php:751
PHP   9. PHPUnit_Framework_TestCase->runBare() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestResult.php:649
PHP  10. PHPUnit_Framework_TestCase->runTest() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestCase.php:804
PHP  11. ReflectionMethod->invokeArgs() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestCase.php:942
PHP  12. PaymentTest->testUnexistingStuff() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestCase.php:942

Fatal error: Class 'Payment' not found in /Users/everzet/example/PaymentTest.php on line 7

Call Stack:
    0.0001     227776   1. {main}() /usr/local/php5-20120823-092307/bin/phpunit:0
    0.0026     532048   2. PHPUnit_TextUI_Command::main() /usr/local/php5-20120823-092307/bin/phpunit:46
    0.0026     532280   3. PHPUnit_TextUI_Command->run() /usr/local/php5-20120823-092307/lib/php/PHPUnit/TextUI/Command.php:130
    0.0136    1956224   4. PHPUnit_TextUI_TestRunner->doRun() /usr/local/php5-20120823-092307/lib/php/PHPUnit/TextUI/Command.php:192
    0.0154    2244344   5. PHPUnit_Framework_TestSuite->run() /usr/local/php5-20120823-092307/lib/php/PHPUnit/TextUI/TestRunner.php:325
    0.0155    2246008   6. PHPUnit_Framework_TestSuite->runTest() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestSuite.php:745
    0.0155    2246040   7. PHPUnit_Framework_TestCase->run() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestSuite.php:772
    0.0155    2247248   8. PHPUnit_Framework_TestResult->run() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestCase.php:751
    0.0161    2305016   9. PHPUnit_Framework_TestCase->runBare() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestResult.php:649
    0.0168    2414800  10. PHPUnit_Framework_TestCase->runTest() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestCase.php:804
    0.0168    2416024  11. ReflectionMethod->invokeArgs() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestCase.php:942
    0.0168    2416056  12. PaymentTest->testUnexistingStuff() /usr/local/php5-20120823-092307/lib/php/PHPUnit/Framework/TestCase.php:942
```

Усещате ли какво става ? Инструментът ни **крещи** ! Крещи ни защото нещо е адски неправилно.

Това му е и работата – да ни направи живота гаден, защото сме счупили нещо. Проблемът е че при TDD, това че нещо не съществува или е счупено не трябва да е проблем; защото все още сме в етап на проектиране. Вместо да ни **крещи**, инструментът трябва да **говори** с нас и да ни предоставя варианти за решаване на проблемът.



PHPUnit възможно най-лошият инструмент ли в е този случай ? Да ! PHPUnit лош инструмент ли е във всички случай – не! Той е отверка и трябва да не го използваме като поялник.

Както виждате – целите са различни.

PHPSpec2 говори с нас през конзолата, защото знае че не сме готови и имаме нужда от помощ.

Xunit ни крещи, защото сме счупили нещо работещо.

Какво става когато използваме грешния инструмент – объркване.

Често чуваме че не правим TDD, защото сме били мързеливи или защото не сме достатъчно опитни. Всъщност най-често TDD не се правят защото се чувстваме некомфортно с инструментите които ползваме. Можем да ползваме бомбардировач, който взривява съседната гора за да ни събуди сутрин за работа, но е доста по-комфортно да използваме будилник.

Всичко в PHPSpec2, включително синтаксиса и CLI интерфейса работи за целта си – да ни помогне да опишем логиката на апликацията. Помага ни да разберем как обектите си взаимодействат и каква работя трябва да извършват. Има ли значение колко пъти ще се извика дадем метод от мок ? В някой случай да, но в повечето не. Ето защо PHPSpec2 никога няма да ни пита подобни въпроси. В повечето случай това няма как да го знаем по време на разработката на дизайна.

Всичко в xUnit, включително синтаксиса, работят за тяхната си цел – да ни помогнат да не счупим апликацията. Има ли значение колко пъти ще се извика функция от някой мок ? Все пак няма да навреди ако кажем колко, все пак кодът е написан и би трябвало да ни е известно. Ако се извика повече или по-малко пъти, вероятно има някакъв бъг. Ето защо в случая сме принудени да отговорим на въпроса. В този случай може да е важно.

Виж http://everzet.com/post/31581124270/fullstack-bdd-2012-wrapup

http://pragprog.com/book/hwcuc/the-cucumber-book

http://www.phpspec.net

http://www.phpunit.de

Как да направим добавка за behat

Има голяма разлика между behat 2.3 и 2.4. При 2.4 целия код се пренаписва, използвайки леко променен Dependency Injection от Sympfony.

Главното е да си регистрираме service

