

PHP로 만드는 서버와 백엔드

《GPT와 함께 만드는 데이터 웹사이트 — 데이터베이스부터 분석 대시보드까지》

2 / 5

웹페이지의 뒤에서 데이터와 명령을 처리하는 방법

완성 교재 · 원리 설명 · 도식 · 단계별 코드 · 데이터 기록 사이트 실습

이 책을 읽는 방법

이 책의 목표는 PHP 코드를 외우는 것이 아니라 요청을 따라가며 읽는 능력을 만드는 것이다. 날짜, 항목, 숫자값, 메모를 입력하는 작은 사이트를 생각하자. 브라우저는 입력창을 보여 주지만, 저장 여부를 판단하고 데이터베이스에 기록하는 일은 서버가 맡는다. PHP는 그 서버에서 실행되는 프로그램을 작성하는 언어다.

각 장은 무엇인가, 왜 필요한가, 작동 원리, 코드, 잘못했을 때의 문제 순서로 진행한다. 코드 뒤의 여섯 가지 설명은 실행 위치, 입력, 출력, 전달 대상, 수정 가능한 부분, 지켜야 할 부분이다. 예제를 읽을 때 먼저 이 설명을 보고 코드를 따라가도 좋다.

짧은 코드는 원리를 보여 주는 독립 예제다. 서로 다른 장의 코드를 한 파일에 모두 붙이지 않는다. 마지막 실습의 파일들은 함께 작동하는 완성된 한 프로젝트다. 복사 실습은 그 파일들을 기준으로 한다. 환경은 PHP 8.2 이상, PDO MySQL 드라이버, mbstring 확장, MySQL 8 계열 또는 호환 MariaDB를 전제로 한다. 지원 상태와 설치 방법은 사용하는 호스팅 안내에서 확인한다.

처음에는 1~10장으로 코드를 읽는 법을 익힌다. 11~17장은 입력과 응답, DB 연결을 공부한다. 18~23장은 로그인, 업로드, 오류, 설정을 다룬다. 24장은 직접 실행하는 CRUD 실습이고 25장은 제3권으로 넘어가는 다리다. SQL 설계와 분석 쿼리는 제3권에서 자세히 다룬다.

이 책에서 ‘서버’는 특별한 모양의 컴퓨터를 뜻하지 않는다. 요청을 받아 처리하는 역할을 뜻한다. 내 노트북도 개발 서버가 될 수 있다. 다만 PHP 개발 서버는 연습용이며 실제 인터넷 서비스용으로 쓰지 않는다.

목차

- 1장 PHP와 백엔드의 역할
- 2장 PHP 파일이 실행되는 전체 과정
- 3장 실습 환경과 첫 요청
- 4장 변수와 값의 생명
- 5장 문자열과 숫자
- 6장 배열과 데이터 묶음
- 7장 조건문으로 판단하기
- 8장 반복문으로 표 만들기
- 9장 함수로 작업 나누기
- 10장 include require와 파일 분리
- 11장 HTML과 PHP 함께 쓰기
- 12장 Form GET POST와 입력 받기
- 13장 서버에서 검증하기
- 14장 JavaScript fetch와 JSON
- 15장 PHP의 JSON 응답과 API
- 16장 MySQL 연결과 PDO
- 17장 CRUD 호출과 Prepared Statement
- 18장 로그인 시스템의 구조
- 19장 Session Cookie와 요청 사이의 기억
- 20장 비밀번호 저장과 검증
- 21장 파일 업로드의 원리
- 22장 에러 예외 로그와 GPT 질문법
- 23장 config와 개발 서비스 설정
- 24장 데이터 기록 사이트 완성 실습
- 25장 제3권으로 이어지는 복습
- 부록 코드 읽기 사전과 공식 참고 자료

1장 PHP와 백엔드의 역할

무엇인가

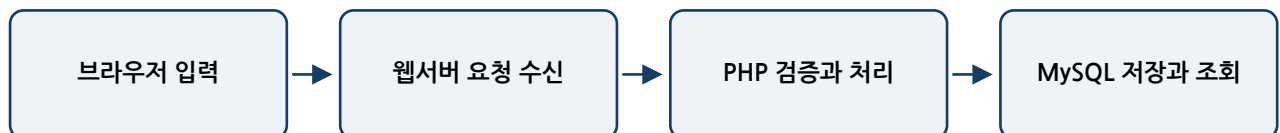
PHP는 서버에서 실행할 프로그램을 쓰는 언어다. 백엔드란 웹페이지 뒤에서 요청을 처리하고 규칙을 적용하는 영역이다. PHP 자체가 데이터베이스는 아니다. PHP는 DB에 명령을 보내고 결과를 받아 웹 응답을 만든다.

왜 필요한가

메모를 HTML 표에 적어 놓으면 작성자의 파일에만 존재한다. 여러 사용자가 브라우저에서 기록을 추가하려면 입력을 받아 저장하는 공통 창구가 필요하다. 브라우저에 DB 비밀번호와 저장 권한을 주면 사용자가 그 정보를 읽고 악용할 수 있다. 서버가 중간에서 입력, 권한, 저장을 통제해야 한다.

작동 원리

브라우저는 화면과 상호작용을 맡는다. 웹서버는 요청 URL을 받아 파일이나 프로그램으로 연결한다. PHP는 검증과 처리, MySQL은 데이터 보관을 맡는다. 서버에서 실행한다는 것은 PHP 언어의 절대적 제약이라기보다 이 웹 구조에서 PHP가 배치되는 위치다. PHP는 명령줄에서도 실행할 수 있다.



기술	주요 실행 위치	우리 사이트에서의 일
HTML	브라우저 해석	입력창과 표의 구조
CSS	브라우저	간격과 색과 반응형 배치
JavaScript	브라우저 또는 서버 환경	이 책에서는 버튼 반응과 fetch
PHP	이 책에서는 서버	검증과 DB 호출과 응답
SQL	DB 엔진	데이터 저장 조회 수정 삭제

코드에서 어떻게 보이는가

코드 1 · php

```

<?php
$siteName = '데이터 기록장';
echo '<h1>' . $siteName . '</h1>';
  
```

실행 위치 서버의 PHP 런타임

입력 코드 안의 고정 문자열

출력 h1 HTML 문자열

다음 단계 웹서버를 통해 브라우저

수정해도 되는 부분 사이트 이름

함부로 수정하면 안 되는 부분 외부 입력을 그대로 HTML에 합치는 방식으로 확장하지 않는다

PHP 태그는 여기부터 PHP 명령을 읽으라는 표시다. echo는 응답 본문에 문자열을 쓴다. 브라우저가 받는 것은 h1 태그이며 PHP 변수는 받지 않는다. 위의 값은 고정 문자열이라 안전한 예제다. 사용자 입력에는 11장의 출력 처리가 필요하다.

잘못하면 어떤 문제가 생기는가

HTML 파일에 PHP 문장을 적기만 해서는 실행되지 않는 환경이 많다. PHP를 지원하지 않는 정적 호스팅에 서도 실행되지 않는다. GPT가 작성한 파일의 확장자만 보고 판단하지 말고 호스팅에 PHP 런타임이 있는지 확인한다.

GPT에게 이렇게 시켜보자

“기록 저장 사이트에서 브라우저, 웹서버, PHP, MySQL이 각각 무엇을 하는지 날짜 입력 한 건을 예로 설명해 줘. 실행 위치를 섞지 말고 요청과 응답을 구분해 줘.”

에러가 났다면

호스팅이 PHP를 지원하는지 확인한다. 파일이 .php인지 확인한다. file://로 열지 않고 서버 URL로 접속했는지 확인한다. PHP 코드가 그대로 보이면 실행 설정을 점검하고 비밀이 들어 있는 파일의 공개를 중단한다.

2장 PHP 파일이 실행되는 전체 과정

무엇인가

HTTP 요청은 브라우저가 서버에 보내는 작업 요청이다. 응답은 서버가 돌려주는 상태 코드, 헤더, 본문이다. URL의 경로는 요청의 목적지를 가리킨다. 파일 경로와 URL 경로는 항상 같은 것이 아니다.

왜 필요한가

“저장 버튼이 안 된다”는 말만으로는 실패한 곳을 찾기 어렵다. 요청이 아예 안 나갔는지, PHP가 오류를 냈는지, DB가 거절했는지 구분할 수 있어야 한다.

작동 원리

index.php 요청을 받은 웹서버는 PHP 실행기로 처리를 넘긴다. PHP는 위에서 아래로 실행하면서 필요한 파일을 읽고 DB를 호출한다. 응답 본문을 만들면 웹서버가 이를 보낸다. 브라우저는 HTML을 해석한 뒤 CSS와 JavaScript 파일을 추가로 요청할 수 있다.



DB를 호출하는 명령은 자동으로 실행되지 않는다. 코드를 실행하는 흐름이 그 줄에 도달해야 한다. 각 HTTP 요청마다 PHP가 다시 실행되는 것이 기본 모델이다. 한 요청에서 만든 일반 변수는 다음 요청에서 자동으로 이어지지 않는다. OPcache가 코드를 캐시해도 사용자별 일반 변수 저장소가 되지는 않는다.

