|  |
| --- |
| **HTML5 Canvas 기반 오픈소스 이미지 에디터 라이브러리 개발**  팀원: 2015104220 정종윤 |

1. **요약**

HTML5부터 등장한 Canvas API를 이용하면 웹에서도 이미지 편집을 비롯한 다양한 그래픽 기술을 처리할 수 있다. 하지만 기본적으로 제공하는 API의 추상화 단계가 낮고, 그래픽 도메인에 대한 지식을 별도로 학습해야 한다는 단점이 있다. 따라서 대부분의 경우에는 Canvas API를 직접 활용하기보다는 완성된 라이브러리를 활용하는 방법을 선택한다. 본 논문에서는 이미지 편집에 초점을 맞추어, 2D 그래픽 기반의 Canvas API를 활용한 오픈소스 이미지 에디터를 구현한다. 이미지 에디터에서 제공하는 다양한 편집 기능들을 어떠한 Canvas API와 디자인 패턴으로 구현 가능한지를 연구하고, 이 과정에서 마주치는 한계점들을 극복하는 방안에 대해 알아본다.

1. **서론**
   1. **연구 배경**

HTML5부터 등장한 Canvas API는 그래픽을 표현하기 위한 웹 표준 API다. 이는 HTML5 이전까지는 웹에서 그래픽을 표현하기 위한 표준이 정해져 있지 않았다는 의미다. 초창기 웹은 단순 문서의 역할에 가까웠기 때문에, 기본적으로 제공하는 기능만으로는 풍부한 그래픽을 나타내기 어려웠다. 따라서 개발자들은 SVG 혹은 Adobe 사의 Flash를 대체제로 이용하곤 했다. 하지만 SVG는 그래픽을 객체지향적으로 관리하기 어려운 형태였고, Adobe의 Flash는 높은 성능을 갖고 있었지만 폐쇄적인 정책과 보안 이슈로 더 발전하지 못했다. 그 사이 웹은 표준에 있어 개방적인 태도를 보였고 이것이 발전하여 HTML5의 Canvas API가 되었다.

이처럼 Canvas API 가 상용화되면서 다양한 그래픽 기술을 웹에서 처리할 수 있게 되었다. Canvas API는 WebGL을 호환하기 때문에, 2D 뿐만 아니라 3D 그래픽 역시 제작이 가능하다. 이를 이용해 애니메이션, 데이터 시각화, 이미지 수정, 실시간 비디오 처리 등 화려한 기능을 별도의 플러그인 없이 구현 가능하다는 장점이 있다.

Canvas API는 그래픽과 관련된 다양한 기능을 제공하긴 하지만, 기본적으로 제공하는 추상화 단계가 낮고 그래픽 도메인에 대한 지식을 별도로 학습해야 한다는 단점이 있다. 따라서 개발과 유지보수에 드는 비용을 절약하기 위해 일반적으로는 필요한 기능을 제공하는 라이브러리 형태로 설치하여 사용하게 된다.

웹 상에서 이미지를 불러와서 편집할 수 있는 이미지 에디터 역시 오픈소스 라이브러리로 제공되는 것이 있다. 하지만 그 수가 매우 적고, 그나마 존재하는 대부분의 라이브러리도 비교적 오래 전에 제작되었기 때문에 제공되는 기능이 한정된다. Canvas 기능을 활용하지 않고 독자적인 방법으로 이미지 에디터를 제작할 수도 있으나, 유지보수나 관리가 까다롭고, 브라우저 간 호환성 문제에 취약하다.

따라서 이번 연구에서는 Canvas API를 바탕으로 오픈소스 이미지 에디터 라이브러리를 제작하되, 기존 라이브러리에서 제공되지 않는 기능을 추가로 구현하여 차이점을 두려 한다.

* 1. **연구 목표**

본 프로젝트는 HTML5 Canvas API를 이용해 오픈소스 이미지 라이브러리를 제작하는 것을 목표로 한다. 이때 오픈소스 라이브러리에서 필수적으로 제공해야 하는 기능들을 포함하면서, 기존 라이브러리에서는 제공하지 않는 기능을 추가로 구현하여 특색있는 오픈소스 라이브러리를 제작하고자 한다.

