Table of contents

1 Markdown, Notable, 그리고 Hugo

해 보고 싶은 것들을 노트에 기록하고 싶었다. 배워가는 과정을, 해 보는 과정을 추적하고 싶었다. 결과물이나 결론도 중요하지만, 그러한 것들을 얻어내는 과정도 못지 않게 중요하다고 생각했다. 그런 것들이 생기면, 일단은 노트를 만들었다가 조금씩 나중에 추가하고 수정할 수 있었으면 했다. 노트 안에 링크도 달고, 이미지도 추가하고, 필요하면 다이어그램도 넣고 하고 싶었다. 노트가 완성이 되면 운영하는 블로그에 원하는 태그와 함께, 내가 노트에 기록한 것과 같은 혹은 비슷한 포맷으로 자동으로 등록이 되면 참 좋겠다고 생각했다. 더 많은 아이디어가 있지만, 일단은 이것 먼저 구현해보았다.

1.1 그냥 순수 텍스트로 컨텐츠를 생산할 수 있다면 어떨까?

노트를 관리하는 수 많은 어플리케이션이 있다. 내가 그동안 거쳐온 어플리케이션이나 서비스들을 돌아보면, 모두들 각각의 장점과 단점이 있는 거 같다. 그런데 그러한 어플리케이션이나 서비스들 중 대다수가 그들 자신의 포맷을 사용하거나, 추후 재사용이 쉽지 않은 포맷을 사용하는 점이 성가셨다.

내 자신에 대한 질문에 답하기 위해 구글링을 해 보고, Markdown 포맷을 알게 되었다. 직장 동료가 개발 문서를 어떤 형식에 기반해서 작성 후에, copy & paste해서 google document로 변환 후 등록한 걸 볼 때만 해도, 왜 저렇게 불편하게 하고 있을까 하고 그냥 지나쳐 버렸다. 나중에 Markdown을 알게 되고 보니, 그게 Markdown이라는 거였다.

1.2 할 일 관리도 할 수 있을까?

그리고 나서, GFM(Github Flavoured Markdown)도 알게 되었다. 다음처럼 할 일 목록, 테이블, 취소된 텍스트, 그리고 자동 링크가 지원되는 변형된 버전이었다.

1.2.1 할 일 목록

```
- [ ] to do 1
- [ ] to do 2
  - [ ] to do 2-1
  - [X] to do 2-2
- [X] to do 3
```

- [ ] to do 1
- [ ] to do 2
  - [ ] to do 2-1

    - [X] to do 2-2
- [X] to do 3

1.2.2 테이블

foo	bar
baz	bim

foo	bar
baz	bim

1.2.3 취소된 텍스트

아베 ~~바보~~ 화이팅!

아베 바보 화이팅!

1.2.4 자동 링크

https://google.com

https://google.com

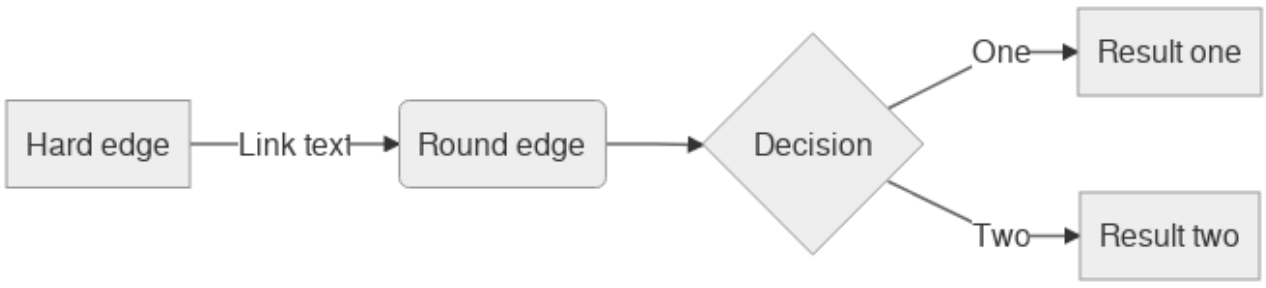
1.3 다이어그램도 가능할까?

