First Personal Project | July 2024

Matrix Calculator

김유진

Project Review Timeline

OVERVIEW

웹 기반 행렬 계산기 프로젝트

HTML과 CSS로 직관적이고 깔끔한 UI를 설계 목표로 하였으며,

JavaScript로 효율적인 행렬 연산과 랜덤 숫자 생성 기능을 구현했습니다.

LANGUAGE

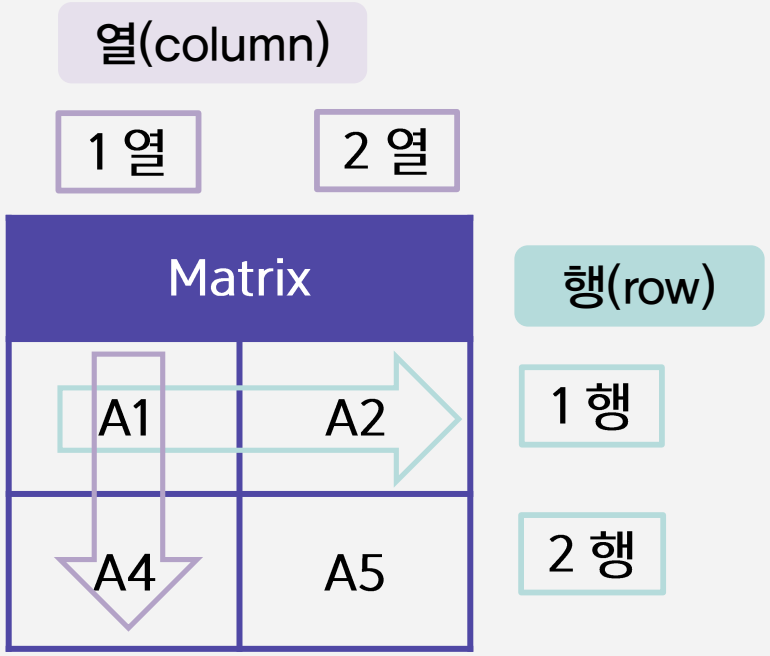
TOOL

Go to Link

DETERMINANT

- 행렬이란 무엇인가요?
- 행렬의 덧셈과 뺄셈
- 행렬의 곱셈

행렬이란 무엇인가요?



- 행과 열로 이루어진 숫자 배열
- 숫자나 수식을 직사각형 배열로 구성한 것
- 가로는 행(row), 세로는 열(column)
- 행렬의 (i, j) 성분은 i번째 행과 j번째 열의 요소
- i개의 행과 j개의 열로 이루어진 행렬은 i x j 행렬

Go to Link

행렬의 덧셈과 뺄셈

연산조건 연산할 두 행렬의 행과 열 개수가 일치해야 합니다.

계산과정 행렬의 동일한 위치에서 값을 더하거나 뺍니다.

A		
A1	A2	A3
A4	A5	A6
A7	A8	A9

$\pm$

B		
B1	B2	B3
B4	B5	B6
B7	B8	B9

A + B		
A1 + B1	A2 + B2	A3 + B3
A4 + B4	A5 + B5	A6 + B6
A7 + B7	A8 + B8	A9 + B9

A − B		
A1 − B1	A2 − B2	A3 − B3
A4 − B4	A5 − B5	A6 − B6
A7 − B7	A8 − B8	A9 − B9

Go to Link

행렬의 곱셈

연산조건 첫번째 행렬 A의 열의 수가 두번째 행렬 B의 행의 수와 같아야합니다.

계산과정 행과 열의 곱: 행렬 A의 각 행과 행렬 B의 각 열을 차례로 곱합니다.
합의 계산: 곱한 결과들을 더하여 결과행렬 C의 해당 위치에 값을 넣습니다.

	3 열			3 행			
A				B		C	
A1	A2	A3		B1	B2	C11	C12
A4	A5	A6	X	B3	B4	= C21	C22
A7	A8	A9		B5	B6	C31	C32

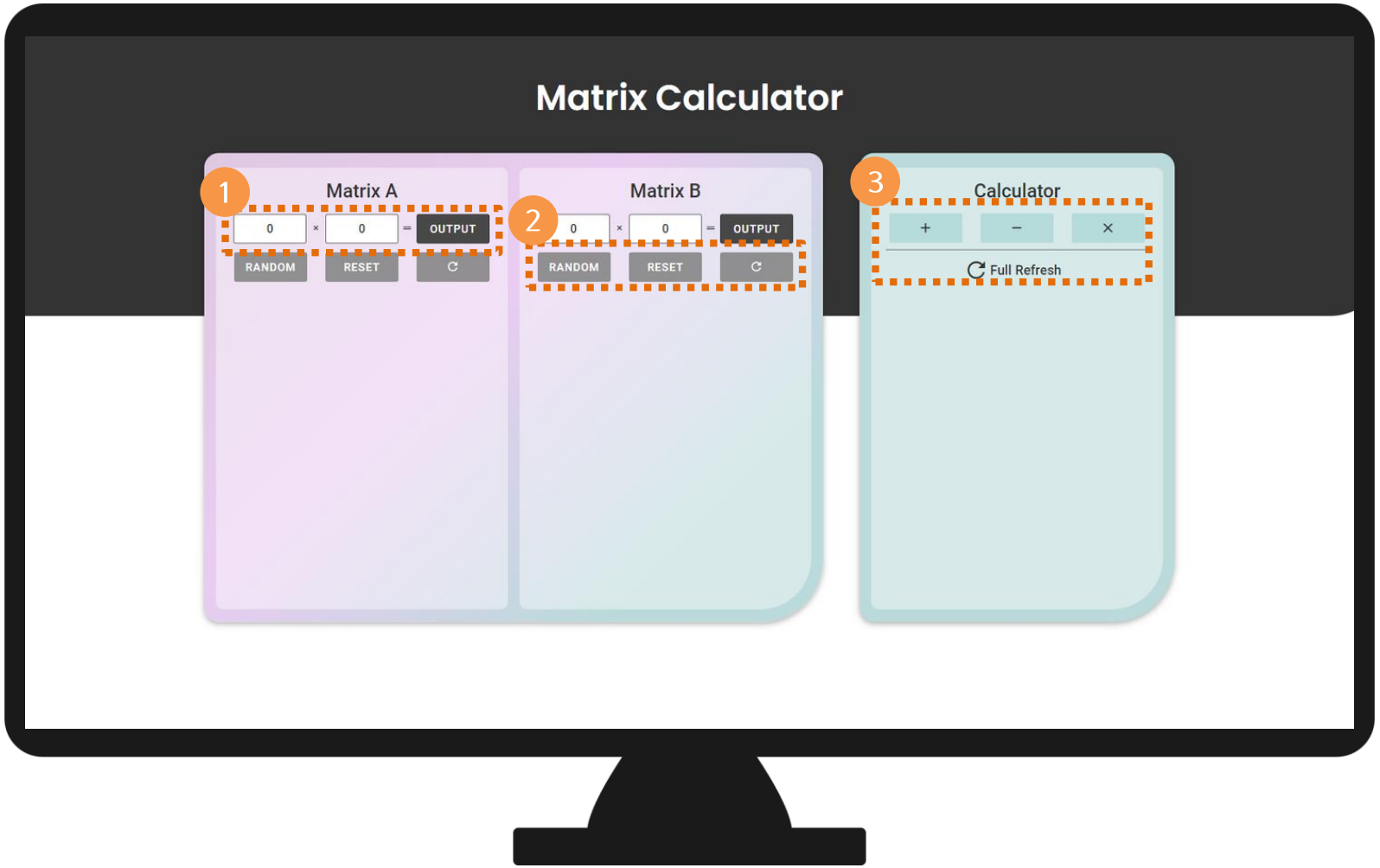
C	
(A1 x B1) + (A2 x B3) + (A3 x B5)	(A1 x B2) + (A2 x B4) + (A3 x B6)
(A4 x B1) + (A5 x B3) + (A6 x B5)	(A4 x B2) + (A5 x B4) + (A6 x B6)
(A7 x B1) + (A8 x B3) + (A9 x B5)	(A7 x B2) + (A8 x B4) + (A9 x B6)