코드에서 어떻게 보이는가

코드 2 · php

```

<?php
header('Content-Type: text/plain; charset=utf-8');
echo "첫 번째 처리\n";
echo "두 번째 처리\n";
  
```

실행 위치 서버

입력 이 파일을 향한 HTTP 요청

출력 텍스트 두 줄과 Content-Type 헤더

다음 단계 브라우저가 텍스트로 표시

수정해도 되는 부분 출력 문구

함부로 수정하면 안 되는 부분 header 호출은 본문 출력 전에 둔다

header는 응답의 성격을 설명한다. text/plain은 HTML 해석 대신 일반 글로 읽으라는 뜻이다. 실제 출력 버퍼 설정에 따라 늦은 헤더가 우연히 동작할 수 있지만 그 동작에 의존하지 않는다.

잘못하면 어떤 문제가 생기는가

PHP 파일 맨 앞의 공백이나 UTF-8 BOM이 출력되면 header나 session_start 앞에 응답이 시작될 수 있다. 순수 PHP 파일은 끝의 닫는 태그를 생략하면 뜻하지 않은 뒤쪽 공백 출력을 줄일 수 있다. sleep으로 처리를 오래 멈추면 사용자는 저장 실패처럼 느낄 수 있다.

GPT에게 이렇게 시켜보자

“이 PHP 파일의 처리 순서를 번호로 설명해 줘. header, 입력 읽기, DB 호출, echo를 구분하고 실제 브라우저에 전달되는 결과를 보여 줘.”

에러가 났다면

개발자 도구 Network에서 요청과 상태 코드를 본다. Response 탭에서 실제 본문을 본다. 404면 경로부터 확인한다. 500이면 서버 로그와 PHP 로그를 확인한다. 200이어도 본문이 기대와 다르면 출력 코드를 확인한다.

3장 실습 환경과 첫 요청

무엇인가

개발 환경은 코드를 고치고 결과를 확인하는 장소다. XAMPP 같은 묶음은 웹서버, PHP, DB를 함께 제공할 수 있다. 이 책은 PHP 내장 서버로 실행하는 방법을 기준으로 설명한다. 이미 XAMPP를 쓴다면 Apache의 문서 루트를 프로젝트의 public 폴더에 맞추는 방식으로 대체할 수 있다.

왜 필요한가

컴퓨터 파일을 더블 클릭하면 브라우저는 파일을 읽는다. PHP 실행 요청은 하지 않는다. localhost URL로 접속해야 개발 서버가 PHP를 실행한다. FTP는 파일을 옮기는 도구이며 PHP 실행기나 DB 생성기가 아니다.

작동 원리

프로젝트 폴더 안에 공개 파일과 비공개 파일을 나눈다. 공개 루트인 public만 웹에서 접근하게 한다. app의 설정과 도우미 파일은 PHP가 내부 경로로 읽는다. public만 공개해도 파일 시스템 권한과 서버 설정은 별도로 점검해야 한다.

파일 위치	웹 직접 접근	용도
project/public/index.php	가능	화면 진입점
project/public/api/records.php	가능	JSON 조회 진입점
project/app/config.php	불가하도록 배치	설정 읽기
project/schema.sql	불가하도록 배치	최초 DB 준비

코드에서 어떻게 보이는가

코드 3 · sh

```
php -v
php -m
php -S 127.0.0.1:8000 -t public
```

실행 위치 터미널과 PHP 실행기

입력 프로젝트 폴더에서 입력한 명령

출력 버전 확장 목록 개발 서버 로그

다음 단계 브라우저가 `http://127.0.0.1:8000` 에 접속

수정해도 되는 부분 사용하지 않는 포트 번호

함부로 수정하면 안 되는 부분 실제 서비스에 내장 서버를 노출하지 않는다

첫 줄은 버전을, 둘째 줄은 확장을 확인한다. 목록에서 PDO, pdo_mysql, mbstring을 찾는다. 셋째 줄은 public을 공개 루트로 지정한다. Windows에서 php 명령을 못 찾으면 설치 경로의 php.exe를 사용하거나 PATH를 설정한다. XAMPP 사용자는 설치된 PHP 실행 파일의 위치를 확인한다.

잘못하면 어떤 문제가 생기는가

Apache와 내장 서버가 같은 포트를 쓰면 충돌한다. localhost는 내 컴퓨터를 뜻하므로 스마트폰에서 같은 주소를 쓰면 스마트폰 자신을 찾는다. 이 책 실습은 내 컴퓨터에서 먼저 끝낸다. 공유 호스팅에서 public 루트를 조정할 수 없다면 비공개 app은 웹루트 밖으로 옮기고 require 경로를 함께 바꿔야 한다.

GPT에게 이렇게 시켜보자

“내 환경은 Windows와 XAMPP다. PHP 버전과 pdo_mysql, mbstring 확인부터 public만 공개하는 방법까지 설명해 줘. 내 호스팅의 폴더 이름을 추측하지 말고 필요한 정보부터 알려 줘.”

에러가 났다면

터미널 현재 폴더를 확인한다. public 폴더와 index.php가 있는지 본다. php 실행 파일 위치를 확인한다. 포트 충돌 메시지를 본다. 화면 문제보다 먼저 가장 짧은 echo 파일이 실행되는지 확인한다.

4장 변수와 값의 생명

무엇인가

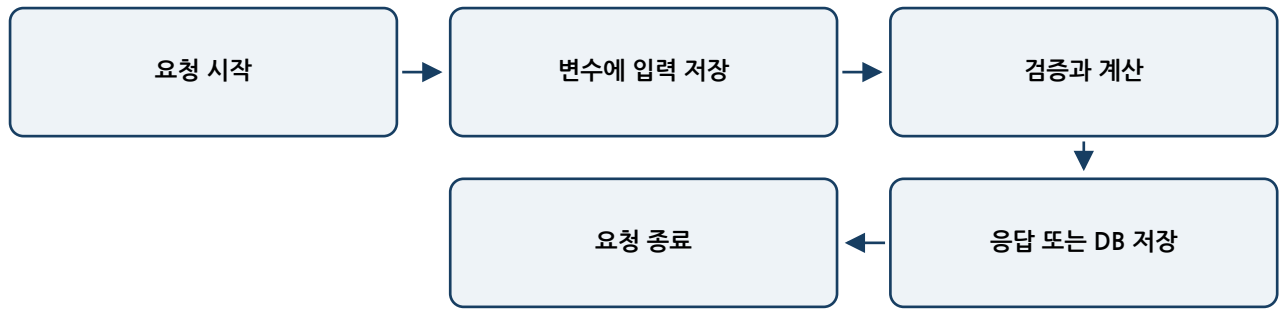
변수는 값에 붙이는 이름이다. PHP 변수는 \$로 시작한다. 이름은 대소문자를 구분한다. \$value와 \$Value는 다른 변수다. =는 값을 대입하는 기호다.

왜 필요한가

날짜, 숫자값, 메모를 여러 줄에서 활용하려면 이름이 필요하다. 변수는 작은 상자에 비유할 수 있지만 정확하게는 실행 중 값에 접근하는 이름이라고 이해하면 된다. 이름을 잘 지으면 코드의 목적이 보인다.

작동 원리

오른쪽 값을 계산한 뒤 왼쪽 이름에 연결한다. 다음 줄이 실행되면 이미 지정한 값을 읽을 수 있다. 새 값을 대입하면 기존 값 대신 새 값을 읽는다. DB에 저장하기 전의 변수는 영구 기록이 아니다.



코드에서 어떻게 보이는가

코드 4 · php

```

<?php
$item = '걸음 수';
$value = 20000;
$nextValue = $value + 1000;
echo $item . ' : ' . $nextValue;

```

실행 위치 서버

입력 고정 항목명과 숫자

출력 걸음 수: 21000 문자열

다음 단계 브라우저

수정해도 되는 부분 항목명과 숫자와 의미가 같은 변수명

함부로 수정하면 안 되는 부분 이름 변경 시 사용하는 모든 위치를 같이 바꾼다

`$value`의 20000을 읽어 1000을 더한 값이 `$nextValue`에 들어간다. 마침표는 문자열 연결이다. PHP의 기본 변수는 선언할 때 타입을 반드시 적지 않아도 된다. 그렇다고 문자열과 배열을 구분하지 않고 받아도 된다는 뜻은 아니다.

잘못하면 어떤 문제가 생기는가

오타가 난 변수는 정의되지 않은 변수 경고를 만들 수 있다. `$value = 10`을 조건에 쓰면 비교가 아니라 대입이 된다. 다음 새로고침에도 값이 남을 것이라고 생각하면 저장 기능을 잘못 설계한다. 일반 변수, 세션, DB는 수명이 다르다.