GFM이 테이블을 지원하는 걸 알고 나니 개발 문서에 사용이 가능한지 알아보고 싶었다. 그런데, 개발 문서에 가끔 필요한 것이 다이어그램이었다. 그동안은 Google document로 작성시, google diagram을 작성한 후 첨부하는 방식으로 해 왔다. 그런데, 그 다이어그램을 사내 Google drive에 접근이 안되는 사람 보내는 이메일에 첨부해서 보내거나 하는 경우에는, 이미지로 변환 후 첨부해야 하는 점이 생각이 났다.

모든 걸 알려주는 구글링이 Mermaid로 이끌어 주었다. Markdown 파일 안에 간단하게 텍스트로 flow chart, sequence diagram, gantt chart를 표현할 수 있었다.

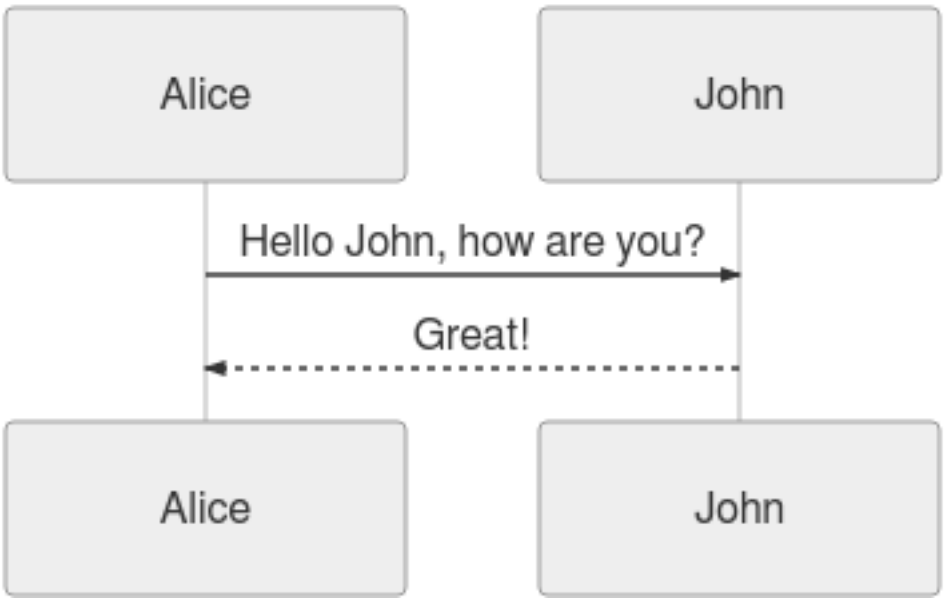
1.3.1 Flow chart

```
graph LR
  A[Hard edge] -->|Link text| B(Round edge)
  B --> C{Decision}
  C -->|One| D[Result one]
  C -->|Two| E[Result two]
```



