Node.js in Toss

이 사람들은 무엇을 하는 사람들인가?

Node.js Chapter lead 김인성

여는 글

어떻게 소개해야 매력적일까...?

여는 글

주제 후보.

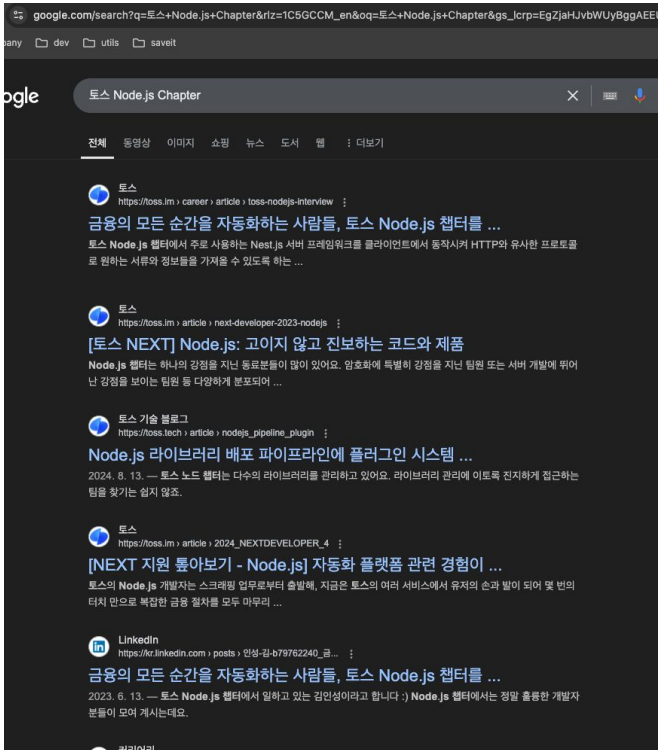
- 토스 커뮤니티 Node.js 라이브러리 배포 방식
- Toss Node.js 서비스 운영 환경 소개 (docker, batch, client side application..)
- 내부에서 쓰고 있는 라이브러리 컨셉 소개하기
- slash에서 발표했던 내용 더 깊게 소개해보기 (재탕하기)
- codegen 소개하기
- …

여는 글

음... 뜬금없이 무엇인가를 발표하기 보단, 소개부터...

Node.js Chapter ?

외부에서 저희를 바라보는 시선에 대해서 알아보기 위해, 일단 구글에 검색부터

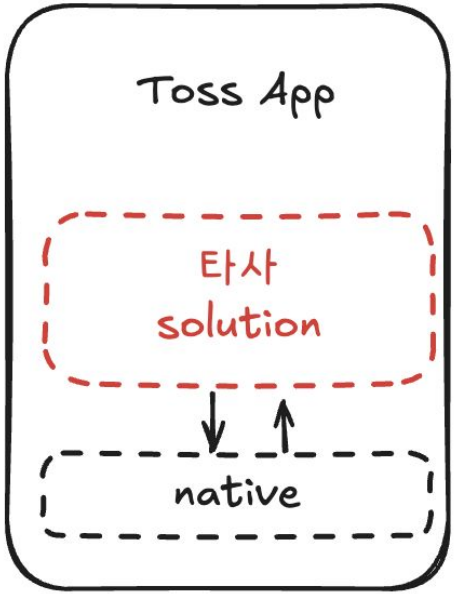


Node.js Chapter ?

Node.js Chapter는 처음부터 Node.js Chapter가 아니였습니다.

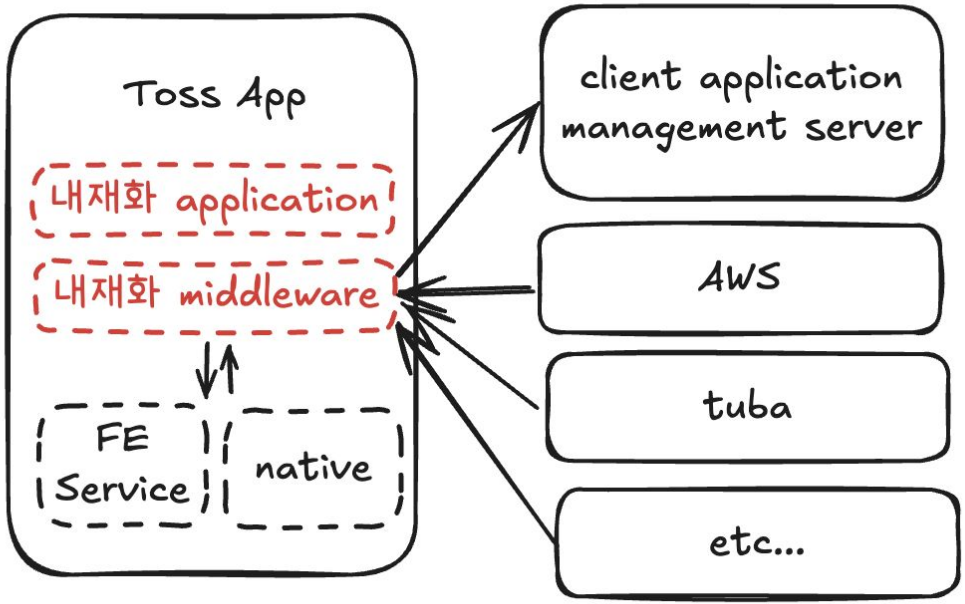
2018~2019 Toss

내재화 이전



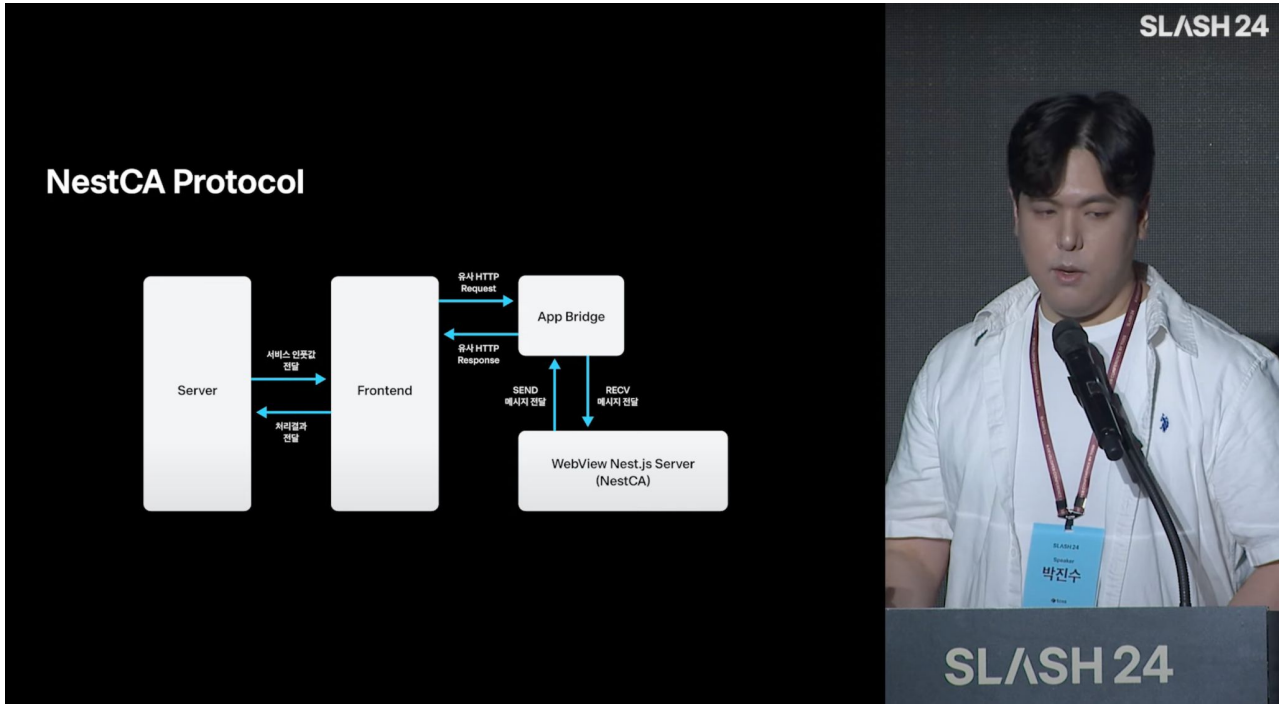
2019~2021 Toss

내재화 이후



Node.js Chapter ?

