idea2Life

Kepler

May 30, 2021

CONTENTS

1 idea2Life Getting Started Guide 3

2 idea2Life How to guides 11

3 idea2Life Developer Guides 35

4 idea2Life Deep dive 45

5 Tips and Troubleshooting 49

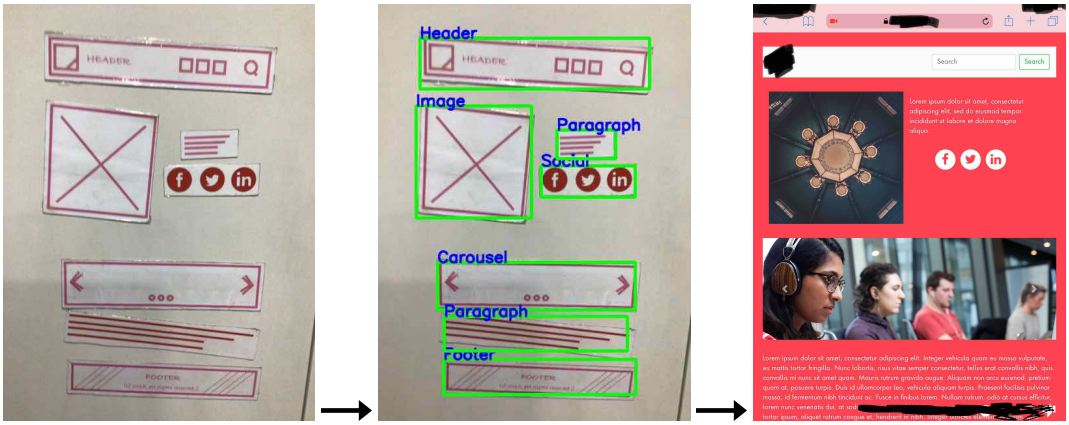
6 Reference 51

7 Indices and tables 57

Index 59

Prototyping a web application is more about ideas than mere sketching wireframes. The biggest barrier to effective prototyping is time and cost. idea2Life is an AI powered rapid prototyping to lower the barrier of prototyping.

With idea2Life, you can create fully functional static websites by just clicking a picture.



Knowing the document organization will help you find and use features effectively and quickly:

- **Getting Started** is guide for starting idea2Life service in shortest amount of time. Start here if you're new to idea2Life.
- **How-to guides** are recipes for idea2Life users. They guide you through the steps involved in addressing use-cases key issues.
- **Developer guides** are recipes for idea2Life developer, who want to contribute and extend idea2Life functionality.
- **Topic guides** discuss key topics and concepts at a fairly high level and provide useful background information and explanation
- **API reference** contain technical reference for APIs, mostly autogenerted from underlying code.

CHAPTER

ONE

IDEA2LIFE GETTING STARTED GUIDE

1.1 Installation

Install Using Docker

1. Download and Install Docker Desktop for Mac using this link docker-desktop. and for linux using this link docker-desktop on linux
2. Clone repo using this link idea2Life repo
3. Change your directory to your cloned repo.
4. Download the model file inside *ai/models*.
5. Open terminal and run following commands

```
cd <path-to-repo> //you need to be in your repo folder
docker-compose build
```

If you want to install idea2Life from source (without docker) Refer this section Install and use idea2Life from source (without docker).

1.2 Supported Hardware and operating system

idea2Life software is supported on the following host operating systems:

- Linux
- mac OS X

Minimum system configuration.:

- Processor: Dual core Processor
- RAM: 4GB of system memory
- Hard disk space: 10 GB

Recommended system configuration:

- Processor: Intel core i7 or higher
- RAM: 8GB of system memory
- Hard disk space: 30 GB

1.3 How to start or stop idea2Life

Start

1. Open terminal and run the following commands:

```
cd <path-to-repo> //you need to be in your repo folder

docker-compose up
```

Stop

2. Open terminal and run the following commands:

```
cd <path-to-repo> //you need to be in your repo folder

docker-compose down
```

1.4 How to use idea2Life

1. Download and print templates. Download link: Idea2life_templates_for_print preferably on magnetic sheets. Cut individual templates like this and paste on magnetic sheets.

Footer

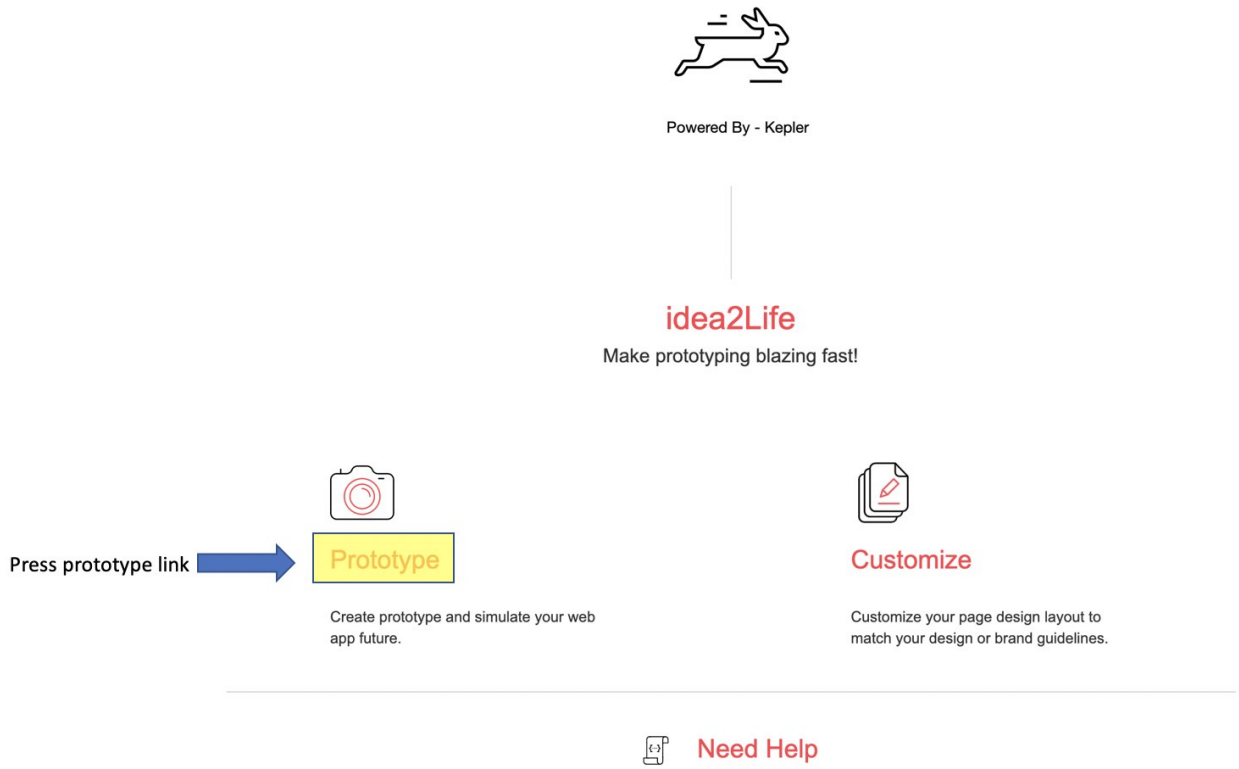
Video

2. Open the app on your tablet browser.:

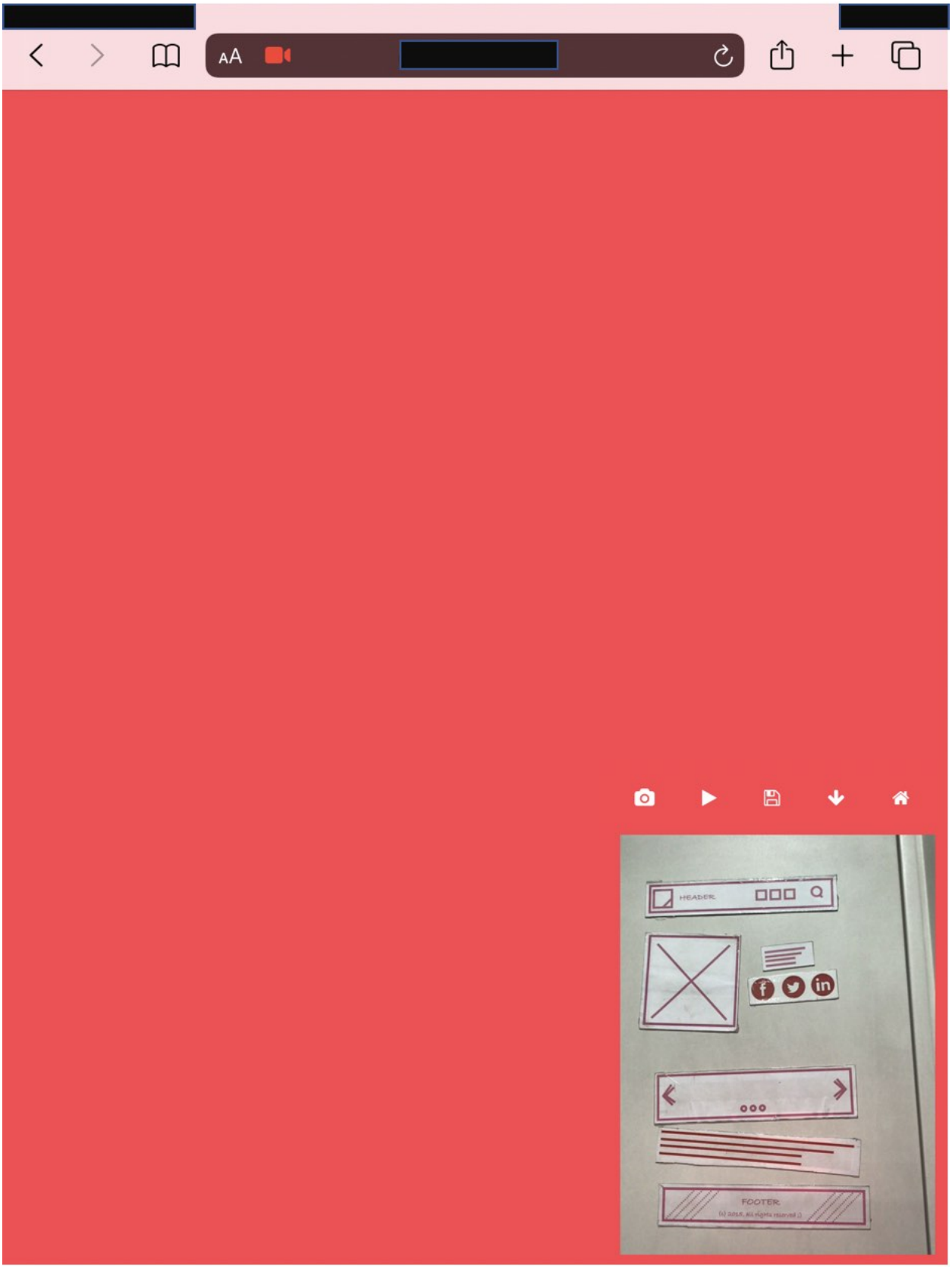
```
https://localhost:1813/
```

   you may have to replace **localhost** with your desktop/server **IP address**, Ensure idea2Life application and tablet is running in same network.

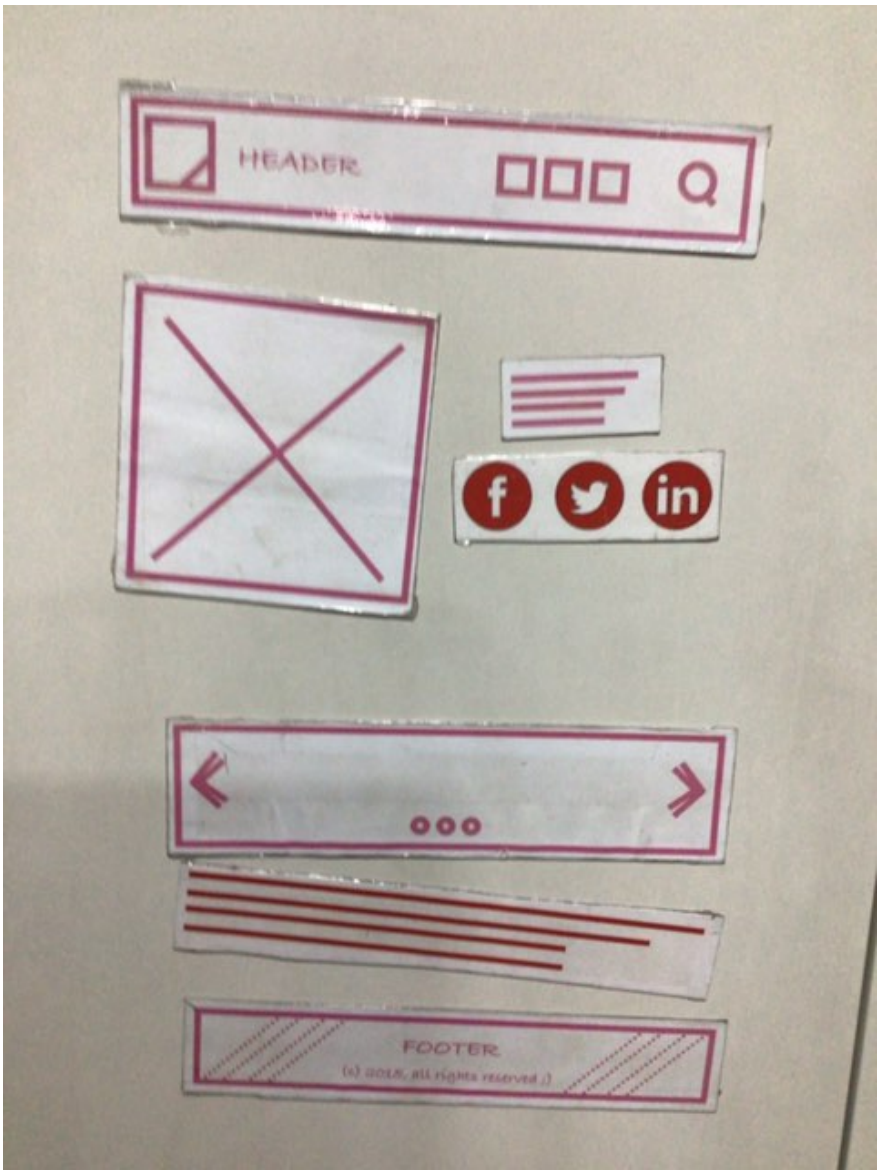
3. Ignore/accept any certificate warning/error in your browser.
4. Click on prototype/camera icon to begin prototyping.



5. You will get to idea2Life prototype page.

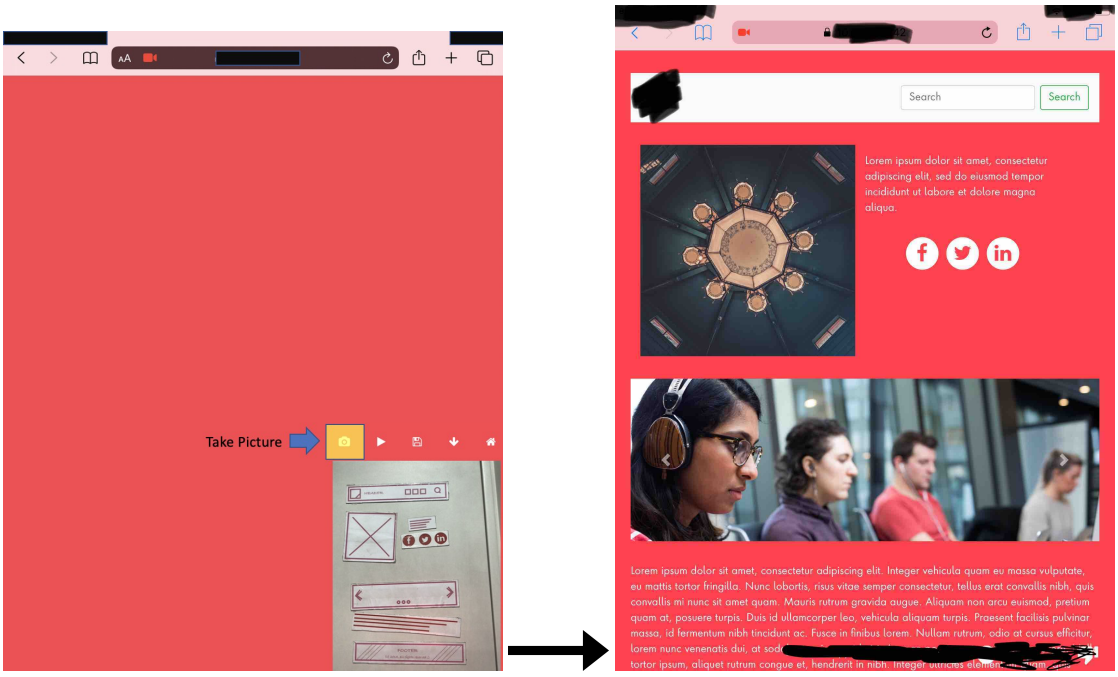


6. **Arrange printed templates** on table or metal surface according to your desired webpage design.

