리액트 프로젝트

2501110200 김재영

CONTENTS

1 개요

2 주제선정이유

3 구현페이지(1학기)

4 구현페이지 및 설명 - 중고판매

5 구현페이지 및 설명 - 물건 업로드

6 구현페이지 및 설명 - 회원가입

7 인사

03
목표 및 기대성과

본 프로젝트는 1학기 웹프로그래밍기초 수업에서 HTML, CSS, JavaScript로 제작했던 'Pick & Go' 가구 처리 서비스 웹사이트를 React 라이브러리를 활용하여 리액트 프로젝트로 재구현한 것입니다. Pick & Go는 사용자가 불필요한 가구를 사진 한 장으로 간편하게 처리할 수 있도록 돕는 서비스로, 가구를 '정리하기(폐기)', '맡겨두기(보관)', '판매하기(중고거래)' 세 가지 방식으로 처리할 수 있는 통합 플랫폼입니다. 1학기 프로젝트에서는 정적인 HTML 페이지와 CSS 애니메이션을 중심으로 시각적 효과에 집중했다면, 이번 기말 과제에서는 React의 컴포넌트 기반 아키텍처와 Hooks를 활용하여 실제로 동작하는 인터랙티브한 웹 애플리케이션으로 발전시켰습니다. 특히 useState를 통한 상태 관리와 useMemo를 통한 성능 최적화 등 React의 핵심 개념들을 실제 프로젝트에 적용하면서 이론으로만 배웠던 내용들을 직접 체험하고 이해할 수 있었습니다. 이를 통해 사용자 인터랙션에 즉각적으로 반응하는 동적인 웹 페이지를 구현할 수 있었으며, 컴포넌트의 재사용성과 코드의 유지보수성을 크게 향상시킬 수 있었습니다.

02
주제선정이유

(1) 실생활 문제 해결에 대한 관심 대학생이 되어 자취를 시작하면서 가구 처리 문제를 직접 겪어봤습니다. 이사할 때나 방 정리를 할 때 쓰지 않는 가구를 어떻게 처리해야 할지 막막했던 경험이 있었고, 중고거래 앱에 올리자니 사진 찍고 설명 쓰는 게 귀찮았고, 그냥 버리자니 아까웠습니다. 이런 경험을 바탕으로 '사진 한 장으로 가구를 쉽게 처리할 수 있다면 얼마나 편할까?'라는 생각에서 이 아이디어가 시작되었습니다. 실제로 우리 주변에는 비슷한 고민을 하는 사람들이 많다고 생각했고, 이런 실생활의 불편함을 해결하는 서비스를 직접 만들어보고 싶다는 마음이 컸습니다. 단순히 과제를 위한 프로젝트가 아니라, 실제로 사용할 수 있는 서비스를 만들고 싶었습니다.

02
주제선정이유

(2)1학기 때 HTML과 CSS로 웹페이지를 만들면서 '이걸 어떻게 실제 서비스처럼 동작하게 만들 수 있을까?'라는 의문이 계속 들었습니다. 버튼을 눌러도 페이지가 새로고침 되거나, 데이터를 입력해도 저장되지 않는 정적인 페이지의 한계를 느꼈습니다. 그러던 중 React를 배우면서 useState로 상태를 관리하고, 클릭 한 번으로 화면이 바뀌는 걸 보고 정말 신기했습니다. '이게 진짜 웹 개발이구나'라는 생각이 들었고, 내가 만든 서비스에 사용자가 실제로 데이터를 입력하고, 필터링하고, 검색할 수 있다는 게 너무 흥미로웠습니다. 특히 16,000개가 넘는 상품 데이터를 필터링하고 페이지네이션으로 보여주는 기능을 구현하면서 '이 정도면 실제 쇼핑몰처럼 동작하는 거 아닌가?'라는 뿌듯함을 느꼈습니다. React의 컴포넌트 재사용성과 Hooks의 강력함을 직접 체험하고 싶었고, 이번 프로젝트가 그걸 제대로 배울 수 있는 기회라고 생각했습니다.

02
주제선정이유

또한 처음 React를 배울 때는 'useState 하나만으로 뭘 할 수 있을까?'라는 생각이 들었습니다. 하지만 이번 프로젝트를 진행하면서 '정리하기' 페이지에서는 드래그 앤 드롭과 이미지 업로드, 슬라이더 구현을, '판매하기' 페이지에서는 필터링과 검색, 페이지네이션을, '맡겨두기' 페이지에서는 폼 데이터 관리를 각각 구현하면서 React로 정말 다양한 기능을 만들 수 있다는 걸 깨달았습니다. 특히 각 페이지마다 다른 방식의 상태 관리가 필요했고, 이를 해결하는 과정에서 React의 작동 원리를 더 깊이 이해할 수 있었습니다. 예를 들어 '판매하기' 페이지에서 위치, 카테고리, 가격, 검색어라는 4가지 필터를 동시에 적용해야 했는데, 처음에는 '이걸 어떻게 한 번에 관리하지?'라는 고민이 있었습니다. 하지만 useMemo를 사용해서 필터 조건이 바뀔 때마다 자동으로 재계산되도록 구현하면서 React의 최적화 기법을 실전에서 배울 수 있었습니다.

