연구주제

(크로스 플랫폼) 웹 브라우저 탭 저장소

1일차

크롬 Extension과 Apps의 차이를 연구하였다.

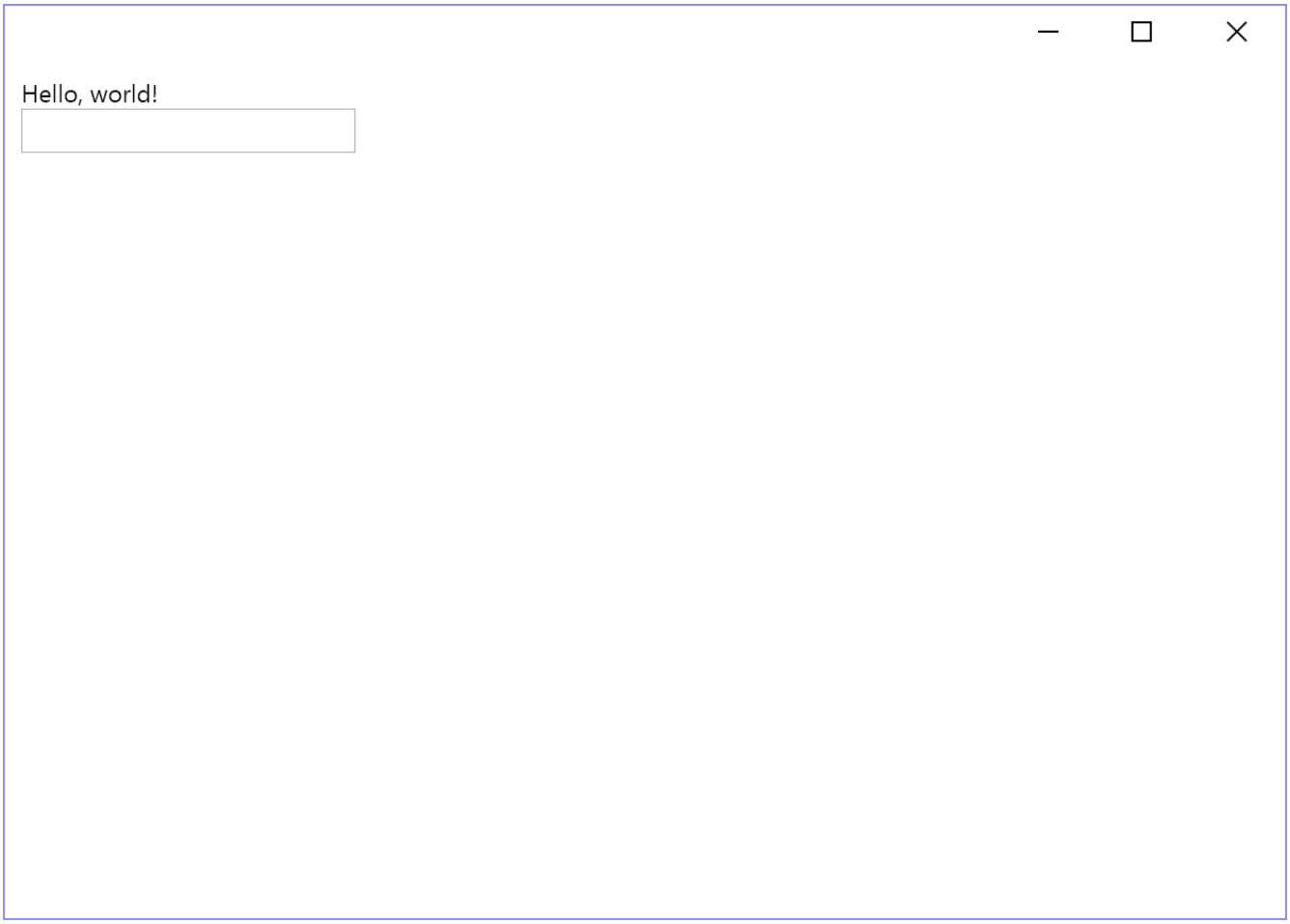
https://developer.chrome.com/extensions

https://developer.chrome.com/extensions/getstarted

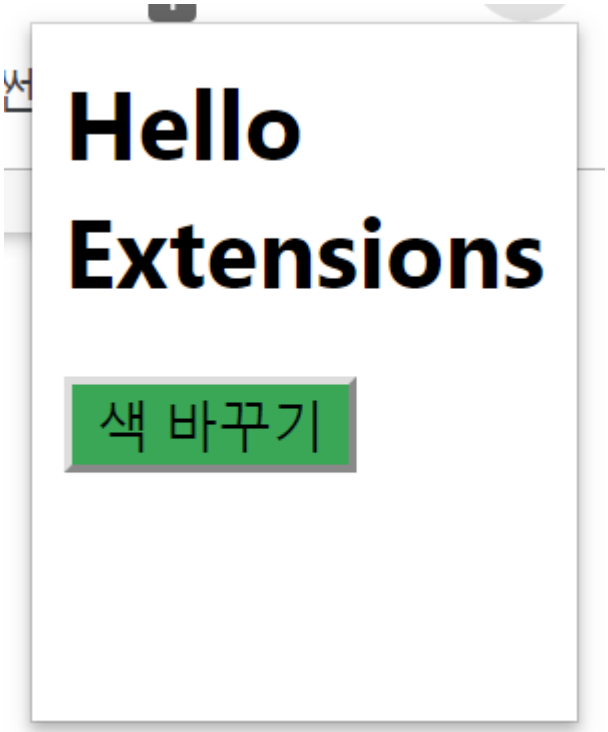
Chrome Apps에서는 웹앱의 형태로, Postman처럼 윈도우에서 사용할 수 있도록 한다. 내가 생각했던 Electron의 형태와 유사한 듯 하다.

Chrome Extension은 크롬 확장프로그램으로 직접 탭을 추가하는 기능을 사용할 수 있을 것이다.

Chrome Extension으로 탭 추가 기능을 하되, 폴더 및 정리 기능은 Chrome Apps에서 지원하는 것을 지향해야 할 것이다.



(Chrome App은 데스크탑에서 작동가능한 응용프로그램의 형태로 구동된다)



(Chrome Extension은 흔히 사용하는 크롬확장프로그램의 형태로 용도에 적합한 형태임을 확인할 수 있다.)

2일차

Chrome Extensions API를 검색하며, 다양한 기능 중 가장 필요하다고 생각하는 Storage에 대해 연구하였다.
Storage 기능은 크롬에 저장 및 Stroage 기능을 활용하여 저장하고, 서버에 저장 및 동기화 작업을 수행할 것이다.

> chrome.storage.sync.set({'key': 'value'} , function(){ /* Write code in here */ })

message('write anything in here') 함수를 이용하여 로그를 남길 수 있음을 확인하였다.

```
chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
