DEV-CAT

웹 애플리케이션 엔지니어 과제 문서

프로젝트명: 미니 트렐로(Mini-Trello)

작성자: 권민호

2022년 3월 3일

DEV-CAT

목차

0. 이용 가이드

서비스 실행 방법

본 프로젝트의 서비스를 실행하기 위해서는 아래 두 프로그램을 설치해야 합니다.

- Node.js (16.13.0 이상)
- npm (노드 패키지 매니저)

프로젝트에서 서비스를 실행하기 위한 과정은 다음과 같습니다.

1. **"웹프로그래머엔지니어-권민호-프로젝트.zip"** 압축 풀기

2. 터미널 창에서 **cd 커맨드**를 사용하여, 프로젝트의 루트 경로로 이동

```
minho ~/Desktop cd 시범/웹프로그래머엔지니어-권민호-프로젝트
minho ~/Desktop/시범/웹프로그래머엔지니어-권민호-프로젝트 main ±
```

3. 터미널 창에 **npm i** 커맨드를 입력하여, 프로젝트에 필요한 패키지들 다운받기

```
minho ~/Desktop/시범/웹프로그래머엔지니어-권민호-프로젝트 main ± npm i
```

4. 터미널 창에 **npm run build** 커맨드를 입력하여, 타입스크립트 파일들을 자바스크립트로 변환하기

```
minho ~/Desktop/시범/웹프로그래머엔지니어-권민호-프로젝트 main ± npm run build
```

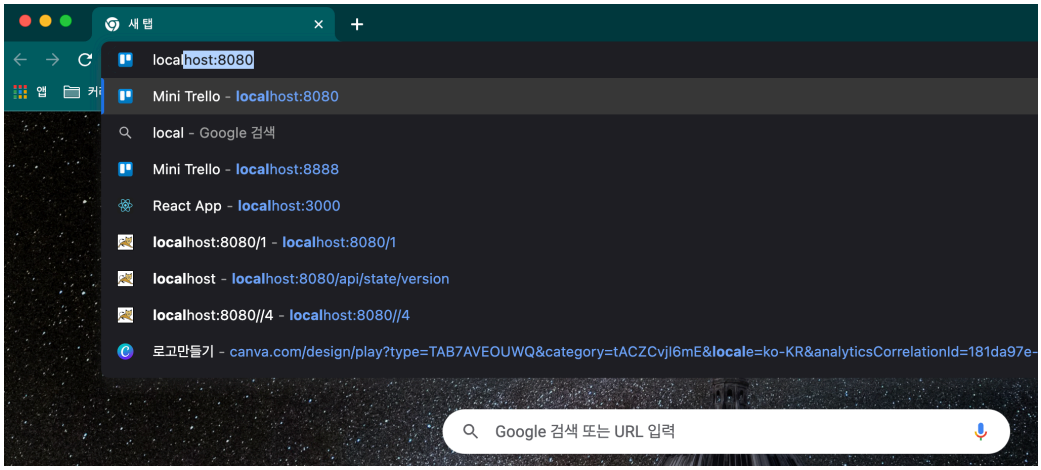
5. 터미널 창에 **npm run start** 커맨드를 입력하여, 서비스 실행하기 (서비스 접속 주소: http://localhost:8080)

```
minho ~/Desktop/시범/웹프로그래머엔지니어-권민호-프로젝트 main ± npm run start

> mini-trello@1.0.0 start
> npm run build & node dist/server.js

App listening on the port 8080

> mini-trello@1.0.0 build
> tsc
```

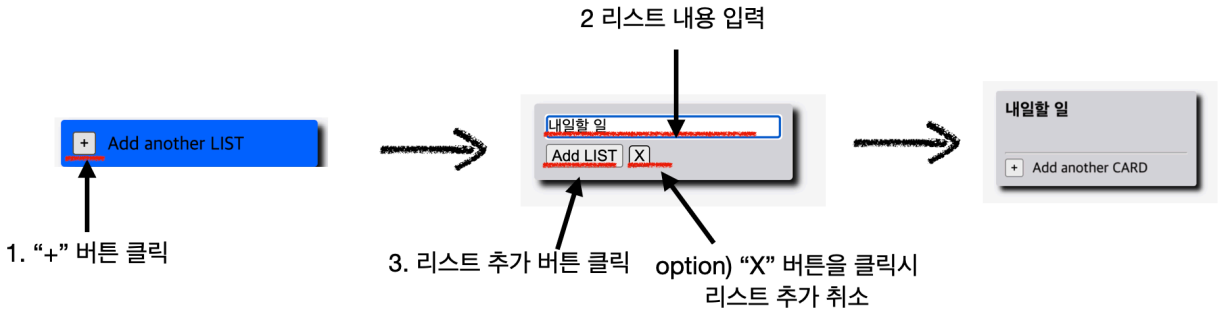
서비스 이용 방법

1. 브라우저[1]로 서비스 접속하기

아래 그림과 같이 브라우저 창을 키고, http://localhost:8080 주소를 입력하여 서비스에 접속합니다.



