크롬 익스텐션에서 데이터를 어떻게 다룰 것인가? (피트스탑과제 – 크롬 익스텐션 개발기)

Oct.08.2021 남현우

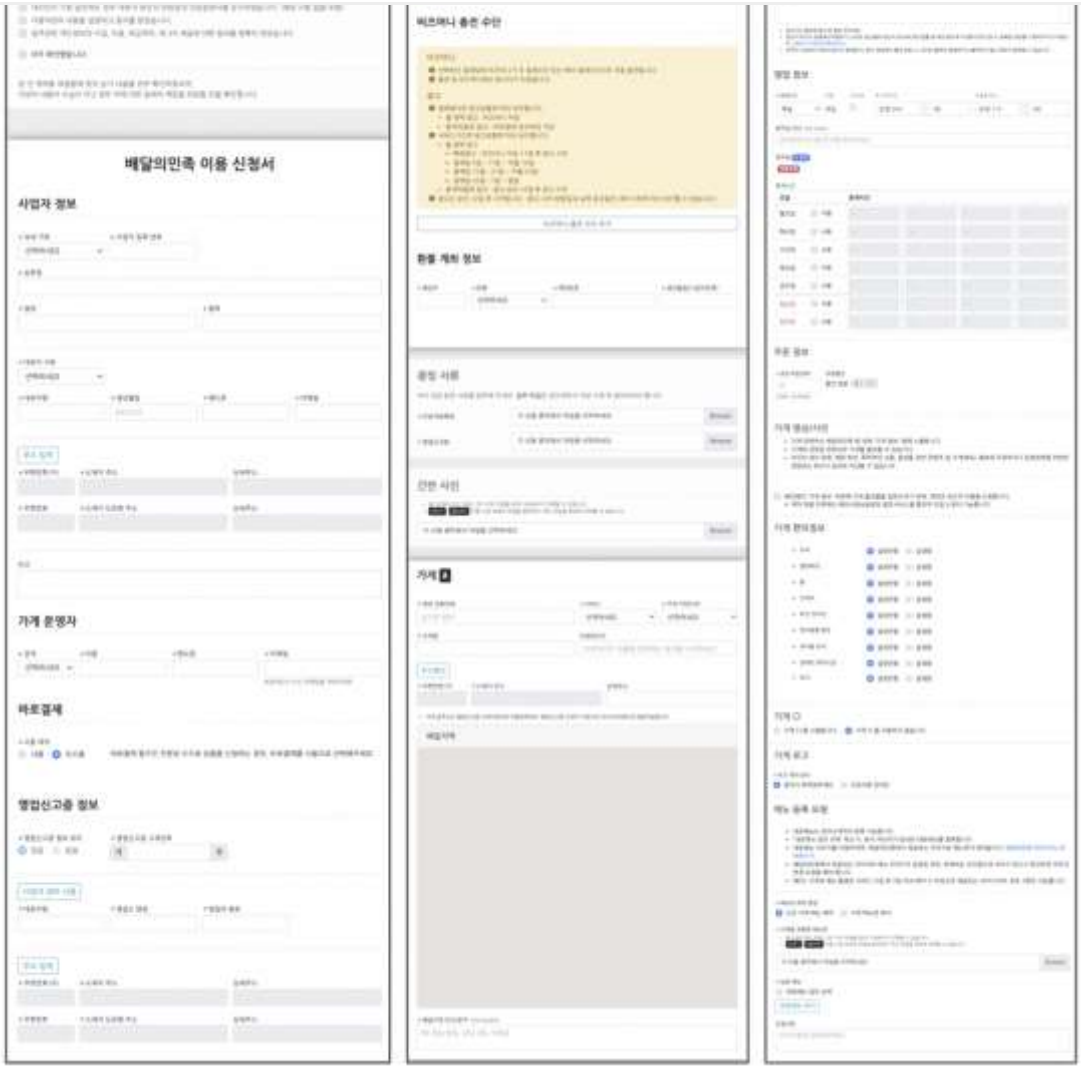
Web Frontend

안녕하세요, 우아한형제들에서 웹프론트엔드 개발을 담당하고 있는 배민셀프서비스팀의 남현우입니다. 피트스탑 기간에 정환님, 미라님, 민희님과 함께 팀 생산성을 높이기 위해 만든 크롬 익스텐션 개발에 대한 이야기를 공유하려고 합니다.

목표

배민셀프서비스팀에서는 피트스탑 과제로 다양한 형태의 폼을 자동으로 채울 수 있게 하는 **크롬 익스텐션**을 개발하기로 했습니다. (피트스탑은 F1 레이싱 중 차량을 정비하는 것과 같이, 사내에서 프로덕트 개발보다 안정성과 생산성 향상을 위해 2주간 재정비하는 시간을 의미합니다). 이러한 의사결정의 배경에는 운영하고 있는 영업 어드민 프로덕트(이하 프로덕트)에서 다양한 형태의 폼을 입력하는 경우가 많았기 때문인데요 개발을 하는 과정에서도 작은 기능 개발을 위해 수많은 폼을 입력해야 테스트가 가능했고, 기획 및 QA 단계에서도 단순 반복 작업의 수고로움을 경험해야 했습니다.