보관 방식	대략적인 수명	용도
일반 변수	현재 요청	검증 중인 값
세션	서버 정책과 세션 수명에 따름	로그인 상태
DB 기록	삭제할 때까지 지속	기록 데이터

GPT에게 이렇게 시켜보자

“이 코드의 변수마다 뜻, 값의 타입, 값이 바뀌는 줄을 표시해 줘. 한 요청이 끝난 뒤 어떤 값이 사라지는지도 설명해 줘.”

에러가 났다면

변수명의 대소문자를 비교한다. 처음 대입한 줄이 실제 실행되는지 본다. 오류 줄 직전의 값과 타입을 개발 환경에서 확인한다. 로그인 정보나 비밀번호는 출력하지 않는다.

5장 문자열과 숫자

무엇인가

문자열은 글자의 묶음이고 정수는 소수점 없는 수, 실수는 소수점 계산을 위한 수다. HTTP Form 입력은 숫자 입력창에서 보내도 PHP에서 보통 문자열로 받는다.

왜 필요한가

'10'이라는 글과 10이라는 수가 비슷하게 보여도 처리 목적이 다르다. 날짜 '2026-10-04'를 빼기 계산식처럼 다루면 안 된다. 메모는 문자열이고 측정값은 숫자로 해석할 규칙이 필요하다.

작동 원리

따옴표 안은 문자열이다. 작은따옴표는 보통 글을 그대로 담고 큰따옴표는 변수나 \n 같은 표현을 해석한다. 값 연결에는 .를, 덧셈에는 +를 쓴다. 계산 가능한 입력인지 검사한 후 변환한다. 소수 실수는 이진 표현 오차가 있으므로 돈 같은 정확한 십진 데이터는 별도 규칙이 필요하다.

코드에서 어떻게 보이는가

코드 5 · php

```
<?php
$raw = '12.5';
if (!is_numeric($raw)) {
    exit('숫자로 입력해 주세요.');
```

```
}
```

```
$value = (float) $raw;
```

```
$total = $value * 2;
```

```
echo '합계: ' . $total;
```

실행 위치 서버

입력 고정 문자열 12.5

출력 합계: 25 또는 오류 문구

다음 단계 브라우저

수정해도 되는 부분 변수와 안내문

함부로 수정하면 안 되는 부분 검증을 없애고 모든 문자열을 강제로 숫자로 바꾸지 않는다

is_numeric은 숫자로 해석할 수 있는지 검사한다. 지수 표현도 허용할 수 있어 우리 기록장의 십진 입력 규칙에는 더 좁은 검증을 사용한다. (float)는 숫자형으로 바꾸라는 요청이다. 'abc'를 무턱대고 변환하면 의도하지 않은 값이 나올 수 있다.

잘못하면 어떤 문제가 생기는가

(0.1 + 0.2)가 수학적으로 기대한 소수 표현과 완전히 같지 않을 수 있다. 본 프로젝트는 DECIMAL(12,2)에 넣을 값을 문자열 상태로 검증하여 바인딩한다. 3자리 소수는 반올림해 숨기지 않고 거절한다. 저장 단위와 소수 자릿수는 사용 목적에 맞게 결정한다.

입력	우리 프로젝트 판단	이유
12.50	허용	소수 2자리
-7	허용	음수 기록 허용
1,000	거절	쉼표 규칙 없음
1e3	거절	평범한 십진 형식만 허용
12.345	거절	DB보다 세밀한 값

GPT에게 이렇게 시켜보자

“숫자 입력을 먼저 문자열로 검사하고 DECIMAL(12,2)에 맞게 저장해 줘. 소수 2자리, 음수 허용, 쉼표와 지수 표기는 거절하고 오류 이유를 설명해 줘.”

에러가 났다면

실제 입력 문자열을 확인한다. 빈 문자열, 공백, 쉼표를 구분한다. DB 열의 범위와 소수 자릿수를 본다. 표시용 숫자 형식과 저장용 값이 섞이지 않았는지 본다.

6장 배열과 데이터 묶음

무엇인가

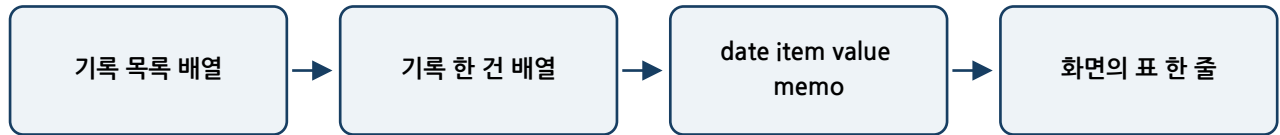
배열은 여러 값을 하나의 이름 아래 모으는 구조다. PHP 배열은 순서가 있는 키와 값의 묶음이다. 숫자 키를 쓰면 목록처럼, 문자열 키를 쓰면 이름 붙인 기록처럼 사용할 수 있다.

왜 필요한가

날짜, 항목, 값, 메모가 한 기록에 속한다는 관계를 표현한다. DB에서 여러 행을 받으면 바깥 배열은 기록 목록, 안쪽 배열은 기록 한 건이 된다.

작동 원리

대괄호로 배열을 만들고 =>로 키와 값을 연결한다. \$record['item']은 record 안의 item 값을 읽는다. 목록의 첫 키는 보통 0이지만 삭제나 직접 키 지정 후에는 연속 숫자가 아닐 수 있다.



코드에서 어떻게 보이는가

코드 6 · php

```

<?php
$record = [
    'date' => '2026-10-04',
    'item' => '걸음 수',
    'value' => '20000.00',
    'memo' => '현장 근무',
];
echo $record['item'];
  
```

실행 위치 서버

입력 한 건의 고정 기록

출력 걸음 수 문자열

다음 단계 브라우저 또는 다음 함수

수정해도 되는 부분 고정 기록 내용

함부로 수정하면 안 되는 부분 키 이름은 읽는 코드와 같게 유지한다

배열은 값을 묶는 구조이고 JSON은 그 값을 전송하기 위한 표현 형식이다. PHP 배열을 그대로 네트워크에 보낼 수는 없으므로 json_encode로 글 형식으로 바꾼다. 숫자 키가 연속되지 않은 배열은 JSON 객체로 표현될 수 있다. 목록 응답이 필요하면 array_values로 키를 정리할 수 있다.

잘못하면 어떤 문제가 생기는가

존재하지 않는 키를 바로 읽으면 경고가 발생한다. ??는 키가 없거나 값이 null일 때 기본값을 준다. 하지만 ??만으로 문자열 검증은 되지 않는다. item[]=x처럼 보내면 값이 배열일 수 있으므로 is_string 검사가 필요하다.

GPT에게 이렇게 시켜보자

“기록 목록과 기록 한 건을 PHP 배열로 표현하고, 어느 대괄호가 목록을 선택하고 어느 대괄호가 항목을 선택하는지 설명해 줘.”

에러가 났다면

키 이름을 확인한다. 값이 배열인지 문자열인지 확인한다. DB fetch 모드가 연관 배열인지 본다. 누락 값을 무조건 빈 글로 바꾸지 말고 필수 항목인지 판단한다.

7장 조건문으로 판단하기

무엇인가

조건문은 조건 결과에 따라 실행할 코드를 고르는 구조다. if는 조건이 참일 때, else는 그렇지 않을 때 실행한다. ===는 값과 타입을 함께 비교한다.

왜 필요한가

저장 요청인지 조회 요청인지, 항목이 비었는지, 로그인했는지에 따라 다른 행동이 필요하다. 조건문은 프로그램의 규칙을 보이게 만든다.

작동 원리

조건을 계산해 참 또는 거짓을 얻고 해당 가지로 이동한다. 일찍 거절해야 할 요청은 먼저 검사하고 exit나 return으로 끝내면 본 처리의 흐름이 단순해진다. 비교와 대입은 역할이 다르다.

코드에서 어떻게 보이는가

코드 7 · php

```
<?php
$method = $_SERVER['REQUEST_METHOD'] ?? '';
if ($method !== 'POST') {
    header('Allow: POST');
    http_response_code(405);
    exit('POST 요청만 허용합니다.');
```

실행 위치 서버의 저장 진입점

입력 HTTP 요청 메서드

출력 405 응답 또는 처리 시작 문구

다음 단계 다음 입력 검증 단계

수정해도 되는 부분 안내 문구

함부로 수정하면 안 되는 부분 메서드 검사와 조기 종료를 임의로 제거하지 않는다

!==는 값 또는 타입이 다르면 참이다. 위에서 GET을 받으면 exit에서 끝나 마지막 echo는 실행되지 않는다. 메서드 검사는 작업 종류를 제한하지만 로그인과 권한을 증명하지는 않는다. POST라고 해서 안전한 요청이라는 뜻도 아니다.

잘못하면 어떤 문제가 생기는가

if (\$value)로 필수 숫자를 검사하면 0이나 '0'을 빈 값처럼 처리할 수 있다. 빈 글인지 검사할 때는 \$value === ''처럼 목적을 분명히 한다. 너무 깊은 중첩은 어느 가지에서 무엇이 실행되는지 읽기 어렵게 만든다.