client application platform



Node.js Chapter ?

client application platform

https://www.youtube.com/watch?v=JDeZNxIk5Ko&t=19s

Node.js Chapter ?

client에서 하던걸 server에서도 하고 싶은데, 양쪽에서 모두 활용할 수 있는 모듈을 못만들까?

우리가 개발할 때 사용하는 typescript(js..)는 둘 다 만들 때, 쓸 수 있잖아.

그리고 client에서 구동하는 application을 잘 만들었는데, server는 못 만들까?

Node.js Chapter ?

이후 사장님 매출장부 서비스, 비즈앱, FDS, AML, 등..

토스 페이먼츠, 토스 플레이스, 토스 뱅크, 토스 증권 등..

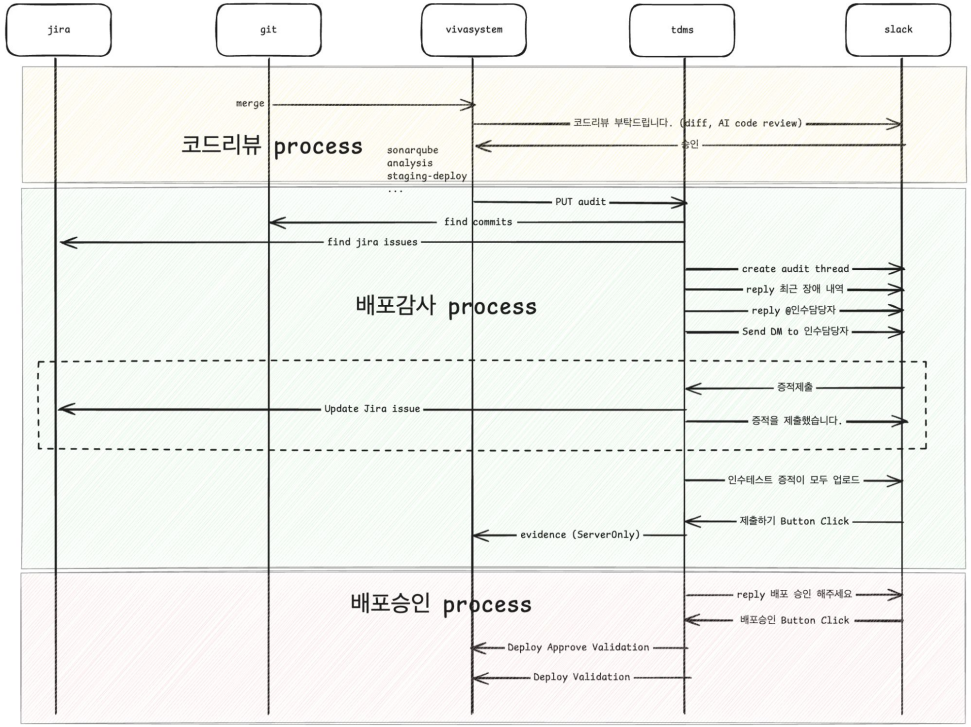
Node.js제품 확장, 계열사 확장 진행

Node.js Chapter ?

그래서 지금은 조금더 구체적으로.. 무엇을 하고 있는가?

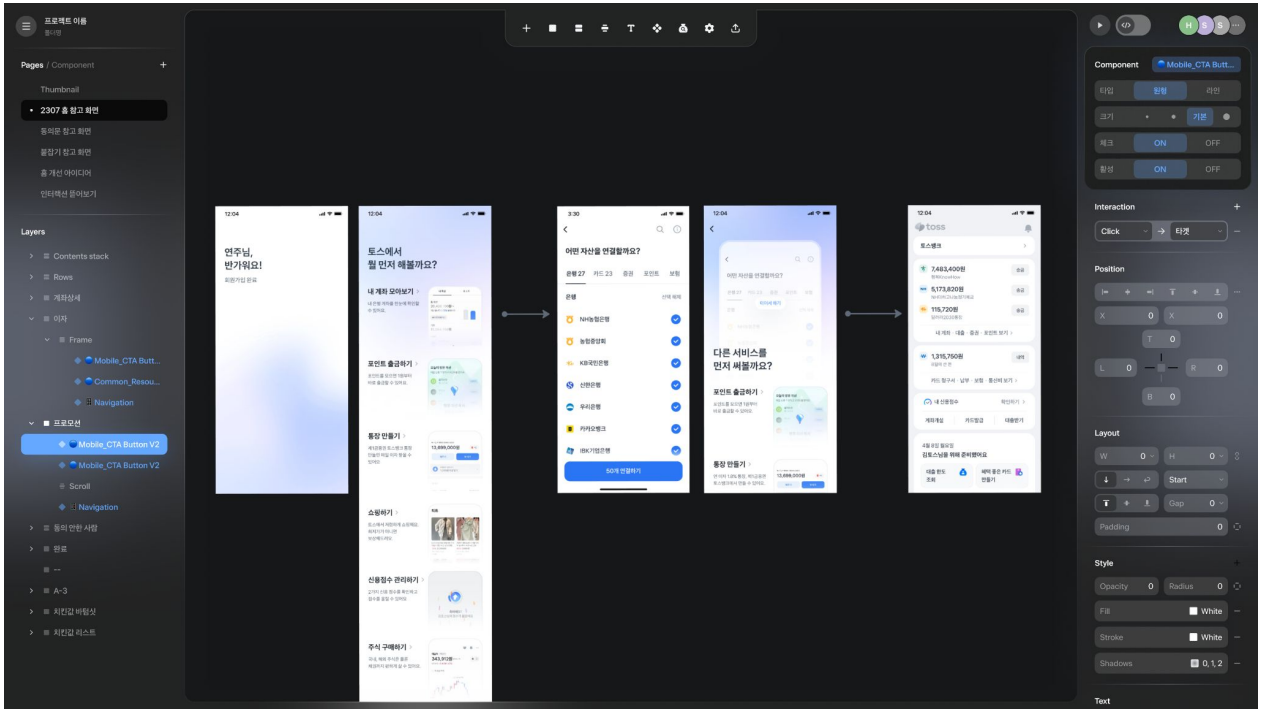
Node.js Chapter ?

TDMS (Toss Deploy Management System)



Node.js Chapter ?

