# GuideAutomator: Automated User Manual Generation with Markdown

**Allan dos Santos Oliveira[1], Rodrigo Souza[1]**

[1] Department of Computer Science – Federal University of Bahia (UFBA) – Salvador – BA – Brazil

allanoliver@dcc.ufba.br, rodrigo@dcc.ufba.br

***Abstract.*** *User manual, also known as user guide, is a technical document for communication designed to assist end users of a product. It aims at filling the gap between what is easily deductible and what is not. A complicating factor for those who write these documents is to keep them up-to-date with changes on the application over time, like addition/removal of features or just changes on user interfaces (documented as screenshots). This work aims at assisting writers of such documents, for web applications only, providing automated screen capture, reducing maintenance cost and user manual inconsistency. We provide GuideAutomator, which enables this automatization through writing of documents under Markdown syntax and encapsulation of Selenium Web Driver.*

https://youtu.be/zXZyNgJOgdY

## 1. Introduction

User experience has been a rising concern in the software industry, due to its decisive role on user engagement and consequently on software's success. A good user experience design enhances user satisfaction by effectively addressing the needs and circumstances of its users, which is done mainly by providing carefully considered user interfaces.

Even though software user interfaces are getting easier to use, applications still require user manuals for several reasons. Some notorious reasons are: unexperienced users, first time users may face difficulties using the application; domains policy, as some domains formally require user manuals to support its users; support critical decisions, some critical actions like delete resources may rise doubts; reveal full potential of an application, some functionalities might not be explicitly visible nor easily deductible.

Therefore, the user manual is an important part of software documentation. However, development teams often ignore it, primarily because of painful standard tools, such as popular text editors like Microsoft Word[1] and LibreOffice Writer[2]. Those are mentioned as painful because they slow down development speed, make versioning challenging to manage due to binary files, and make it harder to keep software documentation synced with the software's growth, especially as these documents contain many screen capture images.

[1] Microsoft Word. Retrieved July 30 from https://products.office.com/en/word
[2] LibreOffice Writer. Retrieved July 30 from https://www.libreoffice.org/discover/writer/

To address these issues, this work assists designers of user manuals for web applications by providing GuideAutomator, a user manual generation tool with automated screen capture implemented upon Selenium Web Driver[3] (a browser automation framework), to be used with Markdown[4] (a lightweight markup language), which can be easily versioned to improve documentation maintainability.

All user manuals aim at providing clear information to their users about how to use an application. In order to accomplish that, there are some good practices for designing such documents (Hodgson, 2007). These practices may include use of summary, instructions on how to use main functionalities, troubleshooting section, glossary, etc. The most important parts, which are the instructions, uses several techniques to facilitate user assistance, like providing a systematic sequence to accomplish a certain task. To support that an effective approach is to make use of screen capture.

Screen captures or screenshots, under this context, are images taken from an application to reproduce either full or part of a user interface in a given application state. It reduces reasoning time for users to find or execute a function and this is the desired scenario to all stakeholders. GuideAutomator comes in to bring automation to this process of screen capture, alongside with Markdown writing as described in the next section.

The remainder of this paper is divided into 5 sections. In section 2 we go through GuideAutomator's description and operation. Section 3 presents the related work. Section 4 draws the conclusion and future work. Finally, Section 5 lists the references for this work.

## 2. An automation tool

GuideAutomator is a command line Node.js application, released under MIT license, which takes as input a mix of a widely used lightweight markup language, Markdown, with GuideAutomator's API for performing actions on web browsers. Our API is a wrapper for a browser automation tool, Selenium Web Driver. Figure 1 presents the application workflow. Essentially, a user manual writer composes the document with Markdown's rich text as in any common Markdown document. However, to reproduce user steps and capture the outcome stages of these steps, the writer must also include our API commands into the Markdown text. Our application processes the API commands and the user manual can be then outputted to PDF and HTML formats.