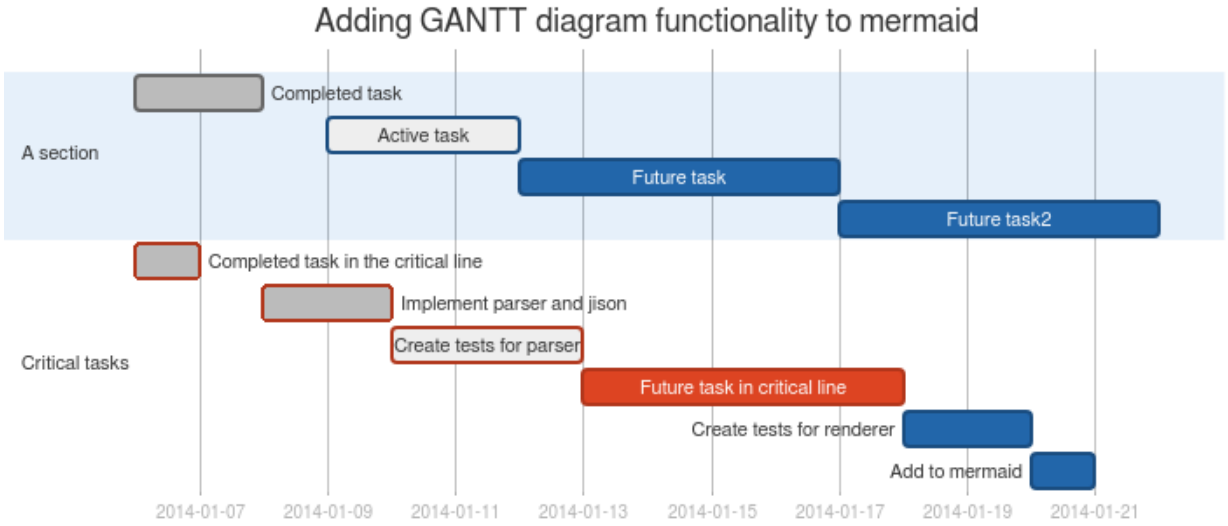
Sequence diagram

```
sequenceDiagram
    Alice->>John: Hello John, how are you?
    John-->>Alice: Great!
```



Gantt chart

```
gantt
        dateFormat  YYYY-MM-DD
        title Adding GANTT diagram functionality to mermaid
        section A section
        Completed task            :done,    des1, 2014-01-06,2014-01-08
        Active task               :active,  des2, 2014-01-09, 3d
        Future task               :         des3, after des2, 5d
        Future task2              :         des4, after des3, 5d
        section Critical tasks
        Completed task in the critical line :crit, done, 2014-01-06,24h
        Implement parser and jison          :crit, done, after des1, 2d
        Create tests for parser             :crit, active, 3d
        Future task in critical line        :crit, 5d
        Create tests for renderer           :2d
        Add to mermaid                      :1d
```



1.4 전용 포맷 vs 일반 텍스트 파일

경험해 봤던 대부분의 어플리케이션/서비스가 전용 포맷을 사용한다. 필요시 export해서 다른 포맷으로 바꾼 후 다시 import할 수도 있다. 하지만 전용 포맷이 사용된다는 점은, 다른 어플리케이션/서비스로 넘어갈 때마다 장벽으로 작용한다. 그래서 일반 텍스트 파일이 사용 가능했으면 했다. 일반 텍스트 파일로 노트가 관리된다면, git나 변경이력이 지원되는 클라우드 서비스에 파일이 관리되는 경우에는, 노트의 변경 이력을 볼 수도 있을테니까. 노트 어플리케이션이나 서비스에 의존하지 않고, 자유롭게 내용 검색도 가능할 것이었다.

1.5 파일명

일반 텍스트 파일로 관리된다고 가정하면, 그 다음에 생각해 볼 것으로는 파일명일 것이다. 이 파일명이 uuid 등, 파일 구분을 위해 랜덤하게 만들어지는 경우에는 파일명 자체는 정보로서의 가치가 0일 것이다. 어플리케이션이나 서비스 측면에서라면, 파일명과 노트의 제목을 구분해서 관리할 수 있어서 구현 측면에서 장점이 많다고 할 수 있을 지 모른다. 하지만, 노트가 일반 텍스트 파일로 관리되는 경우라면, 파일명 또한 사용자가 부여된 대로 정해지는 것이 더 좋다고 생각된다. 파일 관리자에서 노트 파일을 봤을 때 파일명 = 노트 제목이면 검색이나 노트 구분에 있어서도 훨씬 유용할 것이다.

