**Proof of Concept**

**Recip: A Web-based remote-controlled interactive presentation system**

**report**



Chen

January 2018

# Sommaire

[1. Introduction 3](#_Toc43923807)

[2. Specification 4](#_Toc43923808)

[2.1. File 4](#_Toc43923809)

[2.2. Editor 4](#_Toc43923810)

[2.3. Controller 5](#_Toc43923811)

[2.4. Requirements and dependencies 5](#_Toc43923812)

[2.5. Constraints in terms of security, size, portability and quality 6](#_Toc43923813)

[2.5.1. Obsolete file formats 6](#_Toc43923814)

[3. Technical part 7](#_Toc43923815)

[3.1. Modeling and development 7](#_Toc43923816)

[3.1.1. Overview 7](#_Toc43923817)

[3.1.2. Fully dressed use cases and brief technical implementation description 7](#_Toc43923818)

[3.1.3. Directory structure 13](#_Toc43923819)

[3.1.4. .recip.html file structure 15](#_Toc43923820)

[3.1.5. Smartphone controller 16](#_Toc43923821)

[3.2. Tests 17](#_Toc43923822)

[4. Synthesis 20](#_Toc43923823)

[4.1. Final product 20](#_Toc43923824)

[4.2. Perspective and future features 20](#_Toc43923825)

[4.2.1. Editor 20](#_Toc43923826)

[4.2.2. Mobile application 20](#_Toc43923827)

[5. Conclusion 21](#_Toc43923828)

[Bibliography 22](#_Toc43923829)

[List of figures 22](#_Toc43923830)

[Annexes 23](#_Toc43923831)

[Annex 1 – Repository 23](#_Toc43923832)

[Annex 2 – Editor interface screenshot images 23](#_Toc43923833)

# Introduction

“Recip” is a free and open source **re**mote-**c**ontrolled **i**nteractive **p**resentation system in HTML5.

Although several presentation systems in HTML5 already exist, none of them is at the same time FLOSS (free and open-source software) and has full features (not just a framework like reveal.js, but with legit editor), and most of the time, their presentation system is not remotely controllable.

In brief, the project has a desktop editor application that can edit and generate a standalone HTML file as the presentation file, all resources are included in this sole file. This file can be executed by any modern browser like a normal HTML web page file. After opening the file, user can somehow connect the file on PC with a mobile web app (which could be written as an Android application) that can control the page navigation of the presentation file.

# Specification

The remotely controllable system “Recip” can be seen as having three parts: file, editor and controller. In this section, I will talk about the natures, functionalities and behaviors of these three parts, as well as the initial thought and ideas on them. For further technical description and detailed implementation, please read the “[Technical part](#_Technical_part)” section.

## File

Typical Recip presentation files have an extension of “.recip.html”, such file is a simple uncompressed but minified HTML file which contains all necessary resources (.html, .css, .js, images, videos, etc., all base64 encoded and/or embedded in the HTML file). it can be executed with any modern browser, and be viewed and edited using the editor.

The basic structure of the presentation file is based on Reveal.js (see section “[Requirements and dependencies](#_Requirements_and_dependencies)”).

Although it is minified on the code level, this uncompressed file’s size could be a little large if a lot of resources – especially video files – are included (please read section “[Constraints in terms of security, size, portability and quality](#_Constraints_in_terms)”).

## Editor

It is actually a web application that can run in browsers, but the production file is an Electron-compiled executable will also be available, at first for Windows but can also be easily made for other system in the future because of Electron’s cross-platform nature. (Electron: see section “[Requirements and dependencies](#_Requirements_and_dependencies)”)

This web application can create and edit a Recip presentation file, save the Recip presentation file as any Recip format, and open an existed file of any Recip format.

There is already an editor – slides.com – for Reveal.js, but it’s not free or open source. Recip has different positioning – it will be an open system for everyone and any purpose. Also, be aware of that Recip’s editor is not an open source copycat of slides.com, but offers different interface and features.

(Please see Annex 3 “Editor interface screenshot images” to better understand the description of the editor below)

Basic feature of the editor includes insertion of image, text, table, first and second-level titles (headings).