Deus



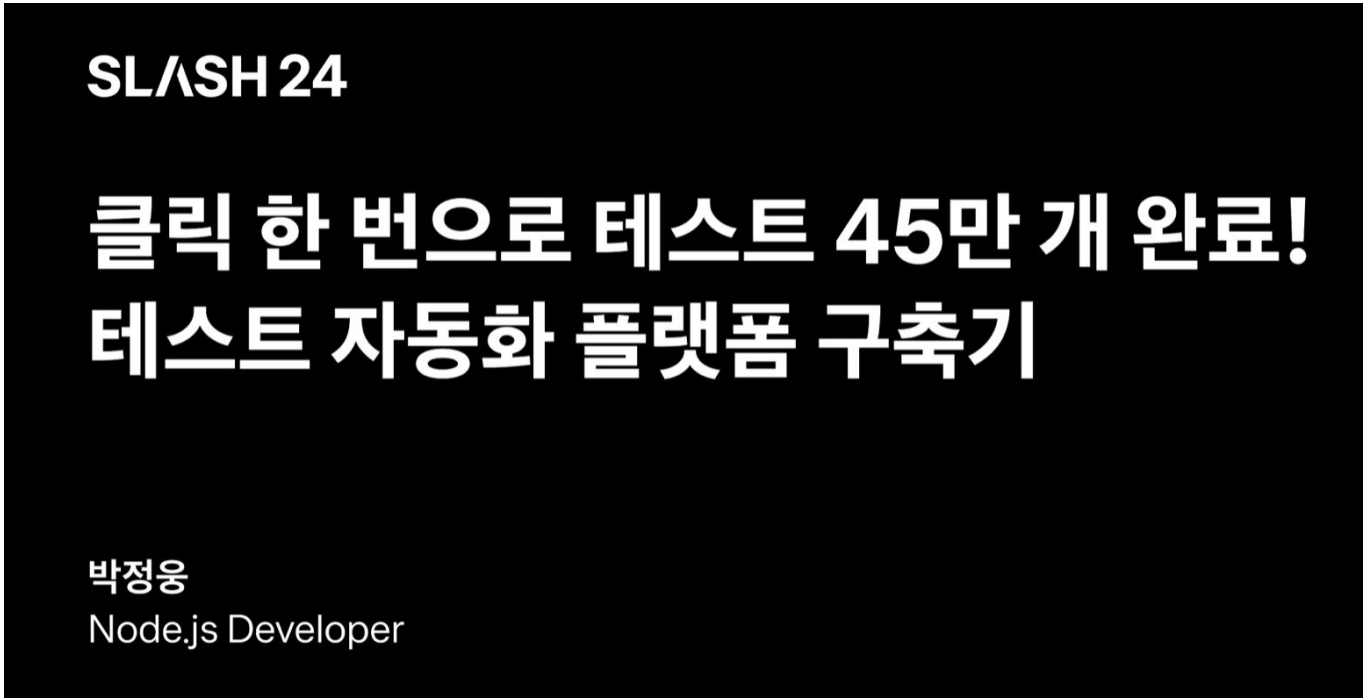
Node.js Chapter ?

운영 업무 효율화 제품

- Slack
- google
- 외부 SaaS 연동을 통한 자동화
  (greenhouse, workday, news platform, viral monitoring, AML, FDS ….)

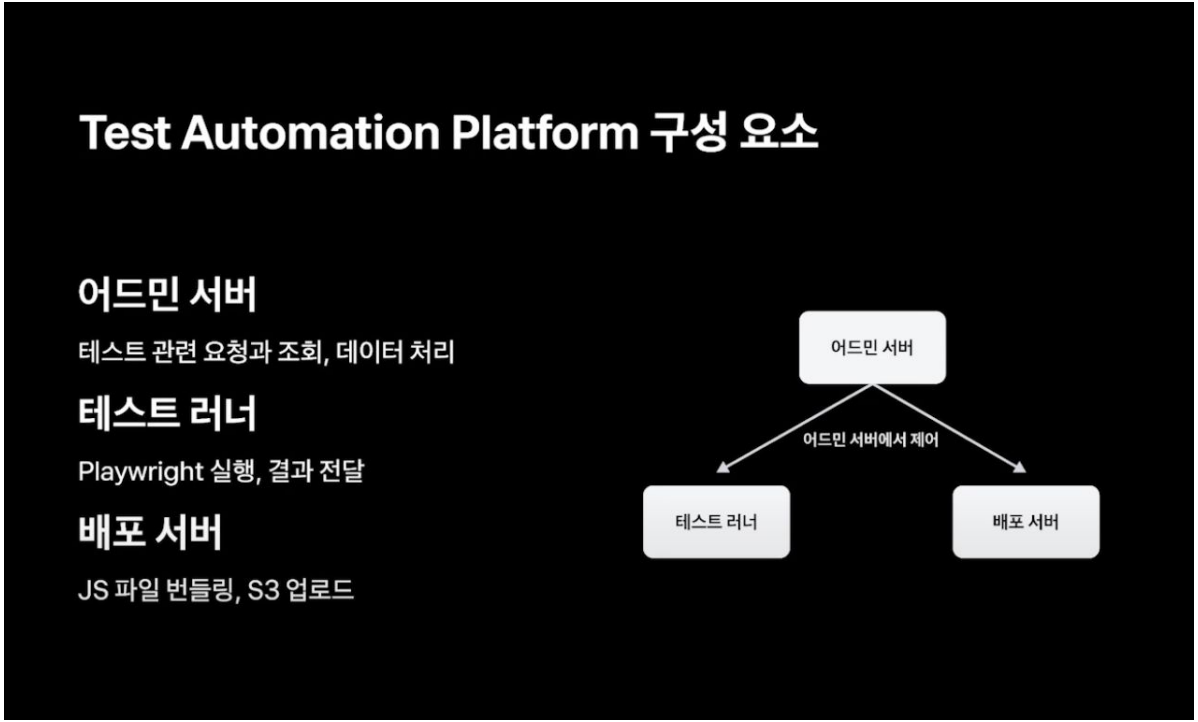
Node.js Chapter ?

테스트 자동화 플랫폼



Node.js Chapter ?

테스트 자동화 플랫폼



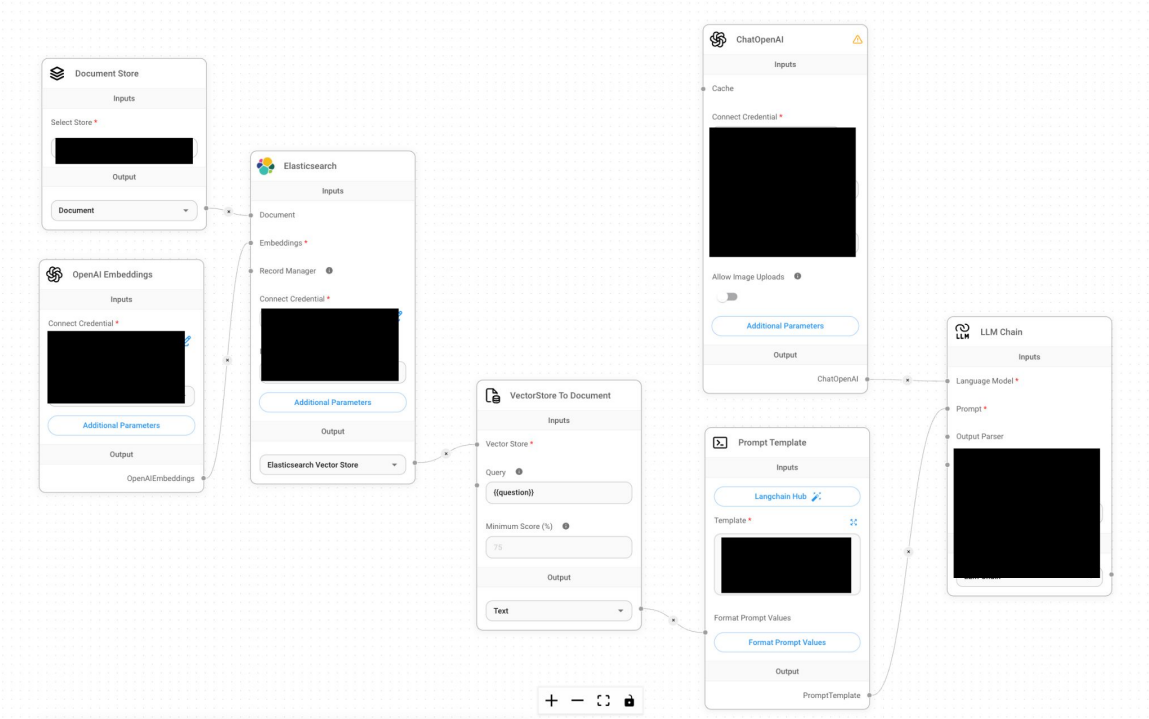
Node.js Chapter ?