02
주제선정이유

마지막으로, 1학기 프로젝트는 '보기 좋은' 페이지를 만드는 데 집중했다면, 이번 프로젝트는 '사용하기 편한' 페이지를 만들고 싶었습니다. 예를 들어 '정리하기' 페이지에서 파일 업로드를 input[type="file"]로 하면 너무 단조롭고 불편할 것 같아서, 드래그 앤 드롭 기능을 추가했습니다. 사용자가 이미지를 끌어다 놓기만 하면 바로 업로드되는 게 훨씬 직관적이고 편하다고 생각했습니다. 또 '판매하기' 페이지에서 251 페이지가 넘는 상품을 어떻게 보여줄지 고민이 많았는데, 아마존이나 쿠팡 같은 실제 쇼핑몰의 페이지네이션을 참고해서 '1 ... 5 6 7 ... 251' 형태로 구현했습니다. 처음에는 1004개의 버튼을 다 만들까 생각했는데, 그러면 페이지가 너무 길어지고 사용자가 헷갈릴 것 같아서 현재 페이지 기준 ±2 페이지만 보여주는 방식으로 바꿨습니다. 이런 작은 고민들이 모여서 실제로 사용 가능한 서비스를 만들 수 있다는 걸 느꼈습니다.

02
구현페이지(1학기)

이 1학기 HTML 페이지는 Pick & Go라는 원클릭 가구 처리 서비스를 소개하는 랜딩 페이지입니다.

상단에는 고정 헤더와 로고/태그라인이 배치되어 브랜드 정보를 항상 보이도록 구성했습니다.

메인 영역에서는 레일 위로 침대·옷장·TV가 이동하고, 트럭과 캐릭터가 반응하는 CSS 애니메이션 시나리오를 통해 “가구 수거/보관/판매” 흐름을 시각적으로 표현했습니다.

오른쪽 컨테이너 아이콘은 진동 애니메이션을 적용해 작업/처리 중인 느낌을 강조했습니다.

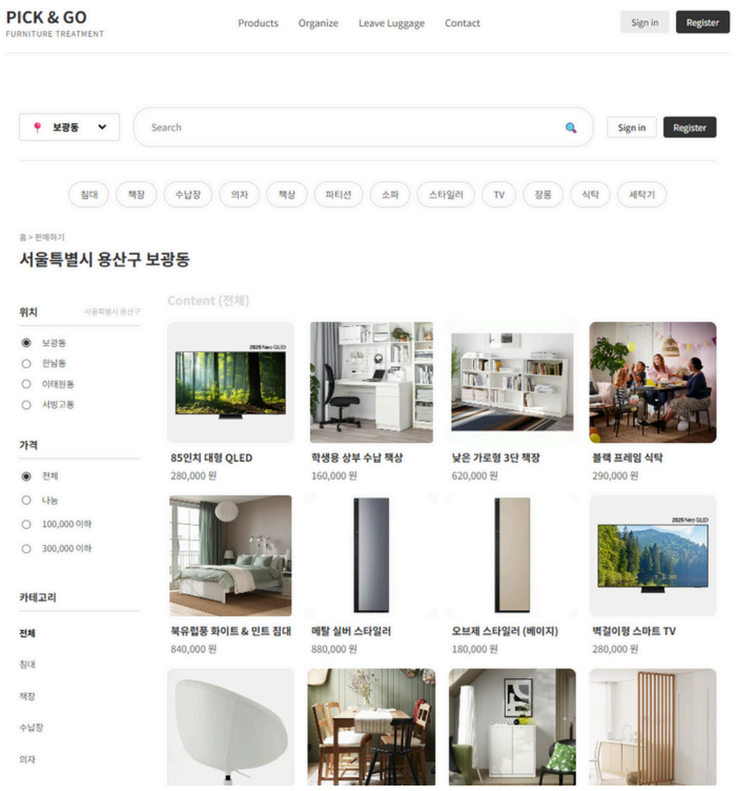
하단에는 정리하기/맡겨두기/판매하기 3개의 서비스 버튼을 카드 형태로 배치하고, 호버 시 살짝 떠오르는 효과로 클릭 유도를 강화했습니다.

전체적으로 배경 그라데이션과 애니메이션을 활용해 정적인 소개 페이지보다 서비스 콘셉트를 직관적으로 전달하는 데 초점을 둔 구성입니다.

이 첫 홈페이지는 css가와 동적 이미지들이 잘나타나있어서 예전에 피그마로 작업했던 다른 웹들을 리액트로 구현하고자 하였습니다.