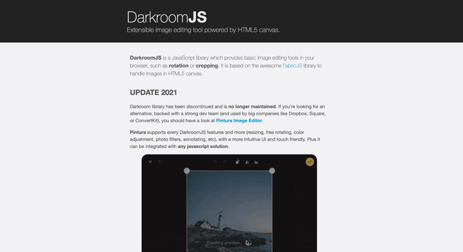
1. **관련 연구**
   1. **관련 오픈소스**
      1. **NHN Toast UI Image Editor, (**[**https://github.com/nhn/tui.image-editor**](https://github.com/nhn/tui.image-editor)**)**

텍스트이(가) 표시된 사진

자동 생성된 설명

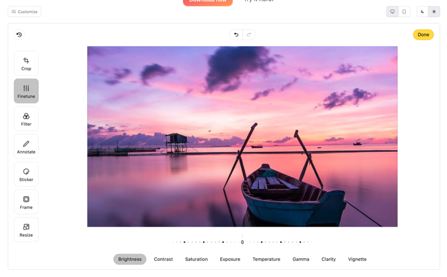
NHN Toast UI 팀에서 개발 및 유지보수하는 Fabric.js 기반 오픈소스 이미지 에디터다. 2021년 9월 기준 Github에서 웹 이미지 에디터를 검색했을 때 나오는 가장 유명한 오픈소스이기도 하다. 자르기, 회전, 실행취소 및 다시실행, 드로잉, 필터 등의 다양한 기능을 제공한다. Fabric.js 기반 UI 한계로 커서나 디자인이 다소 어색하다는 단점이 있다.

* + 1. **DarkroomJS, (**[**https://mattketmo.github.io/darkroomjs/**](https://mattketmo.github.io/darkroomjs/)**)**



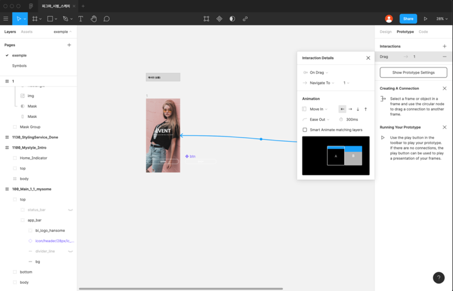
Fabric.js 기반 오픈소스 이미지 에디터로, Github 기준 범용 이미지 에디터 랭킹 2위이다. 자르기, 회전, 실행취소 및 다시실행, 저장 등의 기능을 모듈로 제공하기 때문에 기능을 선택적으로 포함할 수 있다. 하지만 2021년부터 더 이상 유지보수되지 않기 때문에, 아래에서 설명할 Pintura 에디터의 사용을 권장하고 있다.

* + 1. **Pintura, (**[**https://pqina.nl/pintura/**](https://pqina.nl/pintura/)**)**



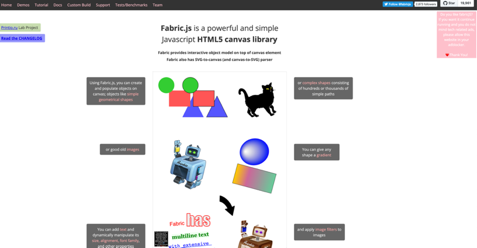
Pintura는 바닐라 JavaScript로 작성된 유료 이미지 에디터다. 유료 제품이기 때문에 오픈소스 프로젝트가 아니며, 이번 프로젝트에서 핵심적으로 다루는 Canvas API를 사용하지 않는다. 다만 기업에서 영리 목적으로 운영하고 있는 제품인만큼 제공하는 기능과 지원이 오픈소스 에디터에 비해 풍부하기 때문에 해당 서비스에서 구현된 기능을 참고하고자 한다.