Slides of two different layout can be added or deleted, their order can be changed.

Several light and dark themes will be available to be chosen from.

There are many visual effects that user can choose from, which can be applied to animation of elements and transition of pages. There are some 3D effects, which differ from the classic PowerPoint effects.

In the editor, advanced user can customize individual element’s CSS style or global JavaScript, CSS style for the whole file, and HTML code for one slide.

User can open an existing .recip.html file, save the currently edited presentation file as a .recip.html, or abandon current work and create a whole new file. Presentation title can be changed.

All these above-mentioned functionalities are provided in a two-level menu bar on the top of the window.

User can navigate the thumbnail/overview of all slides in the sidebar on the right of the window, it’s clickable and scrollable.

The presentation slides are shown in the main window from the center to the left bottom. Every element in the slide is editable in the editor.

## Controller

One way to make remote control possible with the pure HTML presentation file is sending request to a remote server from Android application, the server will offer a file that stores the page number and modify the page number to meet mobile application’s request. This file is checked every 0.5 second by the browser that runs the HTML presentation file, and if the page number changes, the current viewed page should also be changed to the correct one according to the file on server.

The controller can control the page navigation (next page, last page, jump to a page).

## Requirements and dependencies

* Reveal.js, an open source (under MIT License), HTML presentation framework (GitHub homepage: <https://github.com/hakimel/reveal.js>).
* Electron, a tool to build cross platform desktop apps with JavaScript, HTML, and CSS (<https://electronjs.org/>).
* Other library, framework and environment include: jQuery (JavaScript library), Node.js (JavaScript runtime built on Chrome’s V8 JavaScript engine), QUnit (JavaScript unit testing framework), Electron Packager (to compile the web application as executable file), etc.
* Any IDE (WebStorm, NetBeans, etc.) and code editor (Sublime Text, Atom, etc.) could be used, as preferred by individual team members.
* Google Chrome will be the main browser to be used during the development and test.
* Android Studio will be used to develop the Android application (<https://developer.android.com/studio/>).
* Node.js is used on the server side, for the time being it is just a small program and does not require Express.js or other web frameworks.
* Git is used to control and manage the code.

## Constraints in terms of security, size, portability and quality

An executable editor’s size was estimated to be under 300MB. And actually, it is 135MB for the initial released production version, which is reasonable.

A “.recip.html” file (the currently used file format)’s size could be large because it’s uncompressed and all resources are base64 encoded and embedded directly in the HTML file. However, the files of this format are far slimmer than “.recip.exe” (mentioned below in the section “[Obsolete file formats](#_Obsolete_file_formats)”), and are very portable (only one simple HTML file) and compatible (runs in any modern browser), therefore, the file size won’t be a very disturbing issue.

### Obsolete file formats

Initially I planned to have three file formats, whose extension will be “.recip.html”, “.recip” and “.recip.exe” (in Windows) respectively. Apart from the “.recip.html” one that is being used, the other two being:

* “.recip”: a file of this format is a simple compressed file (using an open source compression method such as 7-zip) which contains all necessary resources, it’s not able to self-execute and must be executed with the editor/viewer. “.recip” files are smaller and more portable but user will have to install an editor/viewer to run such files.
* “.recip.exe”: a file of this format is a self-executable file (in Windows) which contains all necessary resources as an equivalent “.recip” file does, but also elements made with Electron that make the whole file self-executable. However, the file size could be large. A plain file’s size, or minimum size, is expected to be 60MB. Although such file is expected to be executable without installing anything (Recip software or browsers), its size is too large. Compared to “.recip.html” file format, “.recip.exe” doesn’t have too many advantages. The fact is, almost every OS has a modern browser thus can execute “.recip.html”.

These disadvantages made me to drop the usage of these two file formats.

Without these two format, I also reasonably dropped the viewer program (the editor is kept), because web browsers are enough for executing the .recip.html file and we don’t need such viewer.