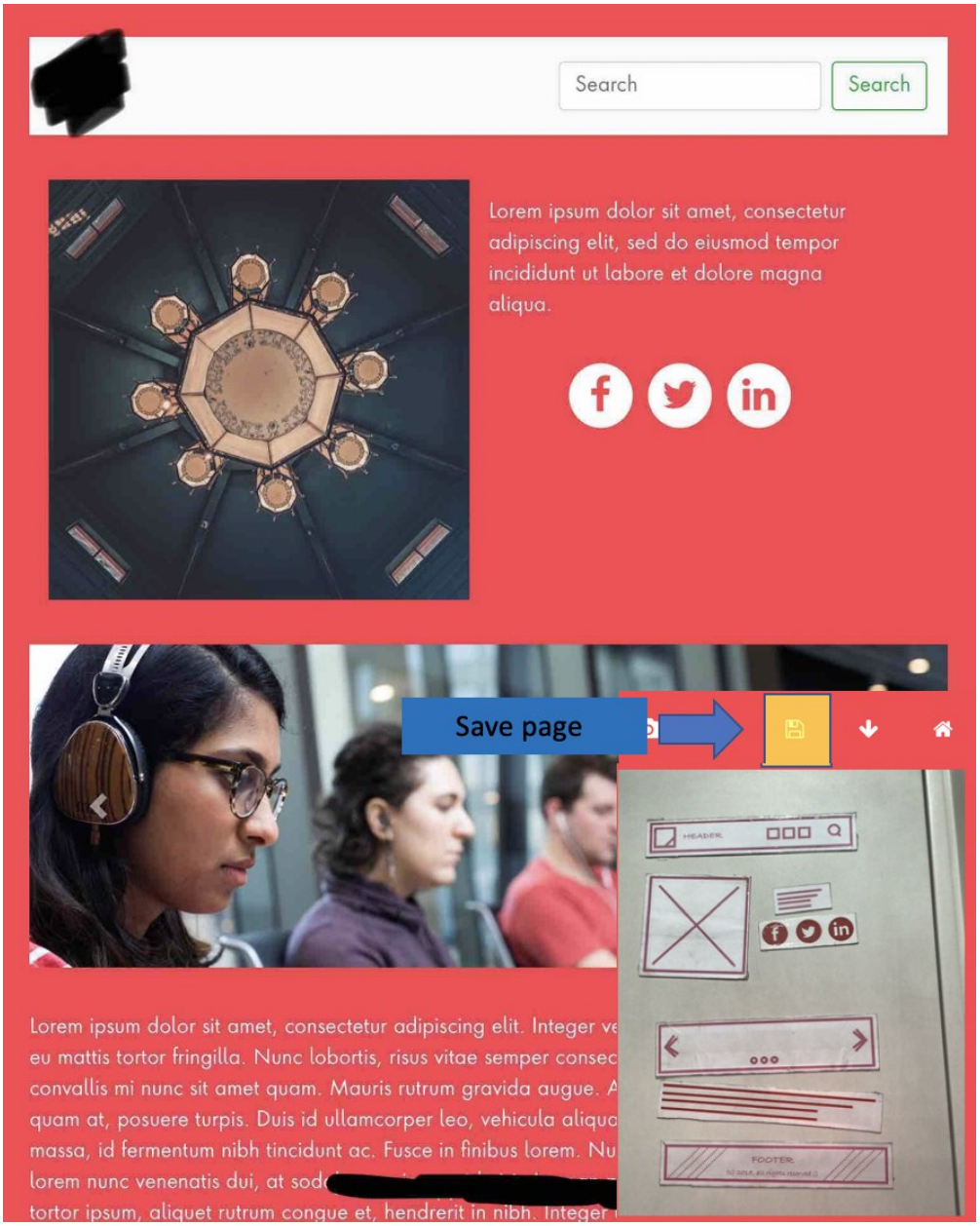


7. **Point camera** towards printed templates and **take picture** by clicking on camera icon in control panel.

8. idea2Life will automatically generate a new webpage based on laid out components.



9. Press third **save icon button** if you want to save currently generated template for *customization* later.



Take a look at demo video for more details:

10. Click on reset button/**play icon button** if you want to design new page.

Refer *idea2Life How to guides*. for advanced use cases.

CHAPTER

TWO

IDEA2LIFE HOW TO GUIDES

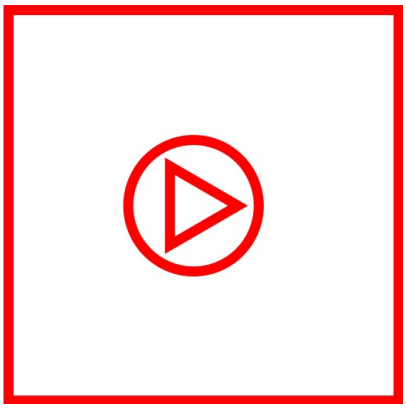
2.1 how to prototype using idea2Life

This guide describes a high level process of adding idea2Life to your brainstorming or ideation process. Imagine you have organized a brainstorming session to decide on the microsite you would like to build to promote the launch or the device.

Download and print templates beforehand. Download link: Idea2life_templates_for_print Cut individual templates like this and paste on magnetic sheets or a cardboard. Think of preparing a deck of components

Footer

Video



Start placing your components as you are brainstorming. for e.g. if the new phone has a very powerful camera One idea may to show a slide show of features at the top.

Below it, may be you decide to few videos shot by the phone.

Now that you have reasonably complete page in place, click the picture and generate a page. Check out *idea2Life Getting Started Guide* for more details.

Continue with the brainstorming, may someone suggest to enable social sharing and you place the social button on the page. May be you really want to focus on images and remove videos. You continue to assemble/reassemble components and generate pages at regular intervals.

After few options in place, you may want to filter it further. Save the choosen ones with some meaningful names. page default names are numbers and hence saving them with proper names helps in searching them easily.

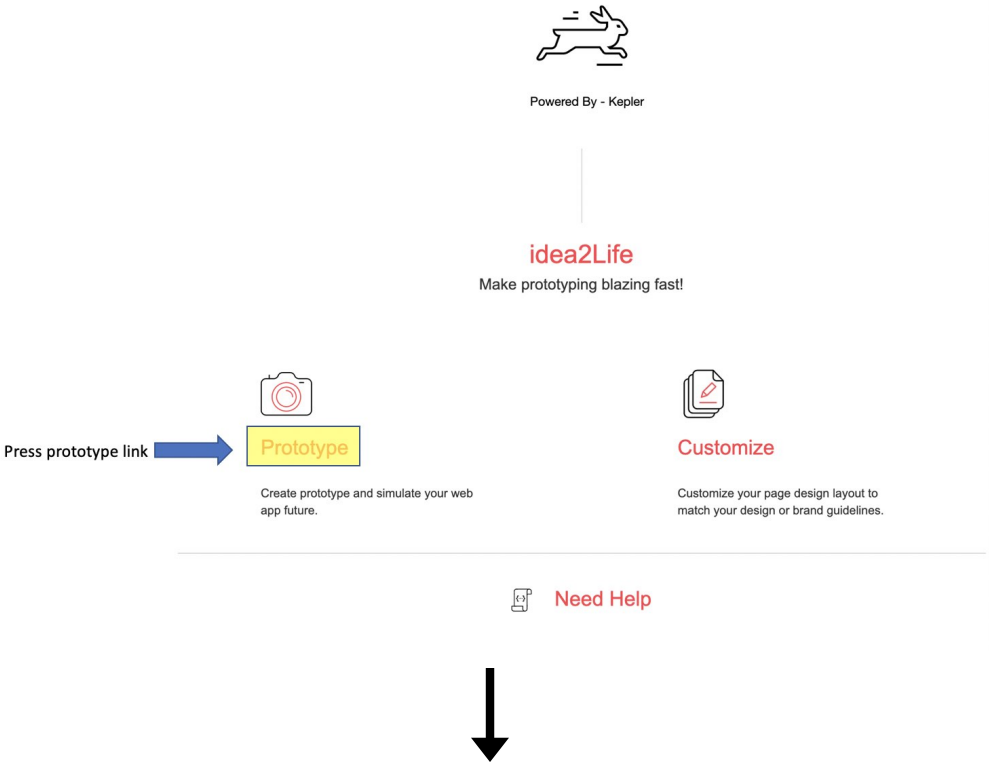
Go ahead and change the assets, styles or copy to make the chosen ones more real. Check out *How to customize generated pages* for more details. If you are brainstorming workflows then connected the final page. Check out *How to link generated pages* for more details.

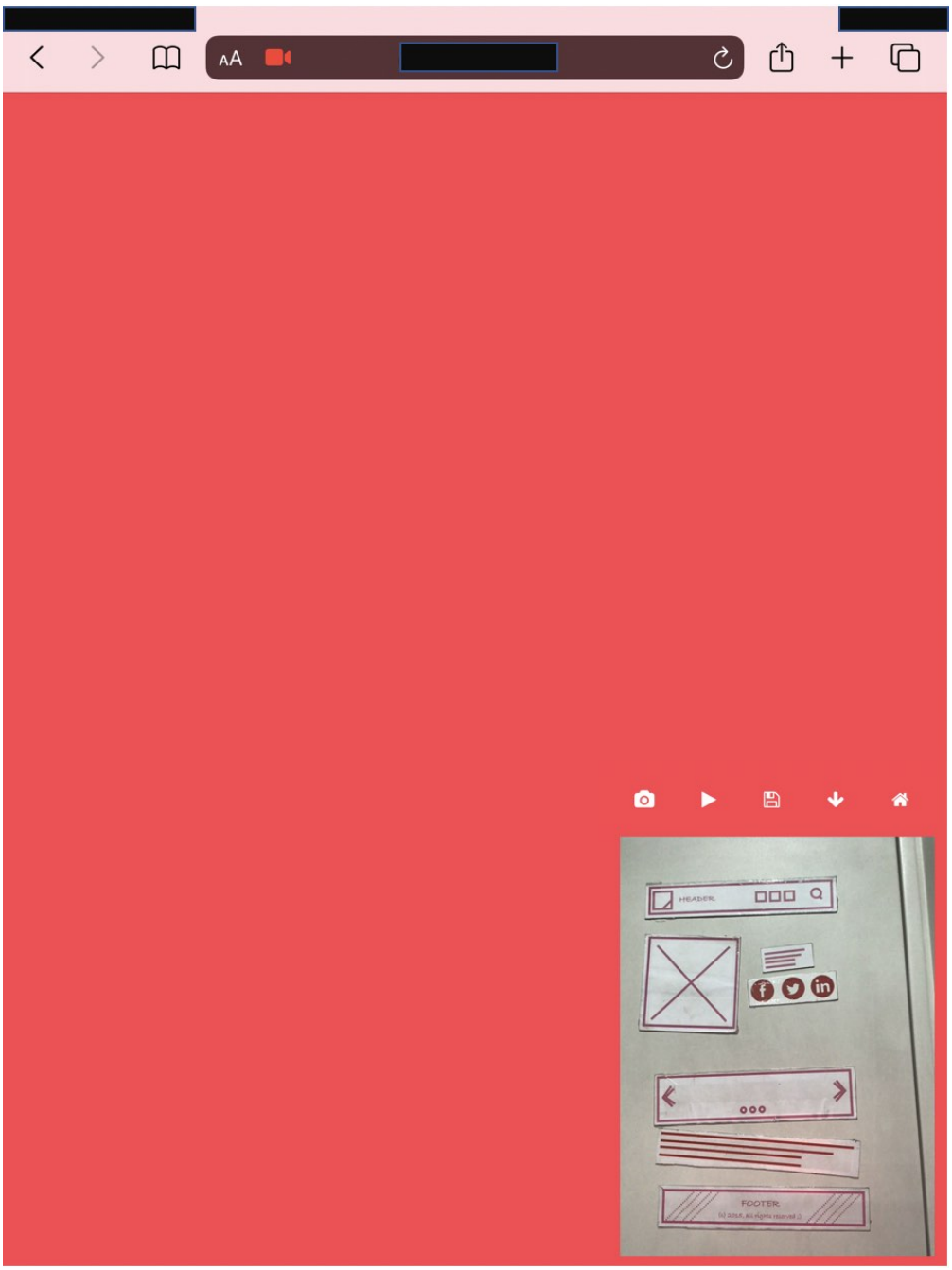
2.2 How to navigate idea2Life home page

Home page is the start point for using the tool. It has links to different functionalities offered by the tool

2.2.1 Prototype

click on the prototype link to begin Prototyping:

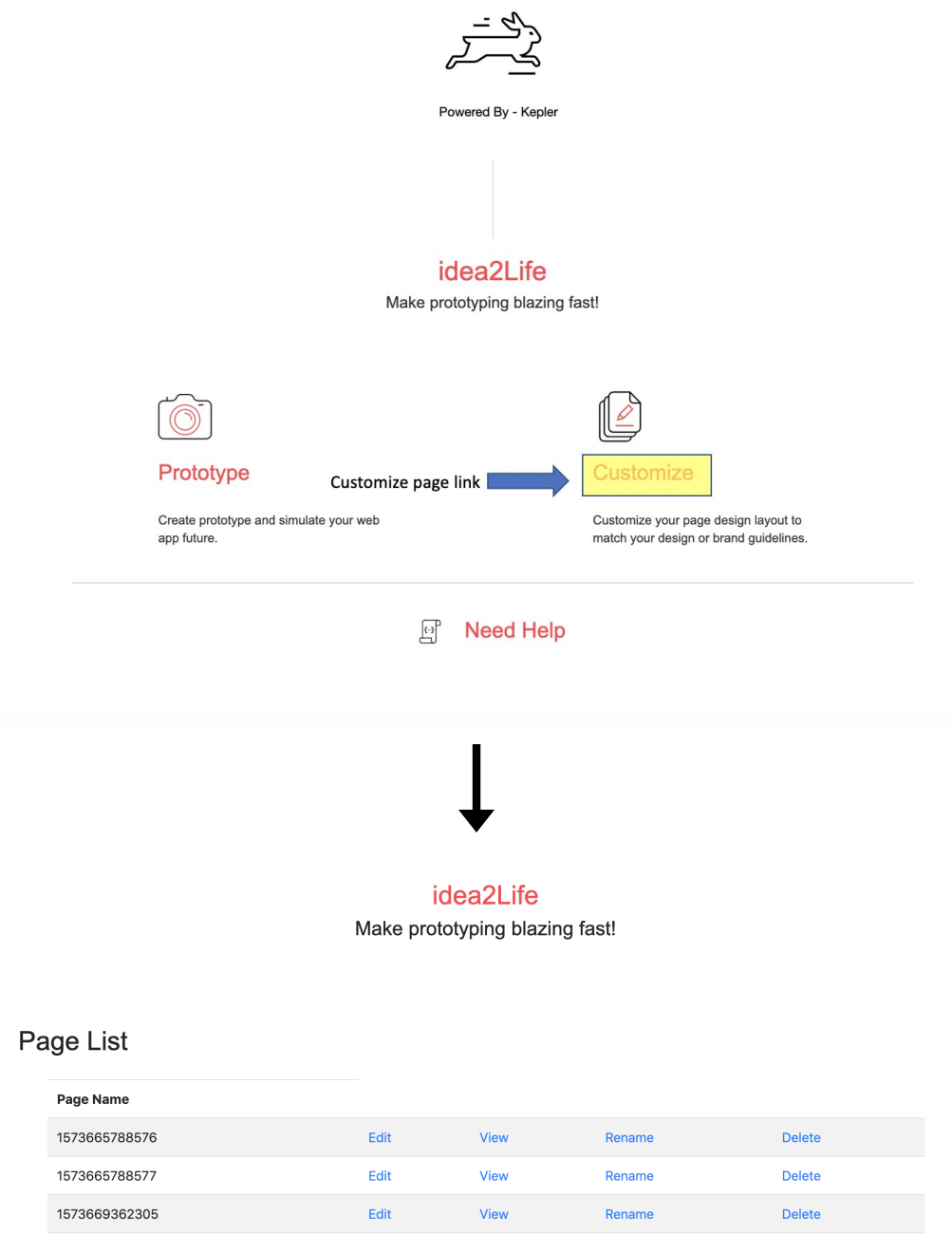




see *how to use prototype page* for more details.