  chrome.tabs.sendMessage(tabs[0].id, {greeting: "hello"}, function(response) {
    console.log(response.farewell);
  });
});
```

이와 비슷하게 message를 탭간의 통신에 활용하는 방법도 있는데, 특정한 탭에 데이터를 전송할 때는 탭을 찾아 값을 보내고, 값이 성공적으로 보내졌을 때 함수가 실행되는 방법으로 실행된다.

```
chrome.runtime.sendMessage({greeting: "hello"}, function(response) {
  console.log(response.farewell);
});
```

더 간단한 방법을 사용할 수 있지만, 이 방법은 특정한 탭에 데이터를 전송할 때 사용하기에는 어렵다.

```
chrome.runtime.onMessage.addListener(
  function(request, sender, sendResponse) {
    console.log(sender.tab ?
                "from a content script:" + sender.tab.url :
                "from the extension");
    if (request.greeting == "hello")
      sendResponse({farewell: "goodbye"});
  });
```

메시지에 응답할 때는 보낸 탭의 정보(sender)와 요청(request) 및 응답(sendResponse)가 함수의 형태임)을 보낼 수 있다. 이때 주의해야 할 점은 sendResponse가 동기 방식으로 진행된다는 것이다. 비동기 방식으로 진행할 경우에는 return true를 통해 기능을 수행할 수 있다.