02

구현페이지 및 설명 - 중고판매




```
import React, { useState, useMemo } from 'react';

const OrganizePage = () => {
  const [dragActive, setDragActive] = useState(false);
  const [uploadedImg, setUploadedImg] = useState(null);
  const [showModal, setShowModal] = useState(false);

  const [saleIndex, setSaleIndex] = useState(0);
  const [transIndex, setTransIndex] = useState(0);

  const baseProducts = [ /* 120개 더미 데이터 */ ];

  const { saleItems, transactionItems } = useMemo(() => {
    const saleSelected = baseProducts.slice(2, 10).map((item) => ({
      id: item.id,
      type: item.category,
      name: item.name,
      desc: `${item.category} 카테고리의 인기 상품입니다.`,
      img: item.img
    }));

    const transSelected = baseProducts.slice(3, 12).map((item) => ({
      id: item.id,
      title: item.name,
      desc: "편안하고 실용적인 디자인의 중고 가구입니다.",
      img: item.img
    }));

    return { saleItems: saleSelected, transactionItems: transSelected };
  }, []);
jsx
코드 복사
  const SALE_VIEW_COUNT = 2;
  const TRANS_VIEW_COUNT = 3;

  const handleSaleUp = () => {
    setSaleIndex(prev => Math.max(prev - SALE_VIEW_COUNT, 0));
  };
  const handleSaleDown = () => {
    if (saleIndex + SALE_VIEW_COUNT < saleItems.length) {
      setSaleIndex(prev => prev + SALE_VIEW_COUNT);
    }
  };

  const handleTransLeft = () => {
    setTransIndex(prev => Math.max(prev - TRANS_VIEW_COUNT, 0));
  };
  const handleTransRight = () => {
    if (transIndex + TRANS_VIEW_COUNT < transactionItems.length) {
      setTransIndex(prev => prev + TRANS_VIEW_COUNT);
    }
  };
```

본 페이지는 1학기 웹프로그래밍기초 수업에서 제작했던 가구 정리/거래 콘셉트를 React 함수형 컴포넌트로 재구현한 대표 페이지입니다.

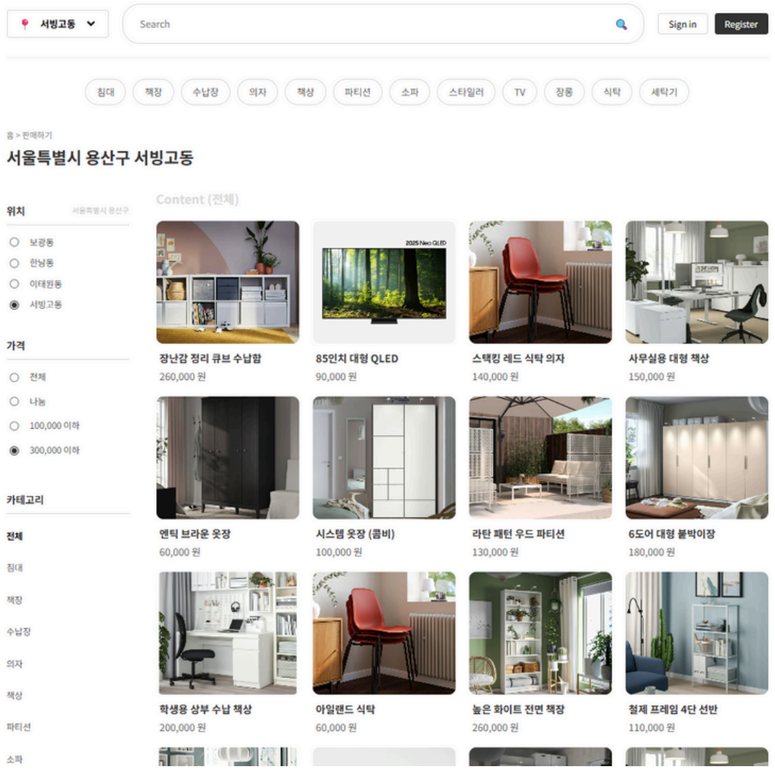
사용자가 한 화면에서 이미지 업로드 흐름과 Sale(세로 슬라이드), Transaction(가로 슬라이드) 콘텐츠를 동시에 확인할 수 있도록 구성했습니다.

제가 구현하고자 한 핵심은가구 처리를 빠르게 도와주는 서비스형 UI였기 때문에, 단순한 목록 나열 대신 슬라이드 방식으로 한 번에 많은 상품을 효율적으로 탐색할 수 있도록 설계했습니다.

또한 더미 데이터가 많아질수록 렌더링 과정에서 불필요한 계산이 반복될 수 있다고 판단하여, useMemo를 사용해 Sale/Transaction에 사용할 고정 서브셋을 한 번만 생성하도록 구성했습니다.

이로 인해 화면이 리렌더링되더라도 동일한 기준의 상품 목록이 유지되며, 화면 안정성과 성능이 함께 개선되었다고 정리할 수 있습니다.

