**Term Project 과제 (3) - 최종 보고서 및 산출물**

**12204948 최승혁**

주제 : 해동라운지 좌석 선택 웹

해동 라운지와 같은 라운지는 직접 가서 좌석을 확인하기 전까지 자리가 비어 있는지 알기 어렵습니다. 따라서 예약은 하지 않지만 해동 라운지를 입장 시 태블릿과 같은 기기로 좌석을 고르고 들어가 좌석 현황을 해동 라운지 오려고 하는 인원들이 간편하게 웹에서 확인할 수 있도록 하는 것이 목표입니다.

"해동 라운지 좌석 선택 웹"을 개발하여 위의 문제점을 해결하고 좌석 관리를 제공합니다. 이 시스템은 단일 웹 페이지로 구성되어 다음과 같은 기능을 제공합니다.

• 실시간 좌석 정보: 사용자는 웹 페이지를 통해 실시간으로 좌석 사용 여부를 확인할 수 있습니다.

• 간편한 선택 및 해제: 사용자는 웹 페이지를 통해 좌석을 쉽게 선택 및 해제할 수 있습니다.

• 점유 시간 : 자리를 선택해서 사용 시 점유 시간이 흘러가 실시간으로 확인할 수 있습니다.

기술 스택:

* 프론트엔드: HTML, CSS, JavaScript,
* 백엔드: Node.js

웹 개발 경험은 이번 수업이 처음이라 아는 선에서 작성하고 계획했습니다.

**중간보고서**

구상도 :

해동라운지의 좌석 배치를 참고했습니다.

텍스트, 스크린샷, 도표, 직사각형이(가) 표시된 사진

자동 생성된 설명

구현된 프론트 현황입니다.

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

자동 생성된 설명

구현완료된 부분입니다.

* CSS의 Flex와 Grid를 사용하여 책상 레이아웃을 배치했습니다.
  + Flexbox는 주로 행과 열 간의 간격을 조절하고, Grid는 각 책상섬의 그리드를 표현하기 위해 사용되었습니다.
  + 레이아웃을 섬형식으로 조절하기 위해 grid-template-columns: repeat(2, 1fr); 를 적용하는 것이 어려웠습니다.
* 각 좌석을 클릭하여 시간 타이머가 시작됩니다.

(시간을 임의로 줄여놓아 시간이 지나면 색이 바뀌는 것을 확인 할 수 있습니다.)

* 타이머가 진행되고 있는 각 타이머를 다시 클릭하게 되면 “종료하시겠습니까” 라는 alert가 발생해 확인을 누르면 타이머가 사라집니다.
* 텍스트, 스크린샷, 직사각형, 폰트이(가) 표시된 사진

  자동 생성된 설명
* 시간에 따라 표시되는 시간 단위를 조절합니다.
* 텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명

**향후 계획**

* 페이지를 새로고침 해도 타이머를 유지할 수 있는 기능을 추가할 예정입니다.
* 피드백 주신 내용인 백엔드를 Node.js가 아닌 멀티 스레드를 지원하는 백엔드라면 다중으로 특정 좌석 점유 요청 시 어떻게 해야 하는지 조사할 예정입니다.

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명**최종보고서**

**최종 구현 기능**

1. **타이머 유지 기능**

새로고침 후에도 타이머가 유지되도록 로컬 스토리지에 시작 시간을 저장했습니다. 각 좌석의 ID 값을 이용하여 타이머 값을 가져옵니다.

2. **‘사용현황’ 보기 토글 기능 추가**

텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명‘사용현황’ 버튼을 누르면 좌석 아래에 각 좌석의 누적 사용 시간을 확인할 수 있습니다. 누적 사용 시간이 1시간, 2시간, 3시간을 경과함에 따라 좌석 색상이 변경됩니다.

3. **‘사용현황 초기화’ 버튼 추가**

‘사용현황 초기화’ 버튼을 통해 각 좌석별로 기록된 시작 시간을 초기화할 수 있습니다.

4. **좌석별 번호 표시**

각 좌석에 번호를 추가하여, 어느 좌석의 통계인지 쉽게 알 수 있도록 했습니다.