* + 1. **Figma, (**[**https://figma.com/**](https://figma.com/)**)**



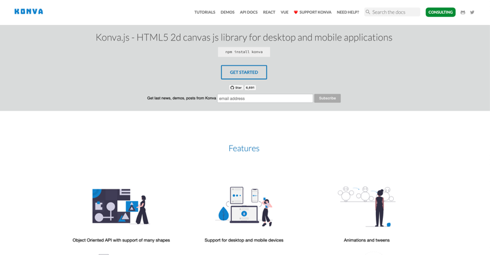
Figma는 웹 기반 UI 프로토타이핑 도구로, 별도의 그래픽 라이브러리 없이 자체적으로 개발한 Canvas 를 사용하고 있다. 프로토타이핑 도구이기 때문에, 이미지 편집 기능 외에도 도형과 레이어 배치 등 부가적인 기능을 제공한다. 이미 많은 사람들이 사용하는 서비스이면서 사용자 친화적인 UX를 갖고 있기 때문에, 라이브러리 UI를 구성하는 데 참고할 예정이다.

* + 1. **Fabric.js (**[**http://fabricjs.com/**](http://fabricjs.com/)**)**



Fabric.js는 SVG 파싱 기능을 갖춘 오픈소스 Canvas 라이브러리다. 이는 라이브러리 제작 목적 자체가 SVG를 객체 지향적이고 프로그래머티컬하게 제어하기 위한 목적으로 Canvas를 사용했기 때문이다. 다른 라이브러리에 비해 오래전에 나온 만큼 안정성이 높고 레퍼런스가 많지만, 구현할 수 있는 UI가 한정적이다.

* + 1. **Konva.js, (**[**https://konvajs.org/**](https://konvajs.org/)**)**

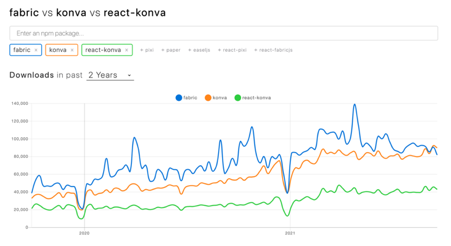


Konva.js는 데스크탑 및 모바일 애플리케이션을 위한 2D Canvas 라이브러리다. Fabric.js와 지원하는 기능은 비슷하지만, 라이브러리 자체에서 노드 간 레이어 관리를 알아서 해 주기 때문에 성능 상 이점이 있다. 또한 React나 Vue등 최신 프레임워크와의 호환성이 뛰어나다. 다만 Fabric.js 에 비해 라이브러리 자체에서 빌트인으로 지원하는 기능이 부족한 경우도 있기 때문에, 이러한 경우에는 더 많은 코드 작성을 필요로 한다는 단점이 있다.

* 1. **그래픽 라이브러리**

Canvas API는 강력한 기능을 제공하지만, 더 높은 추상화 모델을 제공하는 라이브러리를 쓰는 것이 더 나은 생산성을 기대할 수 있다. 이번 프로젝트에서는 Konva.js를 메인으로 선택했고, 그 근거는 다음과 같다.

* + 1. **다운로드 수**



NPM Trend 에서 최근 2년간의 트렌드를 살펴보았다. Fabric 이 근소하게 앞서 나가는 듯 하지만, Konva 역시 상승세가 지속되고 있다. React에서 사용할 수 있게 포팅된 react-konva 라이브러리 역시 Konva에 포함시키면 그 수가 더 많다.

* + 1. **공식 문서**

Konva의 공식 문서와 예제 코드가 Fabric 보다 더 잘 정리되어 있다. 또한 공식 문서에서 Vue나 React 등 다른 UI 라이브러리와 호환해서 사용할 수 있다고 설명해놓았기 때문에, 필요에 따라 UI 라이브러리를 적용하면서 개발 가능하다.

* + 1. **퍼포먼스**

Fabric과 Konva 모두 캐싱과 같은 퍼포먼스 향상을 위한 기법들을 기본적으로 제공한다. 다만 Konva에서는 레이어 관리 기법을 통해 노드 간의 계층 관계를 구현함으로서 독자적인 기능 향상을 기대할 수 있다고 한다.

* + 1. **UI**

이번 프로젝트의 UI는 Figma와 유사하게 만들고자 한다. 그래픽 라이브러리 자체에서 제공하는 커스터마이징 기능들이 Fabric에 비해 Konva가 더 자유로웠다.