Sync(Blocking) vs Async(Non-Blocking)

프로그램 개발에서 자주 헷갈릴 수 있는 개념이기에 연구해보았다. 동기 방식은 수행해야 할 작업을 순차적으로 진행한다. 하지만 비동기 방식은 주로 콜백 방식을 통하여 작업이 끝날 때까지 다른 작업을 수행할 수 있다는 장점이 있다. 이 프로젝트에서는 동기 방식과 비동기 방식을 적절히 활용하여 논리 오류를 발생시키지 않을 것이다.

```
var port = chrome.runtime.connect({name: "knockknock"});
port.postMessage({joke: "Knock knock"});
port.onMessage.addListener(function(msg) {
  if (msg.question == "Who's there?")
    port.postMessage({answer: "Madame"});
  else if (msg.question == "Madame who?")
    port.postMessage({answer: "Madame... Bovary"});
});
```

두 탭간에 통신을 지속적으로 주고받을 때의 메시지 방식은 Long-lived connections를 사용한다. port를 chrome.runtime.connect를 통해 연결을 시도하고, 이를 통해 지속적으로 메시지를 보낸다. 메시지에 응답할 때는 addListener를 사용한다. 메시지 전송은 postMessage를 사용한다.

```
port.onMessage.addListener( function(msg){

});
```

사용된 함수 : console.assert

```
console.assert( condition[boolean], error message )
```

은 condition이 false일 때(보통은 오류 발생시에 해당한다) message를 출력한다.

3일차

크롬 확장 프로그램의 어떤 기능을 사용할 것인지 정의하고, 본격적인 개발을 위한 준비를 한다.