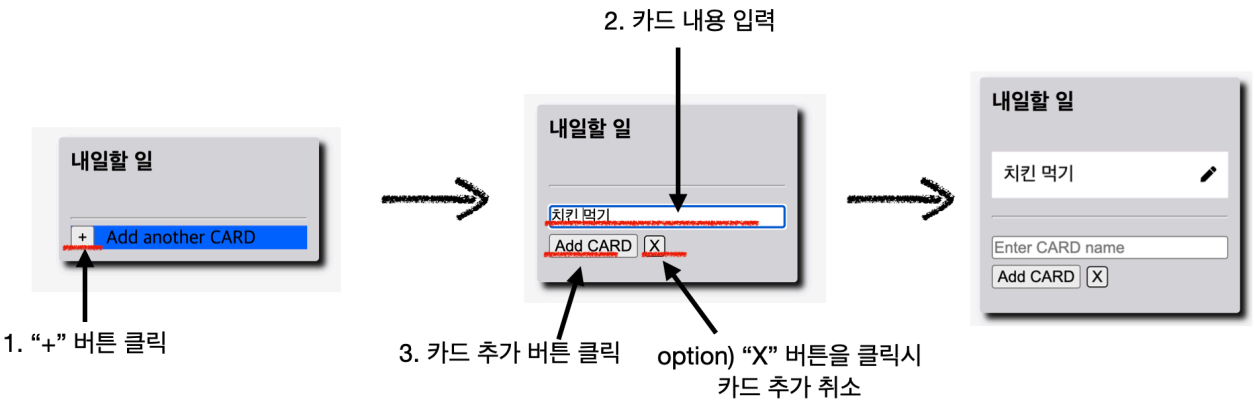
2. 리스트[2] 생성하기



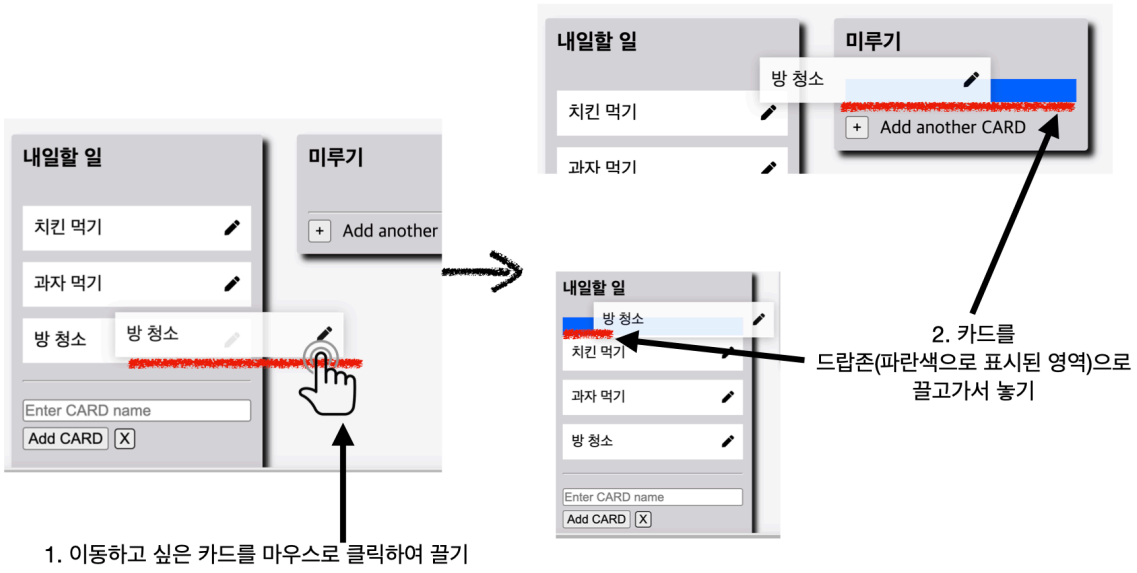
[1] **브라우저**: 웹에서 사용되는 페이지, 이미지, 비디오 등의 콘텐츠를 수신, 전송 및 표현하는 소프트웨어. 대표적인 웹 브라우저는 크롬(Chrome) 브라우저, 웨일(Whale) 브라우저, 엣지(Edge) 브라우저가 있다.

[2] **리스트**: 수행할 작업들에 대한 테마

3. 카드[3] 생성하기



4. 드래그 앤 드랍[4]으로 카드 이동하기



[3] **카드**: 수행할 작업에 대하여 기록된 곳

[4] **드래그 앤 드랍**: 마우스로 물체를 끌어서, 다른 위치로 옮기는 일

1. 명세 분석

프로젝트를 수행하기 위하여 만들었던 것

A. 요구사항 문서, 유스 케이스 다이어그램, 와이어 프레임

- 작성된 요구사항 문서는 용어 정리, 기능적 요구사항, 비 기능적 요구사항이 정리되어 있습니다.
- 작성된 유스 케이스 다이어그램은 구현될 서비스와 사용자 사이의 상호 작용이 표현되어 있습니다.
- 작성된 와이어 프레임은 구현해야할 웹 애플리케이션의 UI가 표현되어 있습니다.
- 위의 문서와 그림들은 다음과 같은 이유로 작성하였습니다.
  - 과제 참고 영상에서 요구 사항들을 추출하여 명확하게 정리하기 위함
  - 과제 참고 영상으로는 생각하지 못한 “애매한 요구 사항"을 찾아내기 위함
  - 추후 협업할 수 있는 다른 개발자가 해당 서비스를 보다 손쉽게 이해할 수 있도록 돕기 위함

B. 데이터 모델링

- 데이터 모델링은 개념적 데이터 모델링과 논리적 데이터 모델링을 작성하였습니다.
- 각 데이터 모델링은 ERD(Entity Relationship Diagram) 형태로 작성되었습니다.
- 데이터 모델링은 다음과 같은 이유로 작성하였습니다.
  - 서비스 구현에 필요한 정보들을 요구사항으로 부터 구체적으로 뽑아내어 추상화하기 위함
  - 서비스 구현에 필요한 정보들을 그룹화하기 위함
  - 그룹화된 정보들 사이의 관계를 파악하기 위함

C. 깃 커밋 메시지 규칙

- 본 프로젝트에서 히스토리는 깃(Git)으로 관리하고, 깃 커밋 메시지 규칙을 정하였습니다.
- 깃 커밋 메시지 규칙은 다음과 같은 이유로 정하였습니다.
  - 프로젝트 작업에 대한 히스토리를 체계적으로 남기기 위함
  - 추후에 협업할 개발자가 프로젝트의 작업 흐름을 보다 쉽게 파악하도록 돕기 위함

명세 분석에 있어서 모호한 부분에 관하여

A. 요구사항 모호함

주제	내용	선택	이유
(카드 업데이트 기능) 구현 vs 미구현	과제 참고 영상의 해설자 음성에서는 카드 업데이트 기능에 대하여 언급하였지만, 영상 자체에서는 카드 업데이트 기능을 보여주지 않음	미구현하기로 결정	1. 주어진 참고 영상과 pdf 파일로는 도저히 카드 업데이트 기능에 대한 요구사항을 명확하게 판단할 수 없기 때문에
(디폴트 리스트) 제공 vs 제공하지 않음	과제 참고 영상에서 보여지는 “To-do”, “Done” 과 같은 리스트를 사용자가 만들지 않아도 기본적으로 주어지는 디폴트 리스트를 제공해야하는가	제공하지 않도록 결정	1. 현재 요구사항에서 리스트의 삭제 기능이 존재하지 않기 때문에, 디폴트 리스트를 원치 않는 사용자는 서비스 이용에 불편을 겪을 수 있음