Go to Link

DESIGN LAYOUT

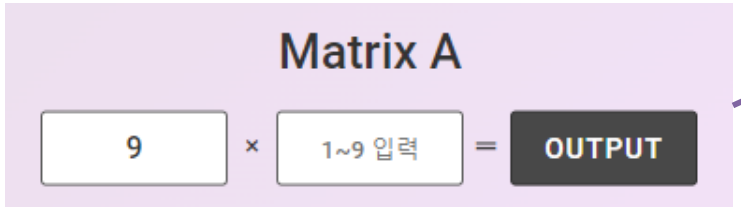
- 기본화면 VIEW
- 기본화면 Description
- 계산화면 VIEW
- 계산화면 Description
- 메뉴바 VIEW
- 메뉴바 Description
- 계산결과 팝업창
- 안내창 (Toast창)

기본화면 VIEW



Go to Link

기본화면 Description



1

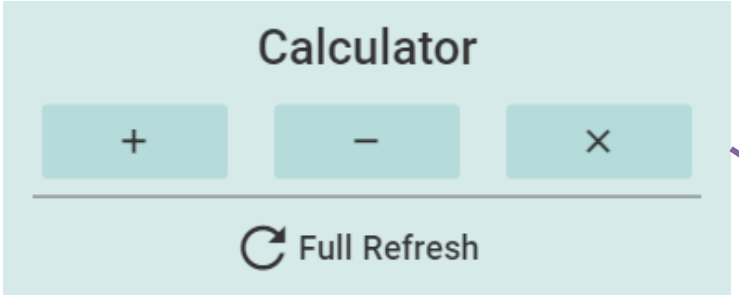
행/열 입력 칸 1~9까지 입력 가능합니다.

2

1. RANDOM 버튼
   모든 입력 칸에 0부터 99 사이의 무작위 숫자가 자동으로 입력됩니다.

2. RESET 버튼
   모든 입력 칸의 값이 0으로 초기화됩니다.

3. 초기화(Refresh) 버튼
   해당 Matrix box부분이 처음 상태로 돌아갑니다.