따라서 각자의 역할에 집중할 수 있는 환경을 만들어, 모두의 생산성 향상을 가져올 수 있도록 하는 **'여러 폼의 입력을 자동화해주는 툴을 만드는 것'을** 이번 피트스탑의 목표로 세웠습니다.



슈퍼 프로덕트의 일부 지면, 전체 폼을 입력해야지만 다음 작업을 진행할 수 있음...

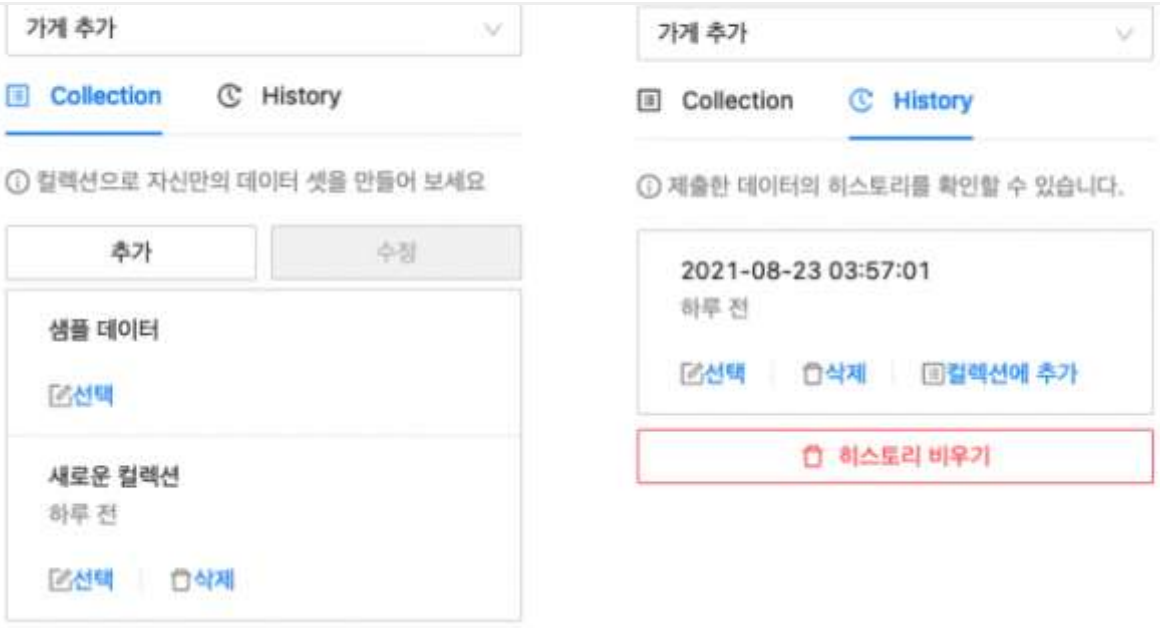
기획의도

기획의 방향은 **입력했던 데이터 셋을 저장하고, 저장된 데이터 셋을 그대로 폼에 입력되게 하는 것**으로 설정했습니다. "테스트용 더미 데이터를 코드에 넣어두고 사용하면 되지 않을까?"라고 생각할 수 있지만 몇 가지 취약점이 존재했습니다.

우선 새로운 feature의 개발에 따라 더미 데이터의 지속적인 업데이트가 필요하다는 점입니다. 실제로 특정 더미 데이터셋을 넣는 기능이 프로덕트 내부에 몇 군데 존재했는데, 배민1이라는 메이저 업데이트에도 불구하고 데이터셋은 이를 follow-up 하고 있지 못했습니다.

또한 데이터가 고정되어 있어, 특정 feature를 테스트하기 위해서 값을 채운 뒤에도 여전히 추가로 입력 값을 수정해야 했습니다. 따라서 더미 데이터를 관리하는 수고도 덜고, 각 사용자마다 커스텀한 데이터셋으로 테스트할 수 있도록 컬렉션과 히스토리, 두 가지를 메인 기능을 개발하기로 결정했습니다.

- 컬렉션 : 현재 입력된 필드의 데이터를 그대로 저장하는 것, 추후에 수정 및 삭제가 가능함.
- 히스토리 : 현재 입력된 데이터를 제출(등록) 했을 때, 실제 API에 전송한 데이터를 저장함.