B. 제약 사항 모호함

주제	내용	선택	이유
(카드, 리스트 최대 생성 개수) 개수 제한 vs 개수 제한 없음	사용자가 생성할 수 있는 카드 혹은 리스트의 최대 생성 개수의 제약이 필요한가	개수 제한 없도록 결정	1. 현재 구현되는 카드나 리스트는 텍스트 기반으로 생성되어, 개수 제한을 두지 않아도 서버에 큰 무리가 없을거라고 판단함 2. 단, 추후 해당 서비스에 추가 기능들이 추가되면, 우선적으로 최대 개수 제약에 대한 기능을 우선적으로 구현할 필요가 있다고 프로젝트에 기록해놓기
(동일한 내용의 카드, 리스트 중복 생성) 허용함 vs 허용하지 않음	사용자가 동일한 내용의 카드나 리스트를 생성할 수 있는가	중복 생성 허용하도록 결정	1. 중복 생성을 하고 싶은 사용자가 있을 수 있는 상황에서, 중복 생성하지 못하도록 하는 것은 서비스를 이용하는 사용자에게 불편함을 줄 수 있다고 생각함. 2. 단, 추후에 동일한 내용의 리스트나 카드를 생성하려고 한다면, 최소한 동일한 내용의 카드나 리스트가 존재한다고 알려주는 기능은 구현할 필요가 있다고 프로젝트에 기록해놓기

2. 설계

전체 아키텍처

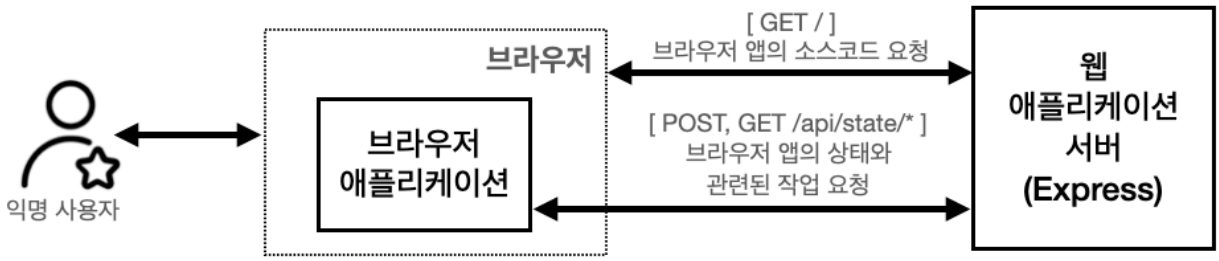
본 웹 서비스 아키텍처는 크게 프론트엔드[5]의 **브라우저 애플리케이션**과 백엔드[6]의 **웹 애플리케이션 서버**로 구분됩니다.

브라우저 앱

- 사용자에게 동적 웹 페이지를 제공하는 브라우저 기반의 애플리케이션 (GUI[7] 제공)

웹 애플리케이션 서버

- HTTP 기반으로 브라우저 앱의 상태를 관리하는 API 서버 (사용자가 서비스를 접속할 때, 브라우저 앱의 소스코드를 사용자한테 전달하는 것 외에는 웹 페이지와 관련된 작업은 수행하지 않음)



이와 같이 프론트엔드와 백엔드를 나누어서 아키텍처를 구성한 이유는 다음과 같습니다.

- **클라이언트 사이드[8] 형태로 웹 페이지를 랜더링하기 위함 :** 해당 서비스는 사용자와 상호작용으로 인하여 웹 페이지 변화가 잦기 때문에, 서버 사이드[9] 형태로 웹 페이지를 렌더링하는 것은 UX 측면에서 좋지 못합니다.
- **하나의 구성 요소[10]는 하나의 역할[11]만 책임지도록 하기 위함**: 이는 하나의 구성요소가 하나의 역할만 담당하기 때문에, 개발자는 구성 요소의 개발 과정에서 집중해야하는 문제가 무엇인지 명확하게 인지할 수 있습니다. 또한, 각 구성 요소가 어떤 문제를 담당하고 있는지 쉽게 파악할 수 있어 프로젝트의 유지보수에 용이합니다.

---

[5] **프론트엔드**: 사용자와 직접적으로 상호작용하는 인터페이스와 로직에 관한 영역

[6] **백엔드**: 사용자가 직접적으로 상호작용하지 못하는 서버나 데이터베이스에 관한 영역

[7] **GUI(Graphical user interface)**: 그래픽 유저 인터페이스

[8] **클라이언트 사이드 렌더링**: 사용자 브라우저 단에서 웹 페이지를 변화시키는 방식

[9] **서버 사이드 렌더링**: 서버를 통해서 사용자가 이용하는 웹 페이지를 변화시키는 방식

[10] **구성 요소**: 여기서 구성요소란, 아키텍처를 구성하는 요소를 의미함

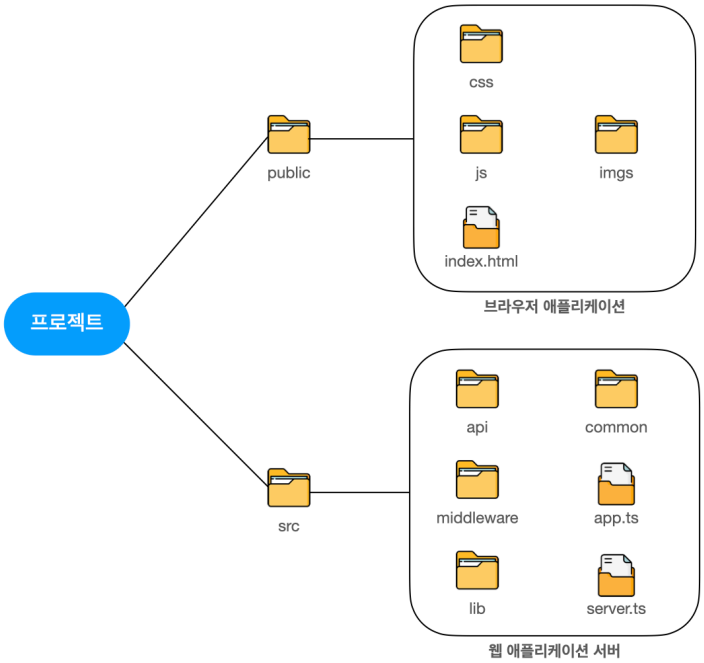
[11] **역할**: 구성 요소가 특정 범위의 문제에 있어서 담당하는 기능

프로젝트 구조

본 서비스의 프로젝트 구조는 전체 아키텍처 구성과 같이 크게 2가지로 구분할 수 있습니다.

public 폴더: 브라우저 애플리케이션의 소스코드가 담긴 폴더

src 폴더: 웹 애플리케이션 서버의 소스코드가 담긴 폴더



public 폴더 구조

브라우저 애플리케이션의 소스코드가 담긴 public 폴더의 구조는 다음과 같습니다.