2.2.2 Customize

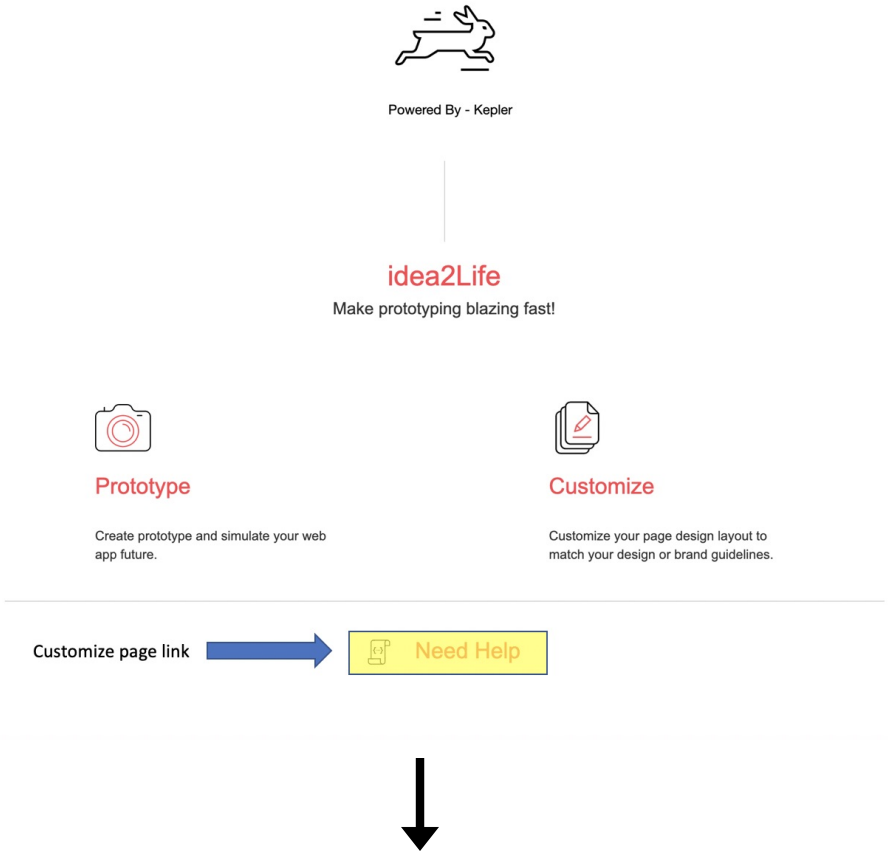
Click on “Customize” to list generated pages. The pages list can be used to view or edit the individual pages.

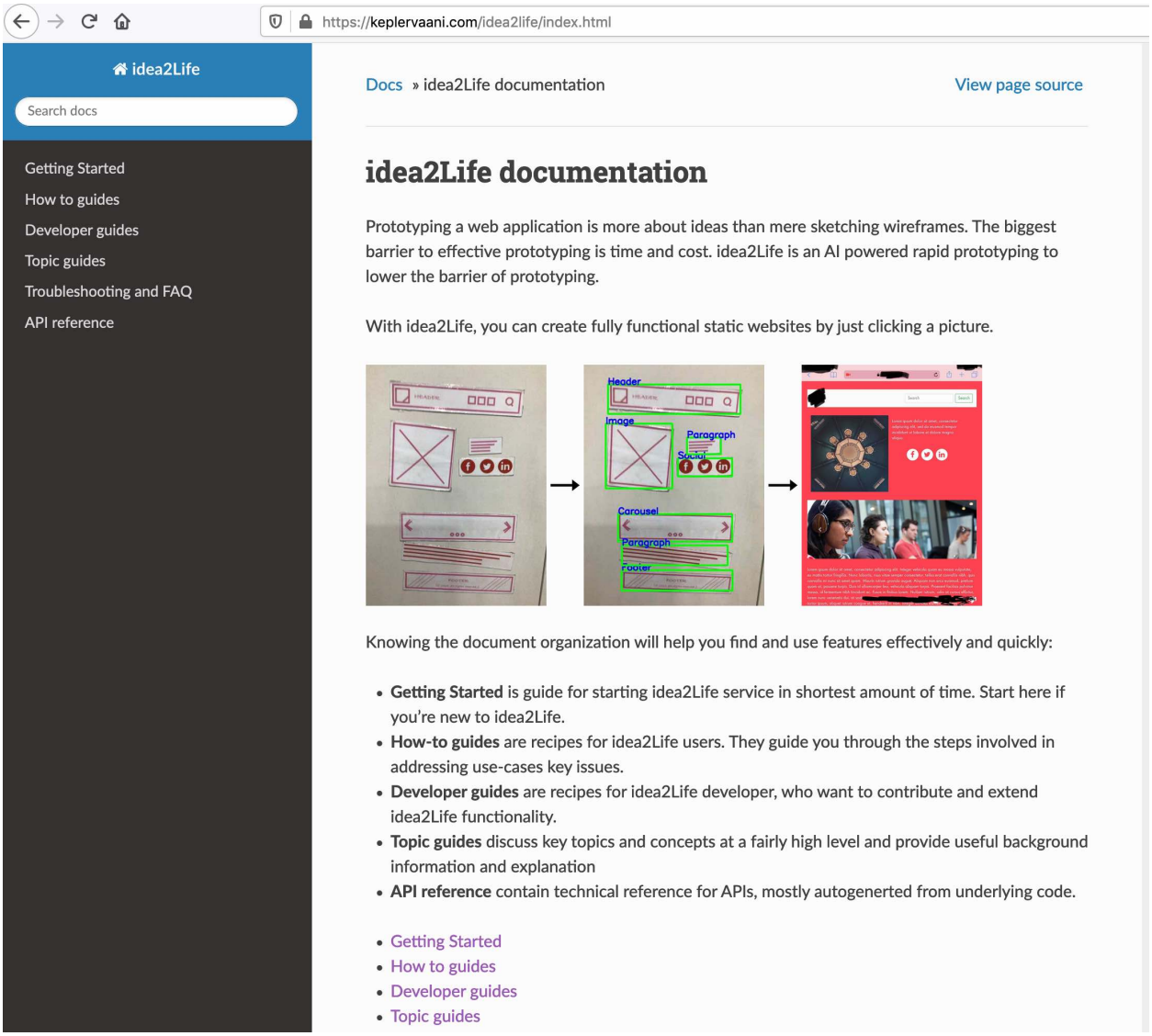


see *How to customize generated pages* for more details.

2.2.3 Need Help

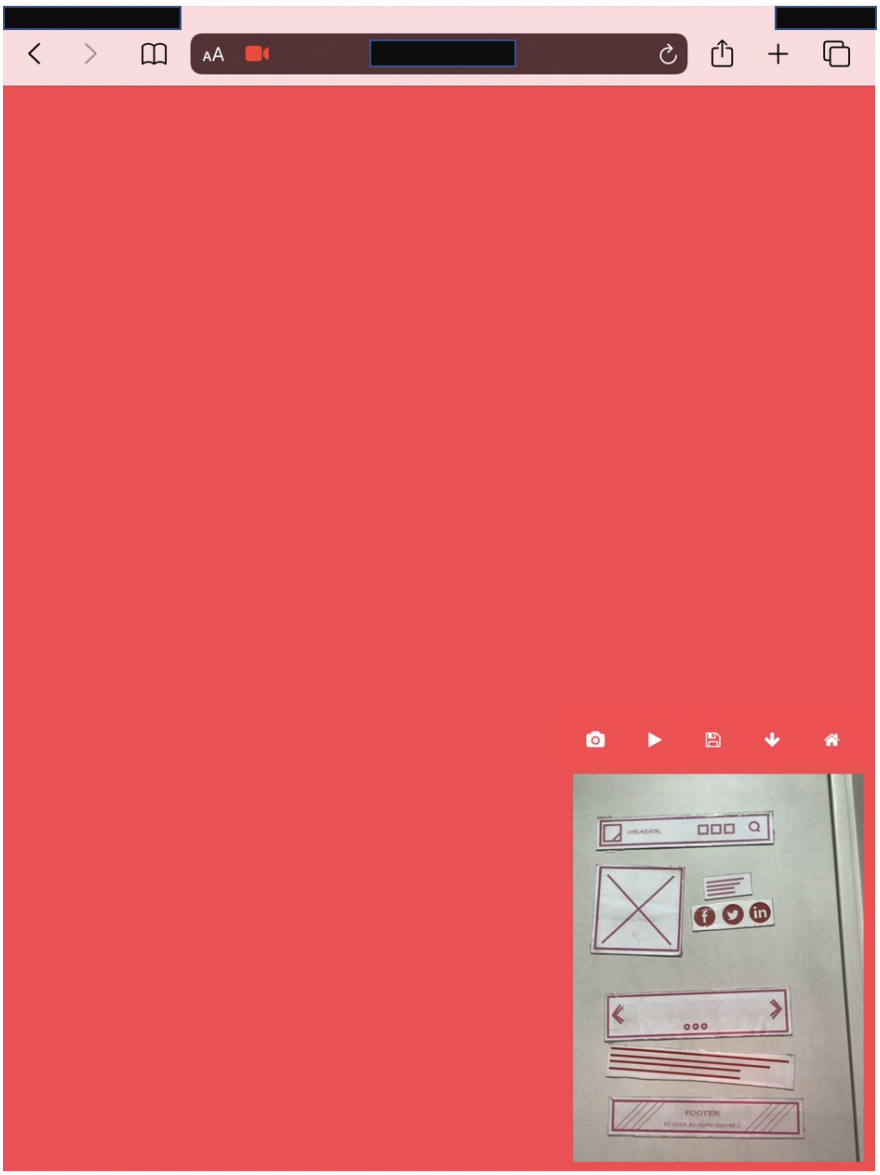
Click on “Need Help” link to launch the documentation.



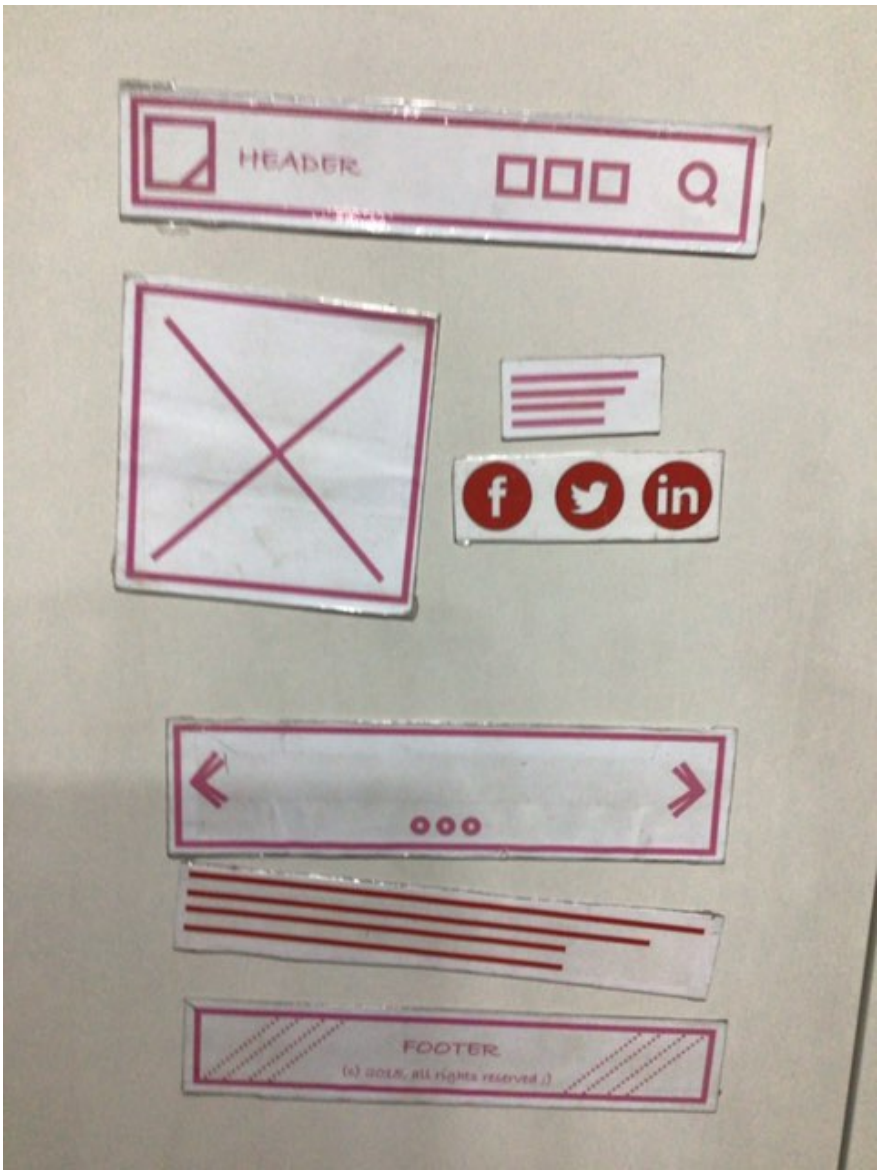


2.3 how to use prototype page

1. After pressing *Prototype* link on homepage, you will get to idea2Life prototype page.

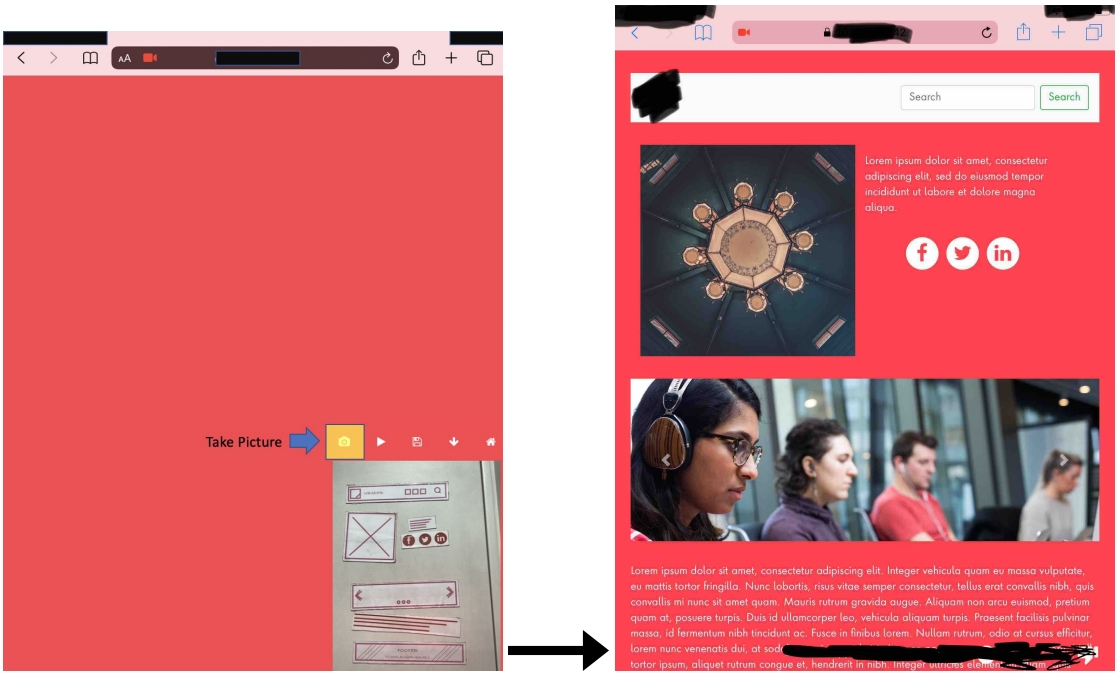


2. **Arrange printed templates** on table or metal surface according to your desired webpage design.