GuideAutomator runs on Windows, Linux and Mac OS platforms, as long as it has Node.js installed. Currently, it only supports Chrome Browser with its respective web driver, ChromeDriver, as discussed on the Selenium Web Driver subsection. The application is available at NPM[5], the default package manager for Node.js, under the

---

[3] Selenium WebDriver. Retrieved April 17, 2016 from http://www.seleniumhq.org/projects/webdriver

[4] Gruber, John. Markdown Syntax Documentation. Retrieved April 17, 2016 from https://daringfireball.net/projects/markdown/syntax

[5] npm, Inc. (2016). guide-automator. Retrieved May 22 from https://www.npmjs.com/package/guide-automator

name of "guide-automator" and runs as the example shown on Figure 2. The application takes two parameters: input file path (Markdown mixed with GuideAutomator's API) and output folder path (for placing generated documents).

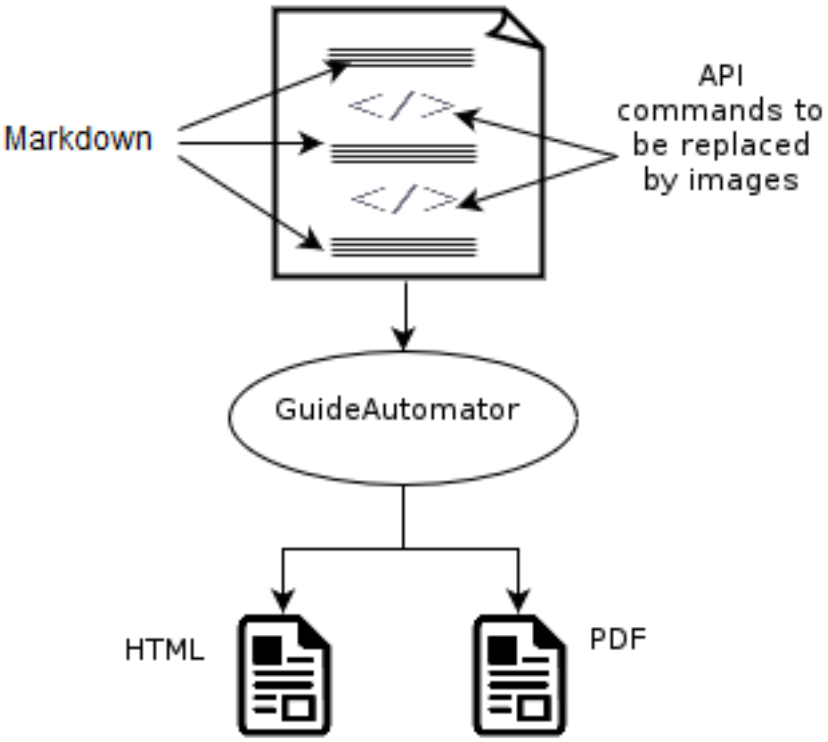


**Figure 1. GuideAutomator's workflow.**

```
guide-automator docs\input.md docs
```

**Figure 2. Example for running GuideAutomator.**

### 2.1. **Markdown**

Markdown is a plain text formatting syntax created in 2004 by John Gruber, originally designed to be converted to valid XHTML or HTML. Over the last decade, many extensions were created making it possible to convert Markdown's language to many other formats, like PDF, LaTeX, RTF, etc. Alongside with all these possibilities, its ease of use has made Markdown very popular in the software community. Some popular applications that make use of Markdown are GitHub[6] and StackOverflow[7]. These applications and many others use either standard Markdown syntax or a custom extension of the original markup language.

---

[6] GitHub Inc. About writing and formatting on GitHub. Retrieved April 17, 2016 from https://help.github.com/articles/about-writing-and-formatting-on-github/

[7] Stack Overflow Inc. How do I format my posts using Markdown or HTML? Retrieved April 17, 2016 from http://stackoverflow.com/help/formatting

Markdown was intended to be simple, easy-to-read and easy-to-write. Its syntax is composed by regular text with a few non-alphabetic characters, which were chosen to look like what they mean. Some of the non-alphabetic characters are "#" (hash) and "*" (asterisk), which are used to style headers and to define unordered lists, respectively. A complete syntax definition is available at the official Markdown page by John Gruber.