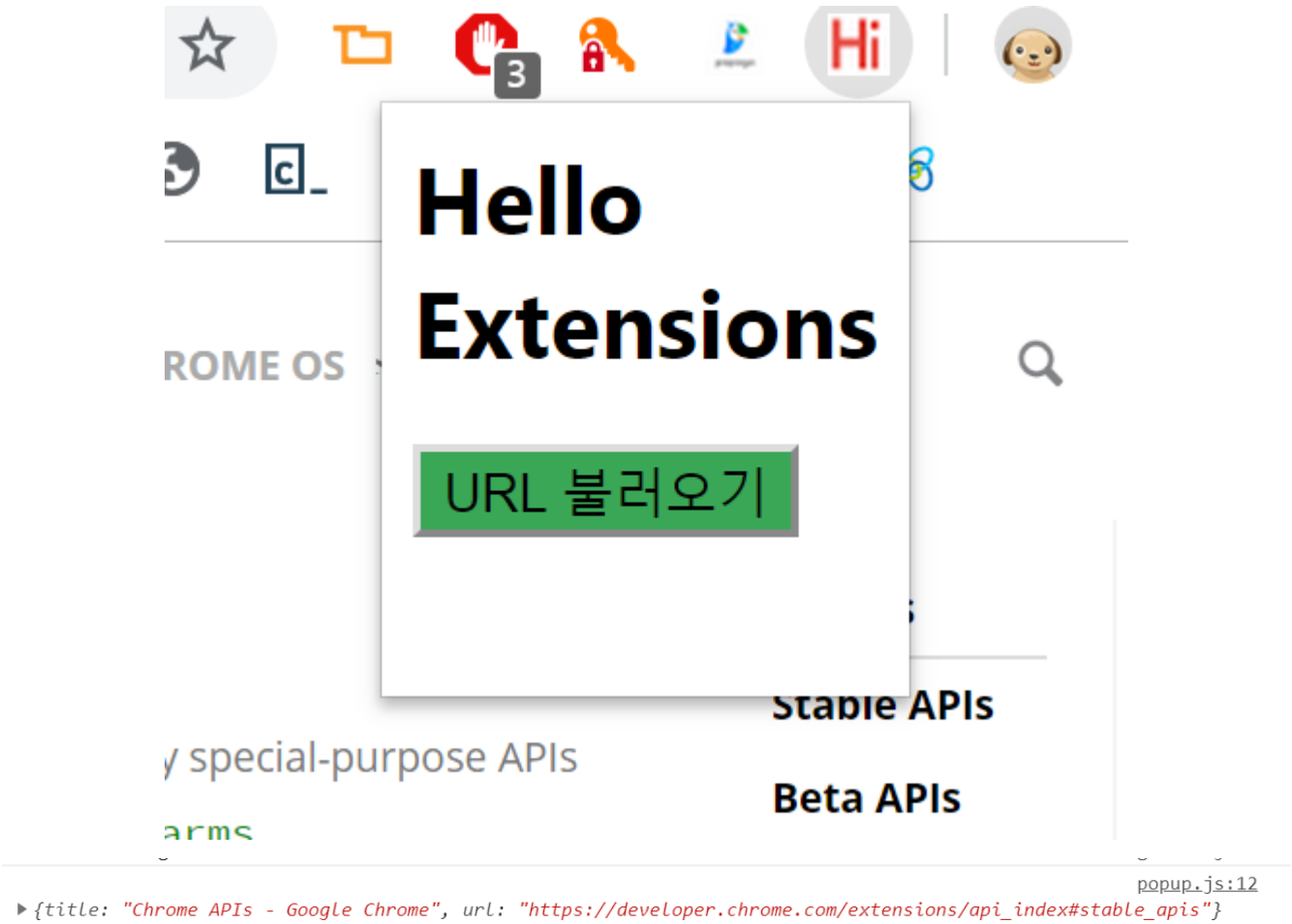
추후의 개발 계획

- 저장된 탭 불러오기
- 저장된 탭에서 새 탭으로 연결하기
- 폴더에 탭 저장하기(storage)
- 열린 탭 확인 및 저장하기
- 탭 / 폴더 이름 설정하기
- 탭 / 폴더에 태그 설정하기
- 탭/폴더/태그 검색 기능 추가하기
    - 태그는 #을 통해, 탭 및 폴더는 텍스트로
- 중복되는 경우 하나만 사용할 것인지 묻기
- 최근 본 탭 다시 열기
    - 한번에 많은 탭이 닫혔을 경우 뱃지로 다시 열 것인지 묻기
- 서버에 탭 동기화하기
    - 2중 암호화를 통해 안전하게 보관하기
    - 탭 저장과 동시에 이루어지지만, 서버에 새로운 정보가 입력되었을 수 있으므로 N분 간격으로 동기화 한다. 수동 동기화가 가능하도록 한다.
- 서버에 있는 탭/폴더 이름 변경하기
- 서버에 로그인하기
- 회원가입하기(사용하는 이메일)
- 방문 횟수 저장하기(빈도)
- 크롬 앱을 통하여 화면에 표시하기

열린 탭 확인하기 기능을 구현할 것이다.

https://developer.chrome.com/extensions/tabs

chrome.tabs에서 지원하는 기능을 통해 현재 열린 탭의 url과 title을 불러올 것이다.



다음과 같이 성공적으로 title과 url을 불러올 수 있었다.