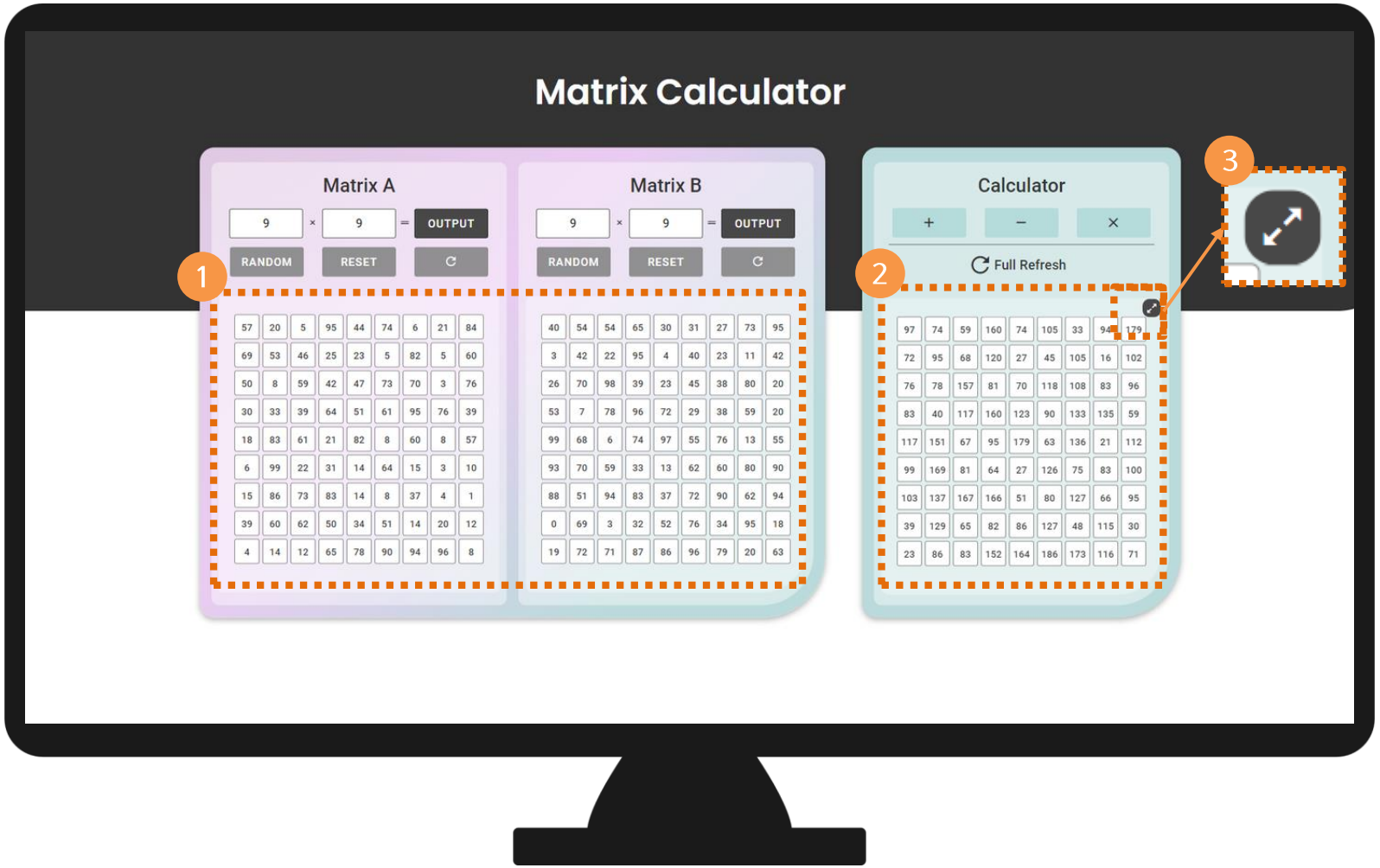
3

1. 덧셈, 뺄셈, 곱셈 연산
   행렬의 덧셈/뺄셈/곱셈 연산이 수행됩니다.

2. Full Refresh 버튼
   입력된 값과 결과가 제거되며, 처음 상태로 돌아갑니다.

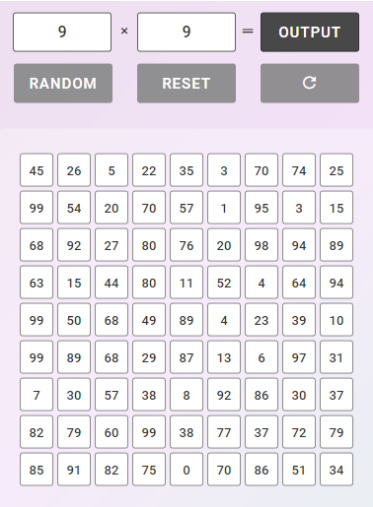
Go to Link

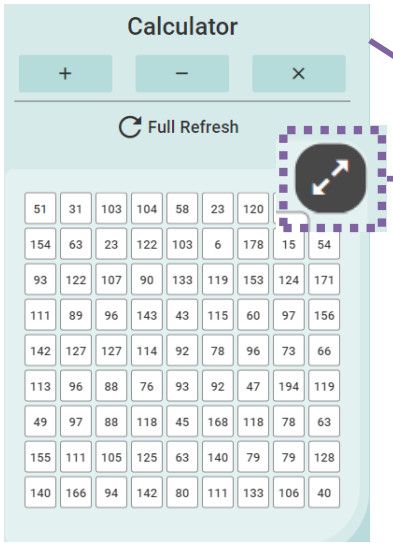
계산화면 VIEW



Go to Link

계산화면 Description





1

1. 행렬 칸에 입력 후 OUTPUT누르면 계산 결과에 따른 input칸들이 생성됩니다.
2. 각 input칸은 입력 가능합니다.

2

1. 행렬 연산 버튼(덧셈, 뺄셈, 곱셈 중 하나 선택)을 클릭하여 결과를 확인할 수 있습니다.
2. 계산 결과는 별도의 출력 칸에 나타납니다.
3. 결과 칸은 readonly로 설정되어 직접 수정할 수 없습니다.