테스트 자동화 플랫폼

- 본 영상 https://youtu.be/cGks5f2f0YE?si=DnAbqSsALG1I8q25

Node.js Chapter ?

ML Service low code platform

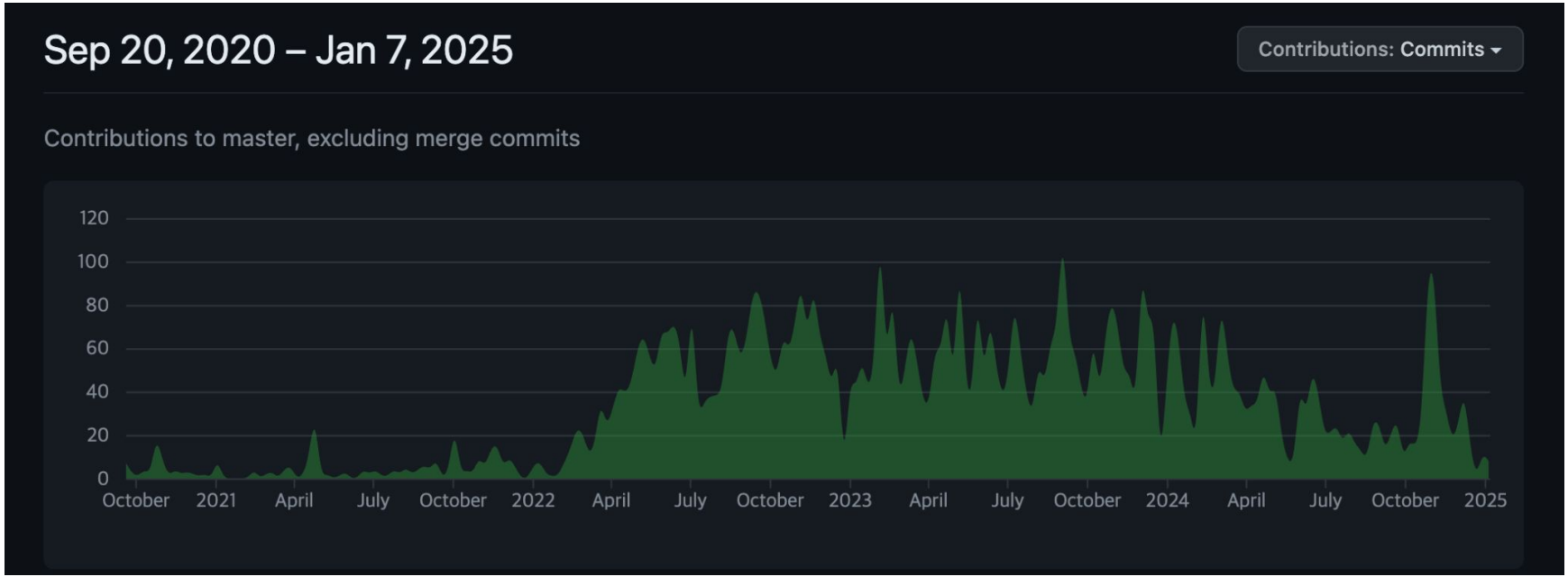


Node.js Chapter ?

이외에도 수많은 서비스/플랫폼/도구를 만들어 내고 있습니다.

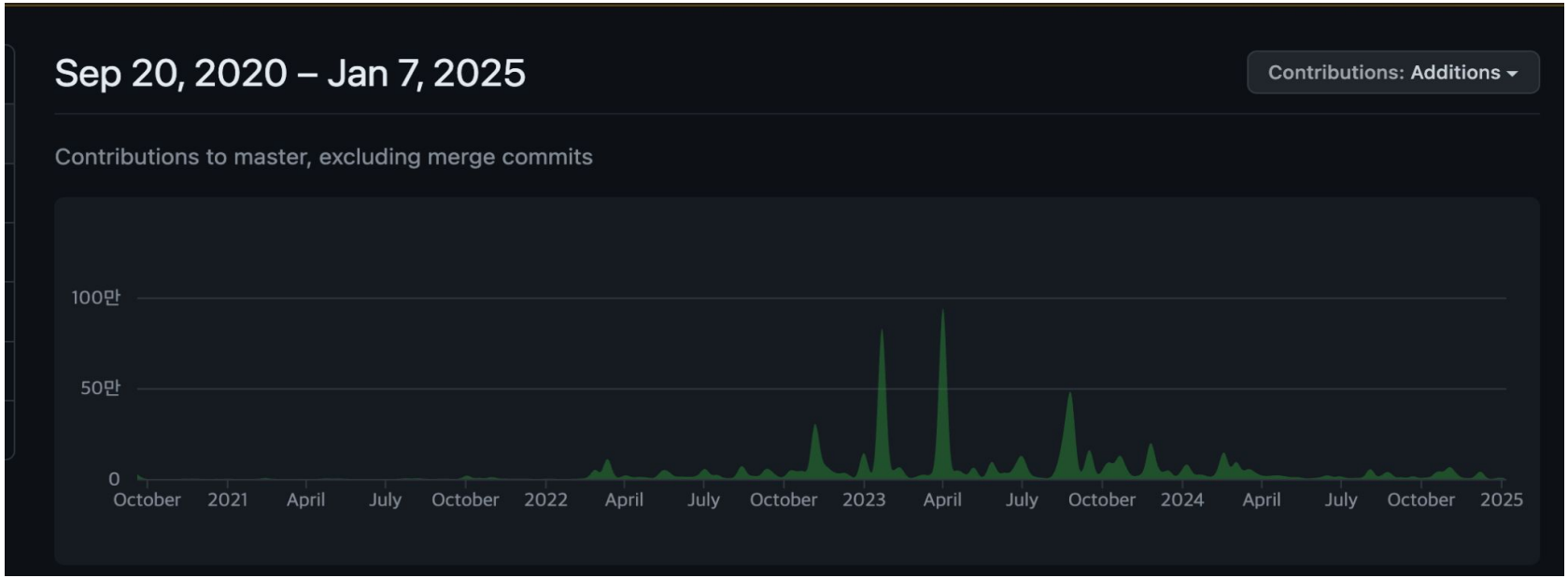
Node.js Chapter ?