표현	뜻	주로 쓰는 곳
=	대입	값 보관
===	타입과 값이 같은가	정확한 비교
!==	타입 또는 값이 다른가	거절 조건
&&	두 조건 모두 참	여러 조건 충족
	하나라도 참	오류 조건 뭉기

GPT에게 이렇게 시켜보자

“이 코드에서 0도 유효한 숫자로 받아 줘. 빈 문자열 검사와 참거짓 검사를 구분하고, 실패하면 일찍 반환하는 구조로 고쳐 줘.”

에러가 났다면

비교 기호를 본다. 실제 값의 타입을 확인한다. 조건을 한 개씩 분리해 읽는다. exit 뒤 코드가 실행되지 않는지 확인한다.

8장 반복문으로 표 만들기

무엇인가

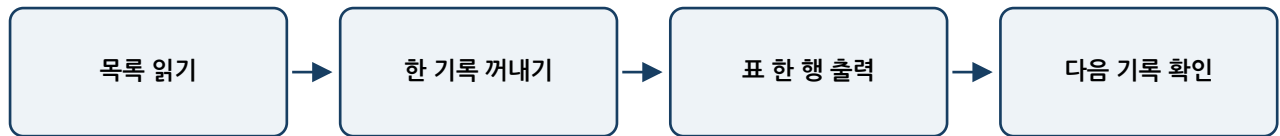
반복문은 여러 항목에 같은 작업을 적용한다. foreach는 배열의 각 값을 하나씩 꺼내는 데 적합하다. for는 정해진 횟수나 숫자 범위를 다룰 때 편하다.

왜 필요한가

기록이 3건에서 300건으로 늘어도 HTML 표 줄을 직접 300번 작성할 수 없다. 데이터 목록을 읽어 한 행을 만드는 규칙을 반복한다.

작동 원리

foreach가 첫 값을 변수에 넣고 본문을 실행한다. 다음 값을 넣어 다시 실행한다. 목록이 끝나면 반복을 빠져 나온다. 배열이 비어 있으면 본문은 한 번도 실행되지 않는다.



코드에서 어떻게 보이는가

코드 8 · php

```

<?php
$values = [10, 20, 30];
$sum = 0;
foreach ($values as $value) {
    $sum += $value;
}
echo $sum;

```

실행 위치 서버

입력 숫자 목록

출력 60

다음 단계 브라우저 또는 합계 표시

수정해도 되는 부분 목록의 숫자

함부로 수정하면 안 되는 부분 합계 초기화는 반복 밖에 둔다

`+=`는 기존 합계에 값을 더해 다시 저장한다. 반복 안에서 `$sum = 0`을 실행하면 매번 합계를 지우게 된다. DB 결과를 HTML로 반복 출력하는 완성 코드는 24장에 있다. 숫자 합계 예제는 원리 이해용이며 돈의 정확한 연산까지 다루지 않는다.

잘못하면 어떤 문제가 생기는가

반복의 종료 조건을 잘못 쓰면 실행 시간이 길어진다. 수십만 행을 한 번에 불러오면 메모리와 응답 시간이 늘어난다. 페이지당 일부 기록을 조회하는 페이지 나누기는 제3권과 이후 실습으로 확장한다. 이 책 프로젝트는 최신 200건만 보여 준다.

GPT에게 이렇게 시켜보자

“DB에서 받은 기록 배열을 `foreach`로 표에 출력해 줘. 배열이 비었을 때는 기록이 없다는 문구를 표시하고 모든 사용자 문자열은 HTML 출력 시 이스케이프해 줘.”

에러가 났다면

배열인지 확인한다. 초기화 위치를 확인한다. 비어 있는 결과인지 본다. 화면에 같은 행이 반복되면 반복 변수 사용 위치를 본다.

9장 함수로 작업 나누기

무엇인가

함수는 이름을 붙인 작업 묶음이다. 매개변수는 함수가 받는 값이고 return은 호출한 곳에 돌려주는 값이다. 함수 정의만 해서는 작업이 실행되지 않고 호출해야 실행된다.

왜 필요한가

여러 화면에서 같은 입력 검사나 출력 처리가 필요하다. 복사해서 여러 군데 고치면 한 곳은 수정하고 다른 곳은 빠뜨리기 쉽다. 함수 한 곳에 규칙을 모으면 유지 보수가 쉬워진다.

작동 원리

호출자가 값을 전달하면 함수 안의 매개변수로 받아 처리한다. return이 실행되면 함수는 끝나고 결과를 호출한 줄로 돌려준다. echo와 return은 다르다. echo는 응답에 쓰고 return은 계산 결과를 돌려준다.

코드에서 어떻게 보이는가

코드 9 · php

```
<?php
function doubleValue(float $value): float
{
    return $value * 2;
}
$result = doubleValue(12.5);
echo $result;
```

실행 위치 서버

입력 함수에 전달한 12.5

출력 반환값 25와 화면 문자열

다음 단계 호출자의 result 변수와 브라우저

수정해도 되는 부분 계산 규칙과 함수명

함부로 수정하면 안 되는 부분 타입 선언과 호출부를 함께 맞춘다

함수 안의 \$value는 그 함수의 지역 변수다. 밖에 있는 같은 이름의 변수를 자동으로 공유하지 않는다. float 와 : float는 입력과 반환의 기대 타입을 적은 것이다. 엄격한 타입 처리 여부는 호출 파일의 strict_types 설정 등의 영향을 받는다. 선언만으로 외부 문자열의 입력 검증이 완성되지는 않는다.

잘못하면 어떤 문제가 생기는가

반환값을 받지 않으면 결과를 놓친다. JSON 응답을 만드는 함수 안에서 디버깅 echo를 넣으면 응답 본문이 망가진다. 함수가 한 번에 입력, SQL, 화면, 파일 삭제까지 모두 맡으면 재사용과 검증이 어려워진다.

GPT에게 이렇게 시켜보자

“이 파일에서 날짜 검증과 HTML 출력 처리를 각각 함수로 분리해 줘. 입력과 반환값을 설명하고 함수 안에서 불필요한 echo는 하지 말아 줘.”

에러가 났다면

함수가 정의된 파일이 로드되는지 본다. 함수 이름과 인자 개수를 본다. 반환값을 사용하는지 확인한다. TypeError라면 전달한 실제 타입을 확인한다.

10장 include require와 파일 분리

무엇인가

include와 require는 다른 PHP 파일을 현재 실행 흐름에 읽어 오는 문법이다. require는 필요한 파일이 없으면 실행을 계속할 수 없게 한다. include의 실패는 경고를 내고 이어질 수 있다. _once는 이미 읽은 파일을 중복 로드하지 않게 한다.

왜 필요한가

DB 연결 설정, 공통 검증 함수, 화면을 분리하면 한 가지를 고칠 때 다른 기능을 덜 건드린다. 특히 DB 비밀번호를 여러 파일에 복사하면 변경 시 누락되거나 공개 파일에 섞이기 쉽다.

작동 원리

웹 URL이 아니라 서버 파일 경로를 지정한다. __DIR__는 현재 파일이 있는 디렉터리다. 경로 기준을 고정하면 터미널의 현재 폴더나 다른 파일에서 호출한 위치가 달라도 예측하기 쉽다.

구문	파일 누락	적합한 대상
include	경고 후 계속될 수 있음	없어도 처리 가능한 보조 부분
require	실행 실패	필수 설정과 함수
require_once	필수이며 중복 로드 방지	공통 초기화 파일

코드에서 어떻게 보이는가

코드 10 · php

```
<?php
require_once __DIR__ . '/../app/bootstrap.php';
echo '공통 설정을 읽었습니다.';
```

실행 위치 서버의 public 파일

입력 bootstrap.php 파일

출력 공통 초기화 후 문자열

다음 단계 현재 요청의 후속 처리

수정해도 되는 부분 실제 배치에 맞춘 경로

함부로 수정하면 안 되는 부분 사용자 입력을 require 경로에 연결하지 않는다

../는 상위 폴더로 이동한다. 24장에서는 public/index.php에서 app/bootstrap.php를 읽는다. api 폴더처럼 한 단계 더 아래에 있는 파일은 ../../app/bootstrap.php가 필요하다. 함수 정의 파일을 중복 include하면 같은 함수를 다시 정의하는 오류가 날 수 있다.

잘못하면 어떤 문제가 생기는가

require \$_GET['page']처럼 외부 입력으로 파일을 선택하면 의도하지 않은 파일을 읽거나 코드가 실행될 수 있다. 파일 분리는 보안과 같지 않다. app이 공개 루트 안에 놓이면 서버 설정 실수 때 노출 위험이 남는다.

GPT에게 이렇게 시켜보자

“화면, DB 연결, 검증 함수를 기능별로 나눠 줘. public을 웹루트로 쓰고 app을 그 밖에 두며 __DIR__ 기준의 require_once를 사용해 줘. URL과 파일 경로를 비교표로 설명해 줘.”