- **index.html**: 브라우저 애플리케이션의 초기 화면을 나타내는 html[12] 파일
- **imgs 폴더**: 브라우저 애플리케이션에 사용하는 이미지 파일이 담긴 폴더
- **css 폴더**: 브라우저 애플리케이션의 화면 스타일링을 담당하는 css[13] 파일들이 있는 폴더
- **js 폴더**: 브라우저 애플리케이션의 핵심 로직이 담긴 js[14] 파일들이 있는 폴더

---

[12] **html**: 컨텐츠의 구조를 정의하는 마크업 언어

[13] **css**: HTML 요소들이 각종 미디어에서 어떻게 보이는가를 **정의**하는 데 사용되는 스타일 시트 언어

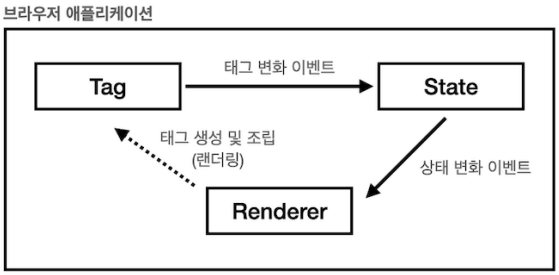
[14] **js**: 프로그래밍 언어 중 하나로, 객체 기반의 스크립트 프로그래밍 언어

public 폴더에서 제일 중요한 요소는 브라우저 애플리케이션의 핵심 로직이 담긴 **js 폴더**입니다.

js 폴더에는 3가지 주요한 폴더가 존재합니다.

- Render folder
- State folder
- View folder

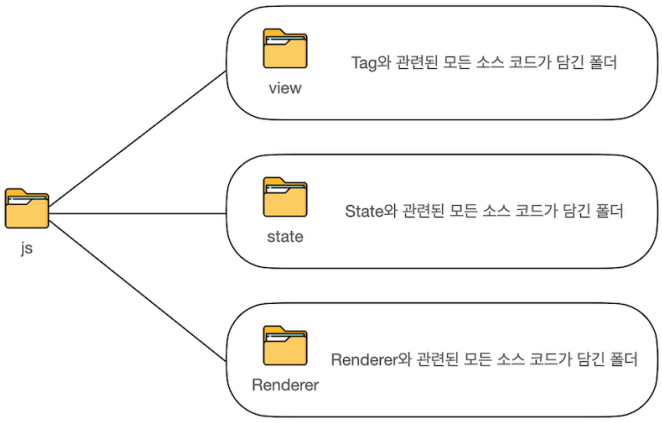
해당 각 폴더들은 **브라우저 애플리케이션의 아키텍처**를 구성하는 각 요소와 관련있습니다.



브라우저 애플리케이션을 구성하는 요소는 크게 3가지로 구분할 수 있습니다.

- 태그(Tag): 브라우저 애플리케이션의 화면을 담당하는 컴포넌트
- 상태(State): 브라우저 애플리케이션의 상태[15]를 담당하는 컴포넌트
- 랜더러(Renderer): 브라우저 애플리케이션의 화면에 대한 동적 변화를 담당하는 컴포넌트

즉, js 폴더의 각 폴더들은 브라우저 애플리케이션의 각 구성하는 컴포넌트와 대응됩니다. (아래 그림 참고)



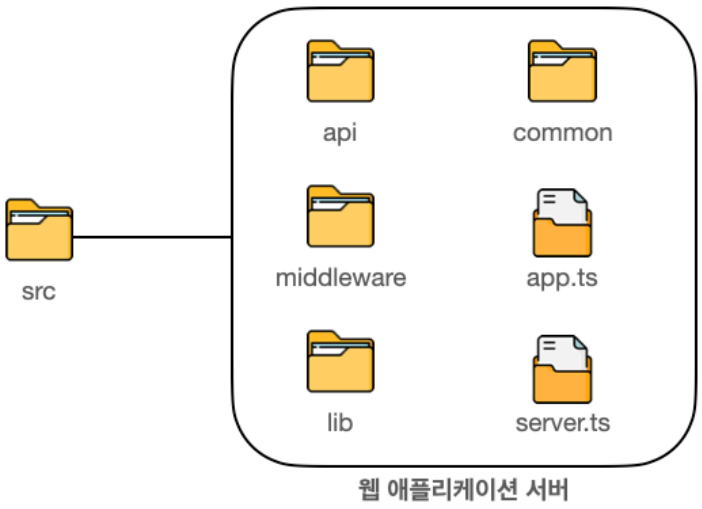
[15] **상태**: 서비스가 실행되는 동안 유지되어야하는 정보

src 폴더 구조

웹 애플리케이션 서버의 소스코드가 담긴 src 폴더의 구조는 다음과 같습니다. (모든 파일은 타입스크립트[16] 파일입니다)

- **api 폴더:** 핵심 서비스 로직을 제공하기 위한 도메인[17] 폴더가 존재하는 곳
- **middleware 폴더:** 미들웨어[18]와 관련된 소스 코드가 있는 곳
- **lib 폴더:** 유틸리티 기능을 수행하는 도구에 대한 소스 코드가 있는 곳
- **common 폴더:** src 폴더 내에서 전역적으로 사용되는 인터페이스, 예외 처리와 관련된 코드가 있는 곳
- **app.ts:** 웹 애플리케이션 서버에 대한 설정 정보[19]가 명세된 클래스 파일
- **server.ts:** 웹 애플리케이션 서버를 실행시키는 소스 코드 파일

(*lib 폴더는 단순히 유틸리티 기능만 수행하는 도구들의 소스코드가 있는 곳이고, common 폴더는 웹 애플리케이션 서버에서 중요하지만 공통적으로 이용되는 소스코드를 모아 놓은 폴더이다)



이러한 구조를 가지는 src 폴더에서 제일 중요한 요소는 **api 폴더** 입니다.

이 두 폴더를 이해하려면, 본 프로젝트의 **웹 애플리케이션 서버에 대한 아키텍처**를 이해할 필요가 있습니다.