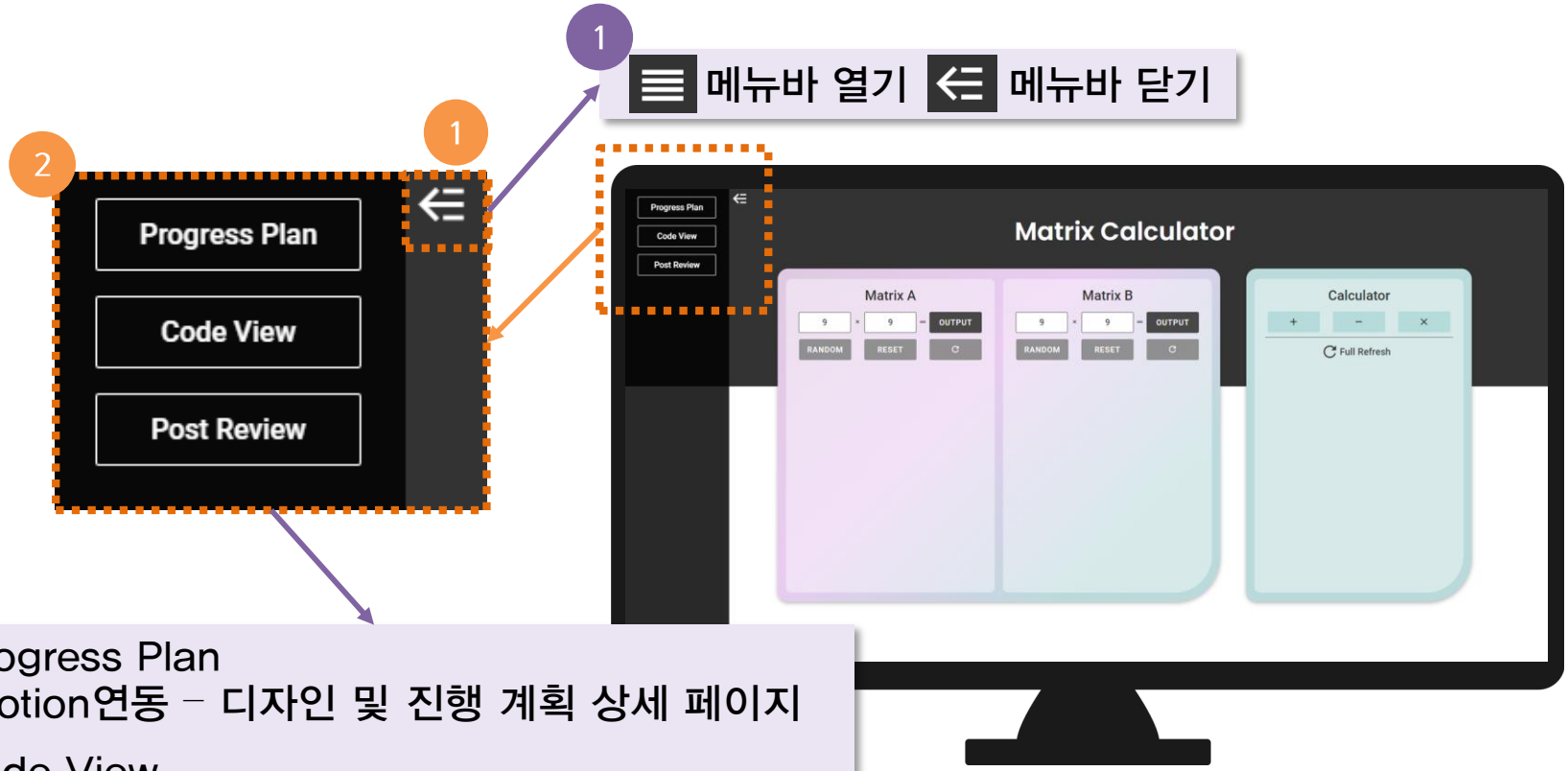
3

결과 화면 팝업창 버튼

1. 결과화면이 출력됨과 동시에 팝업창버튼이 생성됩니다.
2. 확대버튼 클릭 시, 연산 결과를 별도의 팝업창으로 띄웁니다.

Go to Link

메뉴바 VIEW



2

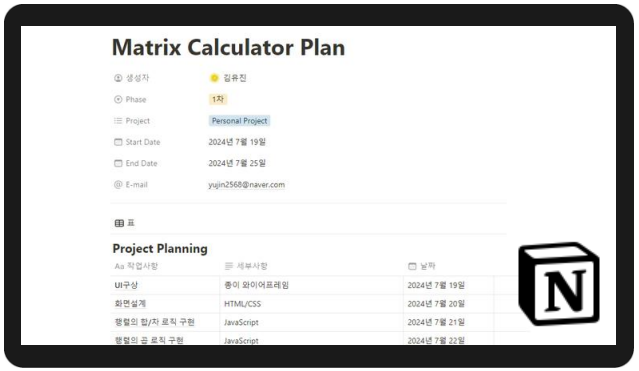
- Progress Plan
  Notion연동 – 디자인 및 진행 계획 상세 페이지
- Code View
  GitHub연동 – 작성 코드 View
- Post Review
  프로젝트 완료 후 기능 설명 및 리뷰 작성 PDF

Go to Link

DESIGN LAYOUT

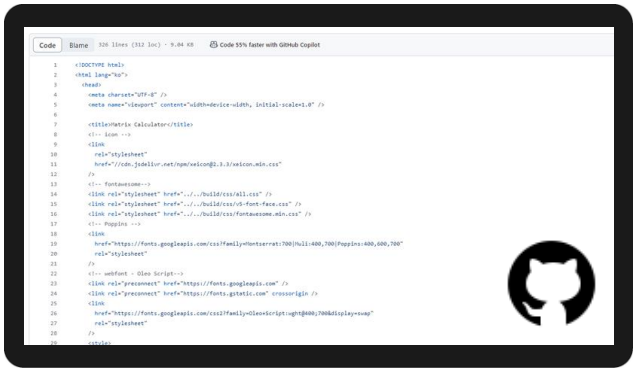
메뉴바 Description

Progress Plan



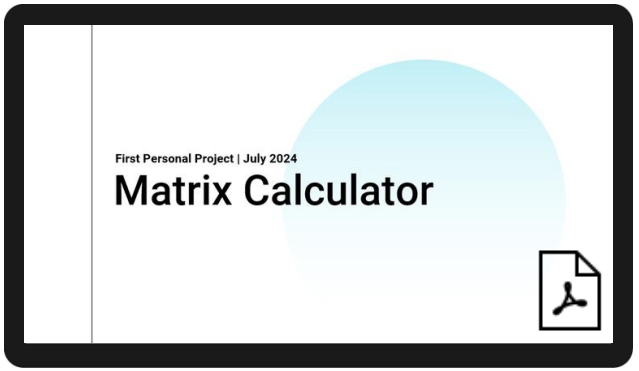
- Project Planning
- UI계획, Paper 와이어프레임
- 행렬의 합/차 로직 구상
- 행렬의 곱 로직 구상

