Comparison of AngularJS framework testing tools

Nina Fat, Marijana Vujovic, Istvan Papp, Sebastian Novak

***Abstract*—AngularJS, the new Javascript framework, with wide usage across web browsers and great expression power, comes with a shortcoming – lack of compiler optimization. Consequently it is strongly recommended that every application written in JavaScript, regardless of the used framework, should include tests that validate both its behavior and performance. This paper presents and evaluates two popular web automation testing frameworks: Protractor and Karma. As an addition to the description of testing frameworks in this paper, we present the results of running tests on optimized and unoptimized web application used in real systems.**

***Index Terms*—Testing, Web Automated testing, Unit testing, End-to-end testing, Karma, Protractor**

I. Introduction

ANGULARJS and newest web frameworks such as ReactJS, KnockoutJS, PolymerJS, used for dynamic development of web application have been recently introduced [1] .These frameworks got their popularity, by using concepts of data-binding, templating, routing, features of single-application pages, MVC and etc. These concepts relieve the developer from writing boiler-plate code, thus easing the development of JavaScript applications in need of fast shipping of final product to customer. Regardless of speed at which they can be learned and implemented, the final product needs to pass a certain level of quality assurance, before it can be delivered to the end user. This is a challenge since there is no static typing, making type checking and IDEs support difficult.

However, by providing continuous integration of tests with source code, the challenge can be overcome. There are a few web frameworks for testing applications such as Jasmine, Mocha and Coffee. Because of its readability and integration across testing tools, Jasmine was chosen. Jasmine tests consist of suites and specs. Suites consist of related tests for application and each test is called a spec. Suite must contain describe() function with brief description of testing and it can contain several specs with it() function. Developer can describe within each it() block what is expected to be result. This block also contains matchers which compare results of test, called actual values with some expected values. This makes documentation easy to generate after test execution because brief string descriptions inside specs and suites explain the main functionality of tested module.

Nina Fat, Author, RT-RK, Narodnog Fronta 23b, 21000 Novi Sad, Serbia (phone: 381-61-6596166; e-mail: nina.fat@rt-rk.com).
Marijana Vujovic, Co-Author, RT-RK, Narodnog Fronta 23b, 21000 Novi Sad, Serbia (e-mail: marijana.vujovic@rt-rk.com).
Istvan Papp, Co-Author, RT-RK, Narodnog Fronta 23b, 21000 Novi Sad, Serbia (e-mail: istvan.papp@rt-rk.com).
Sebastian Novak, Co-Author, RT-RK, Narodnog Fronta 23b, 21000 Novi Sad, Serbia (e-mail: sebastian.novak@rt-rk.com).

To better understand the testing methodologies presented in this paper, it is relevant to understand the AngularJS framework. AngularJS applications' software architecture is based on the Model View Controller pattern (Fig 1.). This means that every AngularJS application consists of:

- Model − lowest level of the pattern responsible for maintaining data.
- View − responsible for displaying all or a portion of the data to the user. In text it is also reffered as HTML template.
- Controller − controls the interactions between the Model and View

Development of every AngularJS application starts with designing a controller which controls the model via view which is updated by any change introduced to the model [2]. Therefore, page's data is refreshed without refreshing the entire page. This is achieved by injecting watchers as event listeners. Every time a watcher is triggered, a digest cycle is started and page is supplied with refreshed data. This is crucial to consider when developing an efficient application.

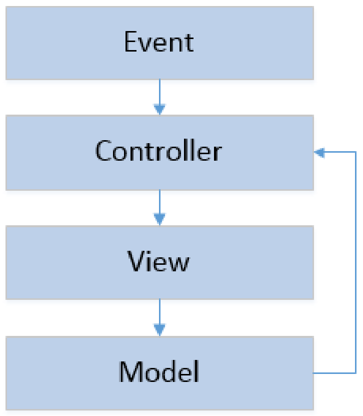

Fig. 1. Model view controller

Description of each testing methodology in parallel with corresponding tool to test AngularJS application will be introduced in the following chapters. Methods and tips for optimizing AngularJS application are described in chapter three. Descriptions of application optimization methods include results measured with each tool. Conclusion provides a detailed result analysis.

II. Unit and End-to-end Testing

The main difference between unit and end-to-end testing methodologies is in part of the application they are granted access.

Unit test can access model and controller (Fig 1.). Unit test tools can inject controller, services, as well as filters. This means that a developer can test every module of an application independently.

End-to-end test can only access the view. Using HTML template of an application, the main functionalities can be tested, such as opening a new view when a button is clicked. End-to-end application testing requires that the application under test is complete and is running on a server. Its advantage is that it conducts tests in a manner similar to manual testing, where a human operator would be testing the application.