# Technical part

## Modeling and development

### Overview

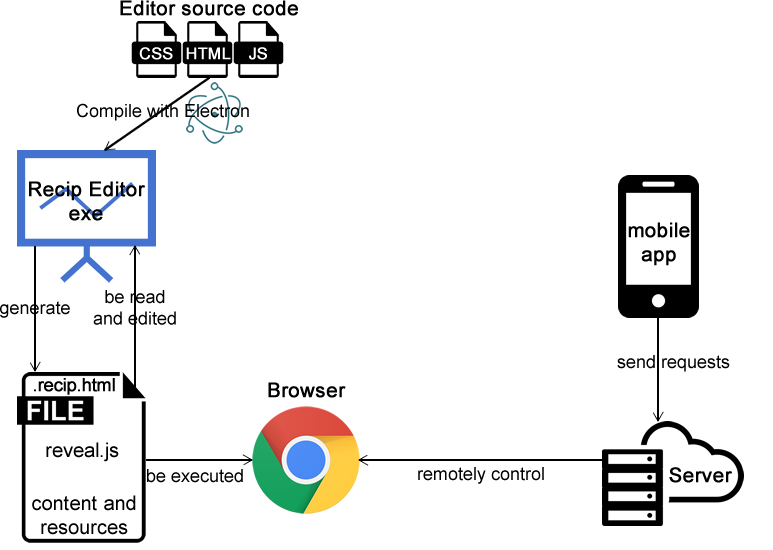


Figure 1 – Recip project overview

The Recip Editor is developed in HTML+CSS+JavaScript and is compiled to executable production file with GitHub’s Electron.

The editor can read, edit an existing .recip.html file and generate/update such file.

The .recip.html file includes necessary resources of reaveal.js framework as well as the content and all resources of the presentation.

The .recip.html file can be executed with modern browsers.

Smartphone web app sends request to the server, then the server sends a message to recip.html program, making it go to the next or the previous page. WebSocket technique is used here.

### Fully dressed use cases and brief technical implementation description

This section includes discussion on all event flows (what the user could do and what the program should treat this) for the editor in the fully dressed use cases tables:

(Please also see Annex 3 “Editor interface screenshot images” to better understand these use cases)

|  |  |  |
| --- | --- | --- |
| Use case name | Main |  |
| Actors | User |  |
| Preconditions | None |  |
| Postconditions |  |  |
| Flow of events | **Actor** | **System** |
| 1. User opens the program | 2. Open the program window |
|  | 3. Load default example presentation and initialize the top menu, putting it on “Add” |
|  | 4. Initialize the buttons |
|  | 5. Bind events |
|  | 6. Update the slide outline view and set the current slide as the first one |
|  | 7. generate a random recip ID for the example presentation |
| Special requirements |  |  |

|  |  |  |
| --- | --- | --- |
| Use case name | Top menu |  |
| Actors | User |  |
| Preconditions | None |  |
| Postconditions |  |  |
| Flow of events | **Actor** | **System** |
| 1. User clicks a top menu button | 2. Make all buttons not the current one (by removing a class name) |
|  | 3. Make all button’s sub-menus not the current one (by removing a class name) |
|  | 4. Make this button the current one (by adding a class name) |
|  | 5. Make this button’s sub-menu the current one (by adding a class name) |
| Special requirements |  |  |

|  |  |  |
| --- | --- | --- |
| Use case name | Slide outline view |  |
| Actors | User |  |
| Preconditions | None |  |
| Postconditions |  |  |
| Flow of events | **Actor** | **System** |
| 1. User clicks a non-current slide in the slide outline view | 2. Make this slide the current one in the slide outline view |
|  | 3. Make its corresponding slide in the main frame the current one |
| Special requirements |  |  |

|  |  |  |
| --- | --- | --- |
| Use case name | Add tab |  |
| Actors | User |  |
| Preconditions | None |  |
| Postconditions |  |  |
| Alternative flows | **Actor** | **System** |
| a1. User clicks the “Text” button | a2. In the current slide, if the last element is UL, then add <li>Text</li> to the last in it, otherwise add a new UL then add this LI |
| b1. User clicks the “Image” button | b2. Open dialog box, show image selection in the box |
| b3. User selects a local image file and clicks OK | b4. Get the file content, Base64 encode it and convert to URI format |
|  | b5. In the current slide, add IMG with the image’s Base64 URI as “src” attribute to the last |
| c1. User clicks the “Table” button | c2. Open dialog box, show ROW x COLUMN selection in the box |
| c3. User enters the ROW and the COLUMN, and clicks OK | c4. In the current slide, add TABLE element with correct row and column number to the last |
| d1. User clicks the “Title” button | d2. In the current slide, add H1 element to the last |
| e1. User clicks the “2nd LVL Title” button | e2. In the current slide, add H2 element to the last |
| f1. User clicks the “Delete last” button | f2. In the current slide, remove the last element if exists |
| Special requirements |  |  |