에러가 났다면

오류에 표시된 실제 파일 경로를 본다. ../의 개수를 확인한다. 파일명 대소문자를 확인한다. 서버 계정의 읽기 권한을 확인한다.

11장 HTML과 PHP 함께 쓰기

무엇인가

PHP 파일에는 일반 HTML과 PHP 처리 구간이 함께 있을 수 있다. PHP는 처리 결과를 HTML 사이에 넣어 응답을 만든다. 브라우저는 최종 HTML을 받는다.

왜 필요한가

기록 한 건을 DB에서 읽어 표 안에 넣고 싶다. HTML 구조를 유지한 채 필요한 값만 PHP로 채우면 화면을 읽기 쉽다.

작동 원리

PHP 바깥 HTML은 본문으로 전달되고 PHP 구간은 실행 결과만 전달된다. <?=는 값을 출력하는 짧은 echo 문법이다. 사용자 문자열은 HTML 문맥에 맞게 이스케이프한다. htmlspecialchars는 HTML의 특수 문자를 문자로 보이게 바꾼다. 저장할 때 무조건 HTML 변환하지 않고 출력 경계에서 처리한다.

코드에서 어떻게 보이는가

코드 11 · php

```
<?php
$item = '<script>예시</script>';
```

```
function e(string $text): string {
    return htmlspecialchars(
        $text, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8'
    );
}
?>
<p>항목: <?= e($item) ?></p>
```

실행 위치 PHP는 서버 HTML은 브라우저

입력 항목 문자열

출력 스크립트 태그가 글로 보이는 p 요소

다음 단계 브라우저 HTML 해석기

수정해도 되는 부분 문구와 HTML 구조

함부로 수정하면 안 되는 부분 사용자 값을 출력할 때 e 호출을 유지한다

ENT_QUOTES는 따옴표도 처리한다. ENT_SUBSTITUTE는 잘못된 문자 시퀀스를 대체한다. 이 함수는 HTML 본문과 따옴표로 감싼 속성 값에 사용한다. JavaScript 코드, CSS, URL 문맥에 그대로 적용하는 만능 보안 함수는 아니다. 사용자 값을 script 문자열에 직접 끼우지 않고 JSON 또는 data 전달 구조를 사용한다.

잘못하면 어떤 문제가 생기는가

외부 문자열을 그대로 echo하면 XSS가 발생할 수 있다. XSS는 공격자 입력을 브라우저가 코드로 실행하는 문제다. SQL Injection과 실행 위치가 다르다. Prepared Statement는 HTML 출력의 XSS를 막지 않는다. fetch 결과를 DOM에 넣을 때도 innerHTML 대신 textContent를 쓰는 이유가 같다.



GPT에게 이렇게 시켜보자

“사용자 항목명과 메모를 HTML 표에 안전하게 출력해 줘. 저장값은 원본으로 두고 출력할 때 htmlspecialchars를 적용해 줘. JavaScript의 innerHTML 사용도 점검해 줘.”

에러가 났다면

원본 데이터와 출력 문자열을 구분한다. 이중 이스케이프가 있는지 본다. 모든 동적 출력 위치를 확인한다. 한글이 깨지면 파일 저장 형식, HTML charset, DB charset을 함께 확인한다.

12장 Form GET POST와 입력 받기

무엇인가

Form은 입력값을 이름과 값으로 묶어 서버에 보내는 HTML 구조다. GET은 보통 조회 조건을 URL 쿼리에 담고 POST는 보통 작업 입력을 요청 본문에 담는다. \$_GET과 \$_POST는 PHP가 정리해 주는 입력 배열이다.

왜 필요한가

HTML 입력창만 있으면 서버는 값을 알 수 없다. name이 전달할 키를 정하고 action이 목적지, method가 전송 방식을 정한다. id는 화면 요소 식별에 쓰며 name의 전송 역할을 대신하지 않는다.

작동 원리

GET Form의 item 값은 ?item=...처럼 쿼리에 실린다. POST Form의 일반 입력은 본문으로 보내져 \$_POST에 들어간다. URL 쿼리가 있는 POST 요청에서도 \$_GET은 채워질 수 있다. \$_POST가 모든 POST 본문을 자동으로 해석하는 것은 아니다. JSON 본문은 15장에서 직접 읽는다.

구분	GET	POST
대표 목적	검색 조회	등록과 변경
Form 입력 위치	URL 쿼리	요청 본문
PHP 읽기	\$_GET	일반 Form은 \$_POST
비밀 보호	HTTPS가 필요	역시 HTTPS가 필요
데이터 변경	하지 않도록 설계	권한과 CSRF 검사를 함께 함

코드에서 어떻게 보이는가

코드 12 · html

```
<form action="receive.php" method="post">
  <label>항목 <input name="item" required></label>
  <button type="submit">보내기</button>
</form>
```

실행 위치 브라우저

입력 사용자가 쓴 항목

출력 POST Form 요청

다음 단계 receive.php

수정해도 되는 부분 라벨과 버튼 문구

함부로 수정하면 안 되는 부분 name과 서버가 읽는 키를 함께 맞춘다

코드 13 · php

```
<?php
$raw = $_POST['item'] ?? null;
if (!is_string($raw)) {
    http_response_code(422);
    exit('항목을 글로 입력해 주세요.');
```

실행 위치 서버의 receive.php

입력 Form의 item 필드

출력 검증 실패 또는 안전한 텍스트

다음 단계 브라우저

수정해도 되는 부분 안내 문구

함부로 수정하면 안 되는 부분 배열 여부 검사와 HTML 출력 처리를 유지한다

잘못하면 어떤 문제가 생기는가

name이 빠지거나 disabled인 입력은 제출되지 않는다. required는 브라우저 편의 검사다. 사용자는 브라우저를 거치지 않고 요청을 만들 수 있다. POST는 암호화 방식이 아니며 서버나 중간 시스템의 로그 정책에 따라 본문이 기록될 수도 있다. 비밀번호는 GET에 넣지 않고 HTTPS를 사용한다.

GPT에게 이렇게 시켜보자

“날짜, 항목, 숫자값, 메모를 POST Form으로 보내 줘. 필드마다 name과 PHP의 배열 키를 표로 맞춰 주고 서버에서도 누락과 배열 입력을 검증해 줘.”

에러가 났다면

Network의 Form Data를 본다. method와 action을 확인한다. name을 확인한다. JSON을 보냈는데 \$_POST를 읽고 있지 않은지 확인한다.

13장 서버에서 검증하기

무엇인가

검증은 입력이 업무 규칙에 맞는지 판단하는 것이다. 정리는 trim처럼 표현을 다듬는 일이고, 이스케이프는 출력이나 명령의 문맥에서 값이 코드로 해석되지 않게 하는 일이다. 세 가지는 목적이 다르다.

왜 필요한가

브라우저에서 날짜 선택과 숫자 제한을 해도 외부 요청은 우회할 수 있다. 존재하지 않는 날짜, 지나치게 긴 메모, 배열 형태의 숫자 입력을 서버가 거절해야 DB와 화면의 일관성을 지킬 수 있다.

작동 원리

먼저 요청 방식, 다음 값의 타입, 필수 여부, 길이, 형식, 범위를 검사한다. 오류가 있으면 저장 전에 응답한다. 입력 검증을 통과해도 DB 제약 조건이 마지막으로 데이터를 거절할 수 있다. CSRF 검사는 타 사이트가 사용자의 로그인 쿠키를 이용해 변경 요청을 유도하는 문제를 줄인다. 이것 역시 값 검증이나 로그인과 다른 검사다.



코드에서 어떻게 보이는가

코드 14 · php

```

<?php
$raw = $_POST['record_date'] ?? null;
$valid = false;
if (is_string($raw)) {
    $date = DateTimeImmutable::createFromFormat('!Y-m-d', $raw);
    $valid = $date !== false
        && $date->format('Y-m-d') === $raw;
}
if (!$valid) {
    http_response_code(422);
    exit('실제 날짜를 YYYY-MM-DD로 입력해 주세요.');
```

```

}
echo '날짜 검증 완료';

```

실행 위치 서버

입력 record_date 문자열 또는 누락값

출력 422 오류 또는 완료 문구

다음 단계 다음 저장 처리

수정해도 되는 부분 안내 문구

함부로 수정하면 안 되는 부분 만든 날짜를 다시 원본과 비교하는 검사를 유지한다

DateTime이 잘못된 날짜를 다음 달로 보정할 수 있으므로 원본과 다시 비교한다. !는 지정하지 않은 시간 요소를 초기화한다. 2026-02-30처럼 달력에 없는 날짜를 저장하려는 요청을 막는다. 최종 프로젝트에서는 DB에 맞춰 연도 1000~9999도 제한한다.