1. **프로젝트 내용**

기존 오픈소스 프로젝트들과 비교하여 설계 고려사항과 시나리오를 정리했다.

* 1. **설계 고려사항**
     1. **추상화**

Canvas API는 기본적으로 제공하는 추상화 단계가 낮다. 이는 캔버스에 렌더되는 컴포넌트들을 객체지향적으로 관리하기가 어렵다는 것을 의미한다. 이미지 에디터는 단순히 하나의 이미지를 회전하고 잘라내는 것뿐만 아니라 도형이나 텍스트의 생성과 속성 조절 기능이 포함될 수 있으므로 컴포넌트 간에도 다양한 이벤트가 발생한다. 하지만 객체지향적인 구조 없이 모든 컴포넌트들을 관리하기는 쉽지 않다.

따라서 본 논문에서는 2D 그래픽에 대한 Canvas API를 JavaScript 기반으로 추상화하여 사용할 수 있게 해주는 Konva.js 라이브러리를 사용하고자 한다. Konva는 Canvas API의 2D Context를 확장하여 추상화된 객체를 제공할 뿐만 아니라 객체 편집 및 중첩 구조 관리, 브라우저 호환, 성능 최적화 기능 등을 추가로 제공하기 때문에 Canvas API를 고수준에서 활용하기에 적합하다고 판단했다.

* + 1. **상태 관리**

바닐라 JavaScript에서 Konva를 활용하는 것도 가능하지만, 여전히 데이터와 뷰(View)가 불일치한다는 문제점이 있다. 따라서 UI 라이브러리로는 React를 사용하고자 한다. React는 선언형 뷰를 지원하기 때문에 뷰가 어떻게 그려질 것인지를 명시하면 데이터에 따라 뷰가 알아서 갱신된다. 또한 반복되는 코드를 컴포넌트 단위로 설계하여 재사용할 수 있기 때문에 코드 관리 관점에서도 유용하다.

* + 1. **패키지 형태로 릴리스**

이번 연구는 라이브러리를 제작하는 프로젝트이다. 따라서 프로젝트의 결과물을 JavaScript 개발 환경에서 패키지 의존성으로 설치할 수 있어야 한다. 일반적으로 JavaScript 생태계에서는 NPM을 패키지 매니저로 사용하기 때문에, NPM에 이번 프로젝트를 릴리스 할 예정이다.

릴리스를 위해서는 여러 모듈로 관리되고 있는 JavaScript들을 단일 파일로 묶어주는 번들링(bundling) 과정이 필요하다. 이 때 JavaScript 표준 모듈 시스템으로 번들링을 가능하게 해주는 Rollup.js를 사용할 예정이다. 이를 ES6 모듈이라 부르는데, 이를 이용하면 정적 코드 분석을 통해 사용되지 않는 코드를 제거하는 트리 쉐이킹(Tree shaking)이 가능해진다. 따라서 라이브러리의 경량화를 함께 챙길 수 있다는 장점이 있다.

* 1. **시나리오**

초기 구상 시나리오는 다음과 같다.

* + 1. **라이브러리 형태로 패키지 릴리스**

사용자(개발자)는 패키지 매니저(또는 CDN)를 이용해 라이브러리를 설치할 수 있다. 그 후 커스터마이징 가능한 옵션을 입력받고, DOM에 Canvas 엘리먼트를 삽입한 후 초기화 과정을 거친다. 초기화 이후 이미지 라이브러리가 DOM에 렌더된다.

* + 1. **Load - 이미지 불러오기**

라이브러리 외부에서부터 이미지를 불러온다. 이때 파일 시스템에서 직접 불러온 이미지 뿐만 아니라 Blob, Base64 인코딩 등 다양한 타입을 호환 가능하게 한다.

* + 1. **Crop & Scaling & Zoom & Flip & Rotation – 크기와 회전 조절**

불러온 이미지를 배경 레이어로 설정하고, 이에 대한 편집 옵션을 제공한다. 이미지 크기를 조절하거나 각도 조절, 확대, 자르기 기능 등을 제공한다.