Code View



- GitHub 코드 View

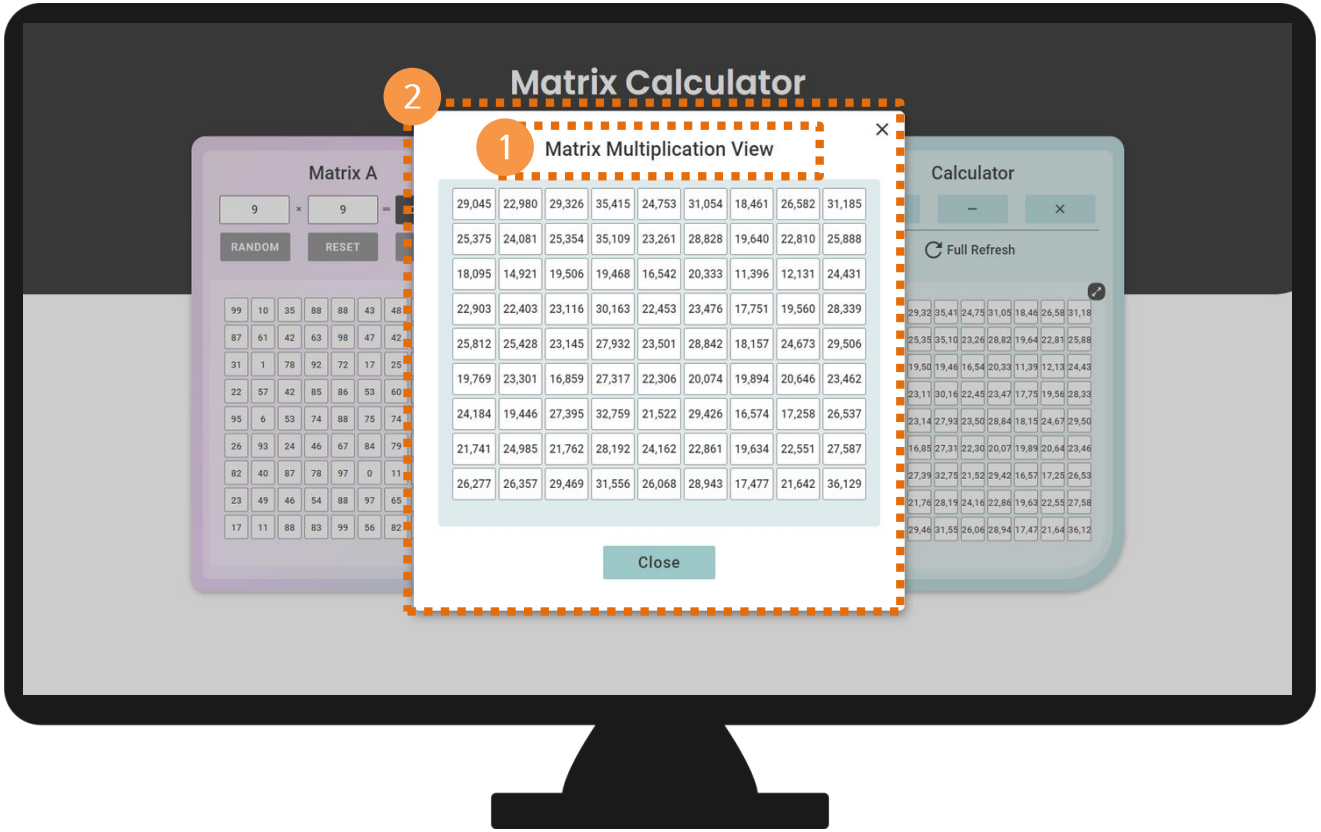
Post Review



- 화면 구현 설명
- 기능 구현 설명
- 개인 피드백

Go to Link

계산결과 팝업창



1

행렬 덧셈 결과 팝업창 제목

Matrix Summation View

행렬 뺄셈 결과 팝업창 제목

Matrix Subtraction View

행렬 곱셈 결과 팝업창 제목

Matrix Multiplication View

2

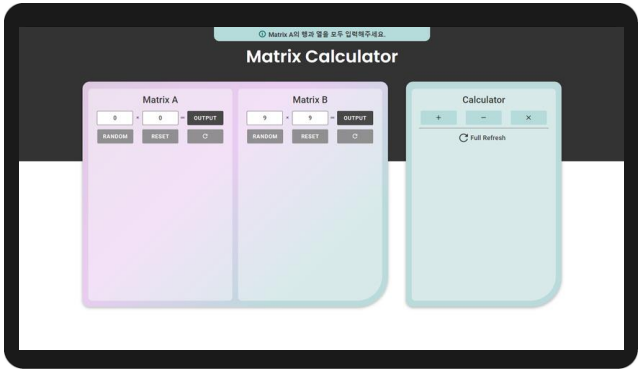
행렬 연산 결과값 팝업 표시

Go to Link

안내창 (Toast창)

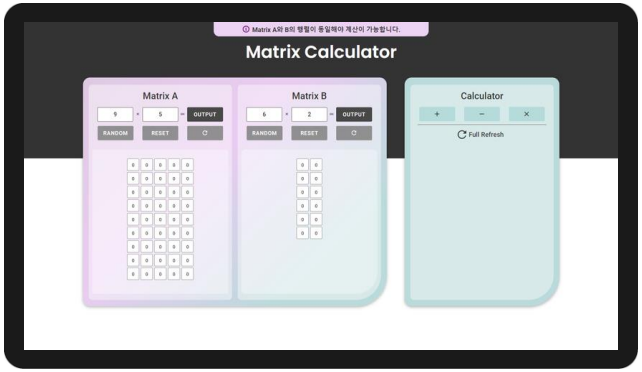
1

행렬칸 중 하나라도 0인 경우



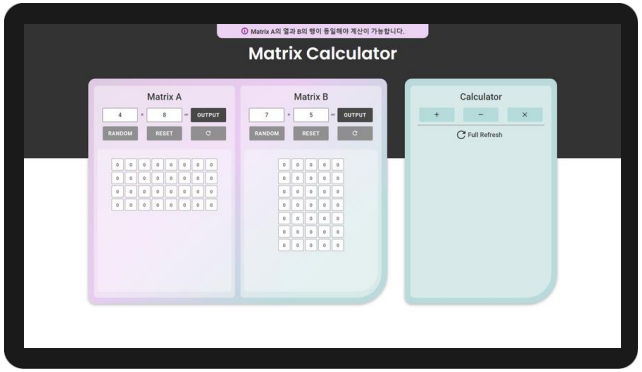
2

덧셈/뺄셈 시 행열 불일치



3

곱셈 시 A의 열과 B의 행 불일치



안내 문구

1 ⓘ Matrix A의 행과 열을 모두 입력해주세요.

2 ⓘ Matrix A와 B의 행렬이 동일해야 계산이 가능합니다.

3 ⓘ Matrix A의 열과 B의 행이 동일해야 계산이 가능합니다.