잘못하면 어떤 문제가 생기는가

검증 전에 INSERT를 하면 잘못된 기록이 이미 저장될 수 있다. htmlspecialchars만 호출하고 안전하다고 생각하면 숫자 범위나 날짜는 검증되지 않는다. CSRF 토큰이 있어도 사용자 권한 검사가 없으면 남의 기록을 수정할 수 있다.

검사	찾는 문제	대체할 수 없는 것
타입과 형식 검증	배열 날짜 숫자 오류	SQL 안전 처리
Prepared Statement	값이 SQL로 해석되는 문제	HTML XSS 방어
HTML 이스케이프	화면에서 코드 실행	사용자 권한
CSRF 토큰	타 사이트 유도 변경 요청	로그인과 소유권 검사

GPT에게 이렇게 시켜보자

“서버 검증을 타입, 필수값, 길이, 실제 날짜, 숫자 범위 순으로 작성해 줘. 실패하면 DB에 쓰지 말고 사용자에게 항목별 이유를 알려 줘.”

에러가 났다면

서버에 도착한 키를 확인한다. 검증 규칙과 DB 열을 비교한다. 정상 입력 한 건부터 시험한다. 오류가 나도 저장되는지 확인한다.

14장 JavaScript fetch와 JSON

무엇인가

fetch는 JavaScript에서 HTTP 요청을 보내는 기능이다. JSON은 배열, 객체, 문자열, 숫자 등을 글 형식으로 표현하는 교환 형식이다. JSON은 실행 코드가 아니며 PHP나 JavaScript만의 형식도 아니다.

왜 필요한가

Form은 보통 새 페이지로 이동하지만 fetch는 현재 화면에서 데이터를 받아 일부만 바꿀 수 있다. 저장된 기록을 가져와 차트를 만드는 이후 단계에서도 같은 구조를 사용한다.

작동 원리

JavaScript가 요청을 보내면 서버가 응답한다. await fetch는 응답 객체를 받고 await response.json은 본문을 읽어 JavaScript 값으로 해석한다. await는 전체 서버가 끝날 때까지 컴퓨터를 멈추는 문법이 아니라 비동기 결과를 기다리는 코드 표현이다. HTTP 404나 500만으로 fetch가 자동 reject되는 것은 아니므로 response.ok도 확인한다.



코드에서 어떻게 보이는가

코드 15 · javascript

```

async function loadRecords() {
  const response = await fetch('api/records.php');
  const data = await response.json();
  if (!response.ok) {
    throw new Error(data.error ?? '조회 실패');
  }
  document.querySelector('#status').textContent =
    받은 기록: ${data.records.length}건 ;
}
loadRecords().catch(error => {
  document.querySelector('#status').textContent = error.message;
});

```

실행 위치 브라우저 JavaScript

입력 api/records.php URL와 status 요소

출력 조회 상태 문구

다음 단계 화면 DOM

수정해도 되는 부분 상태 문구와 표시 형태

함부로 수정하면 안 되는 부분 응답 상태 검사와 catch와 textContent를 유지한다

이 예제는 24장의 API와 함께 사용할 수 있다. 먼저 HTML에 id가 status인 요소가 있어야 한다. 오류 응답이 HTML이면 response.json 자체가 실패한다. catch가 이 경우도 잡는다. JSON의 필드 이름 records는 서버의 이름과 맞아야 한다.

잘못하면 어떤 문제가 생기는가

file://에서 연 HTML은 서버의 API 경로를 제대로 요청하지 못한다. 다른 출처에 요청하면 CORS 정책이 관련된다. 이 책 프로젝트는 같은 서버와 같은 출처를 사용한다. CORS는 로그인 권한을 대신하지 않고, 서버 간 통신에도 같은 제한으로 적용되는 것은 아니다.

GPT에게 이렇게 시켜보자

“같은 출처의 PHP API를 fetch로 조회해 줘. HTTP 오류와 네트워크 오류와 JSON 해석 오류를 구분해 설명하고 사용자 문자열은 textContent로 표시해 줘.”

에러가 났다면

Console 오류를 본다. Network 요청 URL과 상태를 본다. Response가 JSON인지 본다. records 키가 있는지 본다. DOM 요소가 실제 존재하는지 확인한다.

15장 PHP의 JSON 응답과 API

무엇인가

API endpoint는 외부 프로그램이 특정 기능을 요청하는 주소다. api/records.php는 기록 목록 기능의 진입점이다. REST는 자원을 중심으로 URL과 HTTP 메서드를 조직하는 설계 방식이다. 파일 이름에 api를 붙이고 자동으로 REST가 되지는 않는다.

왜 필요한가

같은 데이터로 HTML 표, JavaScript 차트, 다른 앱을 만들려면 화면과 데이터 응답을 분리하는 편이 좋다. 서버는 JSON으로 결과를 주고 화면은 그 결과를 어떻게 보여 줄지 결정한다.

작동 원리

PHP 배열을 json_encode로 JSON 문자열로 바꾼다. Content-Type을 application/json으로 선언한다. HTML과 JSON을 같은 응답에 섞지 않는다. JSON POST 입력은 php://input에서 원문을 읽고 json_decode한다. \$_POST는 JSON을 자동으로 담지 않는다.

코드에서 어떻게 보이는가

코드 16 · php

```
<?php
header('Content-Type: application/json; charset=utf-8');
echo json_encode(
    ['ok' => true, 'records' => []],
    JSON_UNESCAPED_UNICODE | JSON_THROW_ON_ERROR
);
```

실행 위치 서버의 API 파일

입력 여기서는 빈 목록

출력 JSON 응답

다음 단계 브라우저 JavaScript의 response.json

수정해도 되는 부분 추가 결과 필드

함부로 수정하면 안 되는 부분 Content-Type과 JSON만 출력하는 구조를 유지한다

코드 17 · php

```
<?php
try {
    $body = json_decode(
```

```

        file_get_contents('php://input'),
        true, 512, JSON_THROW_ON_ERROR
    );
    if (!is_array($body) || !is_string($body['item'] ?? null)) {
        throw new InvalidArgumentException('item 필요');
    }
    $item = trim($body['item']);
} catch (JsonException | InvalidArgumentException $error) {
    http_response_code(400);
    exit('잘못된 JSON 입력');
}

```

실행 위치 서버의 JSON 입력 처리 구간

입력 JSON 요청 본문

출력 item 변수 또는 400 텍스트 오류

다음 단계 이후 검증 함수

수정해도 되는 부분 업무 필드 이름

함부로 수정하면 안 되는 부분 이 코드는 입력 해석 예제다 JSON API로 쓸 때 오류도 JSON으로 통일한다

두 번째 코드는 JSON 입력 해석만 보여 준다. 완성된 저장 endpoint는 아니다. 실제 JSON 저장 API에는 메서드, Content-Type, 본문 크기, 로그인, CSRF 또는 인증 방식, 길이 검사를 추가해야 한다. 본 실습은 저장을 Form POST로, JSON은 조회로 제한한다.

자원 중심 예시	메서드	의미
/records	GET	목록 조회
/records	POST	새 기록 등록
/records/17	GET	17번 조회
/records/17	PATCH	17번 일부 수정
/records/17	DELETE	17번 삭제

위 URL은 설계 예시다. 우리 PHP 파일 프로젝트에서 이 URL을 곧바로 쓸 수 있는 것은 아니다. 웹서버 rewrite나 라우터가 연결해야 한다. 일반 HTML Form은 GET과 POST를 사용하며 PATCH와 DELETE는 fetch 또는 서버에서 정의한 별도 액션으로 구현한다.

잘못하면 어떤 문제가 생기는가

앞부분의 Warning, HTML, BOM이 JSON을 망가뜨린다. HTTP 200으로 모든 실패를 보내면 클라이언트가 성공과 실패를 구분하기 어렵다. JSON_THROW_ON_ERROR는 변환 실패를 예외로 알리므로 무시하지 않고 처리해야 한다.

GPT에게 이렇게 시켜보자

“PHP 조회 endpoint를 만들어 줘. 성공은 records 배열, 실패는 error 문자열로 반환하고 상태 코드와 Content-Type을 맞춰 줘. HTML과 디버깅 출력을 섞지 말아 줘.”

에러가 났다면

응답 첫 글자가 { 또는 [인지 본다. 서버 경고가 섞였는지 본다. Content-Type을 본다. UTF-8 입력인지 확인한다. 요청 JSON과 Form을 구분한다.