```
<?xml version="1.0" ?>

<container xmlns="http://symfony.com/schema/dic/services"

            xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"


xsi:schemaLocation="http://symfony.com/schema/dic/services
http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>

        <service id="ptbfw.initializer"
class="Ptbfw\Initializer\Extension" />

        <service id="ptbfw.initializer.context.initializer"
class="Ptbfw\Initializer\Context\InitializerInitializer">

            <argument>%ptbfw.initializer.parameters
%</argument>

            <tag name="behat.context.initializer" />

        </service>



    </services>
```

```
</container>
```

Това става при създаването на класа, който служи за връзка между behat и нашия extension

```
class Extension extends \Behat\Behat\Extension\Extension {



    public function load(array $config, ContainerBuilder
$container) {

        $loader = new XmlFileLoader($container, new
FileLocator(__DIR__ . '/services'));

        $loader->load('core.xml');

        $container-
>setParameter('ptbfw.initializer.parameters', $config);

    }



}
```

Остава само да кажем кога да сработва нашия extension

```
use Behat\Behat\Context\Initializer\InitializerInterface;

use Behat\Behat\Context\ContextInterface;



class InitializerInitializer implements InitializerInterface {

    private $parameters;

    public function __construct(array $params) {

        $this->parameters = $params;
```

```
    }

    public function supports(ContextInterface $context) {

        if ($context instanceof
\Ptbfw\Initializer\InitializerAwareInterface) {

            return true;

        } else {

            return false;

        }

    }



    public function initialize(ContextInterface $context) {

        $context->useContext(

            'ptbfw\\Initializer\\Context\\Initializer',

         new \Ptbfw\Initializer\Context\InitializerContext(

            $this->parameters

            )

        );

    }

}
```

Параметрите в конструктора се подават като масив, взет от конфигурационния файл(**behat.yml**), точно както във **feature context**.

Методът **supports** определя към кои класове може да се ползва нашия extension.

В методът **initialize** поставяме кодът за обработка на желания клас.

виж http://everzet.com/post/22899229502/behat-240

http://symfony.com/doc/current/components/dependency_injection/index.html

Как да направим нов driver за Mink

Защо ни е нужно да знаем това ?

Когато въведем и браузъра в цялата картинка service-ите на които разчитаме стават прекалено много. Имаме Behat, който използва Mink, но пък за да това минаваме през MinkExtension. До тук е добре, всичко е на едно, можем лесно да дебъгнем през php и да проследим какво става. Но като намесим и браузър се добавят прекалено много неизвестни. Първо Mink разчита на driver за да си говори с proxy сървър, а proxy сервъра разчита на браузър.

Нека ги разделим на групи.

От една страна имаме Behat, Mink и използвания от Mink driver. От другата имаме браузър и proxy server.

Proxy сървъра показва и пази информация за всички извършени действия. По лога можем ръчно да проследим и симулираме всяко едно действие.

При възникнал проблем можем лесно да се ориентираме от коя страна да търсим наработещата част. Не е и изключено проблемът да е в самия браузър.

Примерен лог:

```
00:02:32.086 INFO - Executing: [find element: By.xpath:
(//html/.//*[self::input | self::textarea | self::select]
[not(./@type = 'submit' or ./@type = 'image' or ./@type =
'hidden')][(((./@id = 'select_b' or ./@name = 'select_b') or
./@id = //label[contains(normalize-space(string(.)),
'select_b')]/@for) or ./@placeholder = 'select_b')] |
.//label[contains(normalize-space(string(.)),
'select_b')]//.//*[self::input | self::textarea | self::select]
[not(./@type = 'submit' or ./@type = 'image' or ./@type =
'hidden')])[1]] at URL: /session/1347210736414/element)
```

Нормално е това да не ви говори нищо, в края на главата ще се изясни.

Реализация

За да използваме driver-а с Mink, трябва да използваме interface-а Behat\Mink\Driver\DriverInterface

Задаване на текуща сесия.