3. **Point camera** towards printed templates and **take picture** by clicking on camera icon in control panel.

4. idea2Life will automatically generate a new webpage based on laid out components.



This step will automatically generate new page which you can edit using these instructions *How to customize generated pages*

5. Press second **play icon button** if you want to continue taking picture and want to generate more pages

6. Press third **save icon button** if you want to save currently generated template for customization later.

7. Press fourth **Minimize button** if you want to hide this control panel and want to see generated page without control panel obstructing it.

8. Press fifth **Home button** if you want to return to homepage.

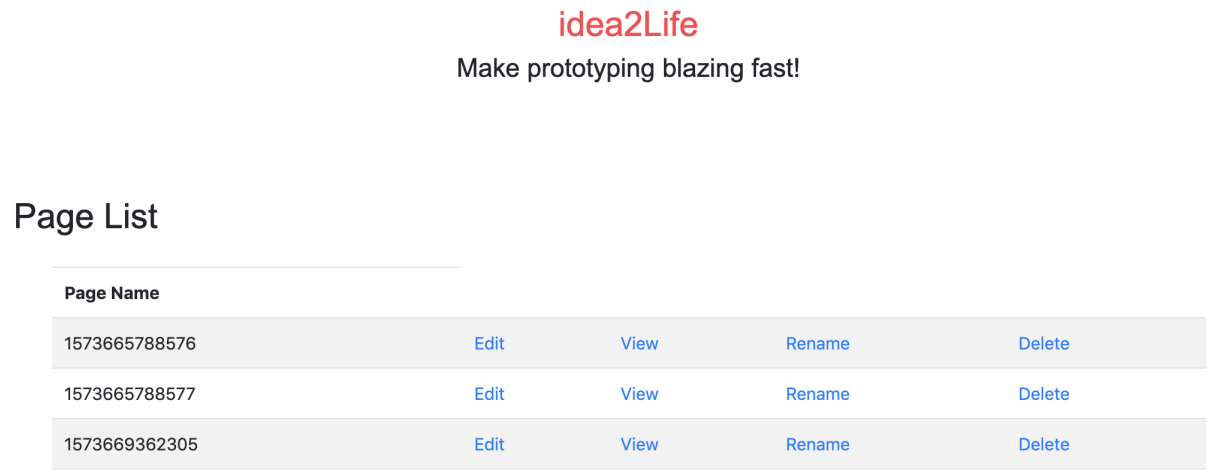
9. Press optional **upload image button** if instead of taking picture using camera you want to upload already clicked image stored on desktop/tablet for page generation. This optional icon will only be visible if you open prototype page using debug option. *Debug idea2Life prototype page*

2.4 How to customize generated pages

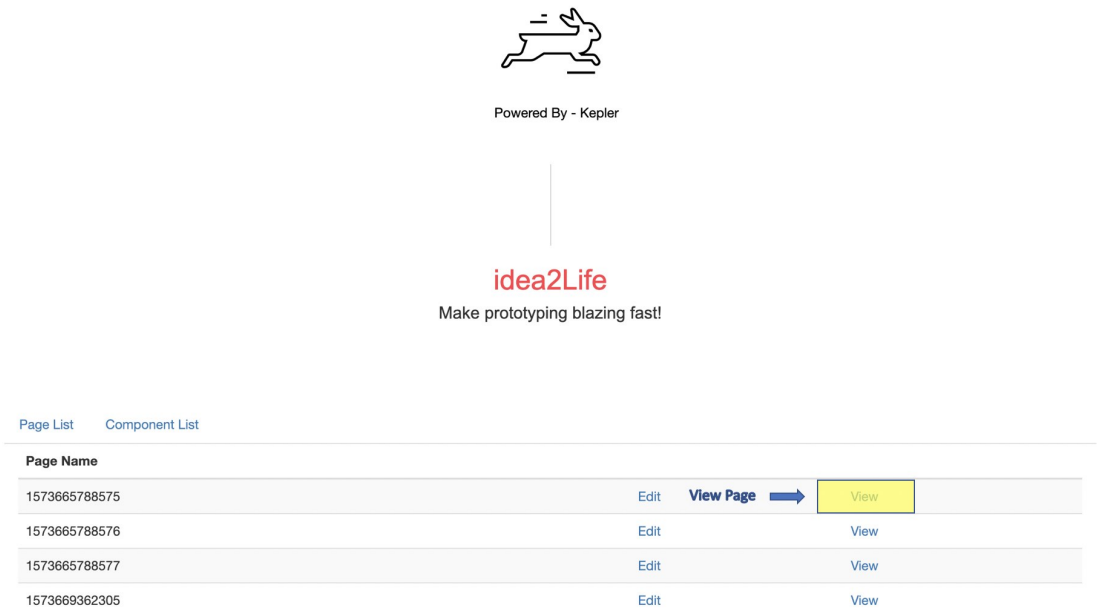
NOTE - For best experience, please use the customize functionality on Desktop.

2.4.1 Preview saved pages

After following *Customize* page instructions from home page. You are greeted with below page where you can see all generated pages.



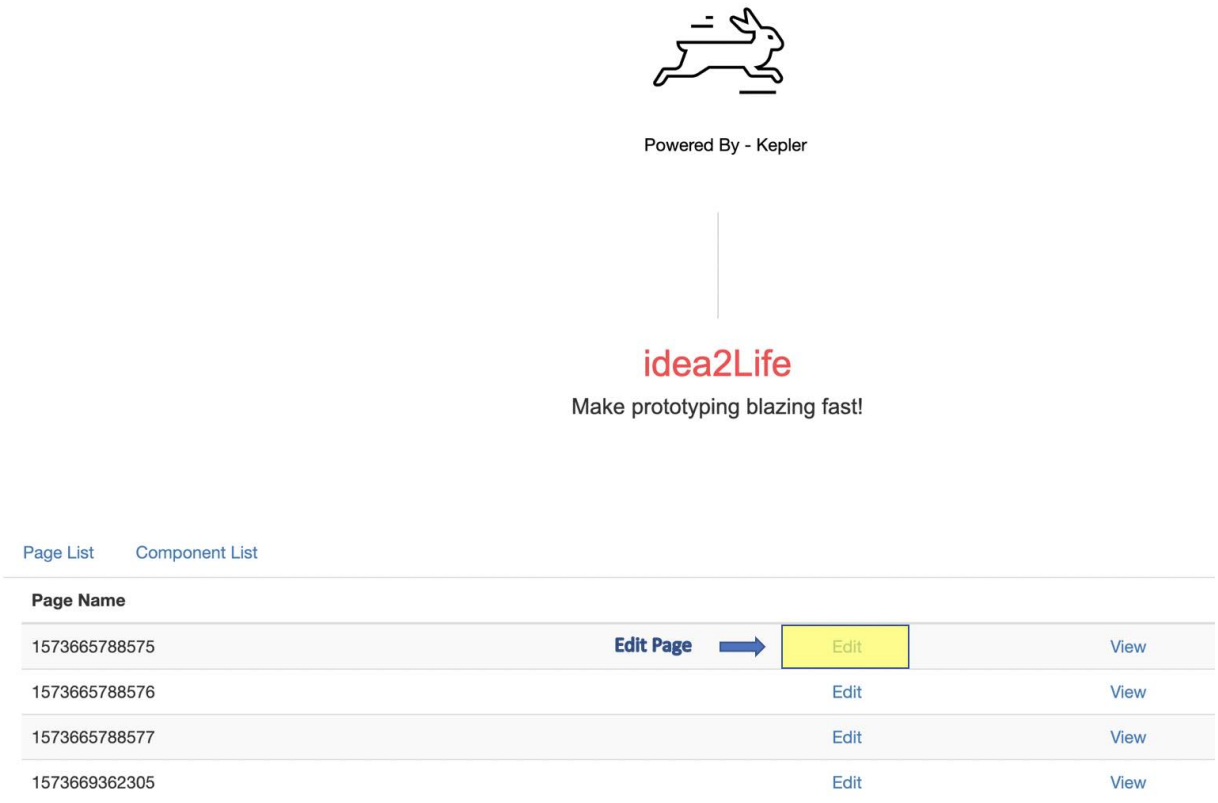
If you want to preview any of the generated pages, press view button (highlighted in yellow).



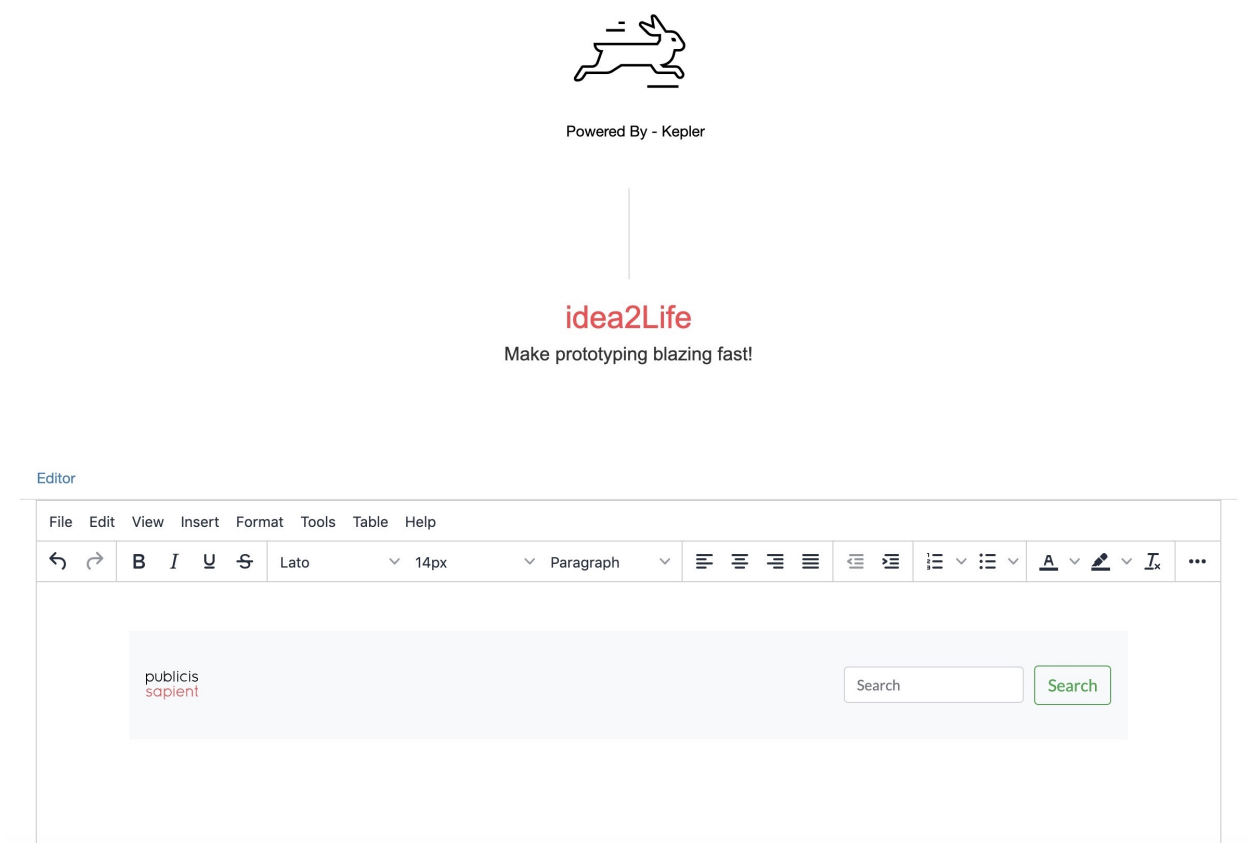
The view button opens the preview of generated page in a new tab.

2.4.2 Edit Page

Clicking on the edit link (highlighted in yellow), opens the page in Edit mode. In edit mode you can modify the generated page.

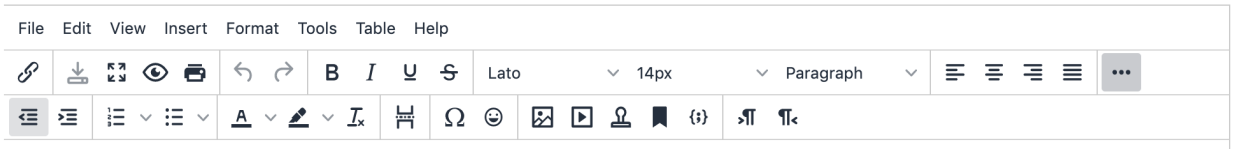


After pressing edit button, you are redirected to page editing mode. which looks like the one below:



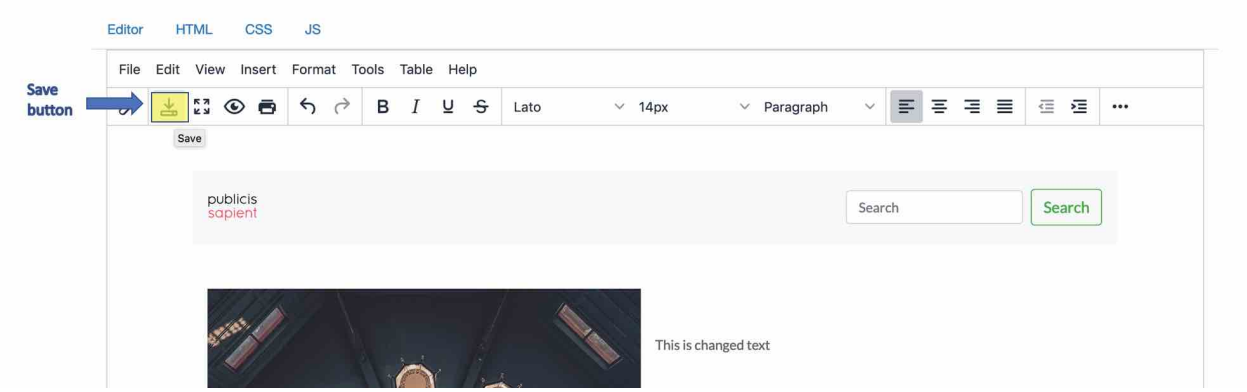
In the editor you have some basic functionality to change the text, font, styling, assets, etc along with features like adding hyperlink. These feature comes in very handy when you are trying to create a website using the HTML pages generated by the tool. Take a look at below panel for checking out available page editing features.