완성된 제품의 UI 모습으로 좌측이 컬렉션, 우측이 히스토리입니다

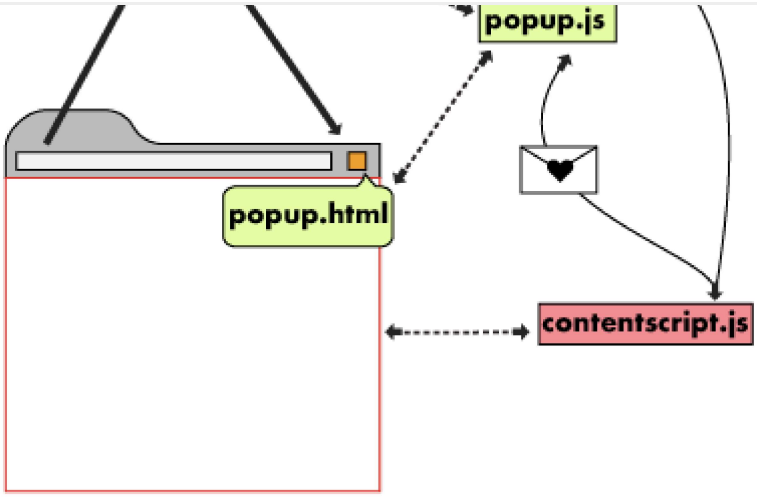
이러한 기획 의도를 바탕으로 실제 개발은 **프로덕트 외부에, 즉 크롬 익스텐션**으로 개발하기로 하였습니다. 이미 같은 팀의 한영재님이 개발하신 셀프서비스용 크롬 익스텐션이 존재했기 때문에 사용자 입장에서 러닝 커브도 낮을 것으로 생각했고, 2주 밖에 되지 않는 짧은 시간 동안 순수하게 필요한 feature 개발에 집중할 수 있었기 때문입니다. 그 외에도 관심사 분리 차원에서 프로덕트와 생산성 향상을 위한 툴을 분리해서 관리하는 편이 유지보수에 더욱 적합할 것으로 생각했습니다.

Part. 1 데이터 주고받기

이 프로젝트에 가장 주요한 기능은 **크롬 익스텐션과 프로덕트 사이에 데이터를 주고받는 것인데요** 프로덕트 내부에 입력된 데이터를 익스텐션으로 전달하여 저장하고, 익스텐션에 저장된 데이터를 프로덕트의 입력폼에 주입해야 합니다. 크롬 익스텐션에서는 **message passing** 기술을 기반으로 브라우저에 띄워진 프로덕트와 익스텐션 간 혹은 익스텐션 내부에서도 통신하도록 구현되었는데, 우선 크롬 익스텐션 개발 환경에 대한 사전 설명을 드릴게요.

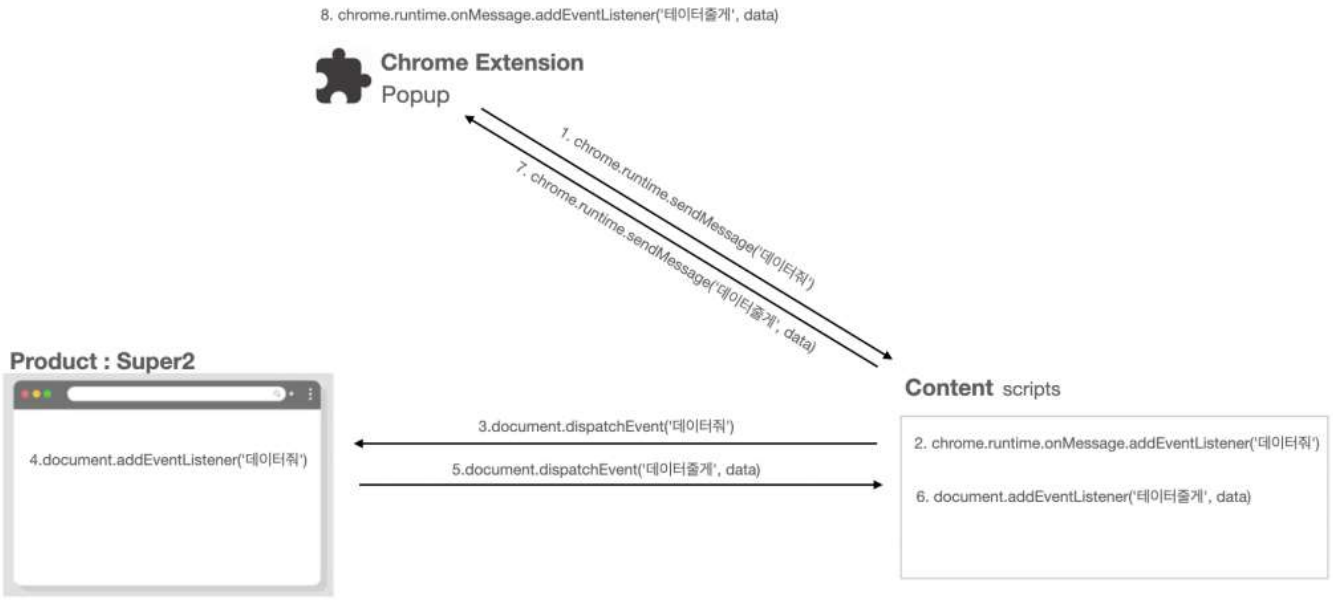
크롬 익스텐션은 background, content, popup 세 부분으로 나누어 개발해야 하는데, 각각은 고유한 컨텍스트와 역할을 가지고 있으며, 메시지를 통해 유기적으로 연결 되어 작동하고 있습니다. 자세한 설명은 공식 문서를 참조해 주세요.