```
setSession(Session $session);
```

Стартиране на браузър

```
function start();
```

Приверка дали браузъра е стартиран

```
function isStarted();
```

Спиране на браузъра.

```
function stop();
```

Рестартиране на браузъра.

```
function reset();
```

Пренасочване към страница

```
function visit($url);
```

Взимане на текущата страница

```
function getCurrentUrl();
```

Презареждане на страница.

```
function reload();
```

Преминаване една стъпка напред в историята на посетените страници.

```
function forward();
```

Преминаване една стъпка назад в историята на посетените страници.

```
function back();
```

HTTP Basic authentication parameters

```
function setBasicAuth($user, $password);
```

Смяна на прозореца на браузъра.

```
function switchToWindow($name = null);
```

Преминаване към използване на iframe

```
function switchToIFrame($name = null);
```

Смяна на използваните header-и

```
function setRequestHeader($name, $value);
```

Взимане на header-ите върнати като отговор от сървъра.

```
 function getResponseHeaders();
```

Смяна на бисквитки

```
 function setCookie($name, $value = null);
```

Взимане на бисквитка

```
function getCookie($name);
```

Взимане на статус кода, върнат от сървъра. Виж http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

```
function getStatusCode();
```

Взимане на съдържанието на страницата.

```
function getContent();
```

Намиране на елемент по xpath.

```
function find($xpath);
```

Взимане на вида на елемент според xpath.

```
function getTagName($xpath);
```

Взимане на текста, отговарящ на xpath.

```
function getText($xpath);
```

Взимане на html кода на елемент, отговарящ на xpath.

```
function getHtml($xpath);
```

Взимане на атрибут на елемент, отговарящ на xpath

```
function getAttribute($xpath, $name);
```

Взимане на стойност на елемент, отговарящ на xpath

```
function getValue($xpath);
```

Смяна на стойност на елемент, отговарящ на xpath

```
function setValue($xpath, $value);
```

Маркиране на чекбокс, отговарящ на xpath

```
function check($xpath);
```

Размаркиране на чкбокс, отговарящ на xpath

```
function uncheck($xpath);
```

Проверка дали чекбокс отговарящ на xpath е маркиран

```
function isChecked($xpath);
```

Избиране на опция от селект елемент, отговарящ на xpath. Грижи се и за обработката на селект елементи с опция за маркиране на няколко опции.

```
function selectOption($xpath, $value, $multiple = false);
```

Кликване с мишката на елемент, отговарящ на xpath

```
function click($xpath);
```

Двоен клик с мишката на елемент отговарящ на xpath

```
function doubleClick($xpath);
```

Кликане с десния бутон на мишката, върху елемент, отговарящ на xpath

```
function rightClick($xpath);
```

Поставяне на файл в поле(отговарящо на xpath) за качване на файл.

```
function attachFile($xpath, $path);
```

Проверка дали елемент, отговарящ на xpath е видим.

```
function isVisible($xpath);
```

Придвижване на мишката до елемент, отговарящ на xpath

```
function mouseOver($xpath);
```

Фокусиране върху елемент, отговарящ на xpath

```
function focus($xpath);
```

Премахване на фокус от елемент, отговарящ на xpath

```
function blur($xpath);
```

Натискане на клавиш или клавишна комбинация в елемент, отговарящ на xpath

```
function keyPress($xpath, $char, $modifier = null);
```

Натискане (само надолу) клавиш или клавишна комбинация в елемент, отговарящ на xpath

```
function keyDown($xpath, $char, $modifier = null);
```

Натискане (само нагоре) клавиш или клавишна комбинация в елемент, отговарящ на xpath

```
function keyUp($xpath, $char, $modifier = null);
```

Влачене на мишката от елемент отговарящ на **sourceXpath** до елемент отговарящ на **destinationXpath**

```
function dragTo($sourceXpath, $destinationXpath);
```

Изпълняване на скрипт.

```
function executeScript($script);
```

Изпълняване на скрипт код и връща като резултат върнатото от изпълнения код..