A. *Karma*

Karma is a tool which uses a web server that executes source code against test code for each browser [3]. Karma watches all the files, specified within the configuration file, and whenever any file changes, it triggers the test run by sending a signal to the testing server. Source files are loaded inside an Iframe, tests are executed and results are recorded.

Testing tool is easily consolidated across many browsers, and supports most testing frameworks (Jasmine, Mocha, Qunit). The developer only needs to install required runner for desired browser.

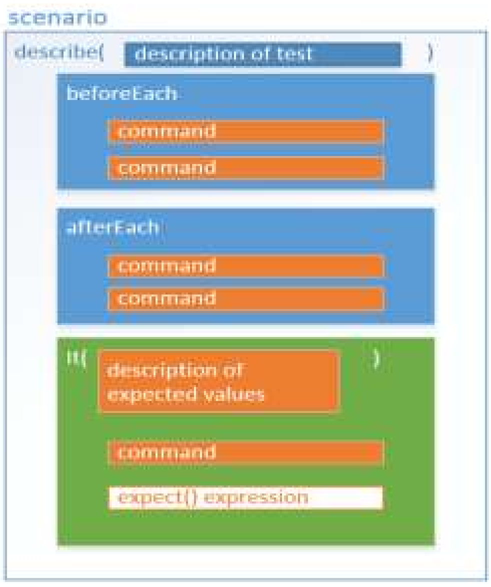

Fig. 2. Visual representation of Jasmine tests

Each stage of testing requires preparation. Before any testing dependencies, services, filters must be instantiated. And then the specification for a test is created. After specification is made, the resulting expression must be matched with a matcher expression. Matcher are often expressions used in expect functions. These can compare different expressions against strings, other expressions, Booleans etc. Series of commands can be run, before and after each test (Fig 2.).

B. *Protractor*

Protractor is a framework also built upon the Selenium Server, developed by the Google Team. The Selenium Server takes care of interpreting commands from the test and forwards them to one or more browsers [4]. Communication between the server and the browser uses the WebDriver's Wire Protocol, a JSON-based protocol. Commands are interpreted by the Browser Driver (Fig 3.). The advantage using the Protractor, as a wrapper for Selenium, is that it adds a way of selecting AngularJS elements such as bindings or repeaters [8].

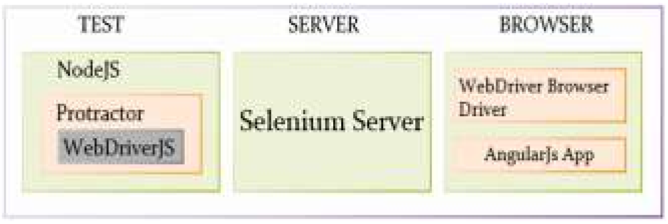

Fig. 3. Model of Protractor tests

Protractor's reliability is backed by the fact that it was developed by the AngularJS team and the fact that AngularJS's documentation includes Protractor tests. It also gives convenient helper/utility methods, e.g. waiting for a page to load, etc. Compatibility with different frameworks and different browsers is also a plus.

Developer must install the webdriver-manager as a helper tool, in order to get an instance of a Selenium Server running. As an addition, the configuration must be instantiated, and elements that need to be tested must be visible inside browser. Visible elements are the only type of elements that can be tested. Developer cannot test a controller, services or filters directly. Those elements can be tested indirectly through their HTML and CSS templates.

III. OPTIMIZATION TECHNIQUES

The optimizations are conducted in the early stages of application's development and include main ideas such as:
1. Caching of data in AngularJS directive ng-repeat.
2. Bypassing watchers
3. Using one time binding syntax where is needed

The first optimization requires the test script to track array elements, used by the *ng-repeat* directive, via $index [6]. Using keep track function user has insinght into all items in the collection and their corresponding DOM elements. If an item is inserted to the collection, ng-repeat will know that all other items already have DOM elements and it will not rerender them. Since AngularJS automatically gives an index to an array, it's considered a minor code change. The advantage of using tracking by $index is also that every element gets its unique key by which error introduced by duplicate keys(which can be assigned manually) is avoided.

Bypassing watchers in an application's code is essential, because with every watcher, the AngularJS digest cycle gets triggered, thus extending execution time. Similar effect occurs when using one-time binding, because every time the value is changed in a controller, the digest cycle is triggered and the DOM is supplied with a new value. By using one-time binding syntax digest cycle is triggered only once, and that's when a page is loaded for the first time or refreshed in browser. The number of digest cycles was decreased significantly [7].