* + 1. **Drawing – 드로잉**

간단한 드로잉 툴을 제공하여 마우스 입력이나 선, 텍스트, 자유 도형 등을 입력할 수 있게 한다.

* + 1. **Drag & Drop & Resizing - 드래그 & 드롭, 리사이징**

드로잉 툴을 이용해 그린 객체를 이동시키거나 크기를 조절할 수 있는 도구를 제공한다. 키보드 입력을 함께 받을 수 있게 구현하며, 필요에 따라 그리드나 스냅 기능을 제공할 수 있다.

* + 1. **Filter – 필터**

Grayscale, Blur, Invert, Brightness 등의 필터 효과를 조절할 수 있게 한다.

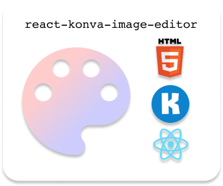
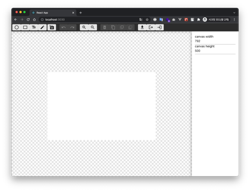
* + 1. **Undo & Redo – 실행 취소 및 다시 실행**

작업 진행 과정을 담은 히스토리를 별도의 자료구조로 정리하여, 실행 취소 기능과 다시 실행 기능을 구현한다. 이 때 키보드 입력을 함께 받을 수 있게 구현한다.

* + 1. **Save & Restore – 저장 및 복구**

현재까지 편집한 이미지를 파일로 저장한다. 이 때 현재까지의 편집 상태를 별도 파일로 저장하여, 필요에 따라 편집 내역을 복구하여 작업을 계속 진행할 수 있게 한다.

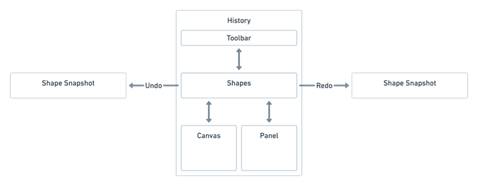
1. **프로젝트 구현 결과**

****

본 프로젝트의 전체 소스코드, 그리고 각 기능에 대한 영상은 Github(<https://github.com/wormwlrm/react-konva-image-editor>)에서 확인 가능하다. 데모 페이지(<https://wormwlrm.github.io/react-konva-image-editor-demo/>)에 접속하면 동작하는 라이브러리를 직접 확인할 수 있다.

* + 1. **아키텍처**

본 프로젝트의 아키텍처는 다음과 같은 구조로 구현되었다.



* **Shapes Layer**: 캔버스에 그려지는 모든 도형, 사용자 상태 등을 관리한다. 또한 캔버스와 패널, 툴바 사이의 상호작용을 중재한다.
  + **Snapshot**: 특정한 사용자 액션(도형 생성, 이동, 스타일 수정 등)이 발생하여 히스토리를 저장해야 할 경우, 현재 도형을 스냅샷으로 저장하고 히스토리에 추가한다.
* **History Layer**: 저장된 스냅샷 배열을 기반으로 실행 취소 및 다시 실행 기능을 지원한다.
* **Canvas Layer**: Shapes Layer에 저장된 도형들을 실제 2D 그래픽으로 표현하고, 사용자로부터 입력받은 상호작용을 Shapes Layer에 넘겨준다.
* **Toolbar Layer**: 사용자 상호작용을 명시적으로 입력받는 영역이다.
* **Panel Layer**: 현재 선택된 도형의 속성을 표시하는 영역이다.
  + 1. **라이브러리 형태로 패키지 릴리스**

데모 프로젝트 확인을 위해 현재까지의 개발 사항을 node.js 패키지 매니저인 NPM (<https://www.npmjs.com/package/react-konva-image-editor>) 에 배포한 상태다. 따라서 터미널에서 다음과 같은 명령어를 입력하면 설치가 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

호스트 측 자바스크립트 코드 내에서는 다음과 같이 호출하여 사용할 수 있다.