02
구현페이지 및 설명 - 중고판매




```
const [maxPrice, setMaxPrice]
= useState(null); // 예: 300000

const filteredSaleItems =
useMemo(() => {
  if (!maxPrice) return
saleItems;
  return saleItems.filter(item =>
item.price <= maxPrice);
}, [saleItems, maxPrice]);
jsx
코드 복사
// UI 예시(버튼/필터 영역)
<button onClick={() =>
setMaxPrice(300000)}>
  30만원 이하
</button>
<button onClick={() =>
setMaxPrice(null)}>
  전체
</button>
```

기본 페이지 구현 이후, 사용자 탐색 효율을 높이기 위해 가격 필터 기능을 추가했다고 설명드릴 수 있는 화면입니다.

초기에는 Sale/Transaction 목록이 모두 노출되는 방식이라 사용자가 원하는 가구를 찾기까지 스크롤과 이동이 많아지는 불편함이 있었습니다.

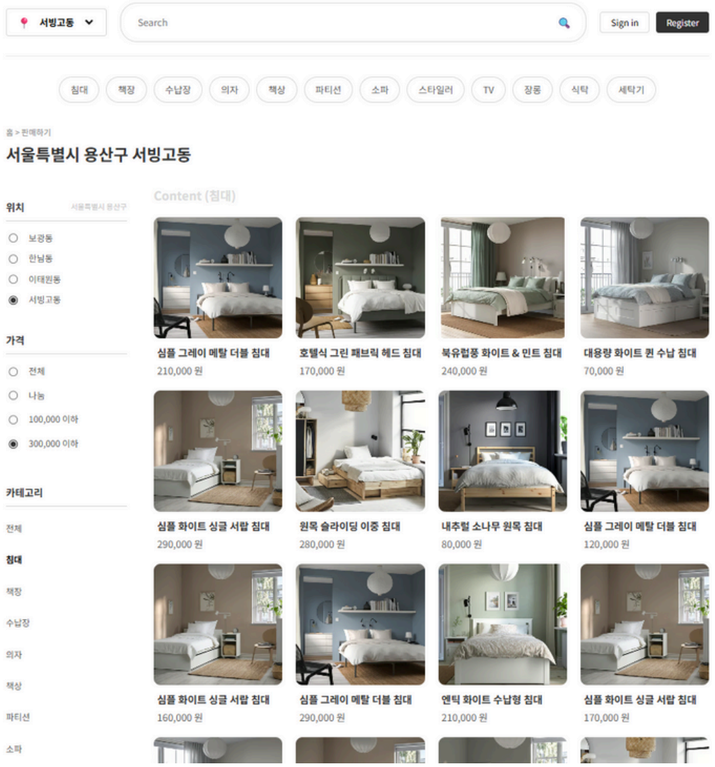
이를 개선하기 위해 30만원 이하와 같은 조건을 선택하면 해당 조건에 맞는 상품만 화면에 즉시 반영되도록 상태 기반 필터링을 적용했습니다.

가격 조건을 상태(maxPrice)로 관리하고, 리스트는 useMemo로 필터링해 렌더링 부담을 줄이면서도 화면 반응성을 확보했습니다.

이러한 방식으로 단순 나열형 UI의 단점을 보완하고, 사용자가 빠르게 후보를 좁힐 수 있도록 사용자 편의성이 향상되었다고 정리했습니다.

02

구현페이지 및 설명 - 중고판매




```
const [maxPrice, setMaxPrice] = useState(null); // 예: 300000

const filteredSaleItems = useMemo(() => {
  if (!maxPrice) return saleItems;
  return saleItems.filter(item => item.price <= maxPrice);
}, [saleItems, maxPrice]);
```

jsx
코드 복사

```
const [maxPrice, setMaxPrice] = useState(null);
// 300000
const [category, setCategory] = useState("전체");
// "침대"

const filteredItems = useMemo(() => {
  return baseProducts.filter(item => {
    const okCategory = category === "전체" || item.category === category;
    const okPrice = !maxPrice || item.price <= maxPrice;
    return okCategory && okPrice;
  });
}, [baseProducts, maxPrice, category]);
```

jsx
코드 복사

```
// UI 예시(카테고리+가격 복합 필터)
<button onClick={() => setCategory("침대")}>침대</button>
<button onClick={() => setCategory("전체")}>전체</button>

<button onClick={() => setMaxPrice(300000)}>30만원 이하</button><button onClick={() => setMaxPrice(300000)}>
  30만원 이하
</button>
<button onClick={() => setMaxPrice(null)}>
  전체
</button>
```

본 화면은 가격 필터만 적용했을 때에도 선택지가 여전히 많을 수 있다는 점을 고려하여, 카테고리 필터를 추가로 결합한 복합 필터링 결과를 보여주는 캡처입니다.