|  |  |  |
| --- | --- | --- |
| Use case name | File tab |  |
| Actors | User |  |
| Preconditions | None |  |
| Postconditions |  |  |
| Alternative flows | **Actor** | **System** |
| a1. User clicks the “Open file” button | a2. Open dialog box, show file selection in the box |
| a3. User selects a local .recip.html file and clicks OK | a4. Get the file content |
|  | a5. Use regular expression to match the slides wrapper element (div.slides), make it “contenteditable” then replace the whole current slides wrapper element with the new code |
|  | a6. Get the title from the new file’s content and update the current one |
|  | a7. Get the theme string from the new file’s content and update the current one in the stored data as well as the style element |
|  | a8. Update the slide outline view |
| b1. User clicks the “Save file” button | b2. Open dialog box, show current presentation title input in the box |
| b3. User enters a new presentation title (or not) and clicks OK | b4. Pop up a .recip.html file selection window |
| b5. User selects a file and clicks OK | b6. Insert current file title, transition string and the slides wrapper element (div.slides)’s outerHTML to a pre-prepared template (see “[.recip.html file structure](#_.recip.html_file_structure)” section and “[Annex 2 – #save-file script snippet](#_Annex_2_–)” for a detailed description), remove “contenteditable” attribute and save all these as the content of the target file |
| c1. User clicks the “File info” button | c2. Open dialog box, show file title input and Recip ID input (Recip ID is not editable) with the current file title in the box |
| c3. User enters a new file title (or not) and clicks OK | c4. Update the file title string in the stored data as well as the HTML TITLE element |
| d1. User clicks the “New file” button | d2. Replace the slides wrapper element (div.slides)’s outerHTML with a pre-prepared, clean one |
| Special requirements |  |  |

|  |  |  |
| --- | --- | --- |
| Use case name | Slide tab |  |
| Actors | User |  |
| Preconditions | None |  |
| Postconditions |  |  |
| Alternative flows | **Actor** | **System** |
| a1. User clicks the “New title slide” button | a2. In the slides wrapper element (div.slides), add a slide (section) with only an H1 element to the last |
|  | a3. Make the newly added slide the current one |
| b1. User clicks the “New content slide” button | b2. In the slides wrapper element (div.slides), add a slide (section) with an H2 element and a UL+LI elements to the last |
|  | b3. Make the newly added slide the current one |
| c1. User clicks the “Delete current” button | c2. Delete current slide |
|  | c3. Make the last slide (if exists) the current one |
| d1. User clicks the “Change order” button | d2. Open dialog box, show slide number selection in the box |
| d3. User select a number and clicks OK | d4. Move the current slide to the number as its index in its parent element |
|  | d5. Update the slide outline view |
| e1. User clicks the “Edit HTML” button | e2. Open dialog box, show HTML editing text area with the current slide’s HTML in the box |
| e3. User makes some modification (or not) to the HTML code then clicks OK | e4. Update the current slide’s HTML |
| Special requirements |  |  |

|  |  |  |
| --- | --- | --- |
| Use case name | Theme tab |  |
| Actors | User |  |
| Preconditions | None |  |
| Postconditions |  |  |
| Alternative flows | **Actor** | **System** |
| a1. User clicks the “Choose a theme” button | a2. Open dialog box, show theme choice in the box |
| a3. User chooses a theme then clicks OK | a4. Make it the current theme string in the stored data |
|  | a5. Update the theme style element |
| b1. User clicks the “Transition” button | b2. Open dialog box, show transition effect choice in the box |
| b3. User chooses a transition then clicks OK | b4. Make it the current transition string in the stored data |
| c1. User clicks the “Global script” button | c2. Open dialog box, show script editing text area with the #custom-script element’s innerHTML in the box |
| c3. User makes some modification (or not) to the code then clicks OK | c4. Update the #custom-script element’s innerHTML |
| d1. User clicks the “Global style” button | d2. Open dialog box, show script editing text area with the #custom-style element’s innerHTML in the box |
| d3. User makes some modification (or not) to the code then clicks OK | d4. Update the #custom-style element’s innerHTML |
| Special requirements |  |  |