텍스트이(가) 표시된 사진

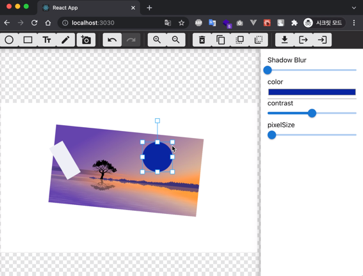
자동 생성된 설명

라이브러리를 설치하여 사용하는 호스트에서 에디터 레이아웃에 대한 옵션값을 다음과 같이 설정할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

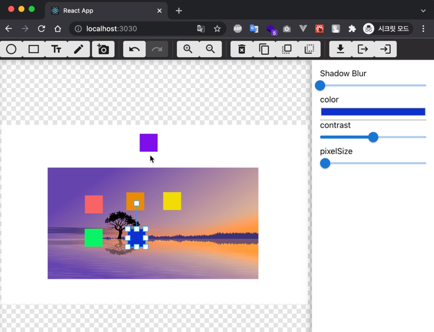
* + 1. **이미지/도형 컴포넌트 생성 및 편집**

****

캔버스에 이미지와 도형 인스턴스를 생성할 수 있다. 각 도형은 드래그 가능하며(Draggable), 자체적으로 회전, 리사이징이 가능(Transformable)하다.

이미지 및 도형 인스턴스는 Shape Layer에서 관리되며, 각 컴포넌트에 대한 속성 정의는 components 폴더에 정의되어 있다. 이미지를 불러올 때에는 이미지 파일을 Base64로 인코딩하여 사용한다.

우측의 패널을 통해 컬러, 필터 등을 적용할 수 있다.

* + 1. **실행 취소 / 다시 실행**

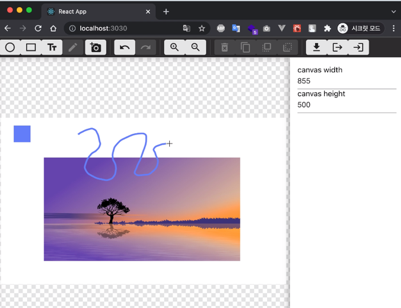
실행 취소와 다시 실행 기능을 위해 메멘토 패턴을 적용했다. 메멘토 패턴은 히스토리를 저장하는 Caretaker 역할, 그리고 히스토리를 가리키는 인덱스를 띄워주는 Originator 역할으로 구성된다.

여기서 Caretaker 역할을 하는 것이 History Layer다. 도형에 대한 사용자 상호작용(드래그, 회전, 리사이징 등)이 발생할 때마다 History Layer는 현재 Shapes Layer의 인스턴스를 스냅샷을 생성하고 배열로 관리한다.

History Layer에는 특정 스냅샷을 가리키는 인덱스가 정의되어 있다. 따라서 실행 취소 이벤트가 발생할 때에는 인덱스를 1만큼 감소시키며, 다시 실행 이벤트가 발생할 때에는 인덱스를 1만큼 증가시킨다.

Shapes Layer는 현재 History Layer에서 가리키는 스냅샷을 가져온다. Canvas Layer는 Shapes Layer의 인스턴스 배열에 종속적이기 때문에, 저장된 인스턴스를 캔버스에 표시한다.

* + 1. **드로잉**

****

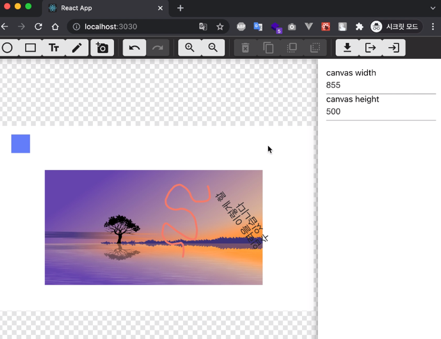
드로잉은 기본적으로 드래그 앤 드롭과 동일한 입력을 받지만 다르게 동작해야 한다. 따라서 툴바에서 드로잉 모드를 선택하게 되면, 캔버스에 있는 도형들은 더 이상 드래그 가능하지 않아야 한다. 이는 도형 위에서 드로잉을 시작했을 때, 도형이 드래그 되는 현상을 막기 위해서이다.