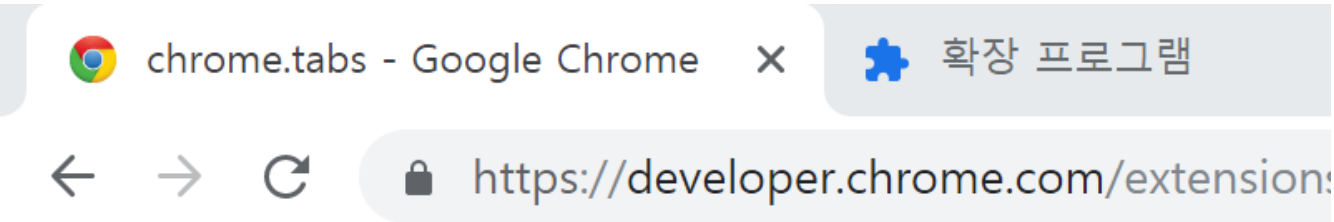
이 때 백그라운드 페이지의 콘솔에 로그를 출력하기 위하여 chrome.extension.getBackgroundPage()로 인스턴스를 불러와 console.log를 사용하였다.

하지만, 현재 탭의 정보만 불러오진다는 문제점이 있다.

4일차

chrome.tabs.query에서 currentWindow와 active 옵션을 모두 true로 하였기에 발생한 문제였다. 조건 active를 true로 하면 크롬 창에서 선택된 탭 하나씩만 불러올 수 있었다. 따라서 조건을 조금 변경하여 currentWindow만 true로 설정하였다. 그 결과 모든 탭의 정보가 불러와지는 것을 확인할 수 있었다.

```
▼0:
    active: true
    audible: false
    autoDiscardable: true
    discarded: false
    favIconUrl: "https://www.google.com/images/icons/product/chrome-32.png"
    height: 588
    highlighted: true
    id: 322
    incognito: false
    index: 0
  ▶ mutedInfo: {muted: false}
    pinned: false
    selected: true
    status: "complete"
    title: "chrome.tabs - Google Chrome"
    url: "https://developer.chrome.com/extensions/tabs#method-query"
    width: 1280
    windowId: 456
  ▶ __proto__: Object
▼1:
    active: false
    audible: false
    autoDiscardable: true
    discarded: false
    favIconUrl: ""
    height: 588
    highlighted: false
    id: 474
    incognito: false
```

정상적으로 두개의 탭이 확인된다.

이제 탭의 정보 중 필요한 정보를 저장할 수 있도록 하는 기능을 구현한 후, 기본적인 UI를 작업할 것이다.

필요한 정보의 저장을 위해서 Storage를 사용할 것인데, 이 기능이 데이터를 반영구적으로 보관할 수 있는지 정확하지 않았다. 이 기능은 나중에 서버에 동기화하는 방식을 통해야 데이터 보관을 안전하게 수행할 수 있을 것이다.

```
chrome.storage.sync.set({key: value}, function() {
        console.log('Value is set to ' + value);
    });
```

데이터 저장에 앞서서 데이터의 형식을 기본적으로 정의할 필요가 있다고 생각하여, 필요한 데이터를 분류하였다.

tabID	탭의 ID(재설정)
title	타이틀 텍스트
favicon?	Favicon의 링크
url	URL 정보
index	탭의 순서

5일차


```
loadUrlElement.onclick = function(element) {
  chrome.storage.sync.get('code', function(result) {
    chrome.extension.getBackgroundPage().console.log(result.code);
  });
};
```

위의 코드를 이용하여 저장된 Url을 불러오는데 성공했다. 이제 각 url별로 해쉬코드를 만들고, 각 해쉬코드를 저장하는 폴더를 만들어야 한다. 또한 저장 기능을 각 사용자별로 이용할 수 있도록, DB 서버를 구축해야 한다. 진행 상황을 버전관리프로그램인 git을 활용하여 기록하기 시작했다.