---

[16] **타입스크립트**: 자바스크립트 언어를 정적 타입으로 명시할 수 있는 프로그래밍 언어

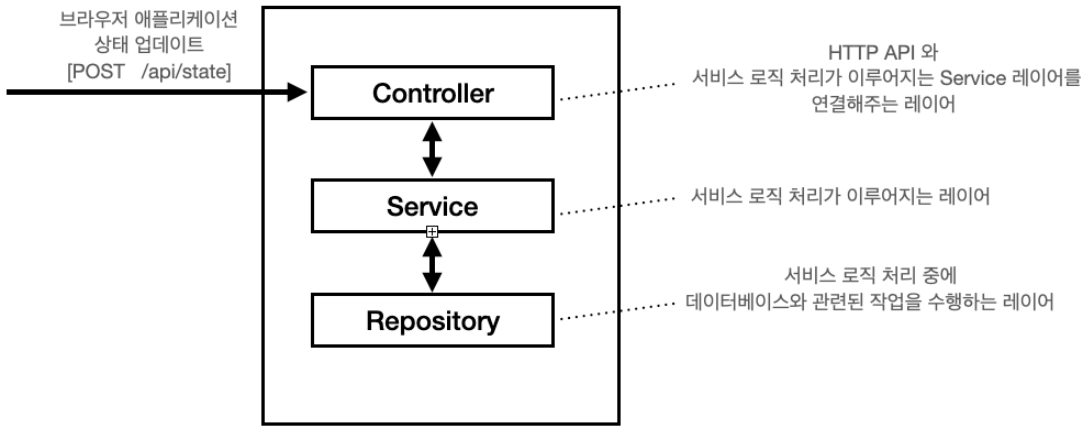
[17] **도메인**: 소프트웨어로 해결하고자하는 비슷한 영역을 모아놓은 것(ex, '회원' 가입 로직, '회원' 탈퇴 로직 —> '회원' 도메인)

[18] **미들웨어**: 핵심 서비스를 제공하기 전/후에 별도의 처리 작업을 수행하는 소프트웨어

[19] **설정 정보**: 서버의 포트, 설정할 미들웨어, 원격으로 호출할 프로그램 정보 등을 말한다.

본 웹 애플리케이션 서버에서는 브라우저 애플리케이션의 **상태 관리에 대한 도메인**(브라우저 애플리케이션의 상태 저장 및 업데이트)이 존재합니다. 그리고 이 도메인에 대한 처리는 핵심적인 **3개의 레이어**를 거쳐서 진행됩니다. (아래 그림 참고)

위의 각 레이어의 역할은 다음과 같습니다.



컨트롤러 레이어(Controller layer)

- HTTP API(http method와 url)와 서비스 로직을 처리하는 서비스 레이어와 연결하는 역할
- 서비스 레이어로 요청을 넘기기 전에, request의 query 혹은 body 값이 서비스에서 유효한 값으로 구성되었는지 검사합니다.

서비스 레이어(Service layer)

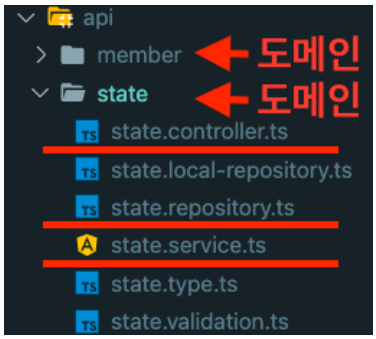
- 핵심 서비스에 대한 로직 처리 역할

리포지토리 레이어(Repository layer)

- 핵심 서비스에 대한 로직 처리 중 데이터베이스와 관련된 작업을 위임하여 처리하는 역할

이와 같이 웹 애플리케이션 서버에서 제공해야하는 전체 서비스를 도메인 단위로 분류하고,

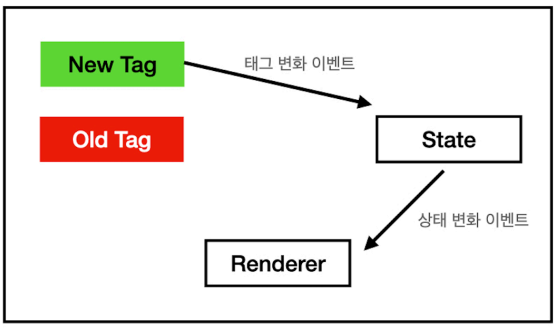
3 레이어 구조로 서비스를 처리하는 구조는 src/api 폴더에 잘 반영되어 있습니다.



설계 및 코드 작성에 대한 주안점

A. 개발하는 프로그램은 언제든지 추가 기능이나 수정 사항이 무조건 발생한다는 생각으로 설계하기

- 브라우저 애플리케이션의 각 요소는 이벤트 기반으로 상호작용하여, 최소한 의존 관계를 맺도록 설계하였습니다.



브라우저 애플리케이션의 각 요소는 서로 직접적으로 의존하지 않고, 자신이 듣고자하는 이벤트 의존합니다.

이러한 방식은 위 그림과 같이 한 요소에 수정 사항이 발생 하여도, 다른 요소의 영향을 주지 않고도

손쉽게 새로운 버전의 요소를 추가하거나 확장할 수 있습니다.

- 수정 사항이 자주 발생할 수 있는 곳은 역할과 구현을 분리하여, 각 클래스들은 역할에 의존하도록 설계하였습니다.

```
export default class StateService {
  private stateRepository: StateRepository = new StateLocalRepository();
```

웹 애플리케이션 서버의 데이터베이스는 현재 로컬 데이터베이스를 사용하고 있지만,

추후에 다른 유형의 데이터베이스를 사용할 수 있습니다. 이러한 이유로, 위 코드와 같이

웹 애플리케이션 서버에서 StateService는 StateRepository(인터페이스) 역할에 의존하도록 작성하였습니다.

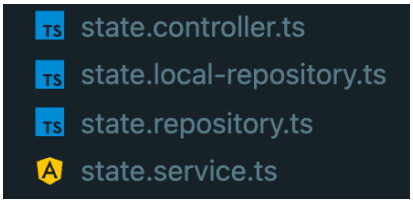
- 대부분의 클래스 혹은 함수는 많은 역할을 부여하지 않고 최소한 책임만 지도록 설계하였습니다.
- 반복적으로 작성된 코드는 한 곳으로 묶어 모듈화하였습니다.

B. 다른 개발자한테 언제든지 프로젝트를 인수인계 할 수 있는 상황이 생긴다는 생각으로 코드 작성하기

- 자바스크립트 언어는 변수의 타입을 명시적으로 나타내지 않기 때문에, 제가 만든 함수나 클래스를 다른 개발자가 사용할 때 변수의 타입을 파악할 수 없어 많은 런타임 에러를 발생시킬 수 있습니다. 그렇기 때문에 모든 함수와 클래스의 주석을 작성하였습니다.

```
/**
 * @private
 * @description: 카드 추가로 인한 상태 업데이트
 * @param {string} id: 카드에 대한 아이디
 * @param {string} title: 카드에 대한 타이틀
 * @param {number} listId: 카드에 대한 리스트 태그 아이디
 */
_addCard(id, title, listId) {
```

- 본 프로젝트에서는 대중적으로 많이 사용하는 3 레이어 아키텍처(Controller, Service, Repository)을 채택하여, 다른 개발자들이 손쉽게 프로젝트 구조를 파악할 수 있도록 하였습니다.