제가 구현하고자 했던 방향은 사용자가 실제로 서비스에서 원하는 조건을 한 번에 설정하고 빠르게 결과를 확인하는 흐름이었기 때문에, 30만원 이하 + 침대처럼 조건을 동시에 적용할 수 있도록 개선했습니다.

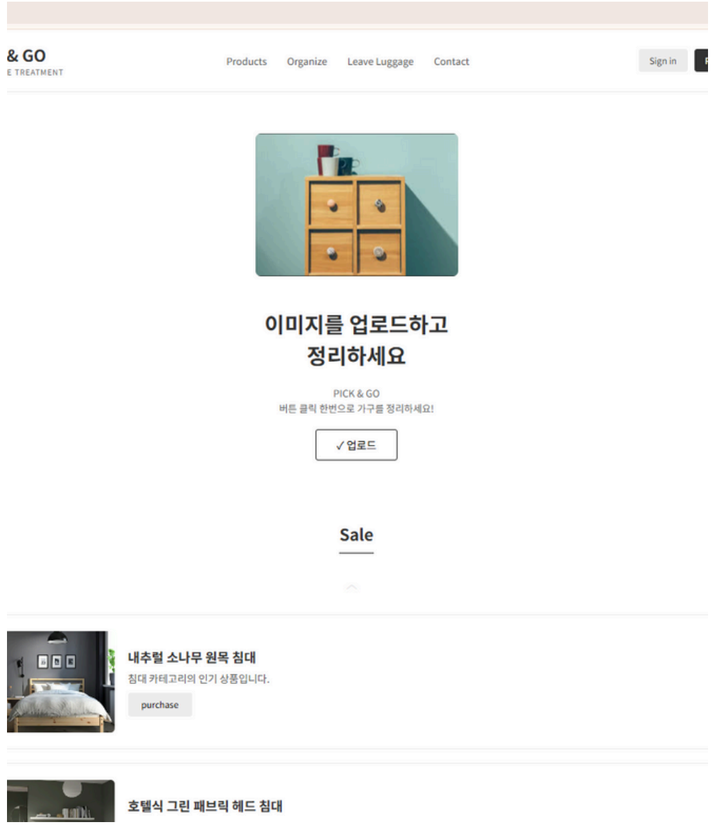
개발 과정에서 단일 조건만 적용하면 사용자가 다시 다른 조건을 조합하기 위해 여러 번 화면을 탐색해야 하는 불편함이 있다고 느꼈습니다.

따라서 가격(maxPrice)과 카테고리(category)를 각각 상태로 분리하고, 두 조건을 AND 방식으로 결합해 조건에 해당하는 상품만 표시되도록 구현했습니다.

이로 인해 사용자의 탐색 과정이 단축되고, 실제 서비스 관점에서도 더 직관적인 정리/거래 경험을 제공할 수 있도록 기능이 향상되었다고 정리할 수 있습니다.

02

구현페이지 및 설명 - 물건 업로드




```
const [dragActive, setDragActive] = useState(false);
const [uploadedImg, setUploadedImg] = useState(null);
const [showModal, setShowModal] = useState(false);

const handleDrag = (e) => {
 e.preventDefault();
 e.stopPropagation();
 if (uploadedImg) return;

 if (e.type === 'dragenter' || e.type === 'dragover') {
  setDragActive(true);
 } else {
  setDragActive(false);
 }
};

const handleDrop = (e) => {
 e.preventDefault();
 e.stopPropagation();
 if (uploadedImg) return;

 setDragActive(false);

 if (
  e.dataTransfer.files &&
  e.dataTransfer.files[0] &&
  e.dataTransfer.files[0].type.startsWith('image/')
 ) {
  setUploadedImg(URL.createObjectURL(e.dataTransfer.files[0]));
 } else {
  alert("이미지 파일만 업로드 가능합니다.");
 }
};

const handleUploadClick = () => {
 if (!uploadedImg) {
  alert("먼저 이미지를 드래그하여 업로드해주세요!");
  return;
 }
 setShowModal(true);
};
jsx
코드 복사
<section
 className={`hero-section ${dragActive ? 'active' : ''}`}
 onDragEnter={handleDrag}
 onDragLeave={handleDrag}
 onDragOver={handleDrag}
 onDrop={handleDrop}
>
 {dragActive && !uploadedImg && (
  <div className="drop-message">파일을 여기에 놓으세요</div>
 )}

 {uploadedImg && (
  <img src={uploadedImg} alt="Uploaded" className="uploaded-preview" />
 )}

 {!dragActive && !uploadedImg && (
  <>
   <img src="/imgs/zimage.png" alt="Main" className="hero-img" />
   <div className="hero-title">
    <h2>이미지를 업로드하고<br />정리하세요</h2>
   </div>
   <p className="hero-desc">PICK & GO<br />버튼 클릭 한번으로 가구를 정리하세요!</p>
  </>
 )}

 <button className="btn-outline" onClick={handleUploadClick}>
  ✓업로드
 </button>
</section>
```

본 화면은 메인 기능인 가구 이미지 업로드 흐름을 사용자가 가장 직관적으로 이해할 수 있도록 구성한 영역입니다.