https://github.com/s2q1ne/tab-storage-extension/

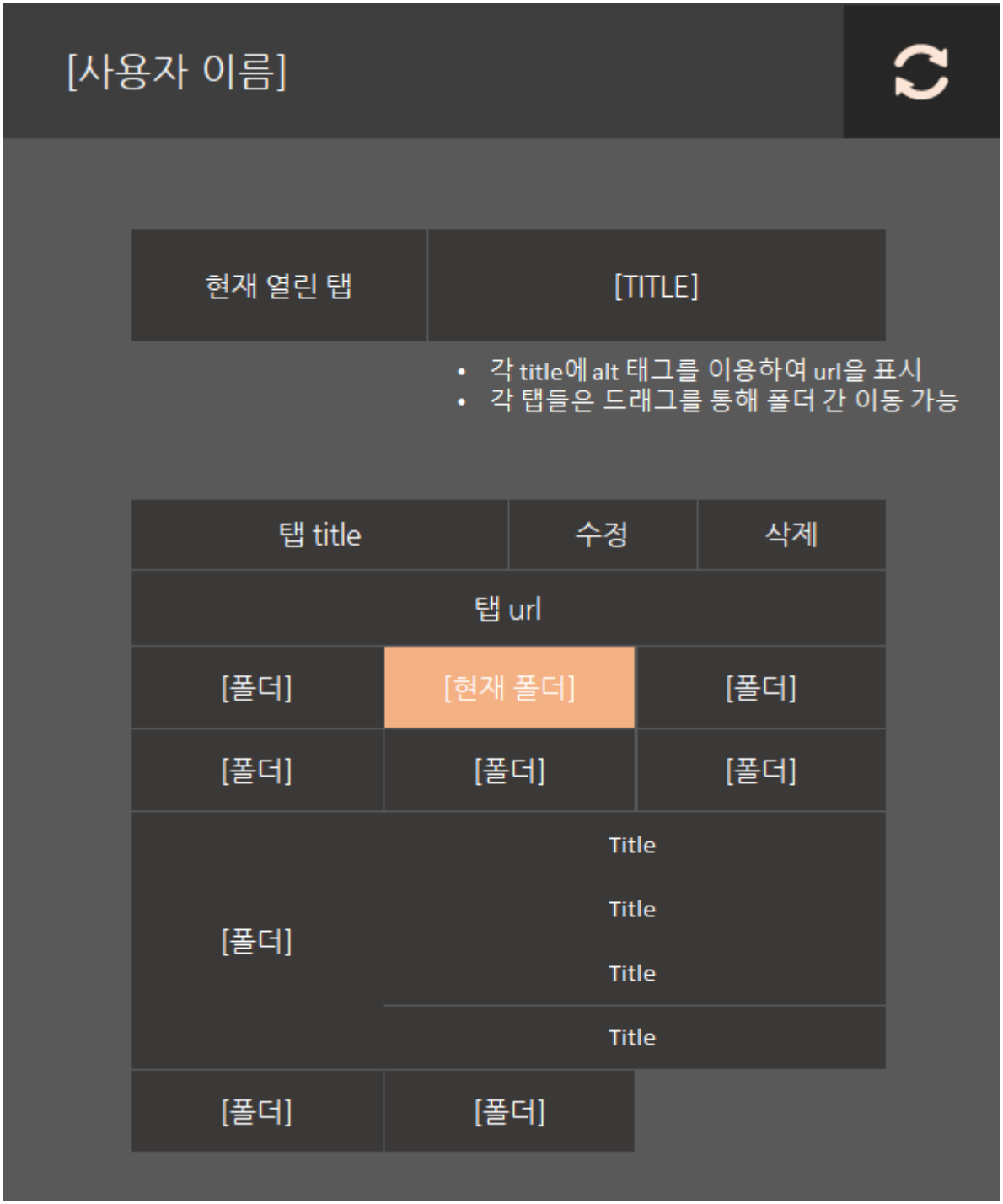
해쉬코드를 생성할 때는 암호화를 적용하여 SHA3-256 기술을 사용할 것이다.