좌석을 클릭하면 타이머가 시작되고, 타이머가 진행 중인 좌석을 다시 클릭하면 타이머가 사라지고 기존의 좌석 번호가 나타납니다.

5. **‘건의하기’ 버튼 추가**

라운지 운영에 대한 불만사항이나 건의사항을 댓글로 작성할 수 있는 ‘건의하기’ 버튼을 추가했습니다. 작성된 댓글은 오른쪽 상단의 X 버튼을 통해 삭제할 수 있습니다.

**개선 사항**

1. **로컬 스토리지 사용해 시작시간을 기록**

const seats = document.querySelectorAll(".seat");

seats.forEach((seat) => {

const savedStartTime = localStorage.getItem(`seat-${seat.id}-startTime`);

if (savedStartTime) {

seat.dataset.startTime = savedStartTime;

seat.classList.add("red");

updateSeat(seat);

}

1. **좌석에 사용현황을 알 수 있도록 통계보기 기능 추가**

// 통계 보기 버튼 이벤트 핸들러

const toggleButton = document.getElementById('toggle-stats');

const statsContainer = document.getElementById('stats-container');

toggleButton.addEventListener('click', () => {

statsContainer.classList.toggle('visible');

if (statsContainer.classList.contains('visible')) {

statsContainer.innerHTML = '';

for (const seatId in seatUsage) {

const seat = document.getElementById(seatId);

const totalUsage = seatUsage[seatId] || 0;

*// 시간, 분, 초 계산*

const hours = Math.floor(totalUsage / 3600);

const minutes = Math.floor((totalUsage % 3600) / 60);

const seconds = Math.floor(totalUsage % 60);

const usageText = `${hours}시간 ${minutes}분 ${seconds}초`;

const seatStats = document.createElement('div');

seatStats.textContent = `${seat.id}: ${usageText}`;

seatStats.classList.add('seat-stats');

if (totalUsage > 3 \* 60 \* 60) {

seatStats.classList.add('pink');

} else if (totalUsage > 2 \* 60 \* 60) {

seatStats.classList.add('blue');

} else if (totalUsage > 1 \* 60 \* 60) {

seatStats.classList.add('green');

} else {

seatStats.classList.add('yellow');

}

statsContainer.appendChild(seatStats);

}

}

});

1. **통계를 초기화하기 위한 버튼 및 초기화 기능 추가**

*// 통계 초기화 버튼 이벤트 핸들러*

const resetButton = document.getElementById("reset-stats");

resetButton.addEventListener("click", () => {

const userConfirmed = confirm("정말로 통계를 초기화하시겠습니까?");

if (userConfirmed) {

localStorage.removeItem("seatUsage");

seatUsage = {};

statsContainer.innerHTML = "";

*// 모든 좌석 상태 초기화*

seats.forEach((seat) => {

clearTimeout(seat.timeoutId); *// 타이머 초기화*

delete seat.dataset.startTime; *// 시작 시간 데이터 삭제*

localStorage.removeItem(`seat-${seat.id}-startTime`); *// 로컬 스토리지에서 시작 시간 삭제*

seat.classList.remove("red", "green", "yellow", "blue", "pink"); *// 모든 색상 클래스 제거*

seat.classList.add("seat"); *// 기본 seat 클래스 추가*

const seatNumberSpan = seat.querySelector("span");

seatNumberSpan.textContent = seat.id.slice(-2); *// 좌석 번호만 표시*

});

statsContainer.classList.remove("visible");

setTimeout(() => {

location.reload();

}, 100);

}

});

1. **‘건의하기’ 버튼 추가**

*// 건의하기 버튼 이벤트 핸들러*

const suggestionButton = document.getElementById("suggestion-button");

const suggestionBox = document.getElementById("suggestion-box");

const suggestionInput = document.getElementById("suggestion-input");

const suggestionList = document.getElementById("suggestion-list");

*// 로컬 스토리지에서 댓글 불러오기*

let suggestions = JSON.parse(localStorage.getItem("suggestions")) || [];

displaySuggestions(); *// 페이지 로드 시 댓글 표시*

*// 건의하기 버튼 클릭 이벤트 핸들러*

suggestionButton.addEventListener("click", () => {

suggestionBox.style.display = suggestionBox.style.display === "none" ? "block" : "none";