- 자바스크립트에서 클래스를 작성할 때, 외부에서 사용하면 안되는 프라이빗 함수는 명시적으로 표현하여, 다른 개발자들이 의도치 않는 실수를 피할 수 있도록 하였습니다.

```
_init() {
  Event.changeTagListener(this._changeTagEventHandler.bind(this));
}
```

  위 사진과 같이 Private 함수의 네이밍은 맨 앞에 '_'를 붙여 명시적으로 나타냅니다.

- 하나의 함수를 만들 때에는 최소한 기능만을 수행하도록하여 최대한 간결한 코드가 작성되도록 하였습니다.
- 저가 만든 아키텍처 혹은 개발 방식에서 다른 개발자가 실수할 수 있는 부분은 명시적인 가이드 라인을 두어 실수를 피하도록 하였습니다.

  브라우저 애플리케이션의 각 요소들은 이벤트 방식으로 통신합니다. 그러나 개발 과정에서 이벤트를 발생시키고, 다른 요소가 이벤트를 캐치하여 각 요소가 통신하도록 하는 작업은 개발자들이 개발 과정에서 상당히 헷갈릴 수 있고 실수를 많이할 수 있다고 생각했습니다.

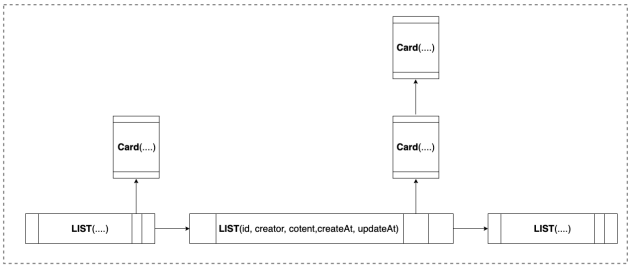
  이러한 이유로, 본 개발 과정에서는 이벤트 생성 및 캐치하는 작업을 한 곳에서 정의하고 관리할 수 있도록 Event 클래스를 만들어 개발 과정에서 헷갈리거나 실수할 수 있는 부분을 보완하도록 하였습니다.

선택하지 않은 설계 대안

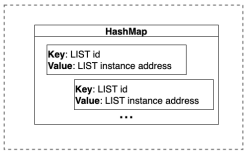
A. 로컬 데이터베이스 구조

본 프로젝트에서 웹 애플리케이션 서버는 메모리 기반의 로컬 데이터베이스를 사용하기로 결정하였습니다.

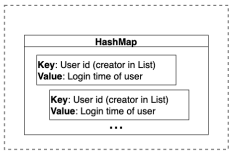
설계 당시에 로컬 데이터베이스의 자료 구조는 아래 그림과 같이 **연결 리스트** 기반으로 구성하였습니다.



LIST&CARD



LIST INDEX



USER

이렇게 설계한 이유는 해당 서비스의 특성상(카드의 생성 및 이동으로 인하여) 데이터에 대한 삽입 및 업데이트 연산이 많이 발생할 수 있다고 생각하였습니다. 그렇기 때문에 삽입과 업데이트 연산에 유리한 연결 리스트와 연결 리스트의 탐색 속도를 보완하기 위한 해시 테이블로 로컬 데이터베이스의 구조를 설계하였습니다.

그러나, 다음과 같은 이유로 개발을 진행하는 과정에서 설계한 구조를 사용하지 않았습니다.

- 저장되는 데이터의 수가 엄청 크지 않은 이상(ex. 최소 1억 개 이상), 배열 기반으로 데이터를 관리하는 것과 연결 리스트 기반으로 관리하는 것의 성능 차이는 크지 않을 거라 생각했습니다. 그리고 저장되는 데이터의 수가 많아지면, 로컬 데이터베이스 대신에 속도와 안정성이 검증된 다른 데이터베이스로 교체될 확률이 크다고 생각했습니다.

3. 프로젝트 자세히 보기

프로젝트의 모든 파일

[public 폴더]

- public
  - **index.html** : 브라우저 애플리케이션의 초기 화면을 나타내는 html 파일
  - **imgs/favicon.png**: 브라우저 애플리케이션에 사용하는 파비콘[20] 이미지

- public/js/render
  - **renderer.js**: 모든 Renderer의 부모 클래스 (Renderer 클래스는 태그 컴포넌트[21]를 생성하고 조립하는 역할을 수행)
  - **board.renderer.js**: 보드 섹션[22]과 관련된 태그 컴포넌트를(카드, 리스트) 생성하고 조립하는 역할 수행

- public/js/state
  - **board-state-manager.js**: 보드 섹션과 관련된 브라우저 앱의 상태를 관리하는 매니저 클래스가 명세된 파일
  - **state.js**: 브라우저 애플리케이션의 전체 상태를 관리하는 서비스 컴포넌트[23] 클래스

- public/js/common
  - **enums.js**: 브라우저 애플리케이션에서 사용하는 enum 타입들이 정의된 파일
  - **event.js**: 브라우저 애플리케이션에서 사용하는 이벤트들이 명세 되어있는 클래스 파일
  - **http.js**: http 클라이언트 모듈이 작성된 파일
  - **uuid.js**: uuid 생성에 관한 유틸 함수가 작성된 파일

- public/css
  - **index.css**: 웹 페이지의 전체 섹션에서(브라우저 애플리케이션의 UI) 사용되는 스타일링 속성이 설정된 css 파일
  - **board.css**: 웹 페이지의 보드 섹션에서 사용되는 스타일링 속성이 설정된 css 파일
  - **header.css**: 웹 페이지의 헤더 섹션에서 사용되는 스타일링 속성이 설정된 css 파public/js/view

---

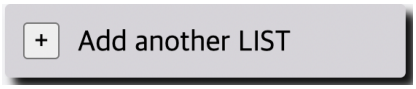
[20] **파비콘**: 웹페이지에 접속했을때, 상단 탭에 보여지는 아이콘

[21] **태그 컴포넌트**: 브라우저 애플리케이션의 UI를 만들때 사용되는 태그의 묶음. 현재 브라우저 애플리케이션의 태그 컴포넌트는 add-item, card, dropzone, list가 존재한다.