Go to Link

CODE

- HTML
- CSS
- JAVASCRIPT

HTML

HTML / Matrix A 행렬 구조 (B행렬 동일)

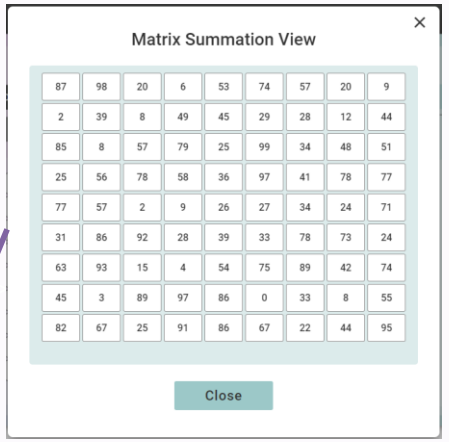
```html
<!-- Matrix A_Content -->
<section class="ContMatrixWrap">
  <article class="subTitleBox">
    <h2 class="subMatrixTitle">Matrix A</h2>
  </article>
  <article class="rcAllInputBox">
    <input id="aMatrixY" class="inputMatrixB" type="number" value="9" min="0" max="9" autofocus placeholder="1~9 입력”/>
    <i class="xi-close-min outIcon"></i>
    <input id="aMatrixX" class="inputMatrixB" type="number" value="9" min="0" max="9" placeholder="1~9 입력" />
    <i class="xi-drag-handle"></i>
    <button id="aOutMatrixBtn" class="outputBtn outIcon">OUTPUT</button>
  </article>
  <article class="matrixBtnBox">
    <button id="aRandomMatrixBtn" class="btnMatrix">RANDOM</button>
    <button id="aResetMatrixBtn" class="btnMatrix">RESET</button>
    <button id="aClearMatrixBtn" class="btnMatrix"><i class="xi-refresh matrixRefreshBtn"></i></button>
  </article>
  <section id="aOutPutWrapB" class="outPutWrapB">
    <article id="aAreaMatrix" class="outPutBox">
      <!-- matrix cell 들어올 자리 -->
    </article>
  </section>
</section>
```

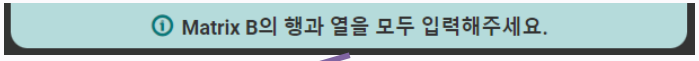
Go to Link

HTML

HTML / 결과팝업창 구조

```
<!-- Popup_Modal -->
<div id="popupModalBg" class="popupModalBackground">
  <div id="popupContent" class="popupWrap">
    <article class="popupHeaderB">
      <h2 id="popupTitle" class="popupTitle">Calculator View</h2>
      <i id="popupTopClosebtn" class="xi-close popupTopCloseIcon"></i>
    </article>
    <section class="popupMiddleB">
      <article class="popupValueB">
        <article id="popupInputBox" class="popupInputBox">
          <!-- result cell 들어올 자리-->
        </article>
      </article>
    </section>
    <article class="popupBottomB">
      <button id="popupBottomClosebtn" class="popupBottomClosebtn">
        Close
      </button>
    </article>
  </div>
</div>
```





HTML / 안내창 구조

```
<!-- Warning -->
<section id="rWarningBox" class="warningBox">
  <i class="xi-info-o warningIcon" id="warningIcon"></i>
  <h2 class="warningBtxt" id="warningBtxt">
    <!-- A의 열과 B의 행이 동일해야 계산이 가능합니다. -->
  </h2>
</section>
```

Go to Link

CSS

CSS / Matrix A, B

```css
.rcAllInputBox {
  width: 100%;
  height: 10%;
  padding: 14px;
}
.inputMatrixB {
  width: 100px;
  height: 40px;
  outline: 0;
  text-align: center;
  border: 1px solid #5f5f5f;
  border-radius: 3px;
  font-size: 16px;
}
.inputMatrixB:focus {
  border-color: #333;
  background-color: #f4f4f400;
  outline: none;
}
.matrixBtnBox {
  width: 100%;
  height: 10%;
  display: flex;
  justify-content: space-evenly;
}
```

Matrix A와 Matrix B는 동일한 클래스를 공유하며, 일관된 스타일을 효율적으로 관리할 수 있도록 구현하였습니다.

Go to Link

CSS

CSS / hover시 박스전환

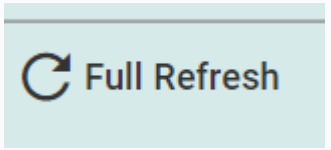
```css
.refreshSubB,
.refreshActiveB {
  position: absolute;
  display: flex;
  align-items: center;
  width: 90%;
  left: 50%;
  top: 40%;
  transform: translate(-50%, -50%);
}

.refreshCont:hover .refreshActiveB {
  opacity: 1;
  cursor: pointer;
}

.refreshCont:hover .refreshActiveB>.refreshTxt {
  font-weight: 700;
}

.refreshCont:hover .refreshBeforeB {
  opacity: 0;
}
```

.refreshSubB



기존

.refreshActiveB

Mouse hover

두 개의 겹쳐진 요소

(.refreshSubB와 .refreshActiveB)를 구현

.refreshCont 요소에 마우스를 올리면

.refreshSubB는 투명해지고, (opacity: 0)

.refreshActiveB 박스가 나타납니다. (opacity: 1)

hover시 이 박스 내부의 아이콘은 회전하며

텍스트는 두껍게 표시됩니다.

Go to Link

Javascript

Matrix A Object

```
// Matrix A_box
const frstMatrixR = {
  aMatrixX: '',
  aMatrixY: '',
  aOutMatrixBtn: '',
  aRandomMatrixBtn: '',
  aResetMatrixBtn: '',
  aClearMatrixBtn: '',
  aOutPutWrapB: '',
  aAreaMatrix: '',
  warningBtxt: '',
  rWarningBox: '',
  warningIcon: '',
  aMatrixVArr: [],
  akeyName: function (item) {···
  aMatrixWork: function () {···
  aMatrixData: function () {···
  aMatrixRandomCreate: function () {···
  aMatrixResetFunc: function () {···
  aMatrixClearFunc: function () {···
  aMatrixXLimit: function () {···
  aMatrixYLimit: function () {···
  aSubBoxValueLimit: function () {···
}; //A_end
```

Matrix B Object

```
// Matrix B_box
const scndtMatrixR = {
  bMatrixX: '',
  bMatrixY: '',
  bOutMatrixBtn: '',
  bRandomMatrixBtn: '',
  bResetMatrixBtn: '',
  bClearMatrixBtn: '',
  bOutPutWrapB: '',
  bAreaMatrix: '',
  rWarningBox: '',
  warningBtxt: '',
  warningIcon: '',
  bMatrixVArr: [],
  bkeyName: function (item) {···
  bMatrixWork: function () {···
  bMatrixData: function () {···
  bMatrixRandomCreate: function () {···
  bMatrixResetFunc: function () {···
  bMatrixClearFunc: function () {···
  bMatrixXLimit: function () {···
  bMatrixYLimit: function () {···
  bSubBoxValueLimit: function () {···
}; //B_end
```

관련된 DOM 요소와 메서드를 객체로 묶어 모듈화하여 기능을 효율적으로 유지하도록 구현하였습니다.