16장 MySQL 연결과 PDO

무엇인가

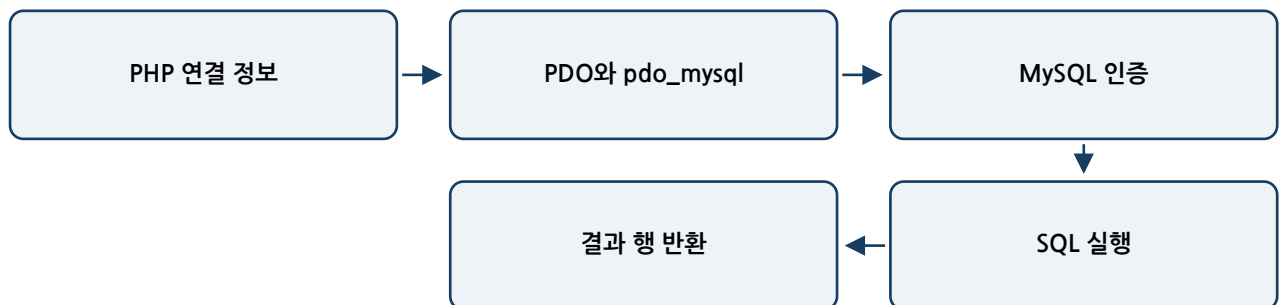
MySQL은 데이터를 보관하고 SQL을 실행하는 DB 시스템이다. PDO는 PHP가 DB 드라이버를 통해 명령을 보내고 결과를 받는 공통 인터페이스다. 연결 정보에는 서버 주소, DB 이름, 계정, 비밀번호, 문자 설정이 포함된다.

왜 필요한가

PHP 변수는 요청 후 사라진다. 계속 남은 기록은 DB에 저장해야 한다. PDO는 SQL 자체를 대신 설계하지 않지만 연결, 실행, 결과 읽기 방식을 제공한다. DB 종류를 바꿔도 SQL과 기능 차이는 따로 수정해야 한다.

작동 원리

PHP가 PDO 객체를 만들면 지정한 주소의 DB 서버에 접속하고 인증한다. 접속이 성공해야 prepare, execute, fetch를 할 수 있다. DB를 만든다는 것과 DB에 연결한다는 것은 다르다. PDO 객체를 만들었다고 records 테이블이 자동으로 생기지 않는다.



코드에서 어떻게 보이는가

코드 18 · php

```

<?php
$pdo = new PDO(
    'mysql:host=127.0.0.1;dbname=record_book;charset=utf8mb4',
    'record_app',
    '여기에_실제_비밀번호',
    [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES => false,
    ]
);
  
```

실행 위치 서버

입력 DSN 계정 비밀번호

출력 PDO 연결 객체 또는 예외

다음 단계 SQL 처리 코드

수정해도 되는 부분 자신의 환경에 맞는 DB 연결 정보

함부로 수정하면 안 되는 부분 비밀번호는 공개 파일에 두지 않고 실습에서는 비공개 설정으로 분리한다

DSN은 연결 대상 설명 문자열이다. utf8mb4는 한글과 다양한 문자를 저장할 수 있게 하는 연결 문자 설정이다. DB와 테이블의 문자 설정도 맞춘다. FETCH_ASSOC는 열 이름으로 읽는 배열을 준다.

ERRMODE_EXCEPTION은 DB 오류를 예외로 처리한다. EMULATE_PREPARES=false는 MySQL 드라이버에서 서버 측 준비 방식을 요청한다.

잘못하면 어떤 문제가 생기는가

could not find driver는 SQL 문법 문제가 아니라 드라이버 문제다. Access denied는 계정, 비밀번호, 접속 출처 권한을 확인한다. Connection refused는 서버 실행, 주소, 포트를 본다. Unknown database는 DB 이름이나 생성 상태를 본다. Linux에서 localhost와 127.0.0.1은 소켓과 TCP 연결 차이가 생길 수 있으므로 호스팅 안내를 따른다.

GPT에게 이렇게 시켜보자

“PDO 연결 코드를 설정 파일과 분리해 줘. pdo_mysql이 필요하다는 점과 각 옵션의 역할을 설명하고 접속 실패의 내부 정보는 사용자에게 노출하지 말아 줘.”

에러가 났다면

드라이버 설치를 확인한다. DB 서버가 켜졌는지 본다. DB 이름과 계정을 본다. 호스팅에서 지정한 주소와 포트를 본다. DB 비밀번호를 GPT 질문에 포함하지 않는다.

17장 CRUD 호출과 Prepared Statement

무엇인가

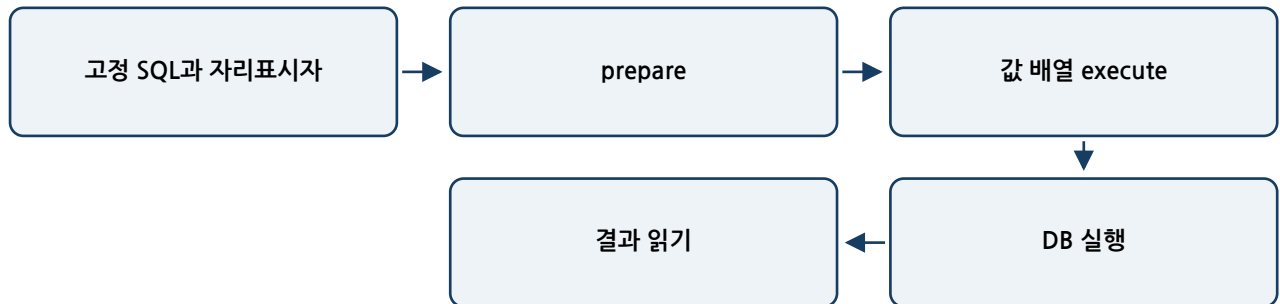
CRUD는 Create, Read, Update, Delete의 줄임말이다. DB에서는 보통 INSERT, SELECT, UPDATE, DELETE가 대응한다. Prepared Statement는 SQL 명령 구조와 입력값을 분리해 실행하는 방식이다.

왜 필요한가

사용자 입력을 SQL 문자열에 붙이면 그 값이 SQL 명령의 일부로 해석될 수 있다. 이를 SQL Injection이라 한다. 안전한 구조는 SQL에는 자리표시자를 적고 값은 execute에 따로 전달하는 것이다.

작동 원리

prepare는 명령 구조를 준비하고 execute는 값을 연결해 실행한다. SELECT 뒤에는 fetch 또는 fetchAll로 결과를 읽는다. INSERT 뒤에는 필요하면 lastInsertId로 생성 번호를 읽는다. UPDATE와 DELETE는 반드시 대상 조건을 분명하게 쓴다.



코드에서 어떻게 보이는가

코드 19 · php

```

<?php
// 16장의 PDO 연결 뒤에서 실행한다.
$id = 17;
$stmt = $pdo->prepare(
    'SELECT id, item FROM records WHERE id = :id'
);
$stmt->execute(['id' => $id]);
$record = $stmt->fetch();
  
```

실행 위치 서버 PHP와 DB 엔진

입력 PDO 객체와 id 17

출력 한 행의 배열 또는 false

다음 단계 화면 처리 또는 JSON 변환

수정해도 되는 부분 고정 조회 열과 필요한 조건

함부로 수정하면 안 되는 부분 외부 id를 SQL 문자열에 직접 붙이지 않는다

:id는 실제 값이 들어갈 자리다. execute의 배열 키는 콜론 없이 id로 쓰고 SQL에서는 :id로 표시한다. 조회 결과가 없으면 fetch는 false일 수 있다. 값이 없다고 무조건 DB 연결 실패는 아니다.

코드 20 · php

```

<?php
// PDO 연결 뒤에서 실행하는 독립 호출 예제다.
$stmt = $pdo->prepare(
    'INSERT INTO records (record_date, item, value, memo)
    VALUES (:record_date, :item, :value, :memo)'
);
$stmt->execute([
    'record_date' => '2026-10-04',
    'item' => '걸음 수',
  
```

```
'value' => '20000.00',
'memo' => '현장 근무',
]);
```

실행 위치 서버와 DB

입력 고정 예시 기록

출력 DB에 새 행

다음 단계 다음 목록 조회

수정해도 되는 부분 예시 데이터

함부로 수정하면 안 되는 부분 실제 입력은 검증 후 전달하고 자리표시자와 키를 일치시킨다

명령	PHP 기본 단계	특히 확인할 것
SELECT	prepare execute fetchAll	조회 권한 결과 수
INSERT	prepare execute	필수 열 값 형식
UPDATE	prepare execute	WHERE id와 소유권
DELETE	prepare execute	대상 확인 CSRF 권한

열 이름과 ASC/DESC 같은 SQL 구조는 값 자리표시자로 바인딩할 수 없다. 사용자 선택으로 정렬을 바꾸려면 허용 목록에서 미리 정한 SQL 조각만 고른다. Prepared Statement가 입력값 바인딩을 안전하게 해도 SQL 구조를 외부 입력으로 만들면 위험이 남는다.

잘못하면 어떤 문제가 생기는가

UPDATE records SET memo=...에 WHERE가 없으면 모든 행이 바뀐다. DELETE에 WHERE가 없으면 모든 행이 사라진다. HTML 이스케이프나 addslashes를 SQL 바인딩 대신 쓰지 않는다. rowCount가 0인 UPDATE는 행이 없거나 같은 값으로 수정한 경우일 수 있어 존재 여부 확인과 구분한다.

