

Zuzu is a static site generator that takes in markdown files and render htmls pages. [This blog](#), for example, has been written using this generator. This enables noobs like me to write blogs without having to learn a lot of code! It is a very simple and easy to use generator. All you need to do is to write a markdown file and it will be rendered as a page ;) You can create a new page by creating a new markdown file.

- [How Does it work](#)
  - [Create a markdown file.](#)
  - [Run the generator and find your blog.](#)
- [The Static Site Generator](#)
  - [Libraries Used](#)
  - [Workflow](#)
  - [Generator Code.](#)

## How does it work?

Zuzu parses the markdown file using *javascript* and renders it as *html documents*. It then saves the html files in the `public` folder. The public folder, with `index.html` file, is the final output of the generator and this can be deployed and hosted in various platforms. This particular blog has been deployed on [Github Pages](#).

### 1. Create a markdown file

```
# This is a title
This is a paragraph
This is another paragraph
This is a list:
* Item 1
* Item 2
* Item 3
This is a code block:
```
print("Hello World")
```

This is a table:
| Column 1 | Column 2 | Column 3 |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
This is a link: [zuzu](https://anubhavp.me/blog/zuzu.html)
```

### 2. Run the generator and find your blog

Run `npm run generate` in the console. You'll now see the blog in the public folder! Run the `index.html` file in your browser to see your blog. You may now deploy your site to a server.

## The Static Site Generator

### 1. Libraries used

- [MarkdownIt](#) *Markdown parser done right.*
- [MarkdownItAnchor](#) *Header anchors for markdown-it.*
- [Glob](#) *"Globs" are the patterns you type when you do stuff like `ls .js` on the command line, or put `build/` in a `.gitignore` file.*
- [Gray-Matter](#) *Parse front-matter from a string or file.*
- [Mkdirp](#) *Create Dirs if they do not exist.*

## 2. Workflow

This is the code for the `generator.js`. The code works in the following way:

1. `fs.readFile()` from `fs` reads all the files from the said directory and stores then in `filename` using `glob`. It is a `glob` that matches all the files in the directory. The `file system` module allows you to work with the file system on your computer.
2. `gray-matter` helps extracting front matter from the a string or file. Converts a string with front-matter, like this:

```
title: Hello
slug: home
---
<h1>Hello world!</h1>
```

Into an object like this:

```
{
  content: '<h1>Hello world!</h1>',
  data: {
    title: 'Hello',
    slug: 'home'
  }
}
```

It then extracts the front matter and stores it in `data`. It then stores the content in `content` and returns the `filename` to the `main()` function. It then repeats the process for all the files in the directory.

3. The `main()` function then takes in one `filename` at a time and then parses it through `markdownit(, {markdownitanchor})`. `markdownit` parses the file and converts the markdown content into HTML files. It then creates a `html` file and writes the parsed content into it. It then saves the `html` file in the `public` folder. This process repeats for all the files in the directory.
4. The converted html files are stored in the specified directories then using `mkdirp`. The `index.html` file is already present in the `public` folder. `mkdirp` creates the directories if they do not exist.

## 3. Generator Code

```
import fs from 'fs'
import glob from 'glob'
import matter from 'gray-matter'
```

```

import mkdirp from 'mkdirp'
import path from 'path'
import hljs from 'highlight.js';
import MarkdownIt from 'markdown-it'
import markdownItAnchor from 'markdown-it-anchor'
import string from 'string'

const slugify = s => string(s).slugify().toString()

const md = MarkdownIt({
  html: true,
  linkify: true,
  typographer: true,
  highlight(str, language) {
    if (language && hljs.getLanguage(language)) {
      try {
        return hljs.highlight(str, { language: language }).value;
      } catch (err) {
        console.log(err)
      }
    }

    return null;
  }
}).use(markdownItAnchor, { slugify });

const readFile = (filename) => {
  const rawFile = fs.readFileSync(filename, 'utf8')
  const parsed = matter(rawFile)
  const html = md.render(parsed.content)

  return {...parsed, html }
}

const templatize = (template, { date, title, content, author }) =>
  template
    .replace(/<!-- PUBLISH_DATE -->/g, date)
    .replace(/<!-- TITLE -->/g, title)
    .replace(/<!-- CONTENT -->/g, content)
    .replace(/<!-- AUTHOR -->/g, author)

const saveFile = (filename, contents) => {
  const dir = path.dirname(filename)
  mkdirp.sync(dir)
  fs.writeFileSync(filename, contents)
}

const getOutputFilename = (filename, outPath) => {
  const basename = path.basename(filename)

```

```

    const newfilename = basename.substring(0, basename.length - 3) + '.html'
    const outfile = path.join(outPath, newfilename)
    return outfile
  }

  const processFile = (filename, template, outPath) => {
    const file = readFile(filename)
    const outfilename = getOutputFilename(filename, outPath)

    const templatized = templatize(template, {
      date: file.data.date,
      title: file.data.title,
      content: file.html,
      author: file.data.author,
    })

    saveFile(outfilename, templatized)
    console.log(`📄 ${outfilename}`)
  }

  const main = () => {
    const srcPath = path.resolve('content')
    const outPath = path.resolve('public')
    const template = fs.readFileSync('./templates/initial/template.html', 'utf8')
    const filenames = glob.sync(srcPath + '/*.md')

    filenames.forEach((filename) => {
      processFile(filename, template, outPath)
    })
  }

  main()

```