- **popup**은 익스텐션 아이콘을 눌렀을 때 뜨는 화면입니다. 현재 띄워져 있는 브라우저와는 별개의 context를 갖고 있으며, 이 프로젝트에서는 react를 이용해서 화면을 구성했습니다.
- **content**는 브라우저내의 웹페이지를 제어하는 영역입니다. 즉, 현재 띄워진 페이지와 context를 공유하고 있어서 프로덕트의 dom에 대한 제어까지 할 수 있는 곳입니다.
- **background**는 크롬 익스텐션이 실제로 동작하는데 필요한 스크립트인데, 다양한 이벤트 핸들러를 다룰 수 있습니다.



슈퍼파트에서는 message passing 기술을 기반으로 데이터를 주고받는 기능 구현을 위해 각 컨텍스트에 맞는 API를 사용했습니다. content와 popup에서는 크롬에서 제공해 주는 chrome.runtime의 sendMessage와 onMessage.addListener를 사용해서 통신했고, 프로덕트와 content에서는 web document API가 제공해 주는 document의 dispatchEvent와 addEventListener를 이용했습니다. content는 앞에서 설명한 것처럼 브라우저 내의 웹페이지에 대한 제어를 할 수 있기 때문에, 크롬 API뿐 아니라 프로덕트 컨텍스트의 web document API까지 모두 이용할 수 있습니다.

아래의 그림은 크롬 익스텐션에서 프로덕트에 데이터를 '요청' 하는 경우의 flow를 보여주고 있습니다. 단순히 message를 주고받는 과정이지만, 여러 과정을 거치기 때문에 다소 복잡해 보입니다. 물론, 실제 제품을 개발하는 과정에서는 이를 추상화해서 복잡한 과정을 반복해서 거치는 수고를 덜었습니다. 덕분에 추상화된 코드로 인해 추가적인 지면에 기능 확장에 대해서도 유연하게 대처할 수 있도록 작업했습니다.



Part2. 데이터 저장하기

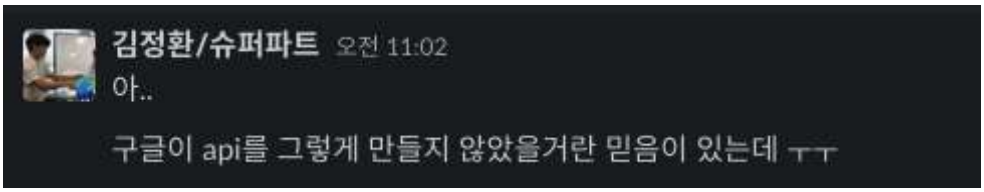
입력했던 데이터 셋을 저장하고, 저장된 데이터 셋을 그대로 폼에 입력되게 하는 기획의도에 따라 데이터를 주고받는 로직 이외에 또 한 가지 필수적으로 필요한 구현사항은 **데이터를 '저장'하는 로직**입니다. 그리고 여기서부터 슈퍼파트의 고난의 행군이 시작되었는데, 문제의 근원은 데이터를 '어디에' 저장할지에 대한

- ~~Chrome.storage.sync~~

최초에 활용했던 저장소는 크롬에서 익스텐션 개발용으로 제공해 주는 API인 chrome.storage중 sync입니다. sync는 브라우저에 저장하는 것이 아닌 크롬에 로그인 된 유저의 cloud 저장소에 데이터를 저장합니다. 클라우드에 저장되기 때문에 데이터가 지워지는(영속성에 대한) 걱정을 하지 않아도 되었습니다. 하지만 구글이 이런 API를 아무 조건 없이 제공해 줄리는 없었고, 저장 용량과 throttling 제한이 걸려 있었습니다. 기본적으로 각 데이터는 약 100kb를 넘을 수 없었는데, **이 제한조건 때문에 저장하려고 했던 데이터 전체가 저장되지 않아 특정 필드의 데이터들이 누락되는 현상이 발견**되었습니다. 이 외에도 최대 512개의 아이템만 저장이 가능했고, 분당 120회(시간당 1,800 회)만 데이터를 쓸 수 있었기에, 이러한 이유들로 다른 대안을 찾게 되었습니다.