});

*// 건의 내용 제출 이벤트 핸들러*

const submitSuggestionButton = document.getElementById("submit-suggestion"); *// 제출 버튼 요소 선택*

submitSuggestionButton.addEventListener("click", () => {

const suggestionText = suggestionInput.value.trim();

if (suggestionText) {

suggestions.push(suggestionText);

localStorage.setItem("suggestions", JSON.stringify(suggestions));

displaySuggestions();

suggestionInput.value = "";

}

});

*// 댓글 표시 함수*

function displaySuggestions() {

suggestionList.innerHTML = "";

suggestions.forEach((suggestion, index) => {

const suggestionItem = document.createElement("div");

suggestionItem.classList.add("suggestion-item");

suggestionItem.innerHTML = `

<span>${suggestion}</span>

<button class="delete-button" data-index="${index}">X</button>

`;

suggestionList.appendChild(suggestionItem);

});

*// 삭제 버튼 이벤트 추가*

const deleteButtons = document.querySelectorAll(".delete-button");

deleteButtons.forEach((button) => {

button.addEventListener("click", () => {

const index = parseInt(button.dataset.index);

deleteSuggestion(index);

});

});

}

*// 댓글 삭제 함수*

function deleteSuggestion(index) {

suggestions.splice(index, 1);

localStorage.setItem("suggestions", JSON.stringify(suggestions));

displaySuggestions();

}

**Node.js와 달리 멀티 스레드를 지원하는 백엔드 (Java (Spring))를 사용하는 경우, 여러 사용자가 동시에 특정 좌석을 점유하려는 상황에 대한 처리 방안에 대한 고민**

**문제 상황:**

여러 사용자가 동시에 동일한 좌석을 예약하려고 할 때, 멀티 스레드 환경에서는 데이터베이스의 좌석 정보가 일관성 없이 변경될 수 있습니다. 예를 들어, 두 명의 사용자가 동시에 특정 좌석을 예약하려고 할 때, 두 요청 모두 좌석이 비어 있다고 판단하여 예약을 진행할 수 있습니다. 이는 데이터베이스의 좌석 상태 정보가 실제와 다르게 변경되는 문제를 야기합니다.

**해결방안**

**1. 동시성 제어 (Concurrency Control):**

* **락 (Lock):** 특정 좌석 데이터에 접근하는 동안 다른 스레드가 접근하지 못하도록 막는 방법입니다. 좌석 데이터에 대한 변경 작업을 수행하는 동안 락을 획득하고, 작업 완료 후 락을 해제하여 데이터 일관성을 유지합니다.
* **세마포어 (Semaphore):** 제한된 개수의 스레드만 특정 자원에 접근할 수 있도록 허용하는 방법입니다. 좌석 개수만큼 세마포어를 생성하고, 좌석 점유 시 세마포어를 획득하고, 해제 시 세마포어를 반환하여 동시 접근을 제어합니다.

**2. 트랜잭션 (Transaction):**

* **데이터베이스 트랜잭션:** 좌석 점유와 관련된 여러 작업 (좌석 상태 변경, 사용자 정보 저장 등)을 하나의 논리적인 단위로 묶어 처리하는 방법입니다. 트랜잭션은 작업의 성공 또는 실패를 보장하여 데이터의 일관성을 유지합니다.
* **낙관적 락 (Optimistic Locking):** 좌석 데이터에 버전 정보를 추가하고, 데이터 변경 시 버전을 비교하여 충돌을 감지하는 방법입니다. 충돌 발생 시 트랜잭션을 재시도하거나 적절한 예외 처리를 수행합니다.
* **비관적 락 (Pessimistic Locking):** 좌석 데이터에 접근하는 동안 다른 스레드가 접근하지 못하도록 막는 방법입니다. 락을 획득한 스레드만 데이터 변경이 가능하며, 다른 스레드는 락이 해제될 때까지 대기합니다.

**산출물 및 실행 방법**

**텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명**

Index.html 실행시 웹 브라우저를 통해 시작

script.js 동적 웹사이트를 위한 스크립트 자바스크립트 파일

styles.css 좌석 배치, 색상 꾸미기 위한 CSS 파일