마우스 커서를 누르면 onmousedown 이벤트가 발생하게 되는데, 이때 x, y 좌표를 구할 수 있다. 따라서 이 좌표를 이산적으로 덧붙이면서 라인 컴포넌트의 경로를 표시할 수 있다.

라인 컴포넌트 역시 도형 인스턴스로 취급된다. 따라서 드래그와 변환이 가능하다.

하지만 마우스를 떼기 전까지 라인 컴포넌트는 아직 생성된 상태가 아니지만, 현재까지의 경로를 캔버스에 표시해주어야 할 필요가 있다. 따라서 마우스를 떼기 전까지 좌표 배열을 임시로 저장하고, 마우스를 뗄 때 저장된 좌표 배열을 이용해 Shape Layer에 새 라인 컴포넌트를 생성한다.

* + 1. **수정 가능한 텍스트 컴포넌트 제공**

****

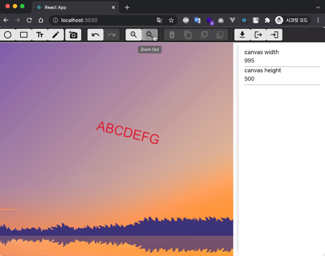
HTML5 스펙에 따르면, 캔버스에서 텍스트를 렌더할 수는 있지만 캔버스 내에서 입력값을 직접 수정할 수 있는 인풋(input) 형태로는 제공되지 않는다. 따라서 해당 기능이 필요한 경우가 있다면 Canvas API를 적절히 우회하여야 한다.

수정 가능한 텍스트 컴포넌트를 만들기 위해서는 Canvas API와 외부 DOM 엘리먼트 간 스타일 및 데이터를 동기화했다. 우선 캔버스에 텍스트를 나타낼 수 있는 텍스트 컴포넌트를 정의한다. 이 역시 도형 인스턴스로 취급되므로 드래그와 변환이 가능해야 한다.

만약 텍스트 컴포넌트에 더블 클릭이나 enter키 등의 이벤트가 발생하면 해당 위치에 <textarea> 엘리먼트를 생성한다. 이 때 캔버스의 줌, 텍스트 컴포넌트의 회전, 줄바꿈을 고려하여 어색함이 없도록 스타일을 적절히 조정해준다.

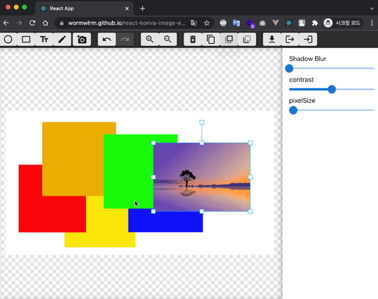
enter키 입력 시 값을 저장하고, shift + enter키를 누를 땐 줄바꿈이 일어나야 한다. 포커스를 잃었을 때에는 자동으로 <textarea> 가 삭제되도록 한다.

* + 1. **캔버스 확대 및 축소**

****

줌 배율을 지정하는 변수를 선언하고, 이를 캔버스 레이어에 적용하고 적절히 리사이징한다. 이 때 기존의 도형 인스턴스 배치에 영향이 가지 않게 한다. 한편 캔버스 영역이 보여지는 화면보다 클 경우 스크롤 가능하게 만든다.

* + 1. **Z-index 조정**

****

도형 간 z-index 조정은 Shape Layer의 도형 인스턴스 배열의 순서를 바꾸는 방식으로 구현 가능하다.

* + 1. **라이브러리 형태로 패키지 릴리스**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

이미지 저장 기능은 현재 렌더된 화면을 이미지 파일로 저장할 수 있는 Canvas API의 toDataURL() 메서드를 사용하여 제공한다.

또한 도형을 인스턴스화했기 때문에, 현재까지의 편집 내역이 담긴 Shape Layer의 도형 인스턴스 배열을 직렬화하여 JSON 파일로 저장할 수 있다. 아래 사진은 실제 직렬화된 JSON 파일의 스크린샷이다.