```
 function evaluateScript($script);
```

Изчакване time секунди, или докато condition не се уцени на вярно

```
function wait($time, $condition);
```

Ако забелязахте задължително трябва да имаме getStatusCode(), но в Selenium2Driver и Sahi пише че нямаме на разположение такава функционалност а те са driver-и за Mink.

За това имаме специален вид exception, Behat\Mink\Exception\UnsupportedDriverActionException, използван за функционалностите които не се поддържат. Пример със selenium2driver и getStatusCode()

```
    /**
     * Returns last response status code.
     *
     * @return  integer
     */
    public function getStatusCode()
    {
        throw new UnsupportedDriverActionException('Status code is not supported by %s', $this);
    }
```

Като сме готови с нашия driver, остава само да го добавим в сесия и да регистрираме сесията в Mink

```
$driver = new МоятНовDriver();

$session = new \Behat\Mink\Session($driver);

$session->start();

$session->visit($options['base_url']);

$mink->registerSession($options['session_name'], $session);
```

Най-вероятно ще искаме нашият driver да е използвания по подразбиране

```
$mink->setDefaultSessionName($options['session_name']);
```

Тук има само xpath селектори, къде са CSS и другите ?

Селекторите са отделни от driver-ите. Може да намерите селекторите в Behat\Mink\Selector. В момента Mink разполага с 3 селектора:

1. CSS (Cascading Style Sheets)
2. Named

Всеки един от тях имплементира interface Behat\Mink\Selector\SelectorInterface. Това изисква всеки един от тях да има метод

```
function translateToXPath($locator);
```

Така Mink транслира селекции различни от xpath до cpath.

Къде е тогава xpath селектора ? Той не ни трябва, защото можем да подадем xpath директно към driver-а.

Можем да си направим собствен селектор, ако искаме да го ползваме с Mink, трябва да имплементираме Behat\Mink\Selector\SelectorInterface.

Какво е .YML

YAML (произнася се /ˈjæməl/,) е език за представяне на серизлизирана информация, във формат четим за хората. Идеята за представянето е взета от програмни езици като C, Perl, и Python, някои идеи от XML и форматът на електронната поща (RFC 2822).YAML е рекурсивен акроним от "YAML Ain't Markup Language". В началото, YAML е значело "Yet Another Markup Language".

YML предоставя възможност за представяне на информацията с възможност новите елементи да се отделят на нов ред и за групиране да се използват интервали, така и на един ред.

Пример за форматиране на лист с използване на нови редове и интервали

```
---# Любими езици

- Php

- C++

- JavaScript
```

Едноредов пример за форматиране на лист

```
---# Любими езици

  [Php, C++, JavaScript]
```

Асоциативни масиви

```
---# използват се два интервала за да разграничим елемтите от
масив

--# ключ: стойност

  име:  Иван

  номер: 1

  възраст: 22
```

Едноредов пример

```
--# използва се запетая + интервал за различаване на
ключ:стойност

{име: Иван, номер: 1, възраст: 22}
```

Пример за асоциативен масив

```
Състезател1:

  име: Иван

  номер: 1

Състезател2:

  име: Ангел

  възраст: 23
```

Пример за асоциативен масив с елементи от тип лист

```
Плодиве: [портокал, ананас, ягода]

Зеленчуци:

  -  зеле

 - домати
```

Това са само главните функционалности на yml, в конфигурационните файлове много рядко ще срещнете нещо повече. За по-подробна информация прочетете статията в wikipedia http://en.wikipedia.org/wiki/YAML.

Bugs & Fixes на използвани приложения

Behat

Включване на файловете от /bootstrap

В началото, когато създавах структурата на първоначалната версия на проекта имах един доста странен проблем с включването на файлове. Би трябвало файловете от директория **/bootstrap** да се включват автоматично в проекта

“`features/bootstrap/` - Every `*.php` file in that directory will be autoloaded by Behat before any actual steps are executed”

Тогава защо някой път ми даваше непознати класове при наследяване ? Наблягам на някой път! Споделих проблемът си в behat@freenode.net. Там се срещнах за шърви път с everzet, който се оказа създателя на Behat, истинското му име е Konstantin Kudryashov. Доста набързо и не помисляйки решихме да се преправи да включва файловете по име. Все още не съм сигурен дали това е най-доброто решение.

По това време Behat, Mink и Gherkin бяха едно, за жалост не мога да изровя точната ревизия на кода, но ще дам пример със сегашната.