1.6 노트 분류

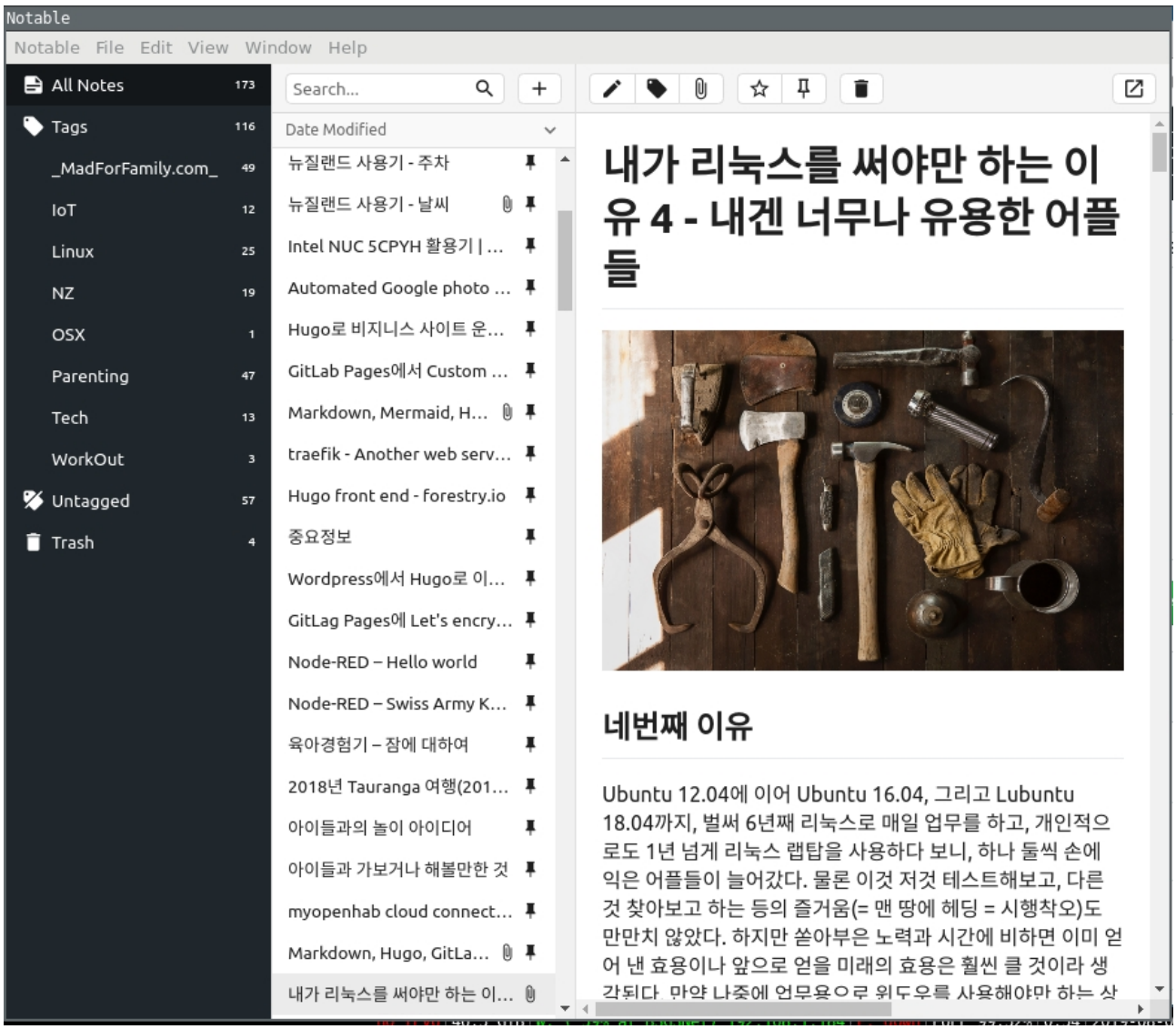
이제, 노트가 유의미한 파일명에 일반 텍스트로 관리된다고 가정해보자. 노트 제목이 파일명이라고는 해도, 그 제목만 가지고 노트를 분류하기가 쉽지 않다. 물론 접두어를 사용해서도 가능하겠지만, 제목과는 별도로, 노트를 분류할 수 있어야 한다고 생각한다. 다만 그 분류가 별도의 트릭이나 db 등이 아니라 노트 파일 자체가 저장되어야, 그 노트를 별도의 어플리케이션에서 열어봤을 때에도, 분류 정보 또한 그대로 참조할 수 있을 것이다. 노트 분류가 여러 단계로 되어도 좋지만, 1 단계만 되어도 실제 사용에서는 충분하리라 생각된다.