IV. RESULTS

Results were conducted on a simple battery application used as a part of a larger home automation system. This application is a truthful representation of an application which uses all main AngularJS components such as filters, services, custom directives, controllers etc.

In first case the fully functional initial version of application is tested. In this version there was no optimization and the application was not tested during development.

The second application was developed after applying the advised optimizations.

Optimizing applications is used to show how testing tools react to applications that load faster and how the speed of loading DOM is connected with testing time.

A. *Execution time of unit tests*

Unit tests are conducted as mentioned with Karma testing tool. With each test, number of elements in array was increased. The application was tested with optimized ng-repeat directive using $index, one-time binding and unoptimized application using normal directive. This difference grows as the number of elements grows, ending with difference 0,303s in average on 1000 elements in ng-repeat (Fig 4.).

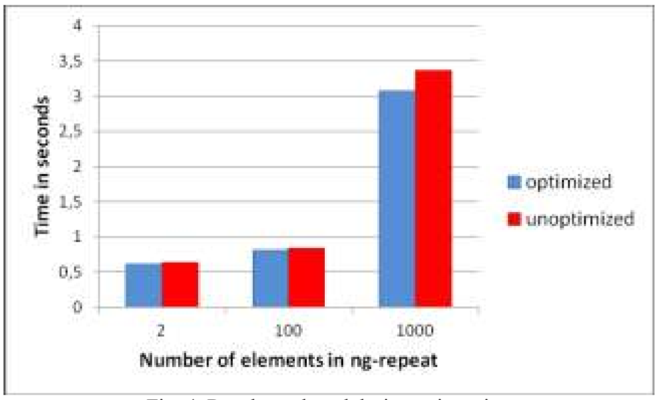

Fig. 4. Results gathered during unit testing

Karma as a testing tool also adds a feature of code coverage where user can see the percentage of a tested application's executed code. Before code coverage can be calculated, part of the code must be instrumented (copied in separate folder as a preparation for tracing) and compared with the code being tested. Code coverage is a great feature when scalability and size of an application is important. It also enables developer to remove unused lines of code.

Results gathered during unit testing coverage show no difference between versions of applications (optimized and unoptimized). Results only differ when application has to process more requests which are inserted in ng-repeat (Fig 5.). This is due to more elements requiring more branches to process data (images, level) inside a custom directive.

B. *Execution time of end-to-end tests*

Protractor as a tool gives us an insight into the whole application and its functionality. For this as mentioned must run a separate Selenium Webdriver and server for application. As a result execution of tests is much slower, then in case of unit tests. The execution also depends on

time needed for application loading since Protractor waits for application to fully load, before it executes a test. Protractor's flaw is that it does not provide code coverage, since all HTML code is loaded in the browser and there is no way to instrument code from scripts since it cannot access the DOM.

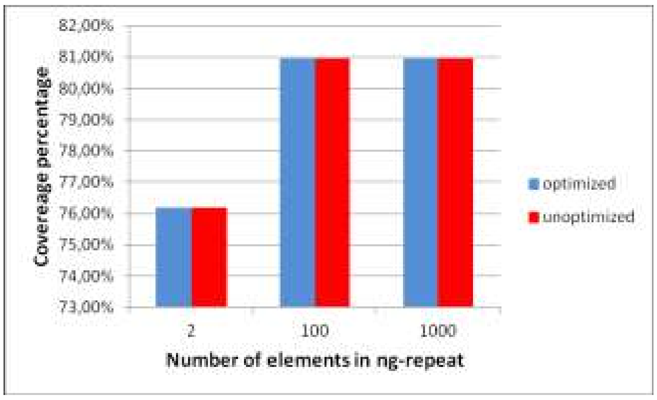

Fig. 5. Results during coverage testing

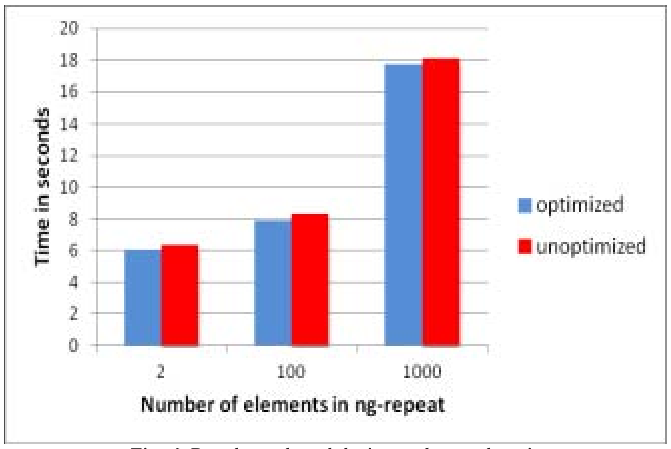

Fig. 6. Results gathered during end-to-end testing

C. *Comparison of available tool options*

Before any testing of an application is conducted, it's advised to reconsider all options of the tools, as well as what each of testing strategies offer.