Editor panel



Using various option on this panel you can *Customize Style*, *Customize assets* or *Customize copy*. You can also link generated pages by following these steps *How to link generated pages*

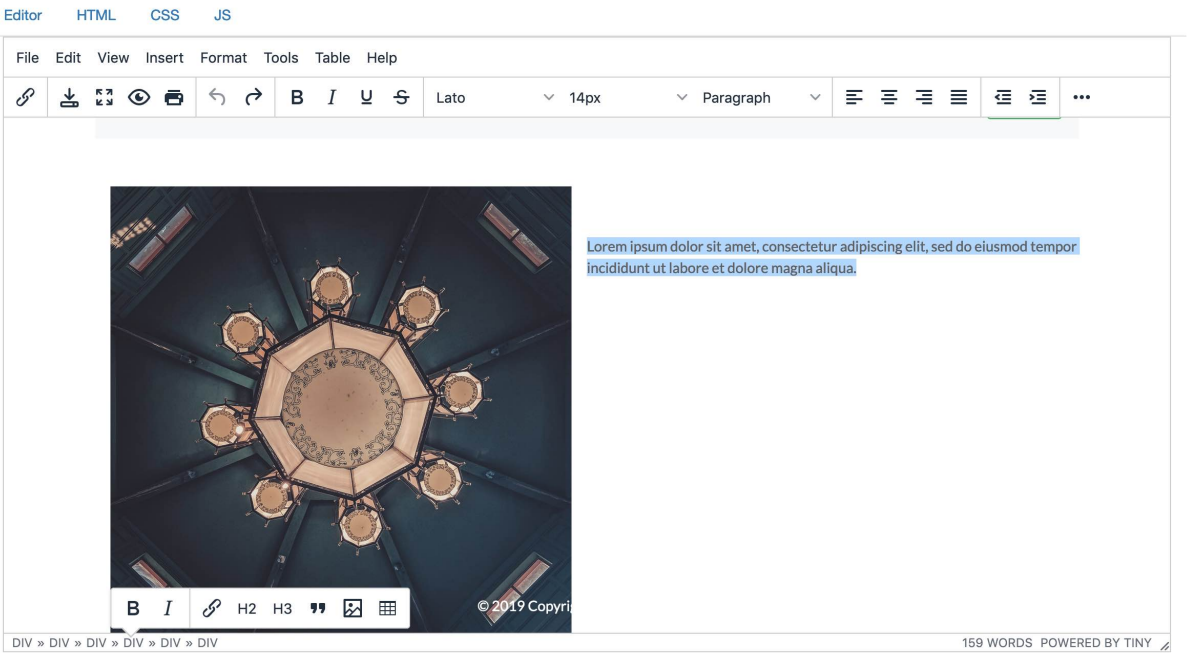
Don't forget to Press save button after you are done customizing page.

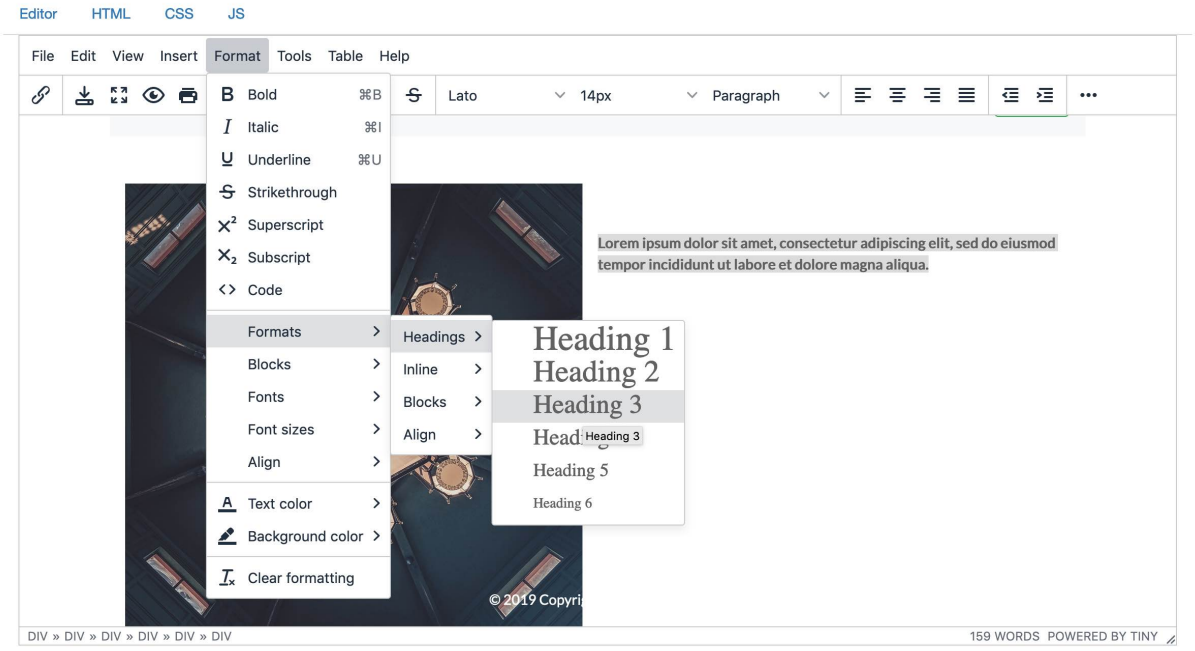


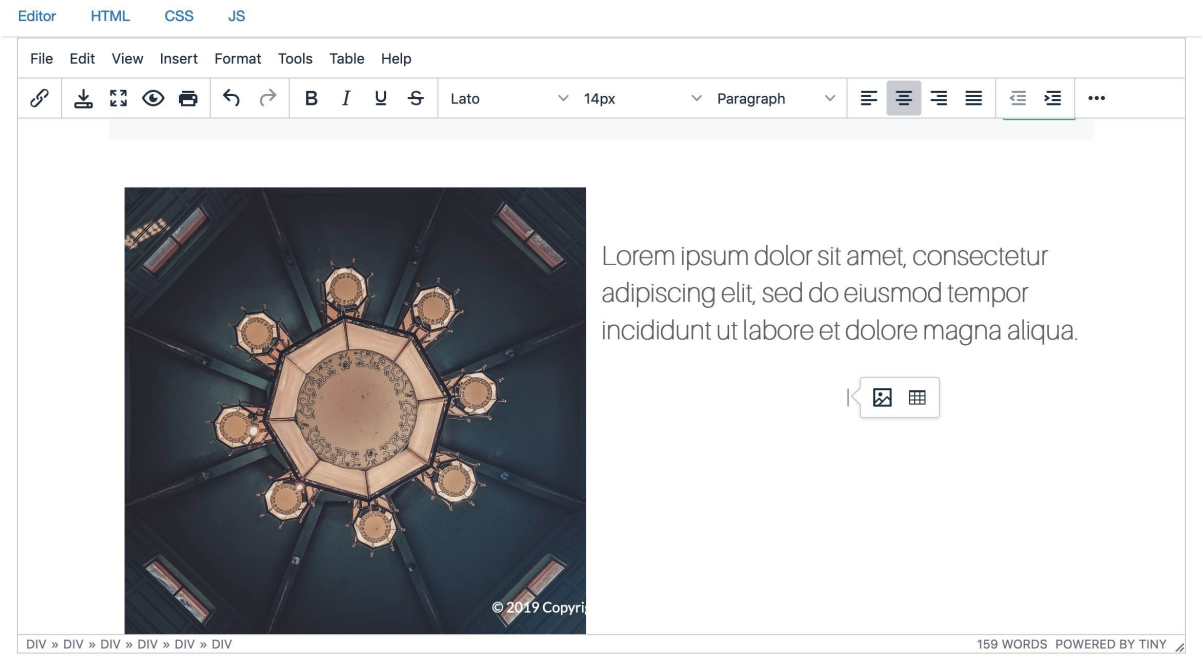
2.4.3 Customize Style

Customize/Modify Style of Generated Pages If you want to change style of text, select text and then, Press Format option in **Editor Panel** choose target font, text format, size etc .

For example here is how you can change format of text.



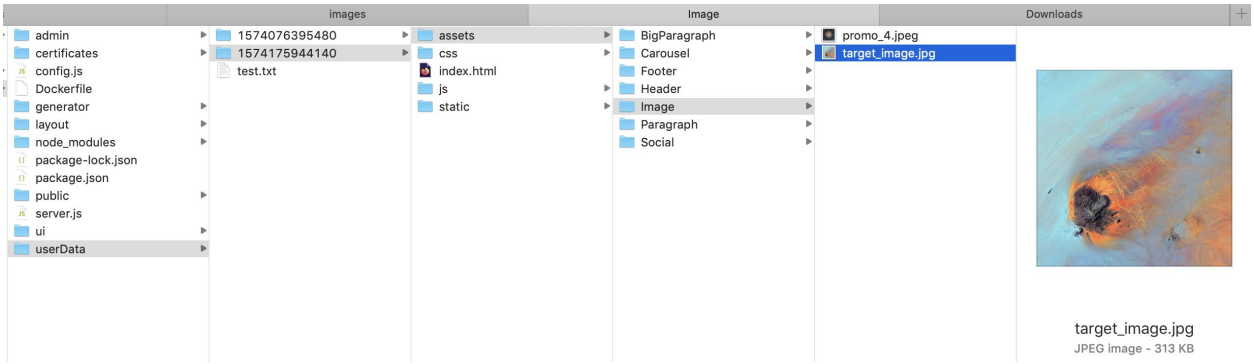




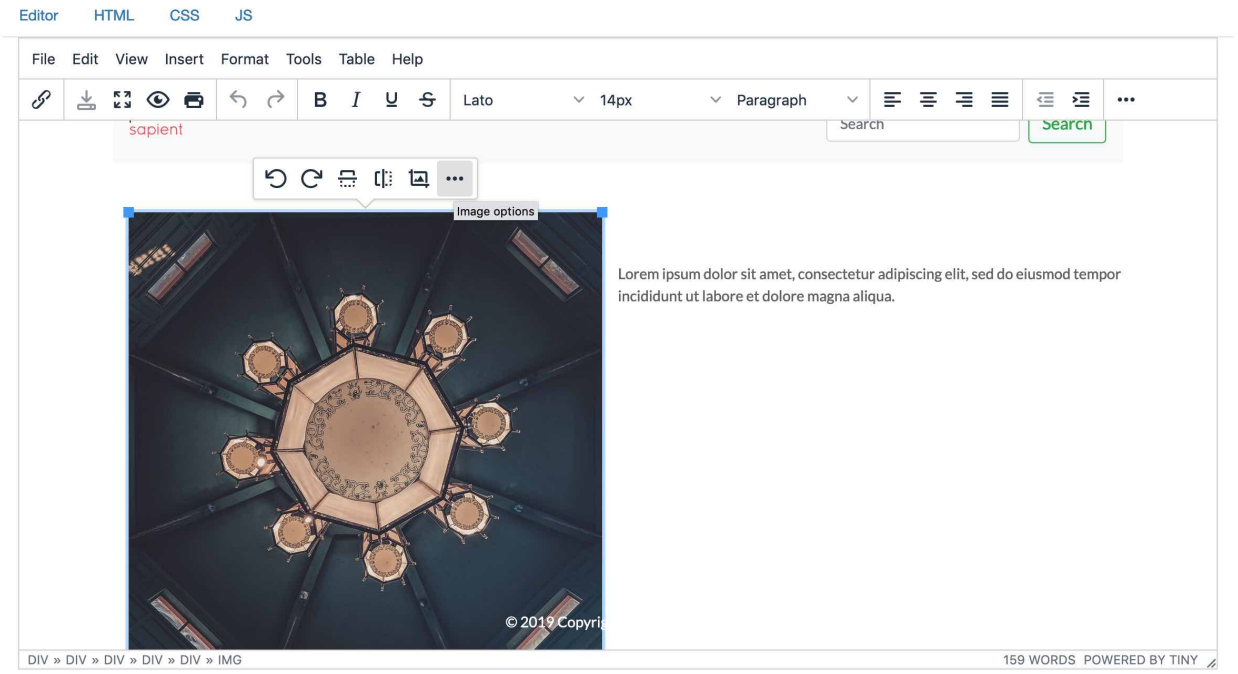
Press save button after you are satisfied with your changed content.

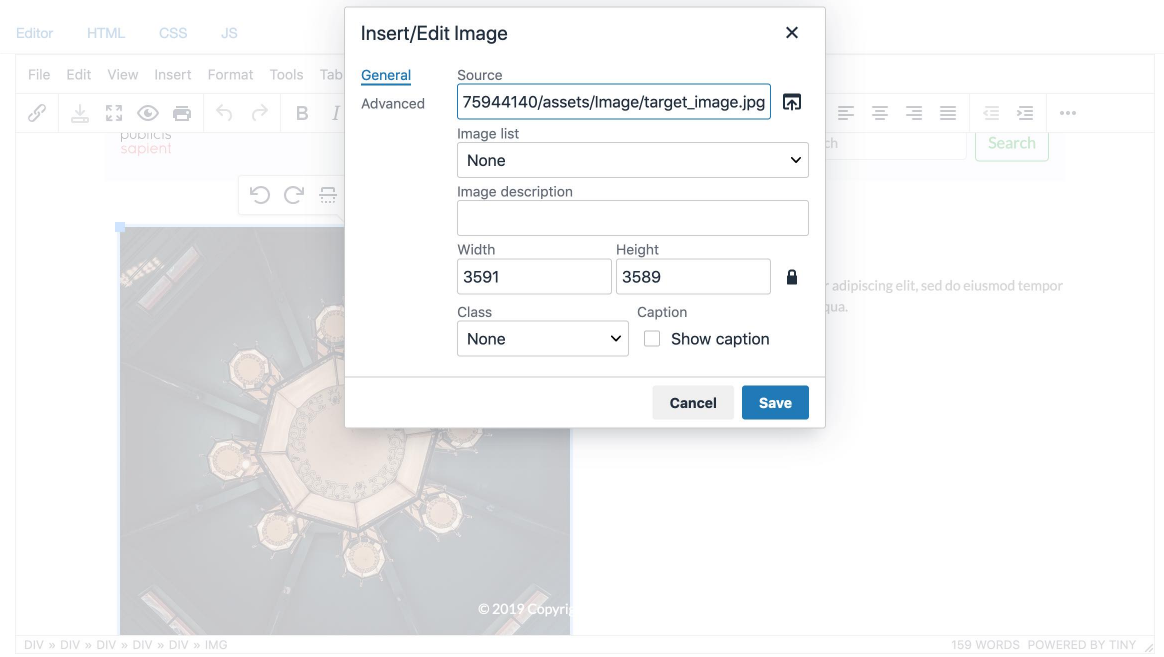
2.4.4 Customize assets

If you want to change asset first save downloaded asset you want to change in asset folder of your site like this.



Then select image/video you want to change and change path of the image to target image/video to save your asset.