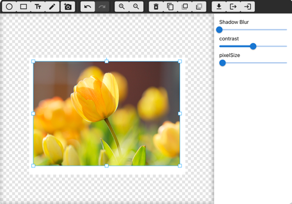
**텍스트이(가) 표시된 사진

자동 생성된 설명**

이후 JSON 파일을 불러올 때, Shape Layer 배열에 적절히 매칭시키는 기능까지 구현된 상태다.

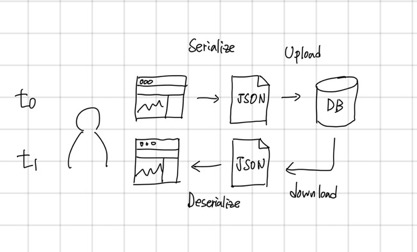
1. **타 라이브러리와의 비교**

**텍스트, 모니터, 화면, 실내이(가) 표시된 사진

자동 생성된 설명** ****

본 프로젝트는 현재까지의 캔버스에 그려진 도형 상태를 JSON 형태로 직렬화할 수 있고, 이를 파일로 다시 불러올 수 있다는 점에서 독창성이 있다. 이를 설명하기 위해, 관련 오픈소스에서 소개한 NHN Toast UI Image Editor와 기능을 비교해본다.

NHN의 이미지 에디터는 결과물로 이미지 파일만 export 할 수 있다. 그렇기 때문에 현재 캔버스가 그려져 있는 웹 브라우저를 닫게 되면 메모리에 올라가 있는 데이터들이 모두 사라지기 때문에, 해당 결과물을 다시 복구하거나 수정할 수 없다.

****

한편 본 프로젝트에서는 데이터 직렬화 기능을 지원한다. 만약 사용자가 작업한 내역을 JSON 파일로 저장할 수 있다면, 차후 해당 JSON 파일을 역직렬화하면서 작업 내역을 복구할 수 있다. 만약 JSON 파일을 다른 사용자에게 공유할 수 있다면, 동일한 작업 내역을 여러 사용자에게 제공할 수도 있다. 이 기능을 호스트 쪽 데이터베이스 등에 저장해둘 수 있다면 이 장점이 극대화된다. 사용자는 원격 서버에 본인의 작업 내역을 저장해두고 언제든 불러올 수 있기 때문이다.

1. **결론 및 기대 효과**

본 프로젝트에서는 HTML5의 Canvas API를 활용하여 이미지 에디터 라이브러리를 제작하고, 각 기능들의 구현 방법과 한계점에 대해 연구하고 소스 코드를 공개했다 (<https://github.com/wormwlrm/reactkonva-image-editor>).

또한 애플리케이션 레벨에서 적합한 고수준의 추상화를 제공하는 라이브러리를 활용하고, 상황에 맞는 디자인 패턴을 사용함으로서 이미지 에디터의 기능을 구현할 수 있었다.

마지막으로 Canvas API의 범위를 넘어서는 요구사항에 대해서는 기술적으로 우회하면서도 사용자 경험을 해치지 않았다.

다만 시간적 여유 상 사용자 편의를 위한 부가적인 기능들(키보드 단축키 지원 등)에 대해서는 구현이 완료되지 않은 부분/ UI 라이브러리를 적용하면서 번들 사이즈가 커진 부분이 있지만, 향후 기능 추가 및 유지보수를 통해 개선해 나갈 예정이다.

1. **참고문헌**
2. Vitaly Friedman, Smashing Book 6: New. Frontiers In Web Design, 2018
3. 장석우, 허문행, HTML5 캔버스를 이용한 플랫폼 독립적인 게임의 구현, 한국정보통신학회논문지, 제18권, 제12호, pp. 3042-3048, 2014
4. 박영수, HTML5에서 직선의 기울기를 이용한 2D to 3D 입체 이미지 변환, 디지털융복합연구, 제12권, 제12호, pp. 521-528, 2014
5. 박미라, 박기호, 안재성, HTML5 Canvas를 활용한
6. 시각적 공간분석 환경의 설계와 구현, 한국지리정보학회지, 제14권, 제4호, pp.44-53, 2011