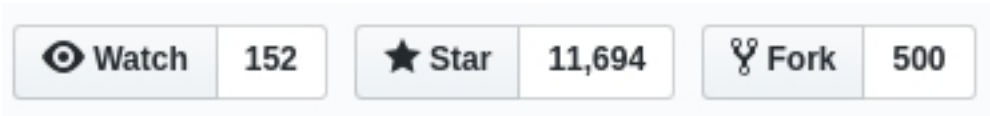
1.7 어떤 어플리케이션 혹은 서비스가 좋을까?

노트를 적는다는 것은 아주 일반적인 행위라, 세상에는 수 많은 어플리케이션과 서비스가 항상 있어왔다. 당연하게도 그 모두를 사용해보지 못했다. 하지만, 현재 사용중인 Notable이, 여태까지 찾아본 어플리케이션과 서비스 중, 위에 나열한 요구 조건 모두를 만족시키는 그 대답 중 하나라는 점은 확실하다.

1.8 Notable

Joplin은, cli, 모바일 앱 지원, 웹 클리핑 extension 지원 등 아주 아주 아주 강력한 후보였지만, uuid 파일명과 특정 파일에 분류 정보가 보관되는 점 때문에 결국 다시 찾고 찾아 Notable로 최종 정착했다.





Github에서 Star 수를 보면 거의 1만 2천에 육박하고 있다. 아마도, 전용 포맷을 사용하지 않고 일반 텍스트 파일로 관리된다는 점이 그 주요한 이유 중에 하나가 아닐까 싶다.

1.8.1 파일 관리

파일 관리 구조는 다음처럼 간단하다. Data 디렉토리 아래에 notes와 attachments 디렉토리가 사용된다. 예상되다시피, *.md는 notes 디렉토리에, 첨부된 파일들은 모두 attachments에 보관된다. 따라서, 동일한 제목의 노트를 생성할 수 없고, 동일한 첨부파일을 업로드할 수 없다. 이러한 제약이 싫은 경우라면 Notable을 절대 사용해서는 안되지만, 이러한 제약을 수용함으로써 얻어지는 장점을 선택해야만 하는 사람이라면 Notable은 당연한 선택이 될 것이다. 비록, Notable이 제공하지 못하는 기능(모바일앱, 웹클리핑 등)에도 불구하고 말이다.

```
/path/to/your/data_directory
├──┬ attachments
│  ├── foo.ext
│  ├── bar.ext
│  └── ...
└──┬ notes
   ├── foo.md
   ├── bar.md
   └── ...
```

1.8.2 Meta data

```
---
attachments: [build.sh, duplicated_titles.jpg, notable_stars.jpg, notable_tags.jpg, notable.jpg, notetaking.jpg, relref.jpg]
pinned: true
tags: [Tag 1/Tag 2, Tech]
title: 'Markdown, Notable, 그리고 Hugo'
created: '2019-04-29T09:14:11.205Z'
modified: '2019-08-04T02:16:46.135Z'
---
```