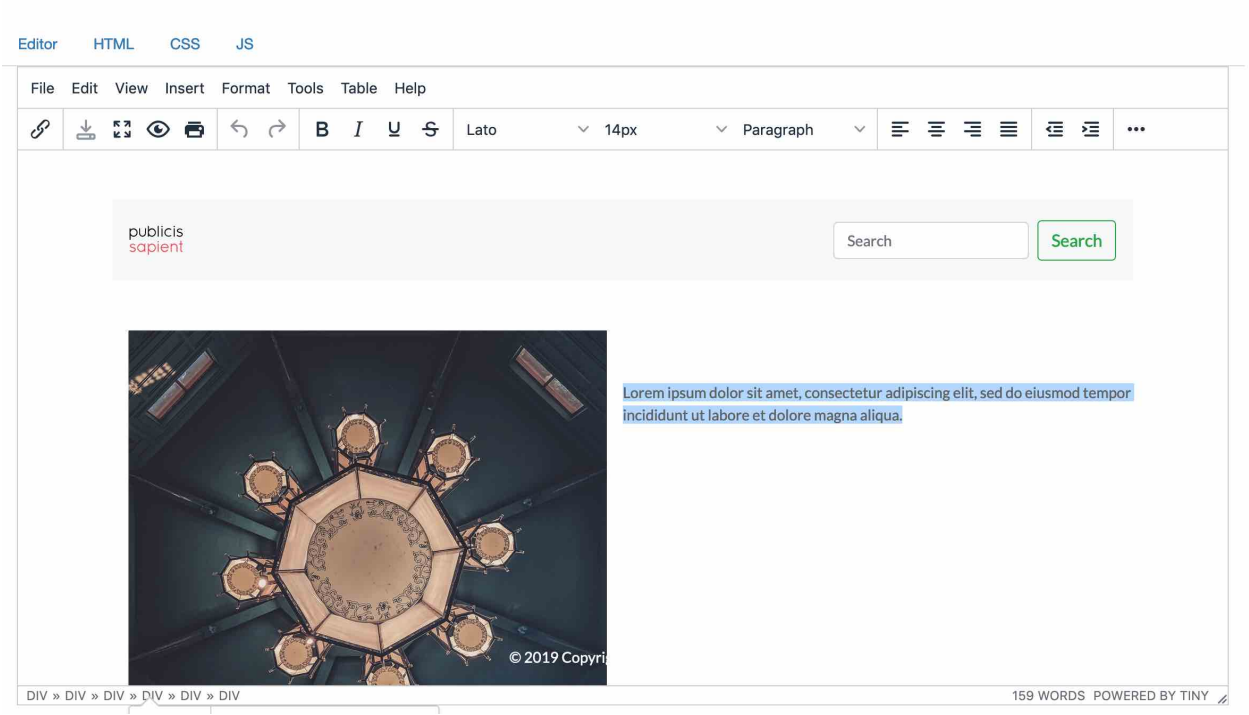




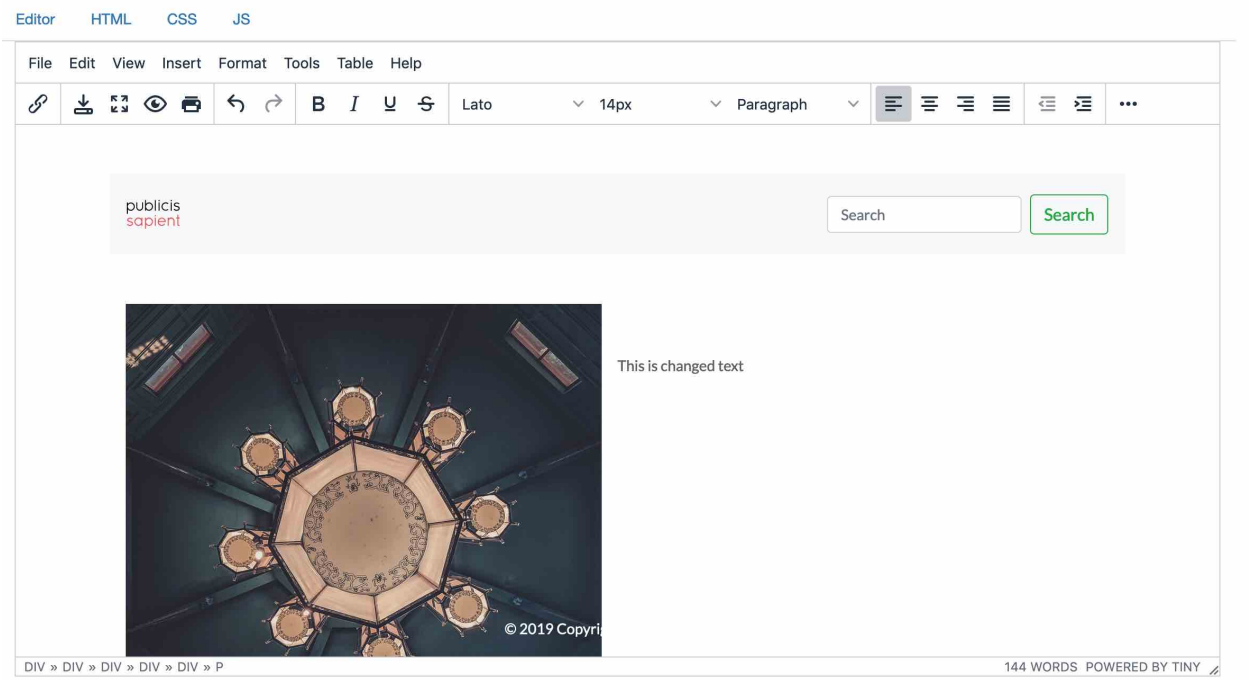
Press save button after you are satisfied with your changed content.

2.4.5 Customize copy

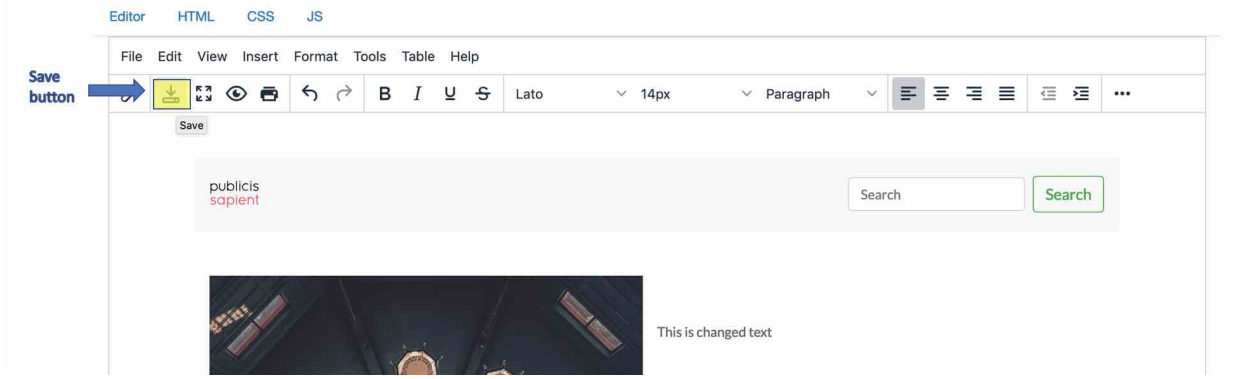
If you want to change copy/content first select text you want to change.



Then start typing text you want to change to like this:



Press save button after you are satisfied with your changed copy content.



2.5 How to link generated pages

NOTE - For best experience, please use the customize functionality on Desktop.

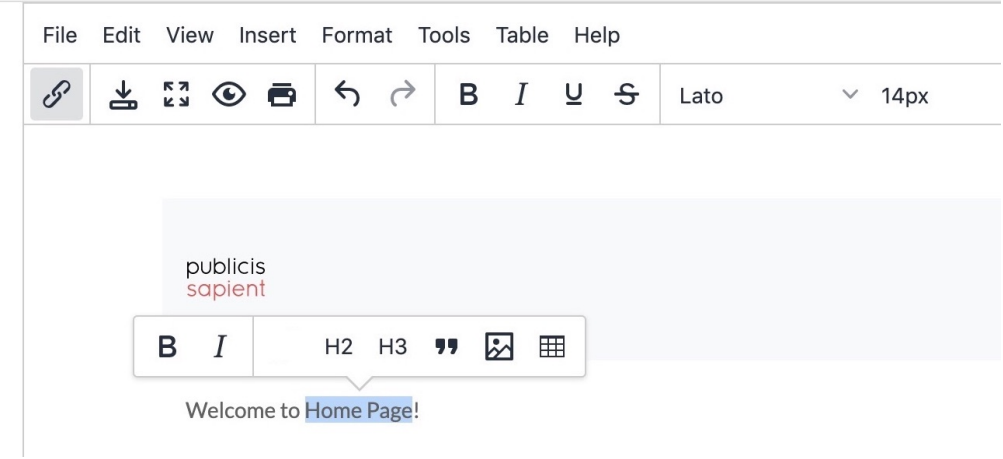
You can link the generated html pages. This helps in prototyping user journeys and workflows. You can link pages using hyperlink on Text, Image and button url. The editor will add an inline <a> for creating link, as shown below:

Welcome to this newpage!

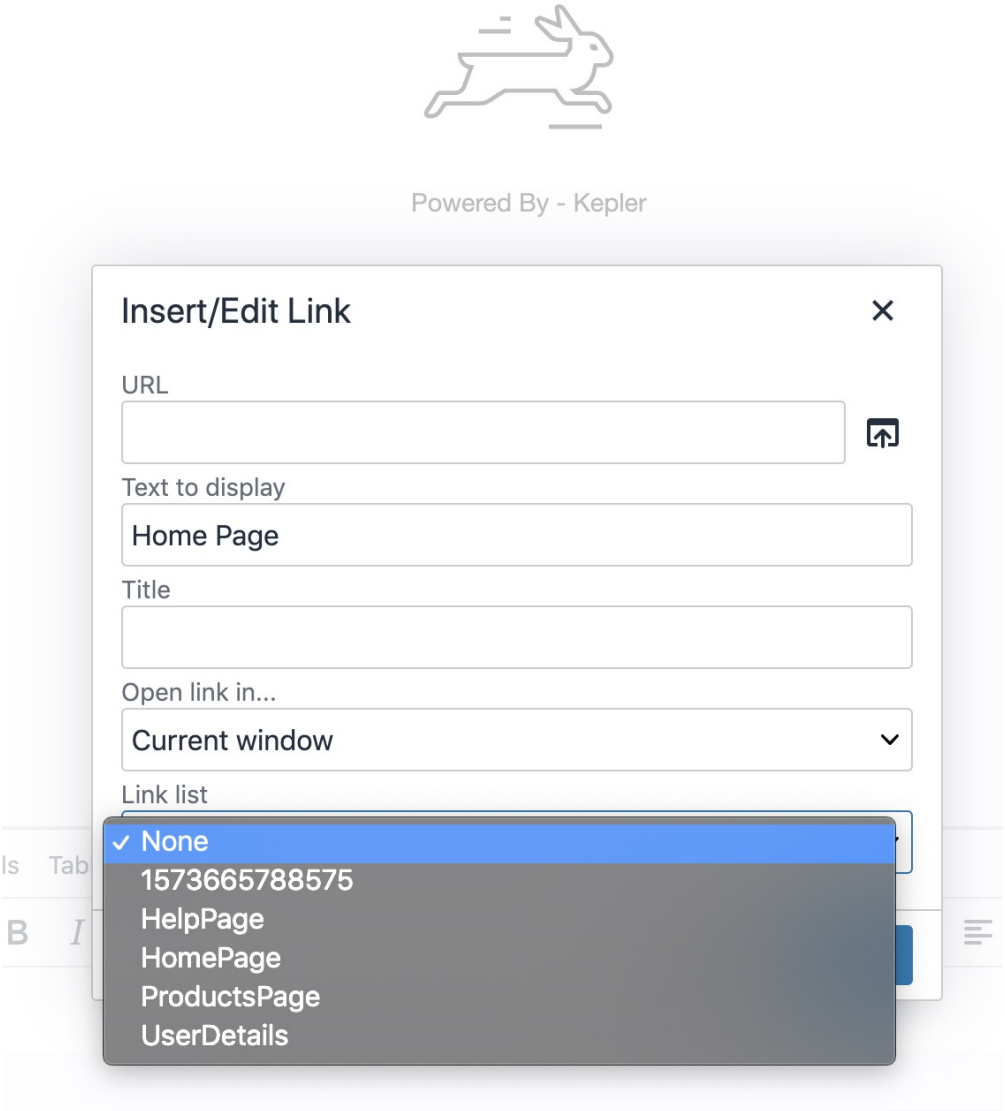
Hyperlinking internal pages shortcut

A quick alternative to link idea2Life generated pages is to select the destination page from “link list” drop down available on bottom of the “Insert/Edit link” pop-up.

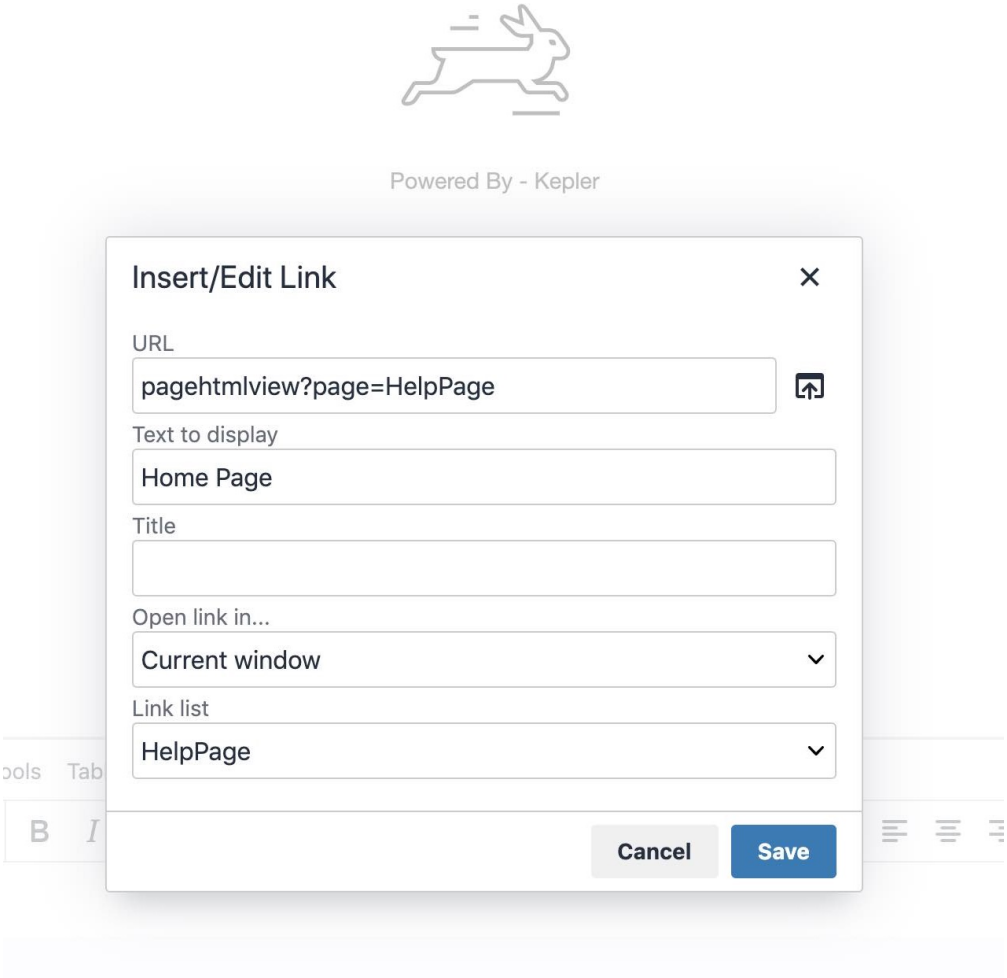
1. Go to the editor screen, and select the text/image/button you want to hyperlink. From the top menu bar, select the “link” button on the toolbar as shown below:



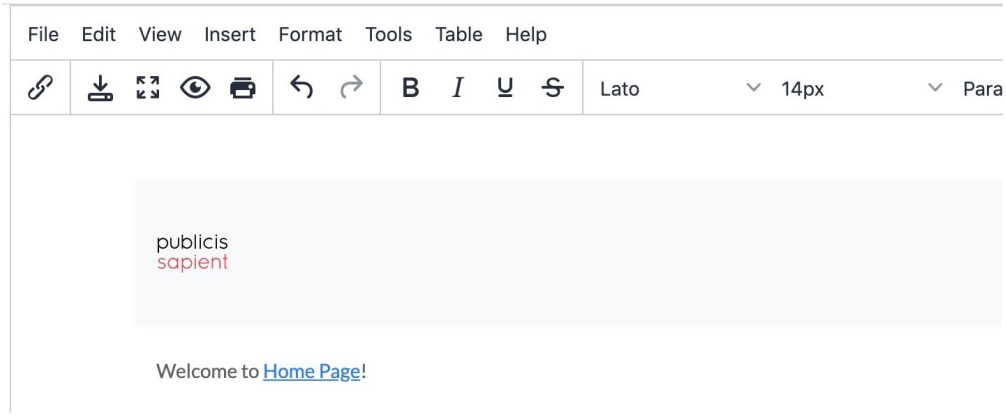
2. Select you page from “link list” drop down. Refer to the screen shot below.