암호화 모듈은 https://github.com/emn178/js-sha3의 소스를 사용할 것이다.

6일차

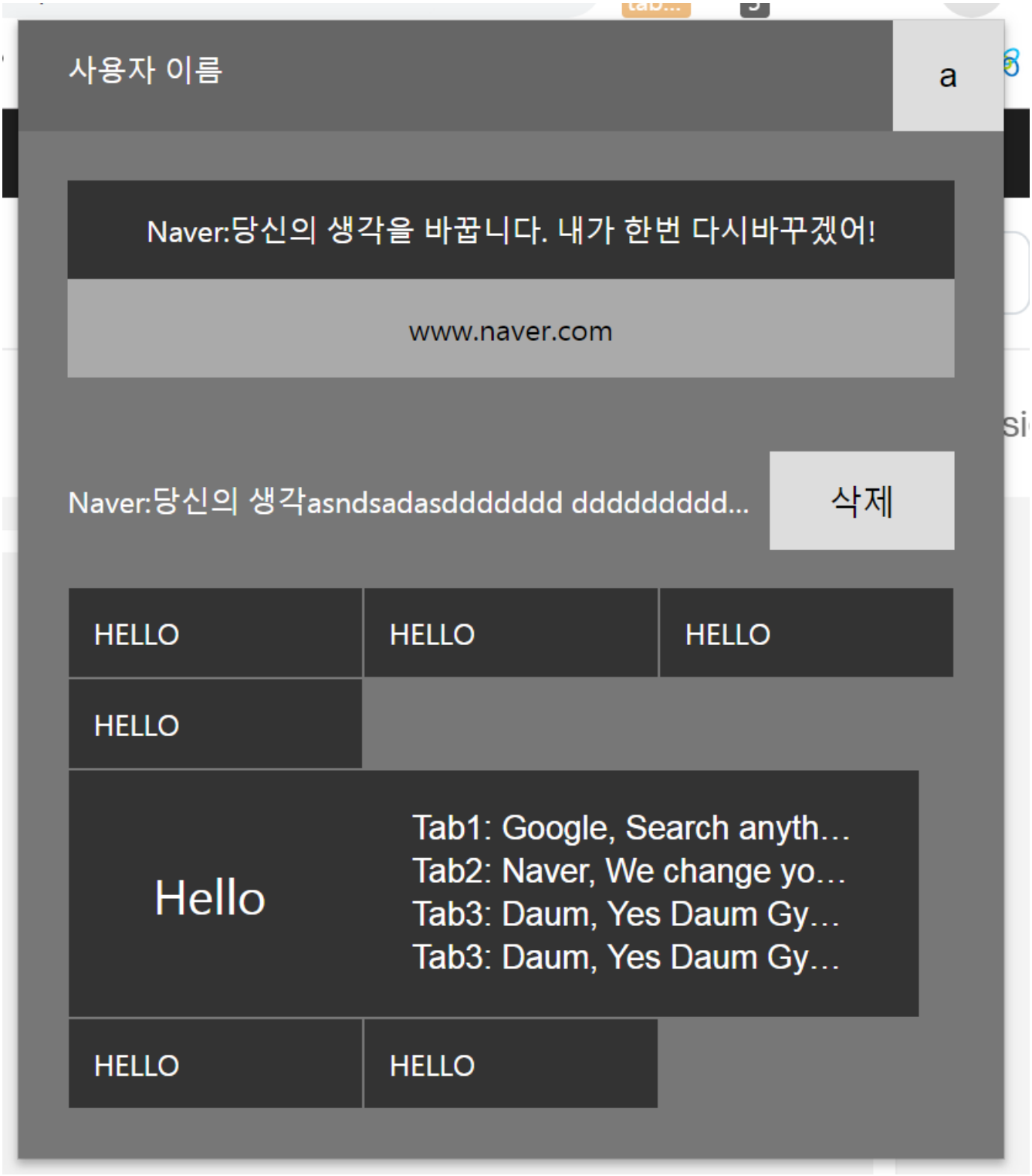
우선 완성도보다는 빠른 완성을 목표로 하여, 암호화 기술, 서버 저장 기술은 나중에 적용하고 프론트엔드부터 우선 제작한다.