- **Chrome.storage.local**

다음으로 고려된 후보 역시 크롬에서 제공해 주는 API 중 하나인 local입니다. sync에서 local로 단어만 변경해 주면 사용하는 API의 인터페이스가 동일했기 때문에 코드 변경사항을 최소로 할 수 있었습니다. 사용자의 브라우저에 데이터를 저장하기 때문에 최대 용량은 5mb 정도까지 여유로웠고, 이마저도 manifest에서 `unlimitedStorage` 옵션을 통해 제한 없이 이용할 수 있었습니다. 로컬 저장소를 이용하기 때문에 익스텐션을 지우는 등의 상황에서는 데이터가 날아갈 수도 있겠다는 걱정을 했지만, 엣지 케이스로 분류하고 계속해서 개발을 했습니다. 그런데 **탭간 전환을 하거나, 브라우저를 재시작 하는 등의 몇 가지 케이스에 저장된 데이터들이 사라지는 현상을 확인**했습니다. 공식문서를 살펴보아도 데이터의 초기화 조건이나 시점등에 대한 내용은 찾아볼 수 없었고, dev 모드가 아닌 실제 빌드된 결과물을 올려 테스트해도 동일한 현상들이 발생해서 다른 방법을 찾아보게 되었습니다.



저도요...

결국 DB가 필요한걸까?

~~믿었던 구글에게 배신당하고 나니~~ 지워질 가능성이 높은, 심지어 컨트롤 할 수도 없는 경우를 겪고 나니 공수가 크더라도 API 서버를 통해 데이터를 관리하는 방향을 생각하게 되었습니다. 복잡한 작업은 아닐듯했으나, 문제는 남은 피트스탑 기간이 넉넉지 않았습니다. 그리하여 또 다른 방법으로 논의되었던 것은 구글 스프레드시트를 이용하는 것이었는데, 결론부터 얘기하자면 구글에서 제공해 주는 스프레드시트용 API는 database를 대체하는 용도였습니다. 따라서 브라우저가 아닌 node 환경에서 사용할 수 있었고, 결국 이것 또한 자체 서버용 API를 만들어야 사용이 가능한 형태였습니다. 분명 영속적인 데이터 저장소가 있으면 좋을 것 같았지만, 사용자들(기획&QA분들)의 피드백을 받고 결정해도 늦지 않을 것이라고 생각했고, 또한 기존의 논의된 개발 범위가 아닌 부분에 더 많은 리소스를 부어야 했기 때문에 즉각 사용할 수 있는 방법들을 고민하게 되었습니다.

클라이언트(브라우저) 저장소

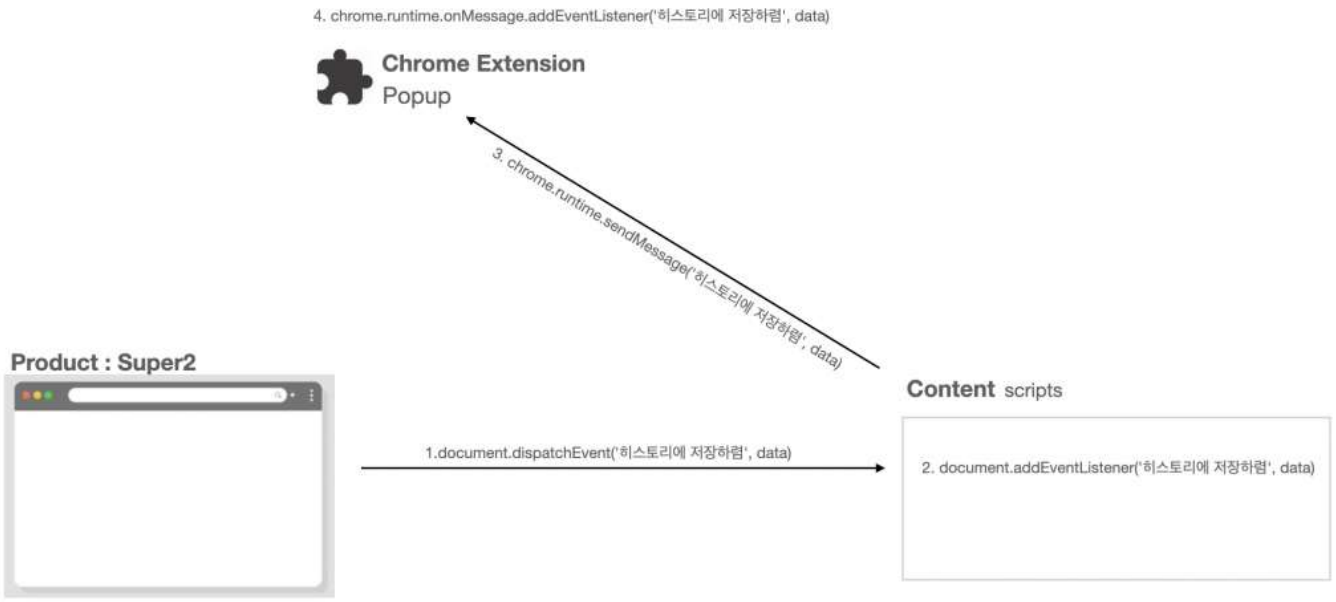
닭 잡는데 소 잡는 칼을 쓸 수는 없으니, 시간과 리소스를 고려해서 선택한 후보군은 **Web Storage(localStorage)**와 **indexedDB**였습니다. 두 가지 모두 도메인 종속적인 웹 브라우저 저장소로 우리가 이용하려는 데이터를 저장하기에는 충분한 저장공간을 제공해 주었습니다(localStorage – 최대