### 2.2. **Selenium Web Driver**

To make automation possible, GuideAutomator encapsulates Selenium Web Driver, a powerful tool for browser automation, which makes it possible to drive a browser natively as a user would. It supports of some of the largest browser vendors, such as Firefox, Chrome, Safari and Internet Explorer. However, due to a few unsupported operations (take screenshot of viewport only and take screenshot of an element) for some webdrivers, GuideAutomator v1.0 guarantees support only to ChromeDriver 2.21[8]. As webdrivers expand their support to required operations, GuideAutomator will expand support to more web browsers.

Selenium Web Driver provides a rich API for manipulating a web browser. Major commands and operations are: fetching pages, locating UI (User Interface) elements, filling in forms and taking screenshots. GuideAutomator's API incorporates these and many other commands as described in the following subsection.

### 2.3. **API**

GuideAutomator provides an API for performing common behavior of web applications users, like clicking elements, filling in inputs, etc. Alongside these actions, GuideAutomator's API provides a command for taking screenshots, so these images can automatically be placed on the user manual, regardless of the output format.

For calling API's commands, the user manual writer must include on their Markdown file blocks of GuideAutomator's code, bounded by specific delimiters, defined as `<automator>` for block code beginning and `</automator>` for block code end, see example on Figure 3. Commands must be separated by special character ";" (semicolon), as shown in Figure 2. GuideAutomator can then compile those commands, and, in case of any failure, it outputs the incorrect token. Otherwise, the application proceeds with processing those commands. Table 1 shows the current list of available API commands for GuideAutomator v1.0.

Once all commands are executed, GuideAutomator gathers all generated images and places them on their appropriate location in the outcome document. Appropriate image locations on the outcome document are defined based on the location of their respective `<automator>` code block, i.e., the code block that encloses their respective `takeScreenshot` or `takeScreenshotOf` operations. This process can be repeated as much as needed to keep a user manual up-to-date with its respective application, as every time GuideAutomator is executed it generates all screen captures all

[8] Google Inc. ChromeDriver - WebDriver for Chrome. Retrieved April 17, 2016 from https://sites.google.com/a/chromium.org/chromedriver/

over again. Figure 3 and 4 illustrate, respectively, the input file for logging into Yii Blog Demo[9] and the output HMTL page containing generated screenshots. Note that captured images are placed where their respective block commands were.

| Command | Description |
|---|---|
| get(url) | Navigates to the page with the specified *url*. |
| takeScreenshot | Takes screenshot of the browser's viewport. |
| takeScreenshotOf(selector) | Takes screenshot of the browser's viewport after the element identified by the css *selector* has been scrolled into view. |
| fillIn(selector,content) | Types *content* on the element identified by the css *selector*. |
| submit(selector) | Submits the form containing the element identified by the css *selector*, if there is one. |
| click(selector) | Performs a click on the element identified by the css *selector*. |

**Table 1. GuideAutomator's API commands, version 1.0.0.**

input.md

```
# How to log into Yii Blog Demo
## First, access http://www.yiiframework.com/demos/blog/
<automator>
get('http://www.yiiframework.com/demos/blog/');
takeScreenshot;
</automator>

## Click on the Login tab
## Fill in the login form with valid username and password
## Submit the login form
<automator>
click('#mainmenu li:last-child > a');
fillIn('#LoginForm_username','demo');
fillIn('#LoginForm_password','demo');
takeScreenshotOf('#login-form');
</automator>

# Now, feel free to use the dashboard
<automator>
submit('#login-form');
takeScreenshot;
</automator>
```

**Figure 3. Simple input example for logging into Yii Blog Demo.**

[9] Yii Software LLC. Yii Blog Demo. Retrieved May 20, 2016 from http://www.yiiframework.com/demos/blog/index.php/post/index

**How to log into Yii Blog Demo**

**First, access http://www.yiiframework.com/demos/blog/**


**Click on the Login tab**

**Fill in the login form with valid username and password**

**Submit the login form**


**Now, feel free to use the dashboard**


**Figure 4. Generated web page from example in Figure 3.**

## 3. Related Work

GuideAutomator has its roots in approaches such as literate programming, automatic documentation generators, standardized markup languages, continuous integration, and functional testing. We comment related work on these topics below.