Unit testing, conducted with Karma is considered more agile. Debugging unit test is easy, with the use of breakpoints. Karma offers code coverage as well as an integrated Webdriver. Most of browsers support unit tests and offer autowatch functions to observe source code changes (Table 1.). Flip side is that unit tests cannot be ran in multiple browsers simultaneously. Additionally, this unit tests cannot test HTTP GET and HTTP POST requests to the server, but these can be emulated using ngMock module. It provides support to inject and mock Angular services into unit tests.

TABLE 1
SUPPORTED BROWSERS FOR RUNNING TESTS

	Karma	Protractor
Chrome	yes	yes
Firefox	yes	yes
Safari	yes	yes
PhantomJS	yes	no
Opera	yes	no
IE	yes	no
SauceLabs	yes	yes
Browserstack	yes	no
iOS	no	no

End-to-end testing is much less flexible and gives fewer opportunities to observe an application fault than a unit test (Table 2.). It is currently not supported by all browsers, but as an advantage it can be run simultaneously in more browsers. Although it doesn't have an integrated autowatch function, autowatch can be integrated in Javascript task runner called Grunt which can autowatch instead of the Protractor tool. End-to-end tests also require separate running instances of Selenium Webdriver during testing. Debugging of a test can be quite complicated because of long error messages and the separation between the browser and the process running the test. One solution it is capture screenshots of the browser during tests to ease debugging. Another solution advices usage of functions such as browser.pause() which pause execution of code and attach debugger in one point in control flow. The downside of browser.pause() is that is not tested enough, and user must be cautious of errors that might appear.

TABLE 2
OPTIONS COMPARISON

	Debbuging	Webdriver	Coverage
Karma	less difficult	integrated	yes
Protractor	difficult	Selenium WebDriver	no

V. CONCLUSION

Testing of applications is a key component to creating a maintainable and reusable code. Using descriptive frameworks like Jasmine and Mocha to write a test is a good start for understanding functionality and writing documentation. Untested software creates problems such as: complicated DSL structures, duplicated code, and unreadable code. Advantages of tests are multiple from easily testing across more browsers, to no need to constantly refresh pages and use primitive debugging techniques.

Before performing testing on an application, it is advisable to write down all expected functionalities, as well as user-specific requirements. This can later help to write more efficient tests. Depending on how large the application's module is, it is always advisable to start from unit tests. These are good indicators of how well internal structures of your application are working and show whether or not the application satisfies main functionality. As seen from results (Fig 4. and Fig 5.), unit tests can be a good indicator of how well is the application is optimized, as well as provide code coverage. This can be essential to writing smaller and better modules of an application. Testing across many browsers is also a plus.

On the other hand, end-to-end test give an overall picture of application. They are not so crucial in determining application efficiency. But they are important in testing a final product, the way application with all connected modules and units works. Their main advantage is that they can help integrate more modules tested with unit tests. This way application can be fully functional and optimized regardless of its size and complexity.

ACKNOWLEDGMENT

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, under grant number TR32014.

REFERENCES

[1] Comparison of Javascript application frameworks, http://noeticforce.com/best-Javascript-frameworks-for-single-page-modern-web-applications, accessed March 2016
[2] A.Lerner, *Ng-book: The Complete Book on AngularJS*." ISBN 978-0-9913446-0-4 AngularJS Documents, Fullstack. 2013. https://docs.angularjs.org/guide/unit-testing accessed March 2016.
[3] Karma testing tool documents and configuration, https://karma-runner.github.io/0.8/config/browsers.html, accessed March 2016.
[4] AngularJS documents, https://docs.angularjs.org/guide/e2e-testing, accessed April 2016.
[5] S. Bailey "*AngularJS Testing Cookbook*", , ISBN 978-1-78398-374-2, Packt Publishing, 2015.
[6] Optimizing application using ngRepeat directive https://docs.angularjs.org/api/ng/directive/ngRepeat#tracking-and-duplicates
[7] Optimizing a Lange AngularJS application, Ng-Conf https://docs.google.com/presentation/d/15XgHRI8Ng2MXKZqglzP3PugWFZmIDKOnlAXDGZW2Djg/ accessed March 2016
[8] Selenium – Web Browser Automation, http://www.seleniumhq.org, accessed in March 2016.