각각의 값이 index를 가지며 string 외의 데이터형도 저장 가능하여, 실제 database와 비슷한 형태로 저장되었습니다. 뿐만 아니라 제공해 주는 API가 비동기로 작동했기 때문에 지연으로 인한 성능 저하 등의 문제를 피할 수 있었고, 기본적인 get, set 이외에도 쿼리를 받아서 데이터를 찾거나, delete, getAll 등의 API 또한 지원해 주었습니다.

결론적으로 **indexedDB**를 선택하기로 했는데, 첫째 이유는 저장 및 조회를 하는 경우에 **데이터를 다루기 더 쉽다**는 점이었습니다. localStorage에서는 string 형태의 값만 저장할 수 있었기 때문에 변환해서 저장해야 했습니다. 실제 value 이외에 구분자로 이용되는 다른 필드들을 key값으로 조합해서 이용해야 했습니다. 예를 들어 어떤 페이지인지, 어떤 데이터 타입인지를 key값에 `history:add-shop` 와 같은 형태로의 변환이 필요했습니다.
반면 indexedDB는 타입을 지정해서 각각의 필드별로 데이터 저장이 가능했으며, 데이터 형을 변환하는 등의 과정을 생략할 수 있었습니다. 또한 indexedDB에서 **기본적인 CRUD를 위해서 제공해 주는 API가 있는 것**이 큰 장점으로 다가왔습니다. 기존에 작업해둔 크롬 스토리지의 API(sync, local) 또한 localStorage와 같이 get, set 메소드만 제공해 주고 있었기 때문에 자체적으로 데이터를 핸들링하고, 변경되는 데이터를 통째로 저장하는 로직이 프런트에 존재했습니다. 작업량이 더 늘어날 수는 있었지만, 추후에 API 서버를 개발한다는 것을 전제로 한다면 미리 작업을 해두는 편이 더 좋을 것으로 생각했습니다. 결론적으로는 더 단순하고 이해하기 쉬운 코드로 변경되었습니다.

Part3. 데이터는 어디에 저장되는걸까?

데이터를 주고받고, 데이터를 저장하는 것까지 완료되었으니 기획적인 구현사항은 마무리되는 듯했습니다. 하지만 ~~너무 식상한 클리셰...~~ 언뜻 보기엔 사소해 보이는 버그 하나가 발견되었는데, **히스토리**(현재 입력된 데이터를 제출(등록) 했을 때, 입력된 데이터를 저장함)**가 특정 상황에서 작동하지 않았습니다.** 문제 상황에 대한 이해를 돕기 위해 Part 1에서 사용된 그림을 이용해 flow를 정리해 보자면, 슈퍼 프로덕트에서 발행한 이벤트를 content에서 chrome.runtime API를 이용해서 최종적으로 popup에 전달하면 popup에서 데이터를 저장하게 됩니다. 그리고 반복되는 테스트 결과로 히스토리 기능이 작동하지 않는 특정 상황은 **Popup이 켜있지 않을 때**였습니다.



혹시 문제의 원인이 어떤 것인지 느낌이 오고 있으신가요? *아니라면 아래의 힌트!*

- **힌트 3.** indexedDB에 접근해서 데이터를 저장하는 로직은 Popup(.ts 파일)에 존재한다. 따라서 크롬 익스텐션 popup 브라우저의 indexedDB에 값이 저장되고 있다.