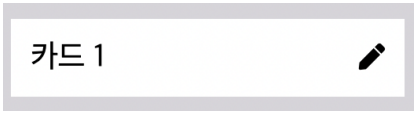
[22] **섹션**: 브라우저 애플리케이션의 UI 영역을 분할하는 단위

[23] **서비스 컴포넌트**: 브라우저 애플리케이션 내에서 서비스 로직을 처리하는 컴포넌트. 현재 브라우저 애플리케이션의 서비스 컴포넌트는 tag, state, renderer, virtual-dom이 존재한다.

- public/view
  - **addItem.js**: Add-Item 태그 컴포넌트 생성과 관련된 소스 코드가 있는 클래스 파일 (아래 이미지 참고)

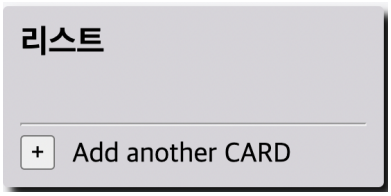


  - **card.js**: Card 태그 컴포넌트 생성과 관련된 소스 코드가 있는 클래스 파일



  - **dropzone.js**: Dropzone 태그 컴포넌트 생성과 관련된 소스 코드가 있는 클래스 파일

  - **list.js**: list 태그 컴포넌트 생성과 관련된 소스 코드가 있는 클래스 파일 (아래 이미지 참고)



  - **virtual-dom.js**:
    - VirtualDom 서비스 컴포넌트에 대한 클래스
    - 서버와 상호작용하여, 서버에서 관리하는 브라우저 애플리케이션의 상태를 업데이트하고 가져오는 역할을 수행

[src 폴더]

- src/api
  - state [브라우저 애플리케이션 상태에 대한 도메인]
    - **state.controller.ts**: 브라우저 애플리케이션의 "상태 도메인" 처리에 대한 컨트롤러가 작성된 클래스 파일
    - **state.local-repository.ts**: 메모리 로컬 DB에 대한 상태 리포지토리 구현체가 작성된 클래스 파일
    - **state.repository.ts**: 브라우저 애플리케이션의 “상태 도메인”에 대한 리포지토리 인터페이스가 작성된 파일
    - **state.service.ts**: 브라우저 애플리케이션의 "상태 도메인"에 대한 서비스 로직이 담긴 클래스 파일
    - **state.type.ts**: 브라우저 애플리케이션의 “상태 도메인”에서 사용되는 타입들이 정의된 파일
    - **state.validation.ts**: 브라우저 애플리케이션의 "상태 도메인" 처리 요청에 대한 쿼리 값이나 바디 값에 대한 유효성을 검증하는 함수들이 작성된 파일

- src/common/exceptions
  - **index.ts**: 모든 예외 클래스의 export 를 담당하는 인덱스 파일
  - **bad-request.exception.ts**: 잘못된 HTTP 요청에 대한 예외 처리를 할 때 사용하는 클래스가 작성된 파일
  - **custom-http.exception.ts**: 모든 예외 클래스의 부모 클래스가 작성된 파일
  - **unauthorized.exception.ts**: 인증되지 않은 사용자 요청에 대한 예외 처리를 할 때 사용하는 클래스가 작성된 파일

- src/common/interface
  - **controller.ts**: 모든 컨트롤러에 대한 공통 규격이 명세된 파일 (컨트롤러 인터페이스)

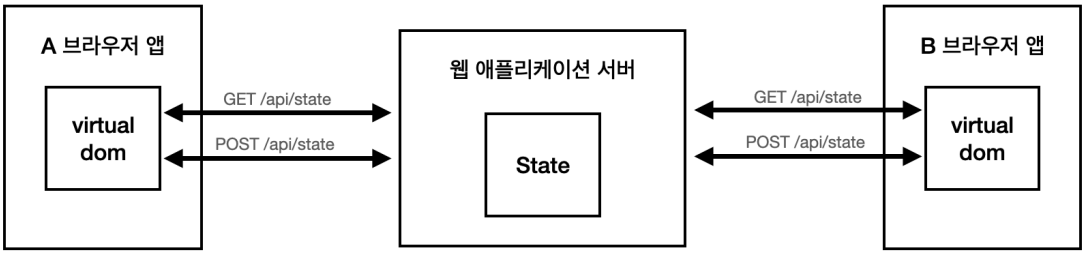
- src/lib
  - **exclude-path.ts**: 특정 미들웨어에서 제외하고 싶은 http의 url들을 설정하도록 돕는 래퍼 함수[24]
  - **request-handler.ts**: 원격 프로그램(컨트롤러-서비스-리포지토리)의 반환값을 json 형태로 변환하고, 에러 발생시에 에러 처리 미들웨어로 전달해주는 래퍼 함수

[24] **래퍼 함수**: 기존 함수를 한 번감싸서 원래 동작에 약간의 처리를 추가하는 함수

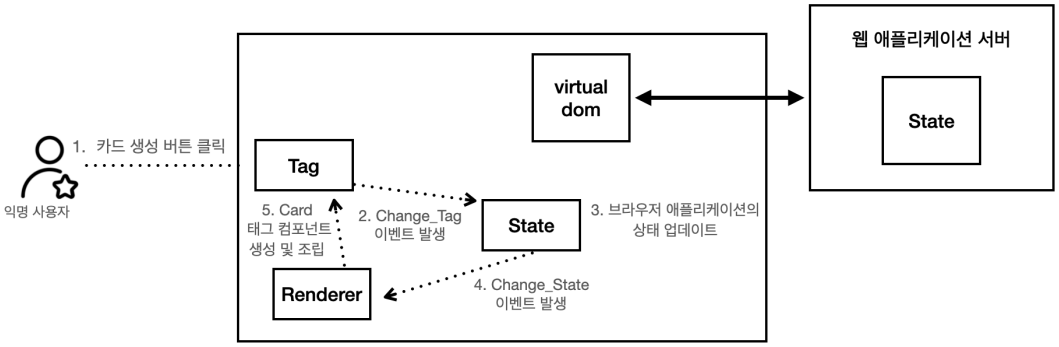
- src/middleware
  - **index.ts**: 모든 미들웨어 클래스의 export 를 담당하는 인덱스 파일
  - **not-found.middleware.ts**: 지원하지 않은 HTTP url과 method에 접근할 경우에, 에러 페이지를 반환해주는 미들웨어가 작성된 파일
  - **exception.middleware.ts**: 웹 애플리케이션 서버에서 발생하는 모든 예외 처리에 대하여 핸들러하는 미들웨어가 작성된 파일
  - **ignore-favicon.middleware.ts**: 브라우저가 자동으로 favicon.icon 요청하는 것에 대하여 필터링해주는 미들웨어가 작성된 파일
- src/app.ts: 웹 애플리케이션 서버에 대한 설정 정보가 명세된 클래스 파일
- src/server.ts: 웹 애플리케이션 서버를 실행시키는 소스 코드 파일

브라우저 애플리케이션의 상태가 웹 애플리케이션 서버에서 관리되는 과정

브라우저 애플리케이션의 상태는(생성된 리스트 및 카드 정보) 웹 애플리케이션 서버에 저장되고 업데이트 됩니다. 그렇기 때문에, 다른 브라우저 애플리케이션에서도 이미 생성된 리스트와 카드에 대하여 접근할 수 있습니다.