여러 repo가 있지만,
그 중 하나 Web Automation Libraries 공용 repo



Node.js Chapter ?

여러 repo가 있지만,
그 중 하나 Web Automation Libraries 공용 repo



Node.js Chapter ?

뭐야 코드 많이 짠게 다는 아니잖아...
코드를 만들면 버그를 만든다는 말도 있는데..

Node.js Chapter ?

이렇게 확장을 할 수 있던 원동력은 여러가지가 있었을 것 같은데요. 그 중, 저는 모듈화가 가장 도움이 되었던 것 같습니다.

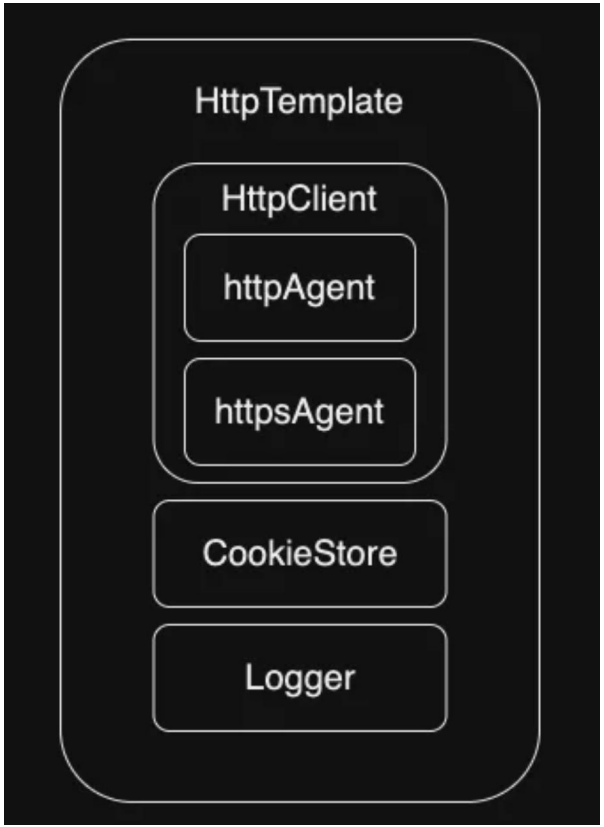
어떻게 모듈화를 했는지, 간단하게 몇가지 예시로 보여드릴까 합니다.

web-automation-libraries

t2-http

- node chapter에서 사용하는 http module
- 스크래핑이 편리하도록 구성한 모듈

web-automation-libraries (t2-http)

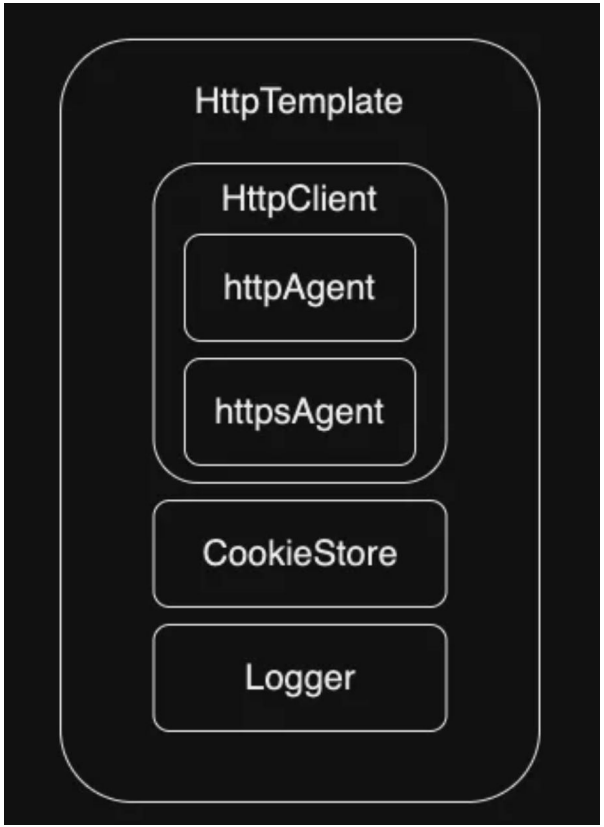


web-automation-libraries (t2-http)

HttpTemplate

- HttpTemplate라는 이름은 통신 라이브러리를 편하게 사용하기 위해 만든 인터페이스 입니다.
- 해당 인터페이스를 통해, 각종 비지니스들을 분리할 수 있게 되고, 관심사별로 나눠 관리할 수 있게 되었습니다.
- 종종 편리함과 엄격한 레이어 분리 사이에서, 많은 논란거리를 만드는 주체이죠

web-automation-libraries (t2-http)

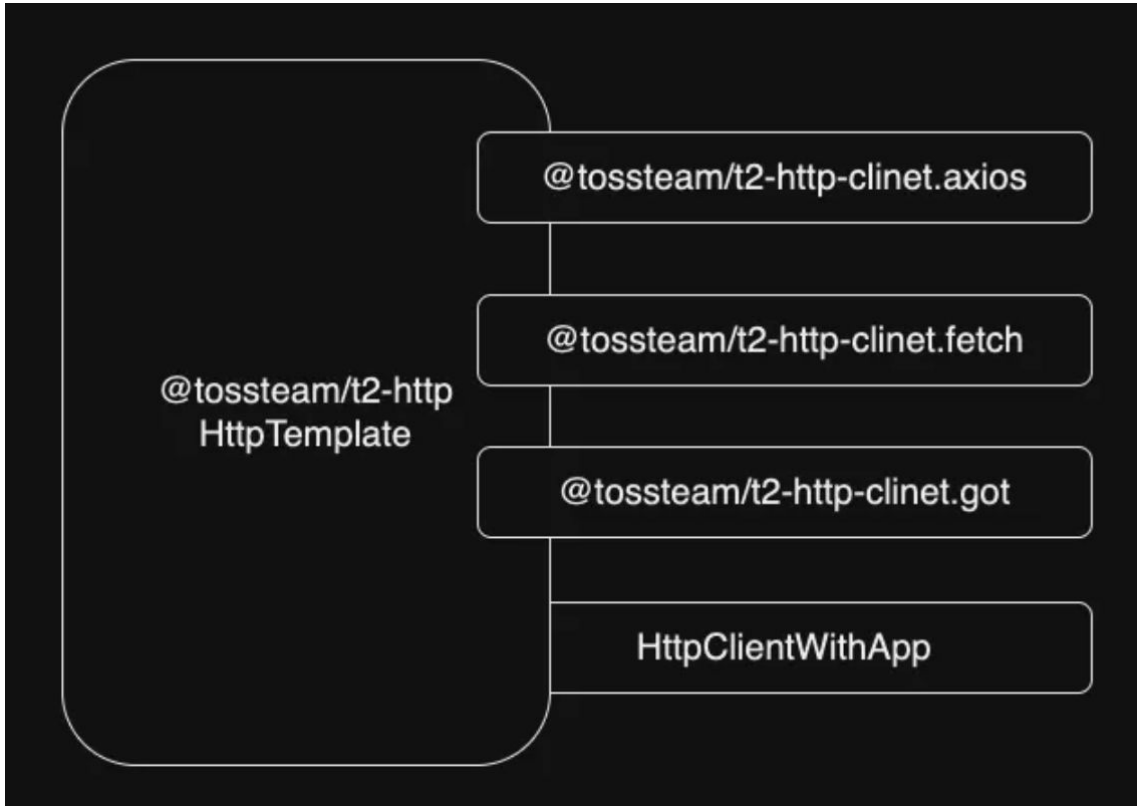


web-automation-libraries (t2-http)