|  |  |  |
| --- | --- | --- |
| Use case name | Current selected tab |  |
| Actors | User |  |
| Preconditions | None |  |
| Postconditions |  |  |
| Flow of events | **Actor** | **System** |
| 1. User clicks an element in the current slide |  |
| Alternative flows | 1a1. User clicks the “Edit style” button | 1a2. Open dialog box, show style editing text area with the currently selected element’s style attribute in the box |
| 1a3. User makes some modification (or not) to the code then clicks OK | 1a4. Update the currently selected element’s style attribute |
| 1b1. User clicks the “Delete element” button | 1b2. Remove the currently selected element from the current slide |
| Special requirements |  |  |

### Directory structure

Recip Editor’s full directory structure is shown below:

|  |  |
| --- | --- |
| RECIP  +---recip-editor  | | blank\_example.recip.html  | | icon64.ico  | | icon64.png  | | index.html  | | LICENSE  | | main.js  | | package.json  | | renderer.js  | |  | +---css  | | style.css  | |  | +---img  | | chart.png  | | content.png  | | css.png  | | delete.png  | | h1.png  | | h2.png  | | html.png  | | image.png  | | js.png  | | name.png  | | new.png  | | open.png  | | order.png  | | save.png  | | shape.png  | | table.png  | | text.png  | | theme.png  | | title.png  | | transition.png  | | video.png  | | | | +---js  | | jquery-3.2.1.min.js  | | script.js  | |  | +---min-resources  | | | head.min.js  | | | reveal.css  | | | reveal.js  | | |  | | \---theme-css  | | beige.css  | | black.css  | | blood.css  | | league.css  | | moon.css  | | night.css  | | serif.css  | | simple.css  | | sky.css  | | solarized.css  | | white.css  | |  | \---reveal (reveal.js 's source code structure is omitted here)  |  +---reveal (reveal.js 's source code structure is omitted here)  |  \---test  qunit-2.4.1.css  qunit-2.4.1.js  test.html  tests.js |

The recip-editor folder is the real working directory for the Editor.

The test and a backed-up reveal folders are outside the recip-editor working directory. The test is the directory where stores and runs JUnit tests, which I will mention below in the “[Tests](#_Tests)” section.

In the recip-editor working directory, there are several necessary files for Electron compilation: main.js (mainly for window initialization), package.json (configuration file).

LICENSE stores the MIT License content. There are also two icons and an example .recip.html file.

index.html is the main interface file, its CSS style file (which controls the appearance of the interface) is stored in the css folder and the main JavaScript file (which manipulates and determines the behaviors of the interface (such as button click event)) as well as a copy of jQuery library are in the js folder.

The img forder stores all the button icons for the editor interface.

index.html, css/style.css, js/script.js and the icons in the img folder are the main efforts.

min-resources folder stores all minified .js and .css files that are used during the content replacement process of the .recip.html file creation. See “[.recip.html file structure](#_.recip.html_file_structure)” section for details.