```
/**
 * Processes data from container and console input.
 *
 * @param InputInterface  $input
 * @param OutputInterface $output
 *
 * @throws \InvalidArgumentException
```

```
 */

public function process(InputInterface $input, OutputInterface
$output)

{

    // code …

     if (is_dir($bootstrapPath = $this->container-
>getParameter('behat.paths.bootstrap'))) {

          $iterator = Finder::create()

               ->files()

               ->name('*.php')

               ->in($bootstrapPath)

          ;



          foreach ($iterator as $path) {

               require_once((string) $path);

          }

}
```

За промяната добави единствено извикване на метод към $iterator, и стана

```
$iterator = Finder::create()

     ->files()

     ->name('*.php')
```

```
    ->sortByName()

    ->in($bootstrapPath)

;
```

С това се зачетох за използваните symphony компоненти и взеха че ми харесаха :)

Gherkin

Няколко истории в един файл?

Когато сценариите се увеличиха, започнах да ги разделям в различни папки и подпапки. На места имах файлове със съдържание

```
Feature: примерен тест 1

Scenario: тестване на функционалности с ляв бутон на мишката

Given Аз се намирам на страницата на НАСА

Then ...



Feature: примерен тест 2

Scenario: тестване на функционалности с десен бутон на мишката

Given Аз се намирам на страницата на НАСА

Then ...
```

Нещо не ми харесваше в структурата, но не знаех какво. Четох книжката за Cucumber и където видя четиво и него минавах. Някъде прочетох че може да имаш само един Feature във файл.

Странно, Behat ми позволяваше. Клонирах кода от https://github.com/cucumber/cucumber и по някакъв начин го подкарах. Там не ми даваше да имам два или повече разказа в един файл.

Дискутирахме до с everzet и той каза че не е проблем да остане така, след няколко дни му цитирах нещо, че не може да имаш две или повече истории в един файл, тествахме как се държи

и cucumber. Оказа се че има проблем. Поради малкия брой тестове, аз лесно си ги оправих. След няколко дни излезе нова версия на behat/gherkin, която оправи проблема. Нямаше много сърдити :)

Това бе най-трудния за обяснение проблем който бях срещал до сега.

Selenium2Driver

При разработката на Ptbfw Selenium driver се натъкнах на проблем при реализиране на метода retry(). Един от моментите в които трябва да повторя дейстие е когато се върне като стойност NULL. Точния проблем беше при getValue(), даваше NULL, когато например имаме

```
<input type="text" value="">
```

Би трябвало да върне “” (празен стринг).

Този проблем го оправих сам, беше проблем в javascript кода използван от webdriver.

Както е редно си направих форк, оправих проблема и пратих PR (pull request), което автоматично доведе до създаване на тикет https://github.com/Behat/MinkSelenium2Driver/pull/1.

Ето и промените по кода: https://github.com/Behat/MinkSelenium2Driver/pull/1/files

```
     }

 } else {

     attributeValue = node.getAttribute('value');

-    if(attributeValue) {

+    if (attributeValue != null) {

         value = "string:" + attributeValue;

-    } else if(node.value) {

+    } else if (node.value) {

         value = "string:" + node.value;

     } else {

         return null;
```

Заключение

В настоящата дипломна работа е реализирахме framework за BDD тестове.

Изградихме система, която ни предоставя бърза инсталация и конфигурация, само с пет команди:

```
$ wget http://selenium.googlecode.com/files/selenium-server-standalone-2.25.0.jar