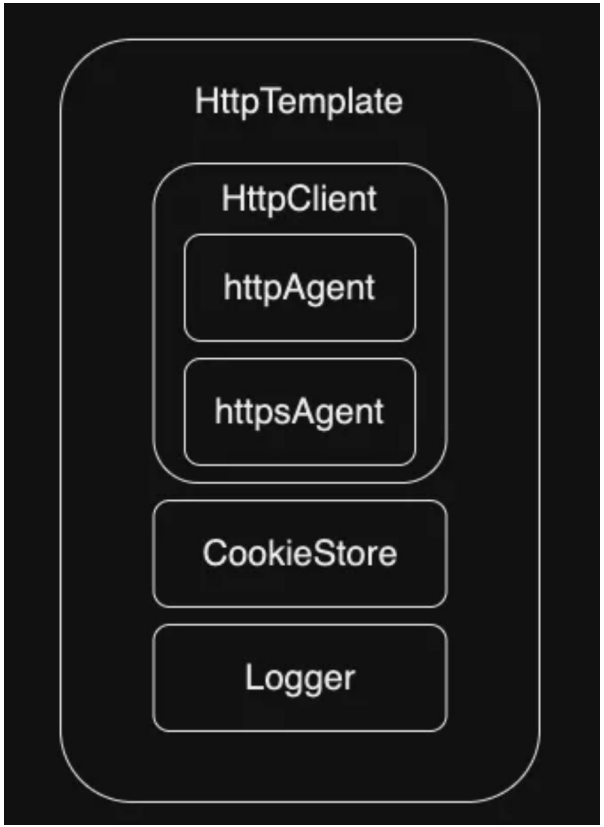
HttpClient

- 실질적인 통신의 주체입니다. 위에서 말씀드린 HttpTemplate는 저희가 사용하기 쉽게 만들어놓은 인터페이스라면, HttpClient는 실제로 통신 스팩을 지정한 인터페이스입니다.
- 해당 인터페이스를 구현한 구현체들은 아래와 같습니다.
    - @tossteam/t2-http-client.axios : axios를 기반으로 구현한 구현체
    - @tossteam/t2-http-client.got
    - @tossteam/t2-http-client.fetcher (브라우저 fetcher를 활용한 httpClient)
    - HttpClientWithApp (앱의 통신 모듈을 활용한 httpClient)

web-automation-libraries (t2-http)



web-automation-libraries (t2-http)



web-automation-libraries (t2-http)

CookieStore

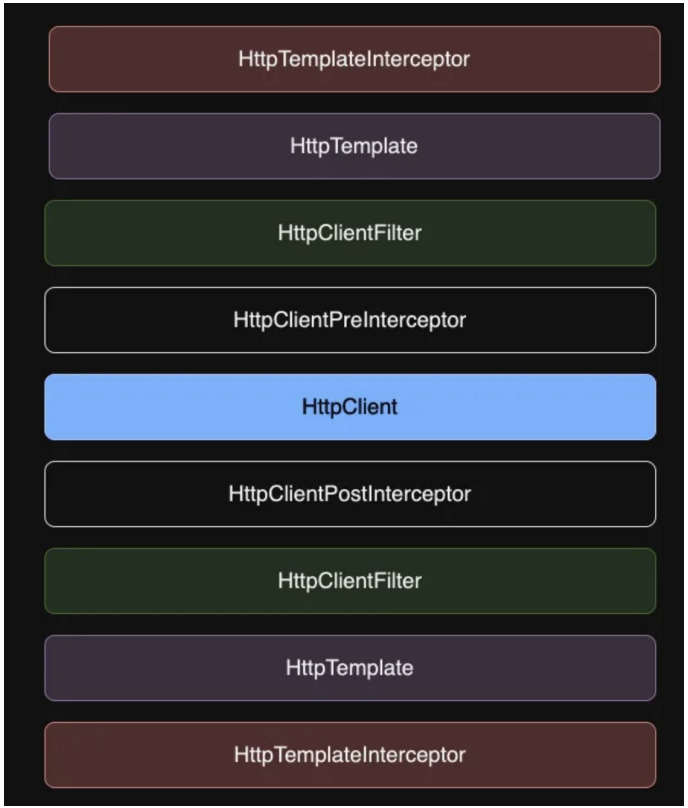
- Http Response의 Set-Cookie 헤더의 값들을 파싱한 결과들을 저장하는 곳입니다. 이 CookieStore를 통해, 저희는 유저들이 브라우저에서 활동할 때 처럼 세션을 유지시켜주며 통신을 할 수 있게 됩니다.

Logger

- Http 요청들의 상세정보들을 로깅하기 위해 존재합니다.
    - HttpClientPreLog, HttpClientPostLog 라는 키워드로 찍히는 로그들을 많이 보셨을텐데요. 이런 로그들을 해당 로거가 서빙해줄 수 있습니다.

web-automation-libraries (t2-http)

filter & interceptor



web-automation-libraries (t2-http)

filter & interceptor

예시들을 보시죠

- HttpTemplateInterceptor
  - 아래의 httpTemplate 요청(HttpTemplateRequestConfig)을 중간에 변조할 수 있게 도와줍니다.

```
export class AuthHeaderInterceptor implements HttpTemplateInterceptor {
  constructor(private readonly value: string) {}
  intercept(
    config: HttpTemplateRequestConfig,
    next: (config: HttpTemplateRequestConfig) => Promise<HttpResponseType>,
  ): Promise<HttpResponseType> {
    config.options ??= {};
    config.options.headers ??= {};
    appendToHeaders(config.options.headers, 'Authorization', this.value);
    return next(config);
  }
}
```

- HttpClientInterceptor
  - HttpClientCookiePostInterceptor, HttpClientCookiePreInterceptor 처럼 cookieStore에 있는 값들을 꺼내 header에 적절한 형태로 만들어 주거나, set-cookie에 있는 값을 꺼내 cookieStore에 저장하고, redirect시 사이드 이팩트가 발생하지 않도록 기존 request요청 헤더에서 cookie를 삭제해줍니다.
    - 요청시에는 다시 최신화된 cookieStore에 있는 값을 전송해줘야 해서, 이렇게 구현한 것 같은데, 여기서 생각보다 많은 예외 상황이 발생합니다.
    - 예를들어 httpTemplate.get(.... { header: “Cookie: ...” }) 처럼 쿠키를 헤더에 직접 세팅해서 요청하는 경우에 redirect가 발생한다면, 해당 쿠키는 정상적용 되지 않습니다.
- HttpClientFilter
  - 로깅, cookie관련 HttpClientInterceptor를 활용한 cookie 처리 로직들이 수행됩니다.

web-automation-libraries (t2-http)