Waits and Yankel (2014) claim that standard documentation tools and processes, based on the collaborative writing of documents using file formats that are binary and proprietary, such as Microsoft Word documents, are not well integrated into software development tools and processes. The main problems of the current approach are the difficulty of merging contributions of different authors, due to limitations of version control systems, and the inconsistency of document presentation across operating systems. They propose writing documentation in plain text files, using a standardized markup language such as Markdown, and then convert it into formats such as PDF and HTML using a tool such as Pandoc[10]. GuideAutomator follows this approach and further automates the process by generating images from user interface scripts interleaved with text in the documentation.

Literate programming is an approach to programming in which the source code is interleaved with text explaining the internals of a program (Knuth, 1984). The focus is on the documentation, and the source code is presented in parts following the structure of the documentation. This document can be transformed either into pure source code, which can be compiled and executed, or into documentation. The approach is currently being used in reproducible research (Madeyski, 2015). Our tool is also based on documentation interleaved with source code. In our case, however, the text is not intended to explain the source code; instead, it explains the system to end users, and the source code is used to generate images that help explain the text.

API documentation generators are tools that extract structured comments in the source code of a program and generate the documentation of its application programming interface (API), aimed at programmers. API documentation generators include Doxygen[11] and Javadoc[12]. GuideAutomator is also an automated documentation generator; however, the documentation is geared towards end users, not programmers.

Continuous integration is the practice of checking in developers' source code changes to a shared repository frequently (Fowler, 2006). Following a check-in, an automated build process compiles the source code and possibly perform other automated tasks, such as running unit tests, performing static analysis, and even deploying the application to servers (in this case, the approach is known as continuous deployment). Because GuideAutomator automates the task of generating user guides containing screen captures, it can be included in the continuous integration cycle and enable the creation and publication of documentation that is in sync with the latest source code change.

---

[10] John MacFarlane. Pandoc: a universal document converter. Retrieved May 16, 2016 from http://pandoc.org

[11] Dimitri van Heesch. Doxygen. Retrieved May 16, 2016 from http://www.doxygen.org

[12] Oracle Corporation. Javadoc. Retrieved May 16, 2016 from http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

Functional testing is the process checking whether the behavior of a software application conforms to its specification, from the end-user point of view, by executing test cases derived from its specification (Myers et al., 2011). For web applications, functional testing can be automated using the framework Selenium (Holmes and Kellogg, 2006). GuideAutomator uses Selenium for a different purpose; instead of comparing the application's output with a specification, it captures the output so it can be displayed in a user manual.

## 4. Conclusion

Development teams often ignore user documentation as a direct result of the pain standard documentation tools and processes cause (Waits, 2014). To overcome this issue, GuideAutomator comes in as an easy to write and maintain automation tool for generating user manuals. Its ease of use and maintenance is due to a high level API for driving a web browser and taking screenshots, and to its plain text syntax that can be even versioned alongside the main project source code.

Screen capture automation dramatically reduces writers' effort. When the application changes, one will not need to update every single image in the user manual as those images will be captured once GuideAutomator is executed to represent the current application state.

As future work, we intend to expand support for more web browsers and create more API commands for browser manipulation. In addition, to provide a better experience to customers, it is our intention to expand options for taking screenshot to allow, for instance, image customization like highlights, borders, resizing, etc. Because we are on the early stages of development, there have been no evaluation of GuideAutomator yet. Therefore, evaluation on real projects is also on the roadmap, so we will be able to validate GuideAutomator's benefits in practical usage.

## 5. References


Hodgson, P. (2007). Tips for writing user manuals. Retrieved April 17, 2016 from http://www.userfocus.co.uk/articles/usermanuals.html

Waits, T. and Yankel, J. (2014). Continuos System and User Documentation Integration.

Knuth, D. (1984). Literate Programming. In *The Computer Journal*, Vol. 27, No. 2, 97-111.

Madeyski, L. and Kitchenham, B.A. (2015). Reproducible Research–What, Why and How. *Wroclaw University of Technology, PRE W, 8*.

Fowler, M. (2006). Continuous Integration. Retrieved May 16, 2016 from http://martinfowler.com/articles/continuousIntegration.html

Myers, G., Sandler, C. and Badgett, T. (2011). The Art of Software Testing, 3rd edition. Wiley, New York, NY.