이것이 가능한 이유는 브라우저 애플리케이션의 **virtual dom** 이라는 서비스 컴포넌트 덕분입니다. virtual dom은 웹 애플리케이션 서버에 브라우저 애플리케이션의 상태를 저장하고 업데이트하는 역할을 수행합니다. virtual dom의 동작 과정을 이해하려면, 먼저 브라우저 애플리케이션 내에서 카드 및 리스트가 어떻게 생성되는지 이해할 필요가 있습니다. 아래 그림은 브라우저 애플리케이션 내에서 새로운 카드가 생성되는 과정을 나타냅니다.

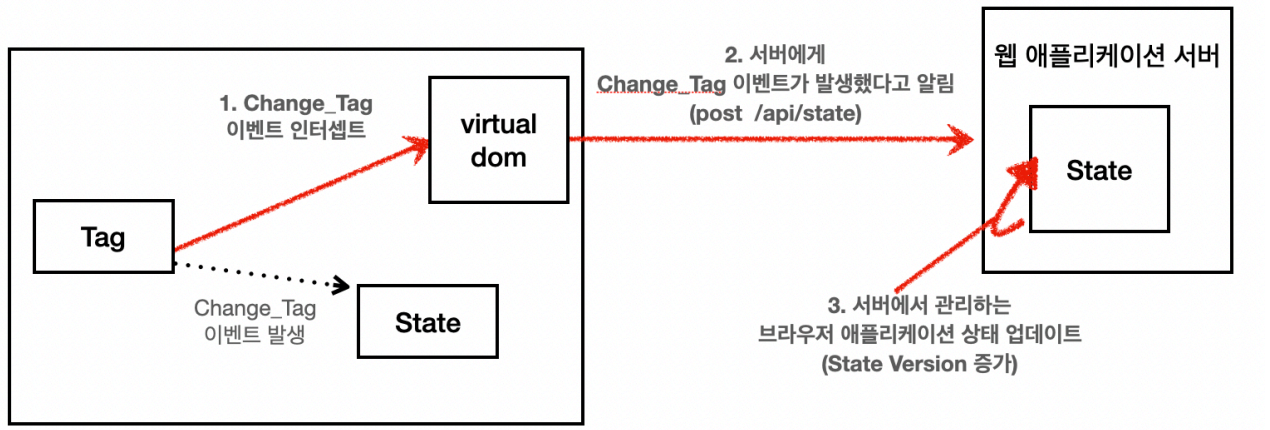


1. 서비스 사용자가 카드 생성 버튼을 클릭하게 되면, 브라우저 애플리케이션 내에서 화면을 담당하는 태그(Tag) 컴포넌트가 “Change_Tag" 이벤트를 발생시킴 (해당 이벤트에는 새로운 카드에 대한 정보가 포함됩니다)

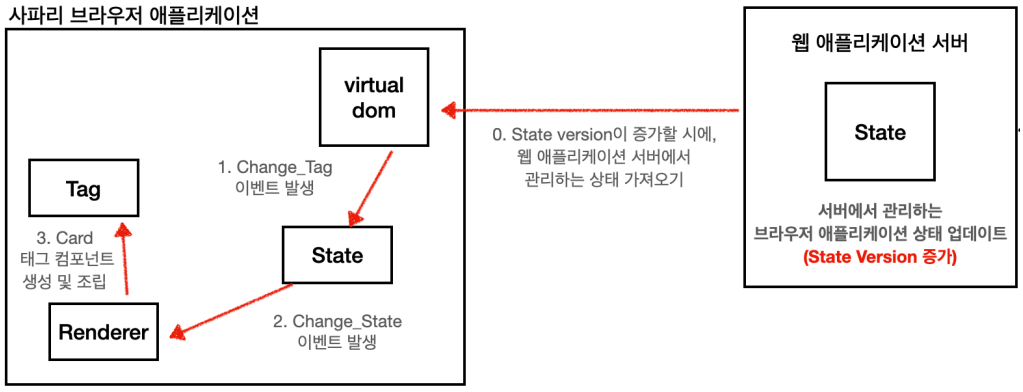
2. “Change_Tag” 이벤트를 리스너하고 있는 상태(State) 컴포넌트는 관리하고 있는 브라우저 애플리케이션의 상태에 새로운 카드 정보를 추가하여 업데이트함. 그리고 상태 컴포넌트가 관리하는 상태가 업데이트되었다고 브라우저 애플리케이션 내에 컴포넌트들에게 “Change_State” 이벤트를 발생시켜 알림

3. “Change_State” 이벤트를 리스너하고 있는 렌더러(Renderer) 컴포넌트는 해당 이벤트에서 새로운 카드 정보를 추출하여 새로운 카드 태그를 생성함. 그리고 이 카드를 보드 섹션에 추가하여 해당 브라우저 애플리케이션의 화면에 새로운 카드를 출력해줌

이러한 과정을 통해 브라우저 애플리케이션 내부에서 새로운 카드가 추가될 때, **virtual dom은 태그 컴포넌트가 발생시킨 "Change_Tag" 이벤트를 가로챕니다.** 그리고 해당 이벤트를 통해 새롭게 추가될 카드 정보를 서버에게 전달하 웹 애플리케이션 서버에서 관리하는 상태를 업데이트 시킵니다.



virtual dom에 의하여 웹 애플리케이션 서버에서 관리하는 상태가 업데이트가 되면, **이 상태에 대한 업데이트 회수를 나타내는 상태 버전이(State version) 하나 증가하게 됩니다. (그림 참고)**



이 상태 버전의 변화는 다른 브라우저 애플리케이션의 virtual dom에게 해당 서버 내의 상태가 새롭게 업데이트되었음을 알려주는 신호가 됩니다. 이 신호를 감지한 다른 브라우저 애플리케이션에서 virtual dom은 서버 내에서 새롭게 업데이트 된 상태를 가져와, 자신이 속한 브라우저 애플리케이션의 상태를 업데이트 시켜 새로운 카드의 정보가 출력되도록 합니다.