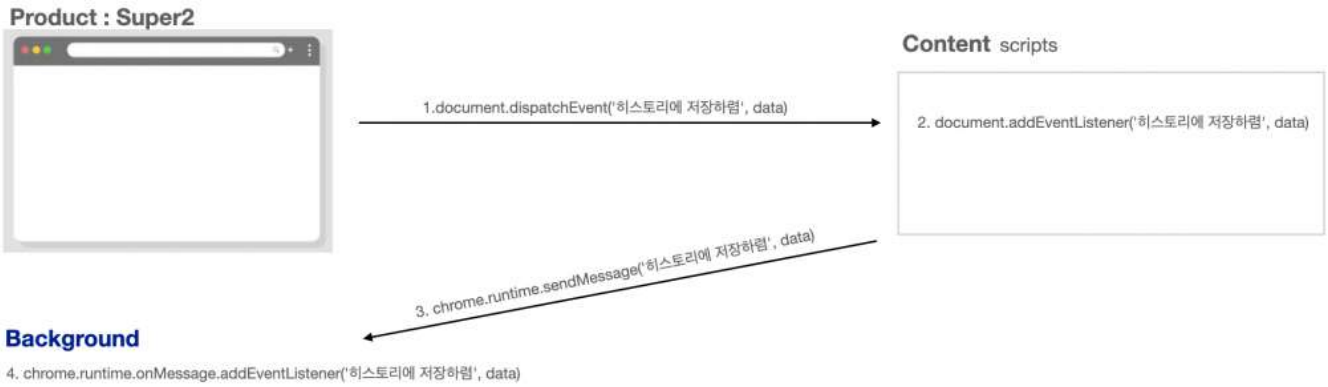
문제의 원인을 위의 힌트들을 기반으로 추론해 보자면, 익스텐션 아이콘을 눌러서 popup창을 띄워두지 않으면 popup 내부의 코드가 실행되지 않습니다. 설령 실행된다고 하더라도, (popup)브라우저가 띄워져 있지 않기 때문에 접근하려고 하는 window 객체는 존재하지 않습니다. 이것이 히스토리 저장 기능이 정상적으로 작동하지 않는 원인이었습니다.

가장 먼저 생각한 해결책은 "혹시 저장소의 위치를 익스텐션의 popup 브라우저 저장소가 아닌, super와 컨텍스트를 공유하는 content로 변경하면 어떻게 될까?" 였는데요 즉 위의 그림에서 3,4 번의 단계는 생략하고, 2번에서 받은 데이터를 바로 저장하는 방법이었습니다. 이 해결방법은 위의 문제는 해결할 수 있었지만, 기존 코드베이스를 전체적으로 변경해야 했습니다. 일례로 popup에는 기존에 저장한 컬렉션, 히스토리 리스트를 window.indexedDB에 접근해서 저장된 데이터를 불러와서 보여주고 있습니다. 하지만 데이터를 저장하는 곳을 content(super2, 동일한 저장소)로 옮기게 되면, 데이터를 불러오는 단계에서부터 message passing이 필요하게 됩니다(ex. popup: "content야 데이터 줘!").

이 외에도 수정하는 등의 로직은 더 많은 수정을 필요로 했고, 시간적인 제약으로 차라리 히스토리 기능을 추후 공개하는 것이 어떤가(API 서버를 개발한 후에)하는 논의까지 하게 되었습니다.

Background

이때 문득 "Part1에서 말한 크롬 익스텐션 개발에 필요한 3개의 파벌 중에, 나머지 background는 도대체 어떤 context를 갖고 있는 거지?"라는 의문이 들기 시작했습니다. 간단하게 슈퍼에서 발행한 이벤트를 popup이 아닌 background로 전달해 보았는데, 코드가 정상적으로 작동하기 시작했습니다. 개발자 도구를 통해 확인해 보니, popup과 background가 동일한 indexedDB를 공유하고 있었습니다. "🤔**왜?**"라는 의문이 앞섰지만, 일단 제품이 정상 작동하도록 하기 위해 작업을 먼저 진행했습니다.



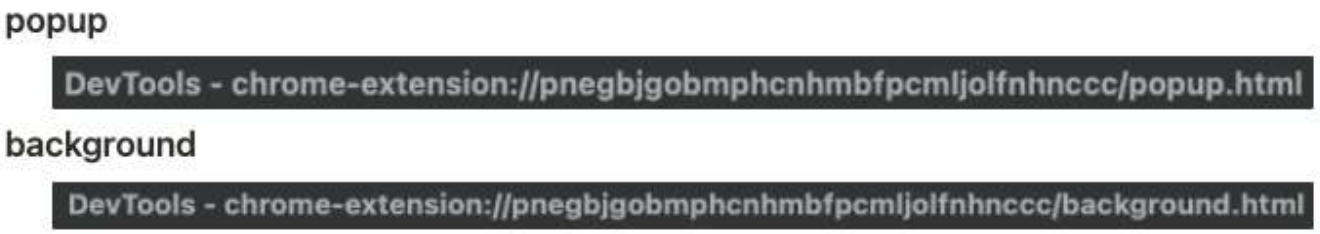
인생지사 새옹지마

indexedDB에 저장하는 로직을 Background로 옮기고 나니, 기존의 message passing보다 더 괜찮은 방법을 발견하게 되었는데요. background에서는 크롬 익스텐션에서 제공해 주는 다양한 이벤트 핸들러에 접근할 수 있다고 설명했는데, 이 중 실제 API request를 후킹 할 수 있는 webRequest.onBeforeRequest API를 제공해 주고 있었습니다. 그리하여 슈퍼의 submit 버튼 들에서 익스텐션으로 이벤트를 발행하던 로직들을 제거하고, background에서 API request를 후킹 해서 실제 API request에 넘겨주는 값들을 저장하는 로직으로 변경했습니다. 결론적으로는 프로덕트에서 외부 서비스