기타 부가 기능들

t2-http에서 제공하는 부가 기능들

- Logger는 Jest환경에서 call stack을 함께 출력해준다 (HttpTemplate 메소드가 호출된 위치)
- HttpTemplateInterceptor Option에 대한 설명
- HttpTemplateInterceptor가 달려있는지 확인하는 Reflection 기능 및 Interceptor 조작 및 연산 기능
- shortcut 기능 (HttpTemplateResponsePromise) `.json .text .buffer`
- HttpSerializer HttpRequest 객체를 curl이나 raw HTTP 혹은 custom한 문자열로 시리얼라이즈 할 수 있음
- parsedRequset Curl 문자열 혹은 Raw HTTP문자열을 그대로 받아서 HttpTemplate Requester를 만들어줌
- HttpAgent, HttpsAgent의 역할 및 ProxyAgent를 끼워넣는 법
- Pre defined UserAgent 활용법 (이게 왜 나왔는지)
- 테스트 관련기능 `t2-http/testing`
  - testHttpTemplateFactory - `testHttpTemplateFactory({ HttpClientWithGot })` 처럼 한줄로 HttpTemplate를 만들어줌
  - mockHttpTemplate - 리퀘스터 의존성에 그대로 넣어줄 수 있으며 HttpClient레벨의 request를 캡쳐해놨다가 스냅샷찍을 수 있음
  - ChromeRepositoryInterceptor - 특정 URL, 도메인에 대하여 크롬 브라우저에서 직접 Cookie를 뽑아서 가져옴. 매뉴얼하게 로그인하고 테스트할 때 사용 가능 by 정현님

web-automation-libraries (t2-http)

invokedToMatchInlineSnapshotInCurl sample

```
it('invokedToMatchInlineSnapshotInCurl 테스트 – 하나하나씩 캡쳐', async () => {
  await withMockHttpTemplate(async (httpTemplate) => {
    const requester01 = new DummyRequester01(httpTemplate);
    const requester02 = new DummyRequester02(httpTemplate);

    await requester02.request({ filter: '*', page: 1, size: 5 });
    httpTemplate.invokedToMatchInlineSnapshotInCurl(`
      [
        "curl -X POST https://api.toss.im/jobs \\
        -H 'Accept: application/json' \\
        -H 'Accept-Encoding: gzip, deflate, br' \\
        -H 'Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7' \\
        -H 'Cache-Control: No-Cache' \\
        -H 'Content-Type: application/json; utf-8;' \\
        -H 'Referer: https://toss.im/career/jobs' \\
        -H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.136 Safari/537.36' \\
        -H 'X-Requested-With: XMLHttpRequest' \\
        --data-raw '{"filter":"*","page":1,"size":5}' \\
        --compressed
      ",
      ]
    `);
```

Node.js Chapter ?

여기까지 설명했더니.. 무슨 얘기를 하다가 여기에 왔는지 헷갈리시는 분을 위해서

==========

이렇게 확장을 할 수 있던 원동력은 여러가지가 있었을 것 같은데요. 그 중, 저는 모듈화가 가장 도움이 되었던 것 같습니다.

어떻게 모듈화를 했는지, 간단하게 몇가지 예시로 보여드릴까 합니다.

web-automation-libraries (slack, slack-listener)

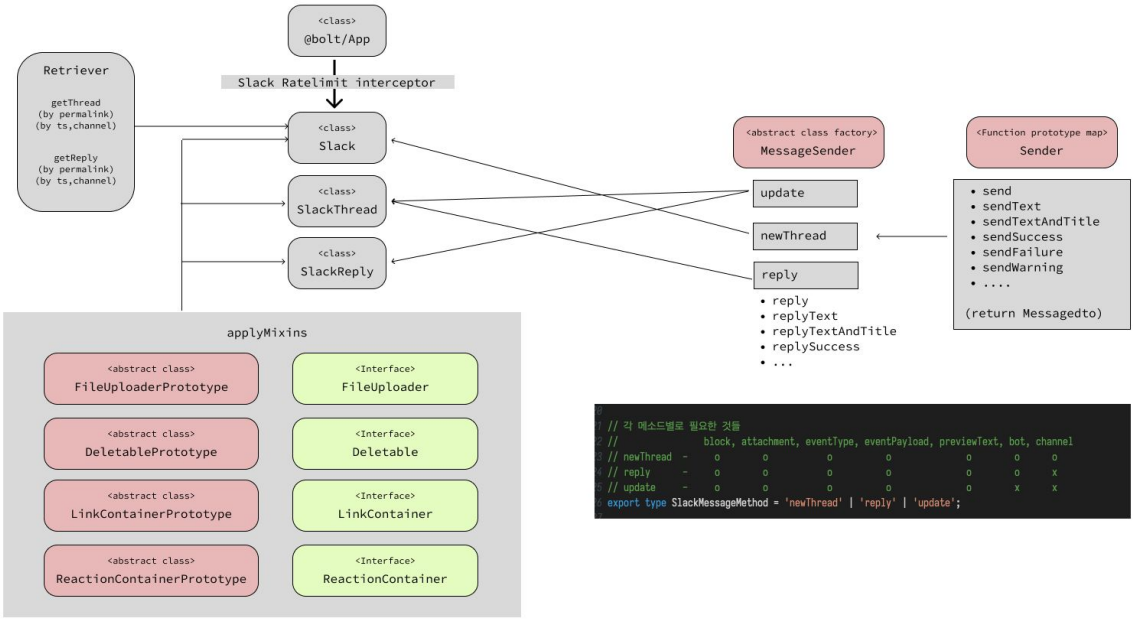
토스에서는 슬랙을 씁니다!

수많은 업무와 논의가 슬랙에서 이뤄지죠.

슬랙에 있는 정보들을 쉽게 다룰 수 있고, 정보를 쉽게 전달할 수 있다면
그만큼 강력한게 없을 것 같습니다.

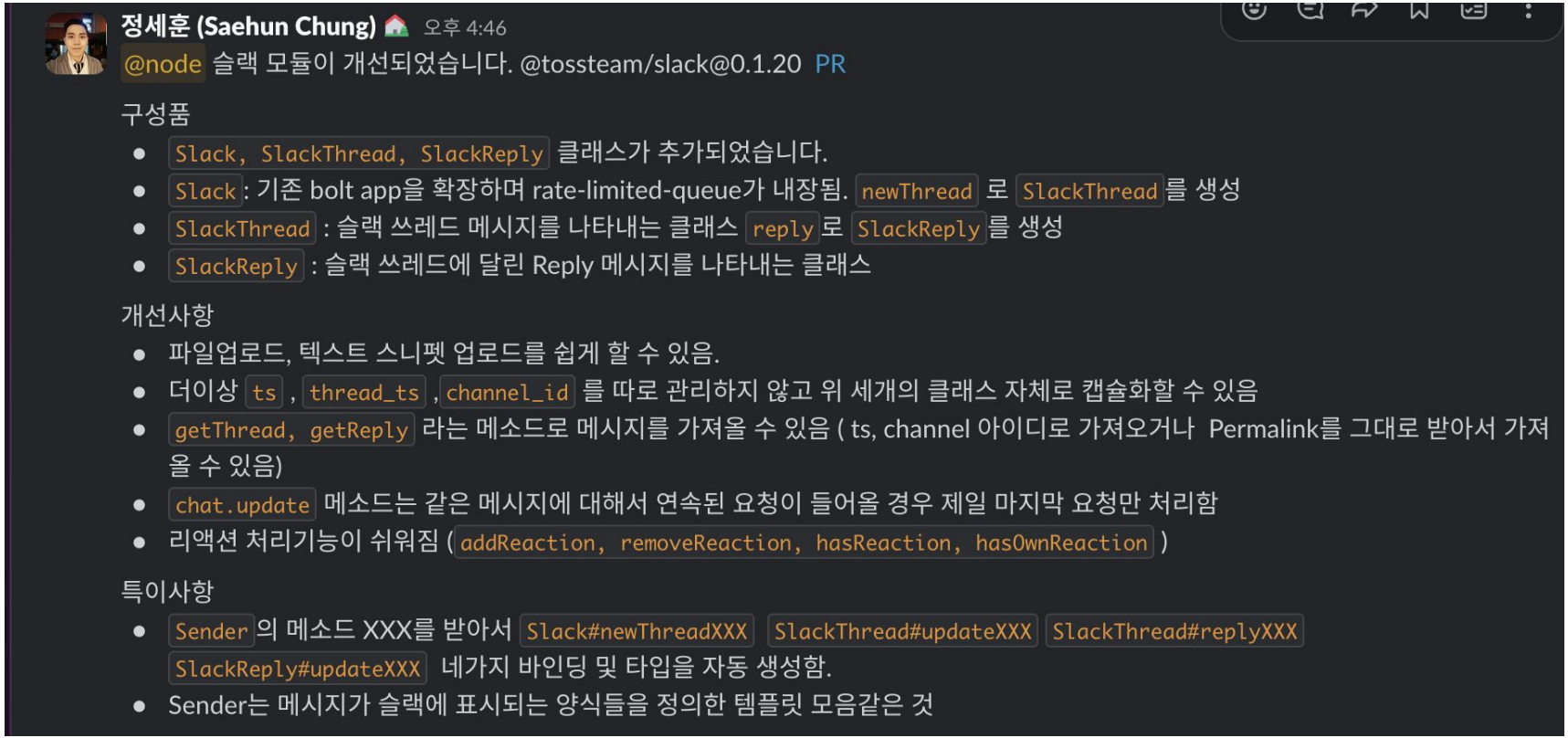
web-automation-libraries (slack, slack-listener)

```
pnpm install @tossteam/slack@0.1.20
```




```ts
// 각 메소드별로 필요한 것들
//              block, attachment, eventType, eventPayload, previewText, bot, channel
// newThread  -    o        o          o          o             o       o     o
// reply      -    o        o          o          o             o       o     x
// update     -    o        o          o          o             o       x     x
export type SlackMessageMethod = 'newThread' | 'reply' | 'update';
```