제가 구현하고자 한 내용은 “사용자가 사진만 올리면 곧바로 정리/거래 과정으로 이어지는 원클릭 흐름”이었기 때문에, 기존의 파일 선택 방식보다 접근성이 높은 드래그 앤 드롭 방식을 적용했습니다.

개발 과정에서 업로드 영역이 눈에 잘 띄지 않으면 사용자가 어디에 파일을 놓아야 하는지 헷갈릴 수 있다는 불편함이 있었습니다.

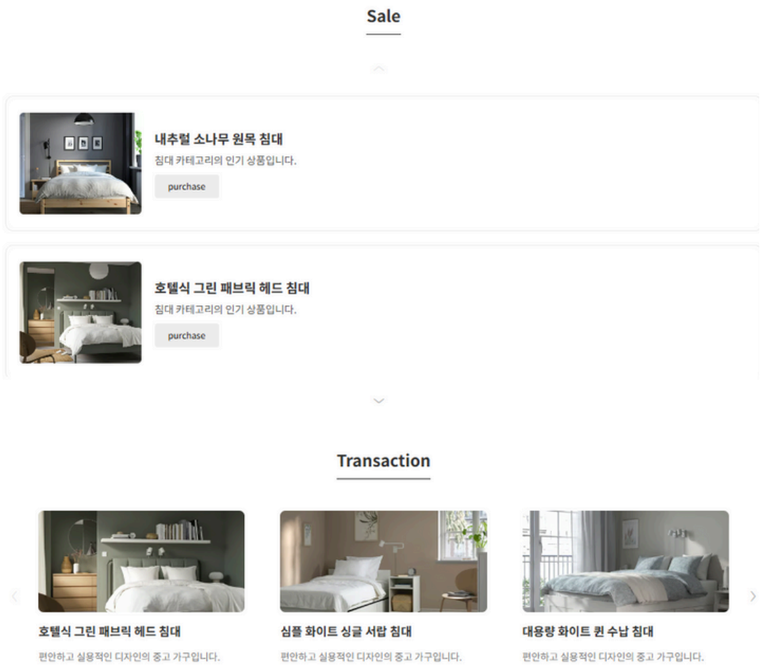
이를 개선하기 위해 dragActive 상태를 두고, 드래그가 감지되면 Hero 영역이 활성화되도록 스타일을 변경하여 드롭 가능한 영역을 명확하게 안내했습니다.

또한 이미지가 이미 업로드된 상태에서는 중복 업로드로 인한 혼란을 줄이기 위해 추가 드래그 동작을 제한하도록 처리했습니다.

이로 인해 사용자가 업로드 과정을 더 빠하고 안정적으로 진행할 수 있도록 UX가 개선되었다고 정리했습니다.

02

구현페이지 및 설명 - 물건 업로드




```
const [saleIndex, setSaleIndex] = useState(0);
const SALE_VIEW_COUNT = 2;

const handleSaleUp = () => {
 setSaleIndex(prev => Math.max(prev - SALE_VIEW_COUNT, 0));
};

const handleSaleDown = () => {
 if (saleIndex + SALE_VIEW_COUNT < saleItems.length) {
  setSaleIndex(prev => prev + SALE_VIEW_COUNT);
 }
};
jsx
코드 복사
<div className="sale-slider-wrapper">
 <button
  className="arrow-btn arrow-btn-vertical"
  onClick={handleSaleUp}
  disabled={saleIndex === 0}
 >
  <span style={{ display: 'block', transform: 'rotate(90deg)' }}>‹</span>
 </button>

 <div className="sale-window">
  <div
   className="sale-track"
   style={{ transform: `translateY(-${saleIndex * 220}px)` }}
  >
   {saleItems.map((item) => (
    <div className="sale-item" key={item.id}>
     <img src={item.img} alt={item.type} className="sale-img" />
     <div className="sale-info">
      <h3>{item.name}</h3>
      <p>{item.desc}</p>
      <button className="btn-gray">purchase</button>
     </div>
    </div>
   ))}
  </div>
 </div>

 <button
  className="arrow-btn arrow-btn-vertical"
  onClick={handleSaleDown}
  disabled={saleIndex + SALE_VIEW_COUNT >= saleItems.length}
 >
  <span style={{ display: 'block', transform: 'rotate(90deg)' }}>›</span>
 </button>
</div>
css
코드 복사
.sale-item {
 border: 1px solid #eee;
 border-radius: 12px;
 transition: box-shadow 0.2s;
}
.sale-item:hover {
 box-shadow: 0 4px 12px rgba(0,0,0,0.05);
}
```

본 이미지는 Sale 영역에서 화살표 버튼을 눌렀을 때 상품 카드가 세로 방향으로 이동하며 변경되는 기능을 보여주는 화면입니다.