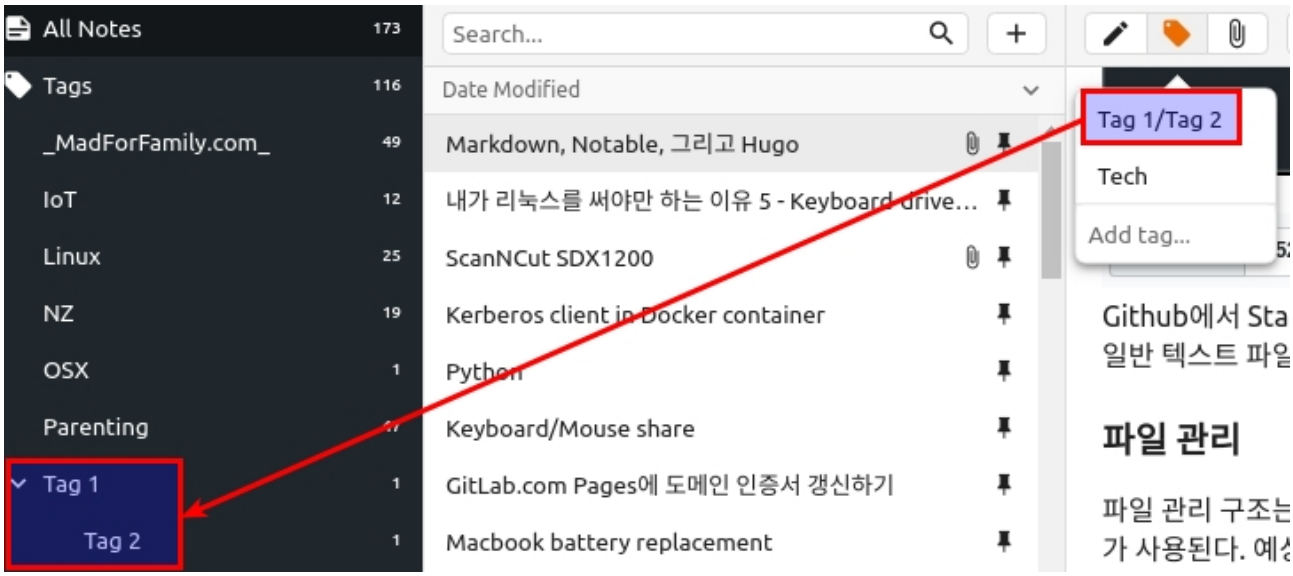
지금 작성하는 이 노트의 meta data이다. 이 영역 아래로는 그저 일반 텍스트일 뿐이다. 보는 것처럼 첨부된 파일들의 목록, tag들, 파일명, 생성/수정 일시가 들어있다.

1.8.3 파일명

노트 안에 첫번째 헤더(#)와 동기된다. 즉, 첫번째 헤더를 고치게 되면 파일명 또한 변경된다. 그런데, 동일 제목 방지가 제대로 되지 않는 것 같다. 여하튼 주의할 필요가 있다. 아무래도 이슈 리포트를 해야 할 듯 하다.

1.8.4 태깅

태그는 무한 단계가 지원되지만, 입력시 그 단계를 모두 입력해야 한다. 즉 "tag 1", "tag 1/tag 2" 같은 식으로 입력하면 그 레벨에 맞게 표시된다.



1.9 Notable + Hugo

Static site generator - Hugo에도 썼지만, 개인 블로그를 Hugo + Academic theme + GitLab Pages로 운영중이다. Notable에서 _MadForFamily.com_를 가진 노트를 저장하면, FSWatch + Git-Sync가 자동으로 Git repo에 Push하고, 그 Git Repo의 CI(.gitlab-ci.yml)에서, _MadForFamily.com_ 태그가 달린 Markdown 파일(약간의 처리 작업)과 그 파일들에서 사용되는 첨부파일들을 GitLab Pages를 가진 또 다른 Git repo로 Push하는 방식이다. 이렇게 함으로써, Notable에서의 노트 작업이 개인 블로그로 자동으로 Publish되고, 싱크되도록 해놨다.

1.9.1 Ugly URL

Notable에서 작성한 노트 파일을 Hugo에서 사용하려면 맨 먼저 Ugly URL이 사용되도록 Hugo를 설정해야 한다. 위에서 살펴본 것처럼, Notable의 노트 파일은 notes에, 첨부 파일은 attachments에 보관하기 때문이다. Notable에서 작성한 노트 파일이 content/post/hello-hugo.md라면, Pretty URL은 http://localhost:1313/post/hello-hugo/에서 접근 가능하고 Ugly URL이라면 http://localhost:1313/post/hello-hugo.html에서 접근 가능하다. 그런데 만약 노트 안에 첨부 파일로의 링크 ./attachments/attached1.jpg가 있다면 Pretty URL의 경우 참조가 불가능한. 그래서 Ugly URL을 써야 한다.