- Storage에 URL 저장하기
- 현재 열린 탭바로 저장하기(폴더)
- 폴더에 URL 저장 바로하기



위와 같은 형태로 프론트 엔드를 제작할 것이다.

7일차



프로토타입을 html 코드와 css 코드를 활용하여 작성하였다. 위의 디스플레이를 실제로 동작하도록, 크롬 API를 적용할 것이다.

8일차

- tabDataUpdate
- tapDataUpdateAndTabElementUpdate
- tabElementUpdate
- update

- updateCurrentTab
- displayMyTab
- saveTheTab

  탭 데이터를 동기화하고 storage에 저장할 수 있도록 위의 함수를 구현하였다.

```
function tabDataUpdate(){
  chrome.storage.sync.get('tabData', function(data){
    data_tabs=data.tabData;
    console.log(data.tabData)
  })
}
```

위의 코드를 통해 storage에 저장된 데이터를 불러와 data_tabs 변수를 다시 불러올 수 있었다. 이 방식은 추후에 getData라는 함수를 통해 각 함수마다 다른 변수를 사용하게 할 것이다.

```
function tabElementUpdate(){
  folders.innerHTML = "";
  for( let key in data_tabs){
    let element = data_tabs[key];
    let tab = document.createElement('DIV');
    tab.title = element.url;
    tab.innerHTML = element.title;
    tab.className = "folder";
    tab.onclick = function(e){
      chrome.tabs.create({
        url: tab.title,
        active: true
      }, function(){})
    }
    folders.appendChild(tab);
  }
}
```

위는 불러온 데이터를 바탕으로 팝업창 안의 블록들을 업데이트 시켜주는 과정을 한다. tab이라는 변수에 element를 생성 후, #folders에 tab을 child로 추가시켜주는 과정을 거친다. 또한 tab을 누를 경우 바로 탭을 만들어 이동할 수 있도록 하였다.

```
function update(){
  updateCurrentTab();
  tabDataUpdateAndTabElementUpdate();
}
```

update함수는 추후에 더 복잡해질 초기화 작업을 편리하게 해주기 위해 작성하였다.

```
function updateCurrentTab(){
  chrome.tabs.query({
    active: true,
    currentWindow: true
  }, function(ctabs){
    let ctab = ctabs[0];
    currentTabTitle.innerHTML = ctab.title;
    currentTabTitle.title = ctab.url;
  })
}
```

updateCurrentTab함수는 chrome.tabs.query함수를 통해 자기 자신의 탭을 불러온 후, 현재 탭의 정보를 알려주는 element에 정보를 대입하는 역할을 한다.

```
function saveTheTab(){
  chrome.tabs.query({
    active: true,
    currentWindow: true
  }, function(currentTabs){
      let cTab = {
        title: currentTabs[0].title,
        url: currentTabs[0].url
      };
      data_tabs[SHA256(cTab.url)]=cTab;
      chrome.storage.sync.set({'tabData': data_tabs}, function() {
        tabElementUpdate();
    });
  })
}
```

saveTheTab함수는 현재 탭을 빠르게 저장할 수 있도록 하는 함수로, 자신의 탭을 chrome.tabs.query함수를 통해 불러온 뒤 앞서 연구한 SHA256 함수를 통해 data_tabs에 저장한다. 그리고 업데이트된 정보를 다시 chrome.storage.sync.set함수를 통해 업데이트 하였다.

앞으로 data를 서버에 동기화 하는 작업과 폴더 기능을 더하면 초기에 구상한 프로그램의 모습과 가까울 것으로 예상된다.