### .recip.html file structure

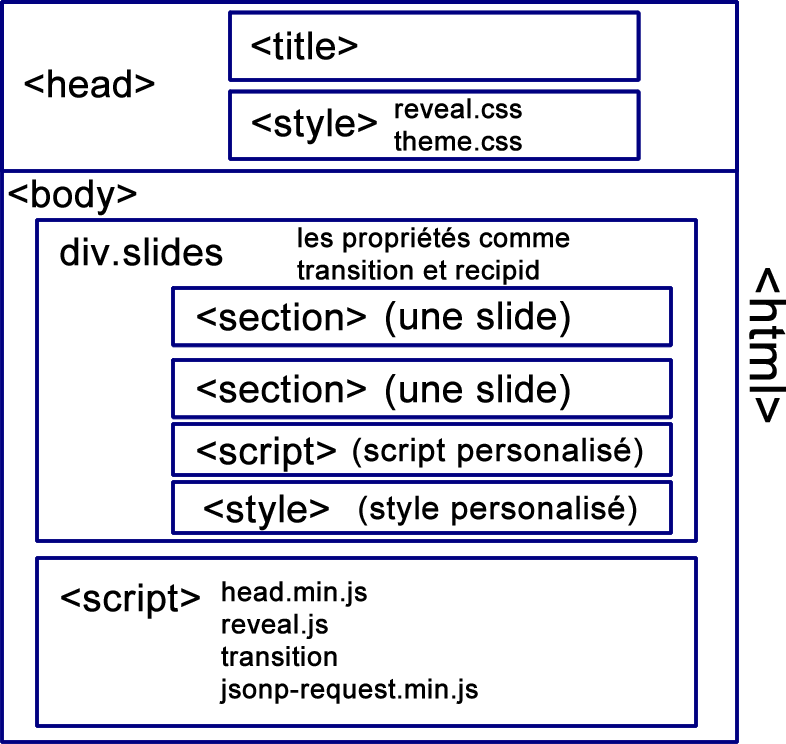


Figure 2 – .recip.html file structure

A .recip.html file has nothing special than a normal HTML file, its typical source code is shown below:

<!doctype html>

<**html**>

<**head**>

<**meta** charset="utf-8">

<**meta** name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">

<**title**>{{{title}}}</**title**>

<**style** type="text/css">

{{{reveal.css}}}

{{{currentThemeName.css}}}

</**style**>

</**head**>

<**body**>

<**div** class="reveal">

<**div** class="slides" contenteditable="true" data-transition="{{{transition}}}" data-title="{{{title}}}" data-recipid="{{{recipid}}}">

<**section** class="current-slide">

<**h1**>Blank Example</**h1**>

</**section**>

<**script** id="custom-script"></**script**>

<**style** id="custom-style" type="text/css"></**style**>

</**div**>

<**script**>

{{{head.**min**.js}}}

{{{reveal.js}}}

Reveal.initialize({

transition: "{{{transition}}}"

});

</**script**>

</**body**>

</**html**>

The “{{{ }}}” are to be replaced with the real values (title and transition name values) or file content (these minified .js and .css files are stored in the min-resources folder as mentioned in the “[Directory structure](#_Directory_structure)” section).

The “Annex 2 – #save-file script snippet” shows how the program replaces these “{{{ }}}” with correct values during the file save process.

The presentation content is stored in the div.slides element, one section element is one slide.

Although jQuery library is used in the editor, .recip.html file does not include the jQuery library, this reduces the file size.

A simple .recip.html file’s size is 110 KB. It is very portable if user doesn’t include too much videos and large images in it. That’s what is recommended: to use text instead.

Images and videos are to be encoded to base64 URI in pure text before being embedded to the .recip.html file (instead of using an external file). Below is an example code of an HTML img element which actually shows the image “”:

<**img** src="data:image/png;base64,">

### Smartphone controller

This idea has been described in the section “Specification” 🡪 “[Controller](#_Controller)”. Now I’m going to implement it.

Every .recip.html file has a random 10-letter-and-digit Recip ID, it can be viewed after clicking the “File info” button. It is expected to be used by server authentication. However, for the current, Proof-of-concept early-release version, a very simple server-side program will be set up without authentication, one sole user connection will be assumed.

WebSocket is the crucial technique that is used here.

The client browsers, where .recip.html file is loaded, is connected with the server using WebSocket, and is listening for a message from the server.

User uses smartphone to access a web app, which could send AJAX request to the server with a signal representing “next page” or “previous page”. Once the request is received, the server sends a message to let the recip.html presentation program in the client browser go to the next or the previous page.

The server-side program is powered by Node.js, but Express.js or other web frameworks are not used for the current simple version.

/\* ===== Server ===== \*/

...

**if** (req.method === 'POST') {

...

wsInstance.send(command);

...