1.9.2 디렉토리 구조 비교

Notable의 디렉토리 구조

```
.
├── attachments
└── notes
```

모든 첨부파일은 attachments 디렉토리에, 모든 markdown 파일은 notes 디렉토리에 보관이 된다.

Hugo의 content 디렉토리 구조

```
.
├── attachments
│   └── image.jpg
└── post
    └── hello.md
```

Notable과 동일한 구조를 이용하기 위해 attachments 디렉토리를 생성했다. Notable의 notes 디렉토리의 파일은 post에, Notable attachments의 파일은 attachments에 복사될 예정이다.

1.9.3 Notable 태그

개인적으로는 _MadForFamily.com_이라는 태그를 가지고 Hugo로 보낼 지를 구분하도록 사용중이다. 그 이외의 태그는 Hugo에서도 tag로 사용된다.

1.9.4 게시물간 링크

Notable은 [링크 타이틀](./markdown_파일명)를 로컬 파일에 대한 링크로 인식한다. Notable 안에서 그렇게 작성된 링크를 클릭할 경우 해당 파일을 열어준다. 주의할 점은 반드시 ./로 시작해야 한다는 점이다.

그런데 이런 로컬 파일로의 링크가 Hugo에서는 동작하지 않는다. Hugo는 기본적으로 markdown으로 작성된 컨텐트를 url이나 .html로 서비스하기 때문이다. 그래서 Notable이 인식하는 markdown 파일로의 파일 링크를 Hugo가 해석할 수 있도록, Hugo의 relref라는 short code로 변경해야 한다.

1.9.5 파일 이름과 첫번째 헤더

Notable은 Markdown 파일 내의 첫번째 헤더를 파일명으로 사용한다. 현재 사용중인 Hugo Academic theme에서는 파일명과 첫번째 헤더가 같은 경우에 2번 연속으로 출력되기 때문에, 첫번째 헤더를 제거해서 Hugo로 처리해야 한다.

1.9.6 사용중인 변환 스크립트

2019년 8월 기준으로 현재 사용중인 build.sh 파일이다. 위에 기재된 사항들을 해내는 스크립트이다.

1.9.7 사용중인 .gitlab-ci.yml

```
build:
  stage: deploy
  script:
    - bash build.sh
```

이렇게 해두면 Notable의 git repo에 변경사항 발생시, 대상 사이트로 markdown 파일들과 사용되는 첨부파일들이 싱크가 되고, 빌드가 된 후 웹사이트에 나타나게 된다.

1.10 이 모든 걸 자동으로 해보자

먼저 Notable을 사용하는 pc에서 Git-Sync와 FSWatch를 적용하면, Notable에서 markdown 파일의 변경사항이 저장될 때마다 자동으로 remote repo에 push가 된다. Remote git repo에서는 위에 기재된 build.sh과 .gitlab-ci.yml에 의해, _MadForFamily.com_ tag를 가진 markdown 파일들과 첨부파일들이 madforfamily.gitlab.io로 싱크가 된다. madforfamily.gitlab.io에 변경사항이 push되면, 싱크된 파일들로 사이트가 새로 빌드된다. 바로 이러한 과정을 거쳐서 이 게시물이 Notable에서 사이트로 Publish되는데, Notable에서 수정할 때마다 이 과정이 이뤄지므로, 한 곳에서만 수정해도 되는 잇점이 있다. 그런데, 주의할 점이 있다. Publish한 노트들의 경우, 제목을 수정하면 안된다. 제목이 곧 파일명이고 파일명이 곧 URL이 되기 때문이다.

그런데, Publish된 게시물을 copy & paste해서 여타의 사이트에 등록한 경우라면, 현재는 수동으로 업데이트해줘야 한다. 나중엔 그 마저도 자동화하고 싶은 욕심이다. 언제가 될런지는 모르겠다.