행렬 데이터 배열을 통해 (aMatrixVArr / bMatrixVArr) 행렬의 결과상태를 한 배열에 담을 수 있도록 구현하였습니다.

Go to Link

CODE

Javascript

Javascript / DOM 요소 가져오기

```
const frstMatrixR = {
  aMatrixX: '',
  aMatrixY: '',
  aOutMatrixBtn: '',
  aRandomMatrixBtn: '',
  aResetMatrixBtn: '',
  aClearMatrixBtn: '',
  aOutPutWrapB: '',
  aAreaMatrix: '',
  warningBtxt: '',
  rWarningBox: '',
  warningIcon: '',
  aMatrixVArr: [], // 결과 할당할 배열
  akeyName: function (item) {
    for (let keyName in item) {
      if (document.getElementById(keyName)) {
        item[keyName] = document.getElementById(keyName);
      }
    }
  }
}
```

akeyName 메서드는 item 객체의 속성 이름을 활용하여, document.getElementById로 해당하는 HTML 요소를 동적으로 찾고 업데이트합니다.

DOM 요소의 ID와 변수 이름을 일치시킴으로써 코드의 일관성을 유지하며 효율적으로 요소를 참조할 수 있도록 구현하였습니다.

Go to Link

Javascript

Javascript / Matrix A행렬 생성 (B행렬 동일)

```
const AyV = Number(frstMatrixR.aMatrixY.value);
const AxV = Number(frstMatrixR.aMatrixX.value);
const ByV = Number(scndtMatrixR.bMatrixY.value);
const BxV = Number(scndtMatrixR.bMatrixX.value);

  this.cOutPutBox.innerHTML = null;
  this.cOutPutWrapB.style.opacity = 1;
  this.cOutPutBox.style.width = 38 * AxV + 'px';

  this.rMatrixVArr = []; // 결과 행렬 배열 초기화
  for (let y = 0; y < AyV; y++) {
    this.rMatrixVArr.push([]); // 새로운 행 추가
    for (let x = 0; x < AxV; x++) {
      this.rMatrixVArr[y].push(0); // 새로운 열 추가 및 값 초기화
    }
  }
}
```

첫 번째 행렬의 행과 열, 두 번째 행렬의 행과 열을 숫자로 변환합니다.

출력 박스 내용을 비우고 출력 박스의 스타일을 설정합니다.

행렬 수 만큼 공배열을 생성합니다.

Go to Link

Javascript

Javascript / 행렬의 값 변경 연동

```
aMatrixData: function () {
  this.aAreaMatrix.addEventListener(
    'change',
    function (event) {
    // 배열값 연동 시키기
     let value = Number(event.target.value);
     let idLen = (event.target.id.length);
     let xIdx = event.target.id[idLen - 1];
     let yIdx = event.target.id[idLen - 2];
     frstMatrixR.aMatrixVArr[yIdx][xIdx] = value;
  });
}
```

행렬을 구성하는 input의 value가 변경된다면,
행렬을 구성하고 있는 데이터(배열)를 변경하여
다시 화면에 렌더링 합니다.

첫번째 행렬의
input값을 변경한 모습

1	0	8	0	8
5	0	6	4	7
0	8	7	0	5
89	6	3	0	3
0	3	0	2	0

```
▶ 0: (5) [1, 0, 8, 0, 8]
▶ 1: (5) [5, 0, 6, 4, 7]
▶ 2: (5) [0, 8, 7, 0, 5]
▶ 3: (5) [89, 6, 3, 0, 3]
▶ 4: (5) [0, 3, 0, 2, 0]
  length: 5
```

Go to Link

Javascript

Javascript / 행렬 덧셈 연산식

```
for (let y = 0; y < AyV; y++) {// 행렬의 행을 반복
  for (let x = 0; x < AxV; x++) { // 행렬의 열을 반복
    const frstV = Number(frstMatrixR.aMatrixVArr[y][x]);
    const scndV = Number(scndtMatrixR.bMatrixVArr[y][x]);
     // 두 값을 더하여 결과 행렬의 (y, x) 위치에 저장
    this.rMatrixVArr[y][x] = frstV + scndV;
  }
}

for (let y = 0; y < AyV; y++) {
  for (let x = 0; x < AxV; x++) {
    let rCellNum = String(y) + String(x);
    let cellValue =Number(calcMatrixR.rMatrixVArr[y][x]).toLocaleString('ko-KR');
    let input = `<input id="rCell${rCellNum}" class="matrixCell" type="text" value="${cellValue}"
readonly>`;
    calcMatrixR.cOutPutBox.innerHTML += input;
  }
}
```

두 개의 행렬(frstMatrixR, scndtMatrixR)을 반복하여 요소별로 더한 값을 결과 배열(rMatrixVArr)에 저장합니다.
이후, 결과 배열의 값을 HTML 입력 필드로 생성하고 cOutPutBox에 할당해 시각화 합니다.

Go to Link

Javascript

Javascript / 행렬 곱셈 연산식

```
const AyV = Number(frstMatrixR.aMatrixY.value);
const AxV = Number(frstMatrixR.aMatrixX.value);
const ByV = Number(scndtMatrixR.bMatrixY.value); //AxV==ByV 일때만 실행하므로 for문 조건엔 필요없음
const BxV = Number(scndtMatrixR.bMatrixX.value);

  this.cOutPutBox.style.width = 38 * BxV + 'px';
  this.cOutPutBox.innerHTML = null; // DOM초기화
  this.rMatrixVArr = []; // 배열초기화

  for (let i = 0; i < AyV; i++) {
    calcMatrixR.rMatrixVArr.push([]);
    for (let j = 0; j < BxV; j++) {
      calcMatrixR.rMatrixVArr[i].push(0);
    }
  }

  for (let i = 0; i < AyV; i++) {
    let calcV = 0;
    for (let j = 0; j < BxV; j++) {
      for (let k = 0; k < AxV; k++) {
        let frstV = Number(frstMatrixR.aMatrixVArr[i][k]);
        let scndV = Number(scndtMatrixR.bMatrixVArr[k][j]);
        calcV += frstV * scndV;
      }
      calcMatrixR.rMatrixVArr[i][j] = calcV;
      calcV = 0;
    }
  }
```

A행렬의 행만큼 새로운 배열 삽입,
B행렬의 열만큼 배열 안에 0을 삽입하여
공배열을 만들었습니다.