}

...

**function** onSocketConnect(ws) {

clients.add(ws);

wsInstance = ws;

ws.onclose = **function**() {

clients.**delete**(ws);

wsInstance = null;

};

}

...

/\* ===== .recip.html ===== \*/

...

socket.onmessage = **function**(e) {

**if** (e.data === '1') {

Reveal.right();

} **else** **if** (e.data === '-1') {

Reveal.left();

}

}

...

## Tests

jQuery’s QUnit, a JavaScript unit testing framework has been used to write the tests for the Recip editor.

An example usage of QUnit code is shown below (part of the “Add buttons functional” test unit):

/\* ===== Add ===== \*/

QUnit.**test**("Add buttons functional", **function**(assert) {

$("#add-text").click();

assert.ok($(".slides section.current-slide").children().last().is("ul"), "Add text button is functional" );

$("#add-table").click();

$("#table-row-width").val("3");

$("#table-column-width").val("3");

$("#dialog-ok").click();

assert.ok($(".slides section.current-slide").children().last().is("table"), "Add table button is functional" );

});

Below is the table that includes all test units I’ve written as well as the detailed description of each assertion:

|  |  |  |
| --- | --- | --- |
| Category | Test unit name | Detailed description of the assertion |
| Top menu | Add button active | Test if 'Add' button has the "current-topmenubtn" class after page load |
| Test if the submenu of 'Add' has the "current-menu" class after page load |
| Top menu button functional | Simulate the click of the 'Theme' button in the first row, test if the sub menu of 'Theme' shows |
| File | File info button functional | Simulate the click of the file info button, enter a string then Click 'OK', test if that string becomes title |
| Outline view | Outline view slide click functional | Simulate the click of a slide in the outline view, test if it becomes the 'current-slide' |
| Simulate the click of a slide in the outline view, test if its corresponding slide in the main frame becomes the 'current-slide' |
| Add | Add buttons functional | Simulate the click of the Add text button, test if it adds text (ul+li) as the last element of the slide |
| Simulate the click of the Add table button, test if it adds a table as the last element of the slide |
| Simulate the click of the Add h1 button, test if it adds h1 as the last element of the slide |
| Simulate the click of the Add h2 button, test if it adds h2 as the last element of the slide |
| Simulate the click of the Delete last button, test if the last element becomes h1 |
| File | New file button functional | Simulate the click of the New file button, test if the slides number count becomes 1 |
| Theme | Theme change functional | Simulate the click of the Theme change button, choose 'sky' then 'OK', test if the theme becomes 'sky' |
| Choose transition button functional | Simulate the click of the Choose transition button, choose 'none' then 'OK', test if the transition becomes 'none' |
| Global script button functional | Simulate the click of the Global script button, enter a string then 'OK', test if the global script becomes that string |
| Global style button functional | Simulate the click of the Global style button, enter a string then 'OK', test if the global style becomes that string |

A screenshot of the successful QUnit tests result is shown below:

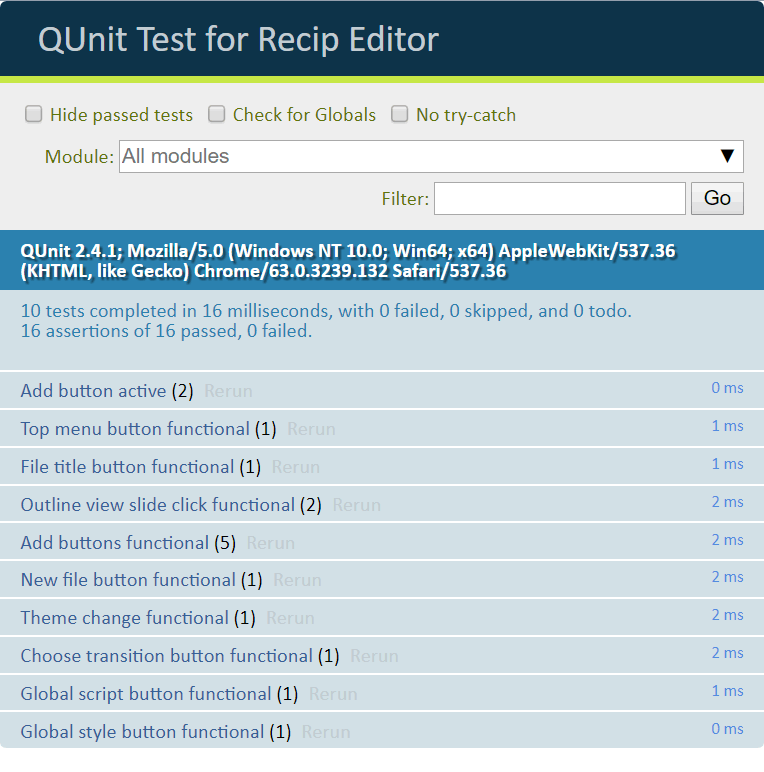


Figure 3 – QUnit tests result page

# Synthesis

## Final product

As I said, for this long-term project, the “final product” at this stage is actually an early release. I’m glad that most of the basic features are successfully implemented, although some plans were changed and some jobs were unable to be finished.

We now have a legit editor which meets most part of the specification. It has a simple interface, top menu bar and sub menu bar, it can open and save file, view file information (title, etc.), create new file, add text, image, table, first-level or second-level title, delete last or selected element or slide, create new slide of two different layouts, change slides' order, choose a theme and transition effect, edit global script, style and slide's HTML, edit selected element style, and has a scrollable and clickable “slide outline view”. (for further functionalities, behaviors and technical details, please read the section “[Fully dressed use cases and brief technical implementation description](#_Fully_dressed_use)” and the section “[Specification](#_Specification)”)

## Perspective and future features

There are some features that are not included in the initial release but could possibly include in the future.

### Editor

The video, basic shape (line, circle, rectangle, triangle, arrow) insertion is not done during the one-month project, but future implementation could be possible.

Table of content is another expected feature that could be automatically generated according to the headings at different levels of every page.

Interactive graphic for data visualization will be another feature. For example, in a clustered column chart, when the cursor is put on a graphical column, all necessaire data will be shown. During the presentation, the presenter could also modify the data to change the chart real-time.

### Mobile application

Future features of the controller - the mobile application - that can be reasonably imagined includes:

Future iPhone support, or even the support of a professional remote control such as Logitech Spotlight, will be possible.

The controller will be able to control the cursor (which serves as both the pointer and the text/graphical content selector (the cursor can hover over or click on such elements)) real-time.

The controller interface currently uses classic buttons. But it is also expected that gestures, or even position sensors (gyroscope and magnetometer) be used to measure the orientation of the smartphone, then pass the right order to the server or the presentation file on the computer to make the cursor move in the way the smartphone move. Also, the controller should be able to show the presentation content in the future.

# Conclusion

There seems to be no open-source editor to create standalone HTML files for presentation. I hope this project could inspire more programmers, and more users will use it and benefit from it.

# Bibliography

1. Android Studio User Guide <<https://developer.android.com/studio/intro/index.html>>
2. MDN web doc <<https://developer.mozilla.org/>>
3. jQuery API <<https://api.jquery.com/>>
4. Oleg Parashchenko: Online syntax highlighting for the masses! <<https://tohtml.com/html/>>
5. 1 Jun 2015, Shawn Rakowski: Getting Started With Standard Dialogs in Electron <<http://mylifeforthecode.com/getting-started-with-standard-dialogs-in-electron/>>
6. 10 April 2016: How to choose , read, save, delete or create a file with Electron Framework <<https://ourcodeworld.com/articles/read/106/how-to-choose-read-save-delete-or-create-a-file-with-electron-framework>>
7. electron-quick-start <<https://github.com/electron/electron-quick-start>>

# List of figures

[Figure 1 – Recip project overview 7](#_Toc43923834)

[Figure 2 – .recip.html file structure 15](#_Toc43923835)

[Figure 3 – QUnit tests result page 19](#_Toc43923836)

# Annexes

## Annex 1 – Repository

GitHub repository where all the source code and the Electron-compiled executable are hosted:

<https://github.com/tomchen/recip>

## 

## Annex 2 – Editor interface screenshot images