3. Below is the pop up view after the page selection.



4. View of edited text with hyperlink added:

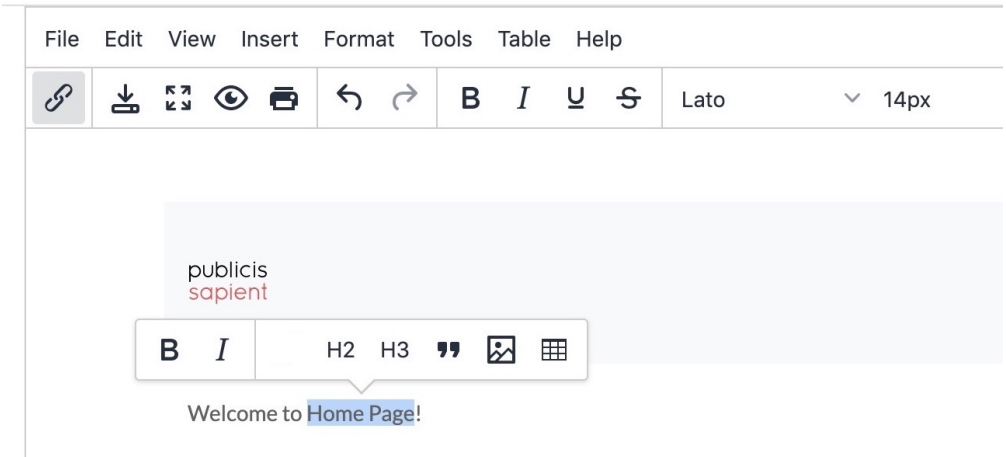


5. Press save button after you are satisfied with your changed content. Similarly, you can link pages using images and button.

images/admin_panel_save_link.jpeg

Linking external Pages

1. Go to the editor screen, and select the text/image/button you want to hyperlink. From the top menu bar, select the "link" button on the toolbar as shown below:



2. Paste the external URL in textbox and save. Now the link will be directed to the external URL.

2.6 How to add new theme

We use EJS (A templating engine for node) to render pages and themes. Simple configurations like changing CSS / ASSETS for theme and components doesn require knowledge of EJS beforehand but advance configurations like adding new components, external libraries will require understanding on the engine.

Adding new theme (quickstart)

1. Create a new folder with *yournewthemename* inside *idea2life/generator/views/themes/*.
2. Copy paste the contents of *default* theme from *idea2life/generator/views/themes/default* to this new folder.
3. Go to *generator/config.js* and change the *current_theme* config name to *yournewthemename*.

Now you have a new theme (exactly same as the default theme) available for any customizations.

Before doing any customization it is important to understand the current theme structure.

Theme structure NOTE - Theme *yournewthemename* has the same structure as the *default*

The *default* theme (located at *idea2life/generator/views/themes/default*) contains the following:

1. cam_component: This is the camera component on the prototyping page.
2. components: This folder contains the *assets/css/html* of supoprted components in idea2life like Paragraph, Button etc.
3. loader: This is the loading icon used on prototping page to indicate backend processing.
4. notification_component: This component display system notifications on protyping page.
5. static: This is the static folder containing all dependent libraries
6. config: Configuration for *default* theme
7. index.ejs: The index file for the theme (this is rendered in the browser with the help of EJS)
8. main_css.ejs: The core css file for the theme (css dependencies are included in index.ejs using this).
9. main_js.ejd: The core js file for the theme (js dependencies are included in index.ejs using this)
10. modal.ejs: This EJS contains the HTML for modal used on the prototype page.

    NOTE - We only support themes which are based on bootstrap and jquery for now.

2.6.1 Changing Styles

Changing styles for your new theme

Global level styling for the pages are plugged from *idea2life/generator/views/themes/yournewthemename/main_css.ejs*.

1. Go to *idea2life/generator/views/themes/yournewthemename/main_css.ejs*.
2. Change the font-family, background-color or color property under *body*.

Changing icon stylesheet for your camera component

Camera component is plugged from *idea2life/generator/views/themes/yournewthemename/cam_component*.

1. Go to *idea2life/generator/views/themes/yournewthemename/cam_component/css/custom/main.css*.
2. Change the styles under .icon or .fa-icon class based on *yournewthemename* requirement.

Changing styles for loader component

Loder components is plugged from *idea2life/generator/views/themes/yournewthemename/loader*.

1. Go to *idea2life/generator/views/themes/yournewthemename/loader/css/index.css*.
2. Change the styles under .loader class based on *yournewthemename* requirement.

Changing styles for notification component

Notification component is plugged from *idea2life/generator/views/themes/yournewthemename/notification_component*. We are using toastr for notification

1. Go to *idea2life/generator/views/themes/yournewthemename/notification_component/css/index.css*.
2. Change the styles under .toast-* classes based on *yournewthemename* requirement.

2.6.2 Changing Assets for idea2Life components

A lot of components like Carousel, Image, Video, Header etc use static assets like images / videos. Changing these assets for the components is a matter of drag and drop as well. Let's take an example - suppose we want to change the logo icon displayed in the header. Following are the steps to do this:

1. All components reside under *idea2life/generator/views/themes/yournewthemename/components*.
2. Find the *Header* component folder inside the components.
3. The component folder has 3 sub folders:

```
1. assets
2. css
3. js
```

4. Under the *assets* folder you can find the *logo.svg* file.
5. Replace this logo with your own but with the same name.

Changing the assets for any component can be done in the same way.

2.7 Hosting Generated Webpages from idea2Life

By default, idea2Life puts your data inside <path-to-repo>/idea2Life/userData. This directory is also mounted inside docker at /usr/src/app/userData. You page name will exist as a folder here which can be copy pasted and hosted on a different server. However, this will work if your static path is mapped correctly.

Lets assume you have a node application (using express server) with package.json

```
{
    "main": "server.js" // assuming this is where http / https server is created
}
```

and your typical application structure with static files looks like

app
├── /static/
├── server.js

To host the page:

1. Copy the folder at <path-to-repo>/idea2Life/userData/pageName into /static/.
2. you should additional static mapping to point to this directory. To do so, put the following code in your server.js:

```
{
    app.use('/static_page', express.static(path.join("static"))); // path to your static
}
```

(This is because idea2Life prepends all dependent static file urls with */static_page/*)

CHAPTER

THREE

IDEA2LIFE DEVELOPER GUIDES

3.1 How to build documentation from source

Build documentation using docker

You can use AI service container created using docker-compose to additionally build documentation for you.

1. If not already started, start docker service from terminal using the following commands:

```
cd <path-to-repo> //you need to be in your repo folder
docker-compose up
```

2. Attach to AI service container instance:

```
docker exec -i -t idea2life_open_ai_1 /bin/bash
```

3. Goto Docs directory in running container:

```
cd ../docs/
```

4. Make documentation using command:

```
make html
```

#. If no error this will build docs in folder docs/build/ in host system. Open index.html for reading documentation

Build documentation alternate instructions

1) Install OpenCV in Anaconda python (python=3.7)

```
conda install --channel menpo opencv
```

2) Install these dependencies from pip:

```
npm install -g jsdoc
pip install sphinx-js
pip install sphinx
pip install sphinx-rtd-theme
```

3) Goto directory docs:

```
cd docs/
```

4) Issue command:

```
make html
```

5) If no error this will build docs in folder docs/build/ open index.html using this from command prompt:

```
open docs/build/index.html
```

3.2 Install and use idea2Life from source (without docker)

Currently we have tested idea2Life installation on MacOSX and Linux systems only. These instructions assumes you have anaconda python with python version 3.7 already installed on your system.

1) Clone idea2Life repo.
2) Create folder *ai/lib*. Install PyYolo inside it. Follow instructions for installation of pyyolo by reading Readme file in pyyolo repo.
3) Inside folder *ai/models* download the model file inside it.

4) Install dependencies **opencv** and **numpy** using either pip or conda. For installing OpenCV on Ubuntu 16.04 and MacOSX use this command on your terminal:

```
conda install --channel menpo opencv
```

5) Go inside the ai folder and run these two commands:

```
cp cfg/yolo-obj.cfg lib/pyyolo/darknet/cfg/
cp data/obj.names lib/pyyolo/darknet/data/
```

6) Goto folder *idea2life*. Run npm install:

```
cd <path_to_repo>/idea2life
npm install
```

Run Service

You have to run ai service and main idea2Life service separately using this command.

1) Open terminal and run the following commands:

```
cd <path-to-repo> //you need to be in your repo folder
export FLASK_APP=ai/server/app.py && flask run -p 5000
```

2) Inside file <path-to-repo>/idea2life/conf.js find this entry:

```
ai: {
        url: 'ai',
        name: 'ai',
        port: 5000
    }
```

Replace field url: **ai** with localhost or ip of address server where ai service is hosted.

2) Open terminal and run the following commands:

```
cd <path-to-repo>/idea2life //you need to be in your repo folder
npm start
```

3.3 How to call AI service

idea2Life AI Service handles template detection task. It is hosted as a flask service separately as python flask service.

URL for this flask service is at: *http://<ip_address_of_flask_server>:5000*

It internally hosts two separate endpoints as listed below.

3.3.1 1) Main endpoint template detection:

Path for flask service: *http://<ip_address_of_flask_server>:5000/svc*
Method: **POST**

Request format:

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "data":
    {
        "imgType": base64, "img": "base64_Image"
    }
}
```

Please note currently imgType of only **base64** is supported, will add imgType **url** in future

In case of error idea2Life will return error in this format:

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": <ERR_CODE>, "message": <ERROR_MESSAGE>
    }
}
```

Response format if Image present but no detection

```
{
"apiVersion": 2.1,
"context": "blank",
    "data":
    {
        "height": "700",
        "results": [],
        "width": "1050"
    }
}
```

Response format For this Image:


```
{
    "apiVersion": 2.1,
    "context": "blank",
    "data":
    {

        "height": "480",
        "results": [
            {
                "bottom": 370,
                "class": "Video",
                "left": 175,
                "prob": 0.789800226688385,
                "right": 375,
                "top": 176
            }
        ],
        "width": "640"
    }
}
```


Possible error list for /svc endpoint:

1. Invalid api version.:


```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 301, "message": "api version not received"
    }
}
```


2. Invalid api version request received.:


```
{
    "apiVersion": "2.1"
    "context": "blank"
```


(continues on next page)

(continued from previous page)

```
    "error":
    {
        "code": 302, "message": "Invalid api version request received"
    }
}
```

3. Context field not found in request.:

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 409, "message": "Context not found"
    }
}
```

4. Invalid request,error string received in request body":

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 410, "Invalid request, Received error in request body"
    }
}
```

5. Data not found: data field in request not found.:

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 411, "message": "data not found: data field in reqest not found"
    }
}
```

6. Unsupported imgType or data, If imgType is different then base64 or url

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 412, "message": "Unsupported imgType or data"
    }
}
```

7. image field in data not found.

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 413, "message": "image field in data not found"
    }
}
```

8. Error in converting base64 image to image.

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 414, "message": "Error in converting base64 image to image"
    }
}
```

9. Detection error, No template detected in image

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 415, "message": "Detection error, No template detected in image"
    }
}
```

3.3.2 2) Debug endpoint :

Path for flask service: *http://<ip_address_of_flask_server>:5000/debug*
Method: **GET**
Request format: NONE

Note: For debug view of previous detections just open URL in your browser

3.4 Debug idea2Life

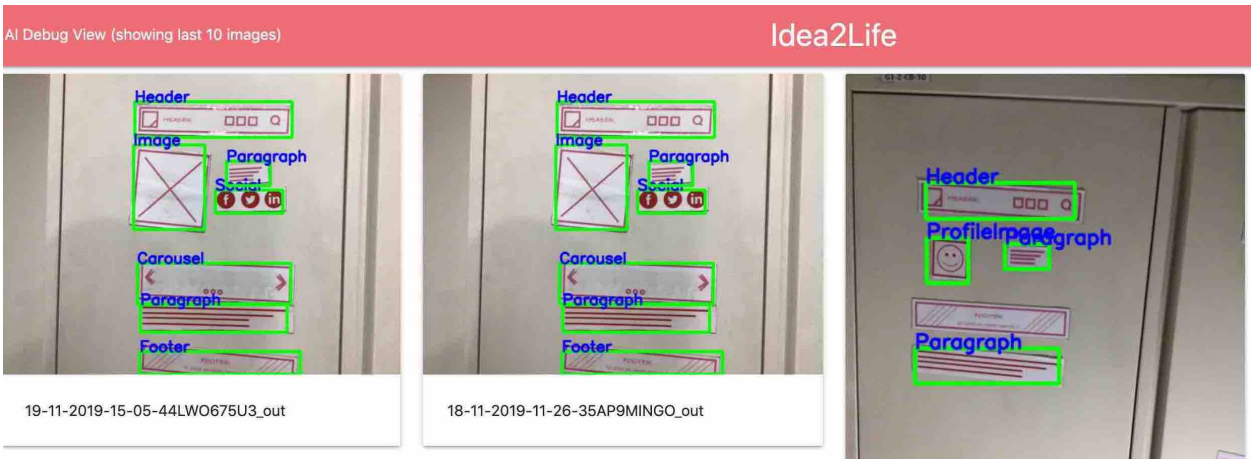
3.4.1 1) Debug ai service:

** Debug endpoint:**

Path for flask service: *http://<ip_address_of_flask_server>:5000/debug*
Method: **GET**
Request format: NONE

Note: For debug view of previous detections just open URL in your browser

After opening url *http://<ip_address_of_flask_server>:5000/debug*: you can see template detection of last 10 images.



you can use this debug view for checking whether photo is clicked correctly, or your clicked photo has some detection error.

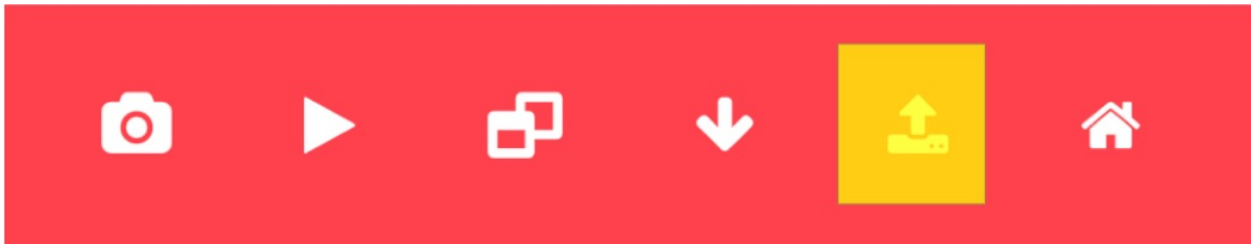
3.4.2 2) Debug idea2Life prototype page:

If you ever encounter error in using application with tablet/ipad or and do not have access of tablet device you can test application without camera using debug upload functionality. For after clicking on *Prototype* icon append */debug* after camera screen. or alternatively you can type full url

https://<IP address>:1813/generator/ui/home/debug

this will take you to following page.

On this page you can see one additional icon to upload (as can been seen in the image below):



Once you click on the upload icon, you will see a pop-up to upload image.

3.4.3 3) Debug idea2Life layout service:

Layout service is responsible for generating XML data from the JSON position data generated from the AI service. The XML will give an intuitive representation of the layout DOM for page. In case you want to manually test the output of the layout module, you can send a POST request to *https://localhost:port/layout/generate* with the ai JSON data in the form:

```
{

    "height": "480",
```

(continues on next page)

(continued from previous page)


```
    "results": [
        {
            "bottom": 60,
            "class": "Header",
            "left": 10,
            "prob": 0.789800226688385,
            "right": 400,
            "top": 10
        },
         {
            "bottom": 370,
            "class": "Video",
            "left": 175,
            "prob": 0.789800226688385,
            "right": 375,
            "top": 176
        }

    ],
    "width": "640"
}
```


Below is the sample XML content generated by the service:

```
<rows>
    <row id='R_R1' width='null' offset='null'>
        <component class='Header'></component>
    </row>
    <row id='R_R2' width='null' offset='null'>
        <component class='Carousel,Video'></component>
    </row>
    <row id='R_R3' width='null' offset='null'>
        <component class='Footer'></component>
    </row>
</rows>
```

The XML shows that we have 3 components in the layout : Header, Carousel & Footer.

Note - At present, we do not support width & offset values for components. You can expect this in furture release.

3.4.4 4) Debug idea2Life Generator (or rendering) service:

Generator service is responsible for generating html from XML data. Generator service use theme configuration to generate pages. Even the prototype page is generated using the generator service based on the selected theme. Generator service creates pages inside *idea2life/userData*, which can then be hosted on different server. Check *Hosting Generated Webpages from idea2Life*.

The protoype page is generated from *https://localhost:port/generator/ui/home*.

To test the html generated by the generator service, send a POST request to *https://localhost:port/generator/create_html* with the JSON data in the form:

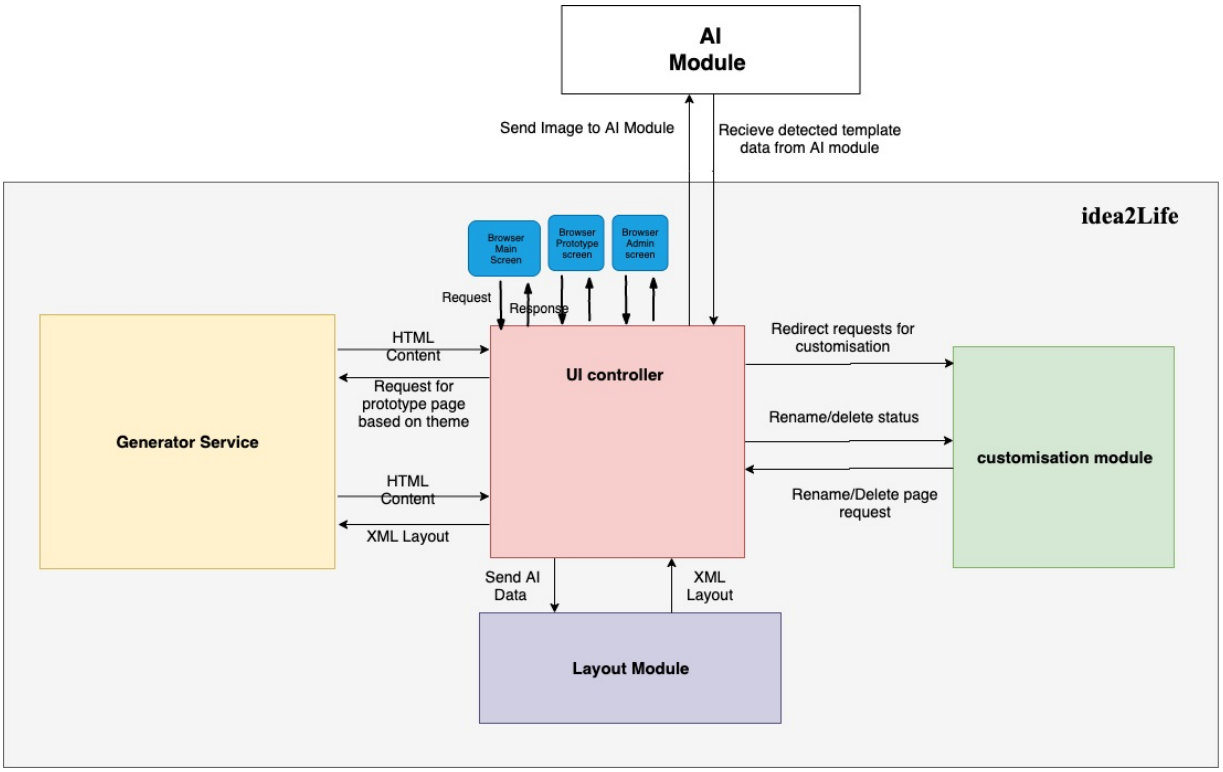
```
{
    xml: xmlData,
    filename: pageName
}
```

This will generate a page inside *idea2life/userData/* with the *filename* and can be viewed at list page (under customization).

3.5 idea2Life architecture

3.5.1 idea2Life system architecture

idea2Life architecture below: Take a look at idea2Life general architecture.



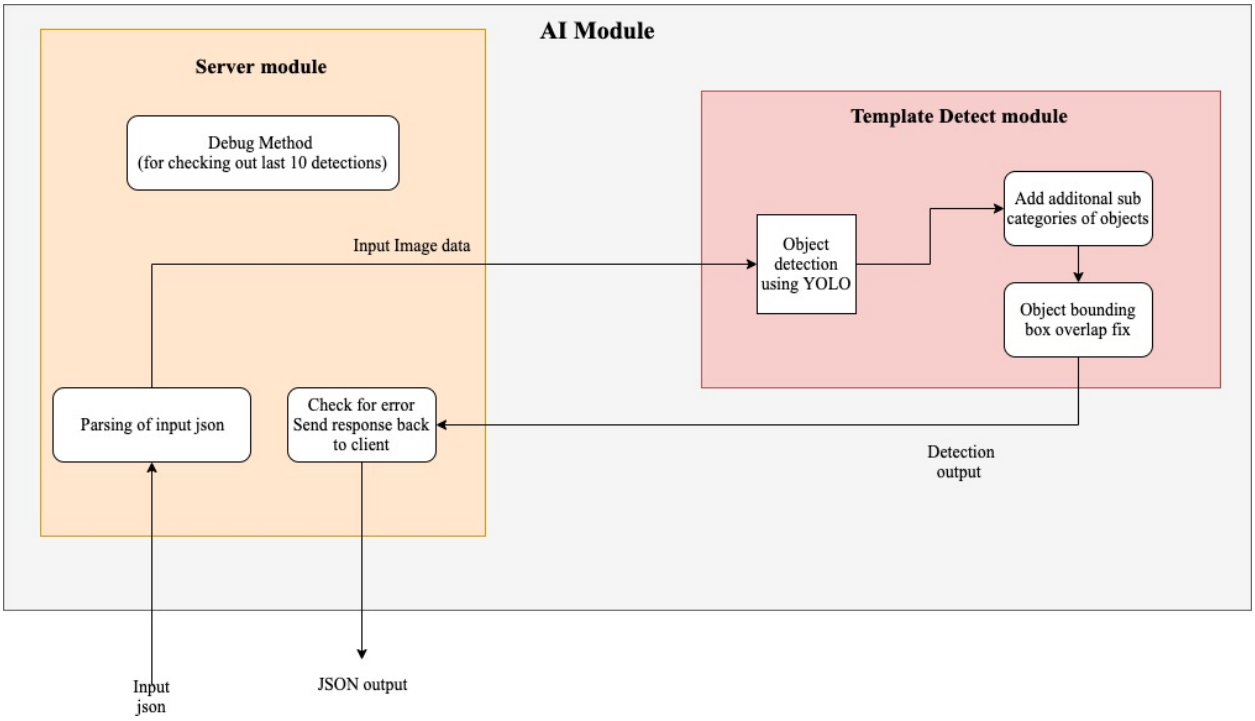
idea2Life could be broadly divided in **main module** and **ai module** here the **main module** could be further divided into following sub modules:

1. **Browser main screen** : This is the main screen hosted at *<idea2Life_url>:1813* which you then connect to when starting out idea2life. For more detail about this screen you can look at following document. *How to navigate idea2Life home page*. On clicking on an of the page link it send requests back to idea2Life UI controller for appropriate redirects.
2. **Browser Prototype screen** : This is the browser Prototype screen hosted at *<idea2Life_url>:1813/generator/ui/* which you get redirected at after clicking on *Prototype* link on main screen. Take a look at this link if you want to know more about Prototype process using this screen. *idea2Life Prototype screen.*
3. **Browser admin/customize screen** : This is the admin screen hosted at *<idea2Life_url>:1813/admin/* which you will get redirected at after clicking on *Customize* link on main page. Take a look at this link if you want to know more about process of customization of generated pages using this screen. *idea2Life customize screen.*

4. **UI controller** : This is the main controller written in Javascript. located at *<idea2life_repo>/idea2life/server.js*. All other services for idea2Life talks to this controller for all requests and response.
5. **Generator service** : This is the generator service for idea2Life, this is the primary module responsible for generating html content from either xml layout or request for prototype page based on theme. Controller service talks to this module for both of these tasks.
6. **Customization module** : This is the Customization module which performs most of the page customization tasks that are requested by *Customize* screen, UI controller sends request received by browser related to page customizations tasks like rename generated pages, change fonts etc to this module. Which then completes these tasks and returns customized pages back to controller.
7. **Layout module** : This is the layout module written in Javascript. This module converts json received from ai module, solves page layout problem and sends result back into xml format to controller module.

3.5.2 idea2Life ai module architecture

idea2Life ai module architecture below:



1. **Server module** : This is the main service module which hosts rest endpoint for performing ai Template detection tasks. Main controller service sends JSON Image data to *svc endpoint* of ai module link. This module then sanitizes json response and then calls Template detect modules for detection of layout template from image. Once detection of templates is completed, This then sends result back to idea2life main controller. Additionally it hosts one *debug endpoint*, use this endpoint if you want to check output of template detection for debug purposes.
2. **Template detect module** : This module is used for actual template detection, It receives Image sent by server module. Converts input image from base64 to binary format, performs image resizing. After this a call to pre-trained object detection neural network model using PyYolo library is performed. This deep learning model file we have trained by using Darknet/YOLO object detection library. After performing object detection using python pyyolo library, It then does further detection of extra sub templates like BigParagraph from existing elements like paragraph, using method **templates_sub_detection**. A further processing is performed using **check_N_fix_overlap** method for fixing overlap between two detected bounding box. Finally result is sent back to ai server module.

CHAPTER

FOUR

IDEA2LIFE DEEP DIVE

4.1 AI service API Details

On doing docker up idea2Life runs ai service:

Path for ai flask service

flask endpoint for ai service is @app.route("/svc", methods=["POST"]

Path for flask service: http://localhost:5000/debug Method: **POST** Request format:

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "data": { "imgType": base64, "img": "base64_Image" }
}
```

imgType could be either base64 or url in future

if error:

```
{
    "apiVersion": "2.1"
    "context": "blank"
    "error":
    {
        "code": 404, "message": "Error message"
    }
}
```

Response format if Image present but no detection

```
{
"apiVersion": 2.1,
"context": "blank",
    "data":
    {
        "height": "700",
        "results": [],
        "width": "1050"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Response format For this Image:

is


```
{
    "apiVersion": 2.1,
    "context": "blank",
    "data":
    {

        "height": "480",
        "results": [
            {
                "bottom": 370,
                "class": "Video",
                "left": 175,
                "prob": 0.789800226688385,
                "right": 375,
                "top": 176
            }
        ],
        "width": "640"
    }
}
```

4.2 idea2Life Layout

idea2Life uses xml for Layout

4.3 idea2Life Supported components

1. 'Button'
2. 'CheckBox'
3. 'Heading'
4. 'Image'
5. 'ImageHorizontal'
6. 'ImageVertical'
7. 'ProfileImage'
8. 'Label'
9. 'Link'
10. 'BigParagraph'
11. 'Paragraph'
12. 'RadioButton'
13. 'TextBox'
14. 'Header'
15. 'Footer'
16. 'Carousel'
17. 'SearchBar'
18. 'Slider'
19. 'Ratings'
20. 'Video'
21. 'VideoHorizontal'
22. 'ComboBox'
23. 'Social'

You can find images for each components in this file Download link: [idea2life_templates_for_print](https://drive.google.com/file/d/1MIeZj1EItCZbk7e1qTOrv3GTdOBdYXXB/view?usp=sharing)

CHAPTER

FIVE

TIPS AND TROUBLESHOOTING

5.1 Error:- Listen tcp 0.0.0.0:1813: bind: address already in use

If you encounter this error, the port 1813 is already in use by some other application. You can either free the port or run the docker application on some other port.

How to release port in use

To release port 1813 on your linux/Mac machine:

Open terminal and run the following commands:

```
lsof -i:1813 // to list the application using the port

kill $(lsof -t -i:1813) // to kill application on that port

OR

kill -9 $(lsof -t -i:1813) // to kill violently
```

How to run docker on another port

To run idea2life docker on another port, do the following:

1. Goto idea2life repo root
2. Open *docker-compose.yml* file
3. Under idea2life, change ports mapping configuration from *1813:1813* to *yourPort:1813*. (yourPort is the port where you want to access the application)
4. Save the file.

Open terminal and run the following commands:

```
cd <path-to-repo> //you need to be in your repo folder

docker-compose build

docker-compose up
```

idea2Life is now accessible on the port you entered.

5.2 Error:- Docker container crashed

If your node server crashes or docker goes down, please raise a issue on github with details. To restart the application, open terminal and run the following commands:

```
cd <path-to-repo> //you need to be in your repo folder

docker-compose up
```

idea2Life will start again

5.3 Error:- Link between 2 pages is broken, when a page is renamed

If you rename a page then the mapping URL to the page also changes, hence the link between the pages is broken. For fixing this, please go back to the editor and remap the link URL for the corresponding page and save. Now the link should work properly.

5.4 Remove all Dangling Docker images

If you are using docker to build and manage idea2Life it may happen that after running *docker-compose build* multiple times you may start to run out of disk space. To recover some of this disk space you can remove dangling docker images using this command:

```
docker rmi $(docker images -f 'dangling=true' -q)
```

CHAPTER

SIX

REFERENCE

6.1 idea2Life AI Service

Idea2life AI Service is implemented at **/ai** in python language. It is further divided into two separate modules.

First one is server module with main **Flask application** located at **/ai/server** path in repo. Second one is **template_detect module** located at **/ai/template_detect** in repo. Details of these could be find in reference below.

6.1.1 Main Flask application server.app

6.1.2 template_detect module

template_detect.template_detect File inside template_detect module has implementation of object detection code, It receives image from main flask application with path for pre-trained model file and cfg file and in turn calls pyyolo library for object detection.

template_detect.utils File inside template_detect module has implementation of implementation of most of the helper class to be used by template_detect.template_detect methods.

6.2 idea2Life Main Modules

6.2.1 UI/Handler/

Module: `handlers_ai`

Local Navigation

- *Function:* `sendImageToAIService`

Description

This is an example of how to document routes.

Function: sendImageToAIService

Send a post request to AI service with base64 image and runs the callback when results are available.

sendImageToAIService()

Arguments

- **sendImageToAIService()** – response object.
- **sendImageToAIService()** – base64 image clicked by the idea2life.
- **sendImageToAIService()** – default filename generated by system (unix timestamp)
- **sendImageToAIService()** – callback function (to be called when data from AI service is available)

Module: handlers_generator

Local Navigation

- *Function: sendXMLToGeneratorService*

Description

This is an example of how to document routes.

Function: sendXMLToGeneratorService

Send a post request to generator service which .

sendXMLToGeneratorService()

Arguments

- **sendXMLToGeneratorService()** – response object.
- **sendXMLToGeneratorService()** – Send a post request to generator service which .
- **sendXMLToGeneratorService()** – callback function (to be called when data is available from the generator service).

Module: handlers_layout

Local Navigation

- *Function: sendAIDataToLayoutService*

Description

This is an example of how to document routes.

Function: sendAIDataToLayoutService

Send a post request to generator service which .

sendAIDataToLayoutService()

Arguments

- **sendAIDataToLayoutService()** – response object.
- **sendAIDataToLayoutService()** – object containing AI data e.g
- **sendAIDataToLayoutService()** – callback function called when data from AI service is available

6.2.2 Generator/

6.2.3 Layout/

6.2.4 Admin module

Module: AdminRouter

Description

Admin - router.js contains all the requests done in the admin module.

Route: "/" - Go to the Home Page.

Method	Path
GET	/

Route: "/errorpage" - Redirected when error is generated

Method	Path
GET	/errorpage

Route: "/view" - View the entire page list.

Method	Path
GET	/view

Route: "/delete" - Delete the selected page

Method	Path
GET	/delete?page=

Parameters:

Name	Type	Description
page	String	Name of the page to be deleted

Route: "/pagehtmlview" - View a selected page

Method	Path
GET	/pagehtmlview?page=

Parameters:

Name	Type	Description
page	String	Name of the page to be viewed

Route: "/pagenames" - List of all the generated pages

Method	Path
GET	/pagenames

Route: "/pageedit" - Open a page in Editor to make few modifications

Method	Path
GET	/pageedit

Route: "/savepage" - Save the edited page

Method	Path
GET	/savepage

Function: scan

Return a list of files of the specified fileTypes in the provided dir, with the file path relative to the given dir

scan()

Arguments

- **directoryName** – path of the directory you want to search the files for
- **fileTypes** – array of file types you are search files, ex: [‘.txt’, ‘.jpeg’]

Return results This is the page list

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

S

scan() (*built-in function*), 55
sendAIDataToLayoutService() (*built-in function*), 53
sendImageToAIService() (*built-in function*), 52
sendXMLToGeneratorService() (*built-in function*), 52