제가 구현하고자 한 내용은 한 화면에 너무 많은 정보를 나열하지 않고, 사용자가 화살표만으로 핵심 상품을 단계적으로 탐색할 수 있게 하는 것이었습니다.

초기에는 상품이 많아질수록 화면이 길어지고 사용자가 스크롤을 반복해야 하는 불편함이 발생할 수 있다고 판단했습니다.

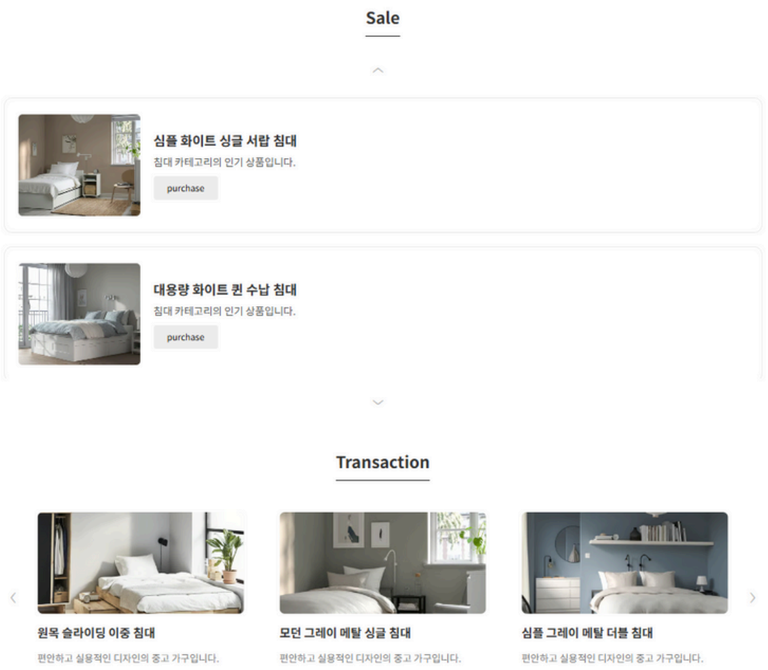
이를 개선하기 위해 saleIndex를 상태로 관리하고, 한 번에 2개씩 이동하도록 SALE_VIEW_COUNT를 설정해 페이지 단위 탐색 방식의 세로 슬라이더로 구현했습니다.

또한 카드에 마우스를 올렸을 때 시각적으로 집중할 수 있도록 그림자와 강조 효과를 적용하여, 단순 리스트보다 더 서비스형 UI에 가까운 느낌을 주도록 구성했습니다.

이 결과 사용자의 탐색 부담이 줄어들고, Sale 콘텐츠의 가독성과 몰입도가 향상되었다고 정리했습니다.

02

구현페이지 및 설명 - 물건 업로드




```
const [transIndex, setTransIndex] = useState(0);
const TRANS_VIEW_COUNT = 3;

const handleTransLeft = () => {
 setTransIndex(prev => Math.max(prev - TRANS_VIEW_COUNT, 0));
};

const handleTransRight = () => {
 if (transIndex + TRANS_VIEW_COUNT < transactionItems.length) {
  setTransIndex(prev => prev + TRANS_VIEW_COUNT);
 }
};
jsx
코드 복사
<div className="slider-wrapper">
 <button
  className="arrow-btn arrow-btn-horizontal"
  onClick={handleTransLeft}
  disabled={transIndex === 0}
 >
  ‹
 </button>

 <div className="slider-window">
  <div
   className="slider-track"
   style={{
    transform: `translateX(calc(-${transIndex} * (100% / 3 + 6.66px)))`
   }}
  >
   {transactionItems.map((item) => (
    <div className="trans-card" key={item.id}>
     <img src={item.img} alt={item.title} className="trans-img" />
     <h3>{item.title}</h3>
     <p>{item.desc}</p>
    </div>
   ))}
  </div>
 </div>

 <button
  className="arrow-btn arrow-btn-horizontal"
  onClick={handleTransRight}
  disabled={transIndex + TRANS_VIEW_COUNT >= transactionItems.length}
 >
  ›
 </button>
</div>
css
코드 복사
.trans-card {
 border-radius: 12px;
 cursor: pointer;
 transition: box-shadow 0.2s, transform 0.2s, border-color 0.2s;
}
.trans-card:hover {
 transform: translateY(-5px);
 box-shadow: 0 10px 20px rgba(0,0,0,0.08);
 border-color: #eee;
}
```

이 이미지는 Transaction 영역에서 좌우 화살표를 클릭하면 카드 목록이 가로 방향으로 전환되는 기능을 보여주는 화면입니다.

제가 구현하고자 한 내용은 중고 거래 추천을 마치 쇼핑몰 카드 UI처럼 직관적으로 탐색할 수 있게 하는 것이었습니다.