행렬의 곱셈 로직은 3중 for문을 사용하여
구현했습니다.

Go to Link

Javascript

Javascript / 예외처리 – 행렬 덧셈 계산 불가 조건

```
const AyV = Number(frstMatrixR.aMatrixY.value);
const AxV = Number(frstMatrixR.aMatrixX.value);
const ByV = Number(scndtMatrixR.bMatrixY.value);
const BxV = Number(scndtMatrixR.bMatrixX.value);
const fsArr = frstMatrixR.aMatrixVArr.length;
const scArr = scndtMatrixR.bMatrixVArr.length;
const RV = fsArr + scArr != 0;

this.cOutPutBox.style.width = 38 * AxV + 'px';
this.cOutPutBox.innerHTML = null; // DOM초기화
this.rMatrixVArr = []; // 배열초기화

if (AyV == ByV && AxV == BxV && RV && fsArr != 0 && scArr != 0) {

            // 행렬 덧셈 로직 구현 부분


    }
```

1. A의 행과 B의 행의 값이 일치해야 합니다.
2. A의 열과 열의 행의 값이 일치해야 합니다.
3. A의 결과값, B의 결과값 모두 0이 아니여야 합니다.

Go to Link

Javascript

Javascript / 예외처리 – 행렬 곱셈 계산 불가 조건

```
const AyV = Number(frstMatrixR.aMatrixY.value);
const AxV = Number(frstMatrixR.aMatrixX.value);
const ByV = Number(scndtMatrixR.bMatrixY.value);
const BxV = Number(scndtMatrixR.bMatrixX.value);

const fsArr = frstMatrixR.aMatrixVArr.length;
const scArr = scndtMatrixR.bMatrixVArr.length;
const RV = fsArr + scArr != 0;

// 출력조건
if (AxV == ByV && RV && fsArr != 0 && scArr != 0) {

    //  행렬 곱셈 로직 구현 부분

}
```

1. A의 행과 B의 열의 값이 일치해야 합니다.
2. A, B 행렬의 결과가 0이 아니여야 합니다.

Go to Link

Javascript

Javascript / 예외처리 – 행렬 개수 입력 제한

```
aMatrixXLimit: function () {
  this.aMatrixX.addEventListener('input', function (event) {
    let value = event.target.value;
    if (value <= 0) {
      event.target.value = '';
    }
    if (value > 9) {
      event.target.value = value.slice(0, 1);
    }
  });
}
```

행렬 생성 시 입력 값이 1~9 범위를 벗어나는 경우 예외 처리를 구현했습니다.

1. 입력 값이 0 이하인 경우에는 공백으로 처리합니다.
2. 9를 초과하는 경우에는 문자열의 앞자리만 잘라서 삽입합니다.

Javascript / 예외처리 – 출력된 input 개수 입력 제한

```
aSubBoxValueLimit: function () {
  this.aOutPutWrapB.addEventListener('input', function (event) {
    let value = event.target.value;
    if (value < -99) {
      event.target.value = -99;
    }
    if (value > 99) {
      event.target.value = 99;
    }
  });
}
```

행렬 생성 후 출력된 input의 입력 값이 –99~99 범위를 벗어나는 경우 예외 처리를 구현했습니다.

1. 입력 값이 –99보다 작을 경우 –99로 설정합니다.
2. 입력 값이 99보다 클 경우 99로 설정합니다.

Go to Link

REVIEW

프로젝트 후기

화면구현 만족도

80%

계산기의 느낌을 유지하면서도 동적인 디자인을 구현하는 것은 추상적이고 어려운 작업이었습니다. 직관성을 높이기 위해 디자인 구조를 네 번에 걸쳐 다듬었습니다.

이 과정에서 폰트와 색상 조화를 신중하게 고려하며, 사용자의 입장에서 최적의 설계를 진행했습니다. 결과적으로, 직관적인 사용성을 우선으로 하여 원하는 조건들을 모두 충족하는 디자인을 완성할 수 있었습니다.

로직이해 만족도

95%

공식을 로직으로 구현하려면 규칙성과 구조를 명확히 이해하는 것이 중요하다고 생각하여, 이를 위해 요소의 인덱스를 하나하나 나열하고 반복문의 구조를 면밀히 분석했습니다.

이 과정을 통해 어떤 반복문이 필요한지 파악하고, 이를 코드로 정확하게 구현할 수 있었습니다. 수학적인 사고를 코드로 전환하는 과정에서 그 중요성을 깊이 깨달았습니다.

유지보수 만족도

75%

사이트를 처음 접하는 사용자의 입장에서 기능을 잘못 사용할 경우에 대한 예외 처리를 신경 써서 편리하게 사용할 수 있도록 구현했습니다. 그러나 현재 구현된 예외 처리에서는 +와 −와 같은 부호가 포함된 값에 대한 처리가 부족하다고 생각합니다.

앞으로는 정규 표현식을 학습하고 활용하여 이러한 부호를 포함한 입력 값에 대해서도 정확하게 예외를 처리할 수 있도록 개선할 계획입니다.

다음프로젝트 목표

1. 기획과 개발의 분리

프로젝트를 진행하며 기획과 개발을 동시에 수행하는 것의 비효율성을 깨달았습니다.
앞으로는 기획과 디자인을 철저히 완료한 후, 화면 구성과 로직 구현을 단계적으로 진행하여,
더 체계적이고 효율적인 개발을 목표로 할 것 입니다.

2. CSS 최적화

반복되는 요소를 중첩 클래스와 적절한 계산을 통해 효율적으로 관리했지만, 여전히 미숙한
부분이 남아 있습니다. 다음 프로젝트 진행 시 더 세밀한 계산과 기획을 통해 여백 사용
(margin, padding)을 줄여 작업하려 합니다.

3. 객체지향 프로그래밍의 심화 학습

객체의 중요성을 인식하고, 향후 프로젝트에서 클래스 문법을 적극적으로 활용하여 객체지향
프로그래밍의 원칙을 따르는 로직을 구현하고자 합니다. 이를 통해 유지보수와 확장성이 뛰어난
코드를 작성하는 능력을 기르려 합니다.

THANK YOU

김유진