그림에도 마지막까지 남은 의문이 있었습니다. **"popup과 background는 어떻게 동일한 indexedDB를 공유하고 있었을까"**? 크롬 익스텐션의 공식 문서뿐 아니라 인터넷에 다양한 정보들을 찾아봤지만, 이에 대한 명확한 답변을 찾아보기는 어려웠습니다. 그래서 찾아본 내용들을 토대로 가능성 있어 보이는 추론을 해보려고 합니다.

> 기본적으로 indexedDB는 여타 다른 클라이언트(브라우저) 저장소와 같이 동일 출처 정책(Same Origin Policy)를 따르고 있다.

크롬 브라우저에서 popup과 background의 개발자도구를 활성화시키면 아래와 같은 URL을 확인할 수 있습니다. SOP 정책에 따라 프로토콜, 포트, 호스트가 동일한 경우에 동일한 출처로 본다는 것에 따라 확인해 보면 두 개의 URL은 `chrome-extension` 프로토콜과 `pnegbjgobmphcnhmbfpcmljolfnhnccc` 라는 호스트를 공유하고 있었기에 indexedDB를 공유하고 있는 것이 아닐까 추측해 보았습니다. 혹시 정확히 알고 계시는 분이 있으시다면 공유해 주시면 감사하겠습니다!😭



후기

완성된 제품을 보니, 안정성과 생산성 향상을 위해 2주간 재정비하라는 피트스탑의 의의에 맞는 시간을 보낸 것 같아 뿌듯했습니다. 길지 않은 시간이었지만 과제 선정부터 기술 스택, 사용자 시나리오까지 파트 내에서 자유롭게 논의하며 진행되었는데, 이를 하나씩 구체화해 나가는 과정들이 의미 있고 흥미로웠습니다. 크롬 익스텐션 개발이라는 새로운 도메인을 접하고, 동료들과 실시간으로 피드백을 주고받으면서 빠르게 진행할 수 있었습니다. 다만 처음부터 좀 더 구체적인 계획을 세웠다면 덜 고생하지 않았을까 하는 생각이 들어 약간은 아쉽기도 했지만, 그것을 해결하는 과정에서 다양한 내용들을 공부하며 오히려 더욱 생산적인 시간을 보낼 수 있어 재미있으면서도 유익한 시간으로 채울 수 있었습니다.

물론 그 결과물 또한 큰 도움이 될 수 있겠구나 체감할 수 있었는데요. 저희가 만든 제품을 사용하지 않았을 때와 사용했을 때를 비교해서 얼마큼 생산성을 올릴 수 있었는지에 대한 효율성을 측정해 보았는데, 최대 92%까지 더 높은 효율을 낼 수 있었습니다! 아직 1.0 버전에서 이 정도의 효율이라면, 더 행복한 미래를 기대할 수 있지 않을까요? **"만드는 사람이 수고로우면 쓰는 사람이 편하고, 만드는 사람이 편하면 쓰는 사람이 수고롭다."**라고 회사 곳곳에 문구가 적혀있는데요. 우아한형제들의 제품을 사용하는 고객들뿐 아니라 옆에 일하는 동료들의 일하는 환경까지 더 편하게 만들어줄 수 있게 앞으로도 열심히 수고롭도록 하겠습니다!

이 글을 읽고 저희와 함께 일하고 싶어지셨다면, [사장님서비스실] 배민셀프서비스팀 에 많은 지원 부탁드립니다. 🤷‍♂️ 🤷‍♀️

긴 글 읽어주셔서 감사드립니다. 이상 끗!

#크롬 익스텐션 #피트스탑

댓글 달기...

전소영

테스터와, 기획, 개발 모두를 위한 효율적인 방안을 찾고 실제로 좋은 피드백도 받아서 정말 좋았습니다. 앞으로도 많이많이 테크조직을 위한 좋은 기술들 발휘해주세요!! b'-'d

좋아요 · 답글 달기 · 1년

박재랑

너무너무 유용하게 잘쓰고 있습니다^_^

좋아요 · 답글 달기 · 1년

Facebook 댓글 플러그인

목록으로 돌아가기

한명의 개발자를 양성하기까지 배민쇼핑라이브를 만드는 기...