$ java -jar selenium-server-standalone-2.25.0.jar

$ git clone git@github.com:ptbfw/ptbfw.git

$ wget http://getcomposer.org/composer.phar

$ php composer.phar install
```

Имплементирахме driver, разширяващ Selenium2, за контрол на браузъра. Driver-ът работи стабилно с firefox. Реализацията е разгледана глава Ptbfw Selenium driver.

Реализирахме е модул за пренасяне на системата в познато състояние преди всеки тест. Реализирахме под-модули за по-лесно връщане на състоянието на бази данни MySQL, модул за изпълняване на команди въведени от потребителя и модул за изчистване на кеширана информация, съхранена в memcached сървър. Изградихме структура, която позволява лесно добавяне на нов под-модул. Реализацията е разгледана глава Initializer.

При реализирането на дипломната работа открихме и решихме проблеми в използваните технологии. Разгледано в глава Bugs & Fixes на използвани приложения.

Остава да направим поддръжка за всички известни браузъри на Selenium. Целта на Ptbdw Selenium2Driver е да тества дали selenium е подходящ за целта. Selenium driver-ът е все още в процес на разработка, решения за проблемите се предлагат от много разработчици. Въпрос на време е да намерим елегантно решение на всички проблеми и Ptbdw Selenium2Driver да стане ненужен.

Перспективите за доразвитие са насочени главно към три области

1. Възможност за тестване на мобилни сайтове
2. Възможност за тестване на API(Application programming interface) реализирано чрез soap(Simple Object Access Protocol), REST(REpresentational State Transfer) или друга подобна технология.
3. Предоставяне на още модули за пренасяне на приложението в познато състояние – повече поддържани бази данни, като neo4j, MariaDB. Предвижда се модул за пренасяне на пощенска кутия до познато състояние. Предвиждат се под-модул за премахване на целия

кеш за най-известните кеширащи системи – APC(Alternative PHP Cache) и memcache.

Системата я подложихме както на ръчно тестване, така и на автоматизирано такова използвайки себе си. След проведените тестове установихме коректна работа на системата.

В резултат на успешно проведения експеримент можем да заключим, че целите на дипломната работа са частично постигнати.

Речник

API - Application programming interface

APC - Alternative PHP Cache

BDD - Behavior-driven development

CSS - Cascading Style Sheets

CLI - Command-line interface

DSL - Domain Specific Language

JSON - JavaScript Object Notation

TDD - Test-driven development

REST - REpresentational State Transfer

soap - Simple Object Access Protocol

Syn - a Standalone Synthetic Event Library

XPath - XML Path Language

XML - Extensible Markup Language

YAML - YAML Ain't Markup Language

Източници

Wikipedia

- http://en.wikipedia.org/wiki/YAML
- http://en.wikipedia.org/wiki/Behavior_Driven_Development

Behat.org

- http://behat.org/
- Behat community
    - #behat@irc.freenode.com
    - google group https://groups.google.com/forum/?hl=bg&fromgroups#!forum/behat
- http://mink.behat.org/
- http://extensions.behat.org/mink/

Composer

- http://getcomposer.org/

Packacgist

- http://packagist.org/

Symphony

- http://www.symfony-project.org/
    - http://symfony.com/doc/current/components/dependency_injection/introduction.html
    - http://symfony.com/doc/current/components/finder.html

Zend framework

- http://framework.zend.com/
- http://framework.zend.com/zf2

everzet's blog

- http://everzet.com/
    - http://everzet.com/post/22899229502/behat-240
    - http://everzet.com/post/31581124270/fullstack-bdd-2012-wrapup

git

- http://git-scm.com/

github

- https://github.com/
    - https://github.com/Behat/MinkExtension-example

Php

- http://www.php.net/
- PDO
    - http://php.net/manual/en/book.pdo.php
- Pear
    - http://pear.php.net/

Apress PHP Objects Patterns and Practice 3rd Edition

- http://www.apress.com/9781430229254/

Selenium

- http://seleniumhq.org/

Sahi

- http://sahi.co.in/

bitovi.com

- http://bitovi.com/blog/2010/07/syn-a-standalone-synthetic-event-library.html

Gherkin

- http://cukes.info/

The Cucumber Book: Behaviour-Driven Development for Testers and Developers

- http://pragprog.com/book/hwcuc/the-cucumber-book

www.w3.org

- http://www.w3.org
    - http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

mozilla.org

- https://developer.mozilla.org/

PHPSpec

- http://www.phpspec.net

PHPUnit

- http://www.phpunit.de

Memcached

- http://memcached.org/