개발 과정에서 세로 나열 방식은 추천 콘텐츠의 흐름을 끊고 페이지가 길어지는 불편함이 있다고 느꼈습니다.

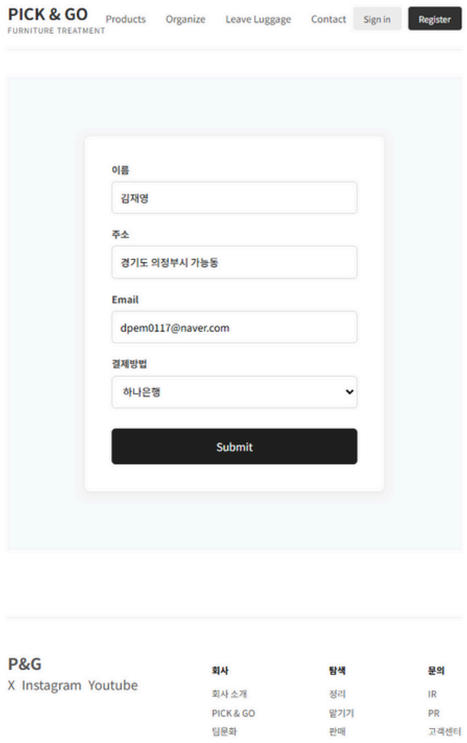
이를 개선하기 위해 transIndex를 상태로 관리하고, 한 번에 3개씩 이동하도록 TRANS_VIEW_COUNT를 설정해 가로 캐러셀 형태의 탐색 구조로 구현했습니다.

또한 카드에 마우스를 올리면 살짝 위로 떠오르는 효과와 그림자를 적용하여, 사용자가 현재 보고 있는 항목을 자연스럽게 강조할 수 있도록 구성했습니다.

이로 인해 단순 리스트보다 더 인터랙티브한 거래 추천 화면이 되었고, 사용자 입장에서 콘텐츠 탐색이 훨씬 편리해졌다고 정리했습니다.

02

구현페이지 및 설명 - 회원가입




```
import React, { useState } from 'react';

const StorePage = () => {
  const [formData, setFormData] = useState({
    name: '김재영',
    address: '경기도 의정부시 가능동',
    email: 'dpem0117@naver.com',
    bank: '하나은행'
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    alert('제출이 완료되었습니다!₩n이름: ' + formData.name);
  };
jsx
코드 복사
  return (
    <div className="store-page">
      <div className="form-wrapper">
        <div className="form-box">
          <form onSubmit={handleSubmit}>
            <div className="form-group">
              <label>이름</label>
              <input
                type="text"
                name="name"
                value={formData.name}
                onChange={handleChange}
              />
            </div>

            <div className="form-group">
              <label>주소</label>
              <input
                type="text"
                name="address"
                value={formData.address}
                onChange={handleChange}
              />
            </div>

            <div className="form-group">
              <label>Email</label>
              <input
                type="email"
                name="email"
                value={formData.email}
                onChange={handleChange}
              />
            </div>

            <div className="form-group">
              <label>결제방법</label>
              <select
                name="bank"
                value={formData.bank}
                onChange={handleChange}
              >
                <option value="하나은행">하나은행</option>
                <option value="국민은행">국민은행</option>
                <option value="신한은행">신한은행</option>
                <option value="카카오뱅크">카카오뱅크</option>
              </select>
            </div>

            <button type="submit" className="btn-submit">
              Submit
            </button>
          </form>
        </div>
      </div>
    </div>
  );
};

export default StorePage;
```

본 화면은 맡겨두기(보관) 서비스 이용 시 필요한 기본 정보를 입력받는 폼 페이지를 React 함수형 컴포넌트로 구현한 결과입니다.

제가 구현하고자 한 내용은 사용자가 보관 서비스를 신청할 때 이름, 주소, 이메일, 결제방법을 한 화면에서 간단하게 입력하고 즉시 제출할 수 있는 흐름을 만드는 것이었습니다.

개발 과정에서 입력값이 늘어나면 값 관리가 복잡해지고, 입력 UI가 많을수록 상태 처리가 분산되어 실수가 발생할 수 있다는 불편함이 있었습니다.

이를 개선하기 위해 useState로 formData 객체를 구성하여 입력값을 한 번에 관리했고, handleChange에서 name 속성을 기준으로 값을 갱신하는 방식으로 코드를 단순화했습니다.

또한 제출 시에는 onSubmit 이벤트에서 기본 새로고침을 막고, 입력된 이름 정보를 알림으로 확인할 수 있게 하여 사용자가 정상 제출 여부를 바로 피드백받도록 구성했습니다.

이러한 방식으로 기존의 정적 폼보다 상태 기반 입력 흐름이 명확해졌고, 실제 서비스 페이지처럼 구조화된 신청 UI를 구현할 수 있었다고 정리했습니다.

THANK YOU