web-automation-libraries (slack, slack-listener)



정세훈 (Saehun Chung) 오후 4:46

`@node` 슬랙 모듈이 개선되었습니다. @tossteam/slack@0.1.20 PR

구성품

- `Slack, SlackThread, SlackReply` 클래스가 추가되었습니다.
- `Slack` : 기존 bolt app을 확장하며 rate-limited-queue가 내장됨. `newThread` 로 `SlackThread` 를 생성
- `SlackThread` : 슬랙 쓰레드 메시지를 나타내는 클래스 `reply` 로 `SlackReply` 를 생성
- `SlackReply` : 슬랙 쓰레드에 달린 Reply 메시지를 나타내는 클래스

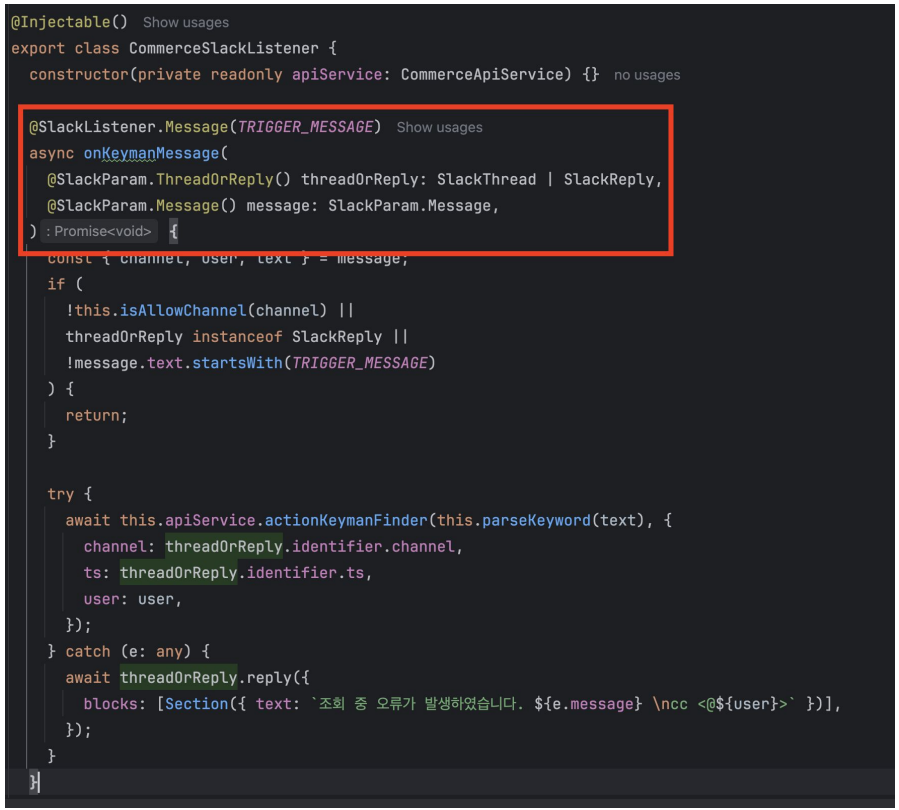
개선사항

- 파일업로드, 텍스트 스니펫 업로드를 쉽게 할 수 있음.
- 더이상 `ts` , `thread_ts` , `channel_id` 를 따로 관리하지 않고 위 세개의 클래스 자체로 캡슐화할 수 있음
- `getThread, getReply` 라는 메소드로 메시지를 가져올 수 있음 (ts, channel 아이디로 가져오거나 Permalink를 그대로 받아서 가져올 수 있음)
- `chat.update` 메소드는 같은 메시지에 대해서 연속된 요청이 들어올 경우 제일 마지막 요청만 처리함
- 리액션 처리기능이 쉬워짐 (`addReaction, removeReaction, hasReaction, hasOwnReaction`)

특이사항

- `Sender` 의 메소드 XXX를 받아서 `Slack#newThreadXXX` `SlackThread#updateXXX` `SlackThread#replyXXX` `SlackReply#updateXXX` 네가지 바인딩 및 타입을 자동 생성함.
- Sender는 메시지가 슬랙에 표시되는 양식들을 정의한 템플릿 모음같은 것

web-automation-libraries (slack, slack-listener)



```
@Injectable()  Show usages
export class CommerceSlackListener {
  constructor(private readonly apiService: CommerceApiService) {}  no usages

  @SlackListener.Message(TRIGGER_MESSAGE)  Show usages
  async onKeymanMessage(
    @SlackParam.ThreadOrReply() threadOrReply: SlackThread | SlackReply,
    @SlackParam.Message() message: SlackParam.Message,
  ) : Promise<void> {
    const { channel, user, text } = message;
    if (
      !this.isAllowChannel(channel) ||
      threadOrReply instanceof SlackReply ||
      !message.text.startsWith(TRIGGER_MESSAGE)
    ) {
      return;
    }

    try {
      await this.apiService.actionKeymanFinder(this.parseKeyword(text), {
        channel: threadOrReply.identifier.channel,
        ts: threadOrReply.identifier.ts,
        user: user,
      });
    } catch (e: any) {
      await threadOrReply.reply({
        blocks: [Section({ text: `조회 중 오류가 발생하였습니다. ${e.message} \ncc <@${user}>` })],
      });
    }
  }
```

마무리하는 글

Toss Node.js Chapter는 토스 커뮤니티의 기술적 엣지를 만들어내는 조직이라고 생각합니다.

마무리하는 글

👆

Node.js Chapter 채용

👆

Next 채용
지원하기!