GPT에게 이렇게 시켜보자

“PDO의 SELECT INSERT UPDATE DELETE 예제를 작성해 줘. 모든 입력값은 자리표시자로 바인딩하고 UPDATE와 DELETE에는 검증된 id 조건을 넣어 줘. rowCount 0을 해석할 때 주의할 점도 알려 줘.”

에러가 났다면

SQL 열 이름과 테이블 이름을 본다. 자리표시자와 배열 키 수를 맞춘다. 데이터 타입과 DB 제약을 본다. 조회와 변경 대상 id를 확인한다. 운영 DB에서 실험하지 않고 테스트 기록으로 확인한다.

18장 로그인 시스템의 구조

무엇인가

로그인은 사용자가 누구인지 확인하는 인증 과정이다. 권한 검사는 그 사용자가 특정 기록을 읽거나 바꿀 수 있는지 확인하는 과정이다. 로그인 창을 화면에 보여 주는 것만으로 보호가 되지 않는다.

왜 필요한가

개인 기록을 공개하지 않으려면 서버가 모든 접근에서 권한을 판단해야 한다. 화면 메뉴를 숨겨도 API 주소를 직접 호출할 수 있으므로 조회 API와 수정 처리 파일도 확인해야 한다.

작동 원리

아이디로 사용자 한 명을 조회한다. 저장된 비밀번호 해시와 입력 비밀번호를 비교한다. 일치하면 서버 세션에 사용자 번호를 보관한다. 다음 요청은 세션을 통해 로그인 상태를 읽는다. 각 기록의 owner_id가 로그인 사용자와 일치하는지 SQL 조건에도 포함한다.



코드에서 어떻게 보이는가

코드 21 · php

```

<?php
// 세션을 시작한 보호 페이지의 앞부분이다.
$userId = $_SESSION['user_id'] ?? null;
if (!is_int($userId) || $userId < 1) {
    http_response_code(401);
    exit('로그인이 필요합니다.');
```

실행 위치 서버의 보호 진입점

입력 이전 로그인에서 만든 세션

출력 사용자 번호 또는 401 오류

다음 단계 해당 사용자의 기록 조회

수정해도 되는 부분 안내 문구

함부로 수정하면 안 되는 부분 이 검사를 화면에만 넣고 API에서 빼지 않는다

이 코드는 접근 검사 조각이며 세션 시작이나 로그인 처리 전체를 포함하지 않는다. 19장과 20장을 연결해야 한다. 사용자 id를 Cookie나 hidden 입력으로 받아 믿으면 다른 사용자로 가장할 수 있다. 서버가 확인한 값만 신뢰한다.

잘못하면 어떤 문제가 생기는가

로그인 검사 뒤 `SELECT WHERE id=:id`만 사용하면 다른 사용자의 번호를 바꾸어 조회할 수 있다. 소유권이 있는 사이트는 `WHERE id=:id AND owner_id=:user_id` 같은 조건이 필요하다. 로그인 실패는 계정 존재 여부를 지나치게 드러내지 않고 비슷한 메시지로 처리한다. 반복 시도 제한, HTTPS, 세션 만료와 재인증은 실제 서비스에서 함께 설계한다.

GPT에게 이렇게 시켜보자

“로그인 인증과 기록 소유권 검사를 분리해 줘. 화면, 조회 API, 등록, 수정, 삭제의 모든 진입점에서 세션을 검사하고 기록 조회는 `owner_id` 조건을 포함해 줘.”

에러가 났다면

세션 시작 위치를 본다. 로그인 성공 시 저장한 키와 읽는 키를 비교한다. API도 보호되는지 직접 확인한다. 다른 사용자 id로 접근하는 시험은 자신이 만든 테스트 계정 두 개로 한다.

19장 Session Cookie와 요청 사이의 기억

무엇인가

Cookie는 브라우저가 보관하고 조건에 맞는 요청에 함께 보내는 작은 값이다. Session은 서버가 요청 사이에 사용자 상태를 이어 가는 구조다. 일반적인 PHP 세션은 브라우저 Cookie에 세션 식별자를 넣고 실제 상태는 서버에 둔다.

왜 필요한가

로그인 후 다음 페이지 요청에서도 사용자 정보를 알아야 한다. 매 요청마다 비밀번호를 다시 보낼 수는 없다. 기록 저장의 CSRF 토큰도 세션에 보관해 제출한 값과 비교한다.

작동 원리

`session_start`는 세션 식별자를 읽어 서버 상태를 불러오거나 새로 만든다. 로그인 성공 직후 `session_regenerate_id(true)`로 식별자를 바꾼다. 이는 로그인 전 식별자를 고정해 악용하는 위험을 줄인다. `HttpOnly`는 JavaScript의 Cookie 읽기를 제한하고 `Secure`는 HTTPS 전송을 요구한다. `SameSite`는 다른 사이트에서 유발한 요청의 Cookie 전송을 제한한다.

항목	보관 위치	사용자 변조 가능성	예시
일반 Cookie 값	브라우저	가능	화면 테마
세션 식별자 Cookie	브라우저	탈취되면 위험	PHPSESSID

항목	보관 위치	사용자 변조 가능성	예시
세션의 user_id	서버	브라우저가 직접 수정 못함	로그인 사용자
DB 기록	DB	접근 권한으로 통제	측정값

코드에서 어떻게 보이는가

코드 22 · php

```
<?php
ini_set('session.use_strict_mode', '1');
session_set_cookie_params([
    'httponly' => true,
    'secure' => true,
    'samesite' => 'Lax',
    'path' => '/',
]);
session_start();
```

실행 위치 HTTPS 서비스의 서버

입력 요청의 세션 Cookie

출력 세션 상태와 필요 시 Set-Cookie

다음 단계 로그인 처리와 접근 검사

수정해도 되는 부분 Cookie 경로와 정책을 환경에 맞게 설정

함부로 수정하면 안 되는 부분 HTTP 개발에서 secure true를 그대로 쓰면 Cookie 전달이 안 될 수 있다

HTTPS 운영 예제다. 24장의 localhost 실습은 HTTP라 secure=false를 개발 모드에서만 사용한다. reverse proxy 환경에서는 HTTPS 판단을 단순 사용자 헤더에 맡기지 않고 서버 설정으로 확인한다. SameSite=Lax 만으로 모든 CSRF를 막았다고 판단하지 않고 변경 요청에는 CSRF 토큰도 사용한다.

잘못하면 어떤 문제가 생기는가

session_start 전에 출력하면 헤더 문제가 생긴다. 세션의 존재만으로 인증된 사용자는 아니다. CSRF 토큰용 세션도 생길 수 있다. session_destroy는 서버 세션을 끝내지만 Cookie 삭제와 현재 배열 비우기도 로그아웃 처리에서 고려해야 한다. 로그아웃도 POST와 CSRF 검사로 처리한다.

GPT에게 이렇게 시켜보자

“개발 HTTP와 운영 HTTPS의 세션 Cookie 설정을 나눠 줘. 로그인 성공 시 세션 id를 갱신하고 로그아웃은 POST와 CSRF 검사 후 세션 상태와 Cookie를 정리해 줘.”

에러가 났다면

Cookie가 생성됐는지 본다. 다음 요청에 전달되는지 본다. Secure와 HTTPS를 비교한다. path와 도메인을 확인한다. 서버 세션 저장 경로 권한을 본다.

20장 비밀번호 저장과 검증

무엇인가

비밀번호 해시는 비밀번호를 검증하기 위해 만든 단방향 결과다. 암호화처럼 나중에 원문을 되찾는 저장 방식이 아니다. PHP의 password_hash와 password_verify를 사용한다.

왜 필요한가

DB가 유출되면 평문 비밀번호는 바로 읽힌다. 사용자가 다른 서비스에서 같은 비밀번호를 쓸 수도 있어 피해가 커진다. 일반 SHA-256이나 MD5 한 번 적용도 비밀번호 저장용 설계로 충분하지 않다.

작동 원리

가입 시 password_hash로 해시를 만들고 DB에는 그 해시만 저장한다. 로그인 시 사용자가 입력한 원문과 저장된 해시를 password_verify로 검사한다. 해시에는 알고리즘과 검사에 필요한 정보가 포함된다. 기본 함수는 필요한 salt를 관리하므로 임의의 고정 salt를 덧붙이지 않는다.

코드에서 어떻게 보이는가

코드 23 · php

```
<?php
$password = '학습용-예시-비밀번호';
$hash = password_hash($password, PASSWORD_DEFAULT);
$matched = password_verify($password, $hash);
echo $matched ? '일치' : '불일치';
```

실행 위치 서버의 독립 원리 예제

입력 고정 예시 비밀번호

출력 해시 변수와 일치 결과

다음 단계 가입에서는 해시를 DB에 로그인에서는 세션 처리로

수정해도 되는 부분 학습용 예시 문자열

함부로 수정하면 안 되는 부분 실제 원문을 echo 로그 DB 평문으로 남기지 않는다

PASSWORD_DEFAULT는 기본 알고리즘 선택이다. 향후 기본 알고리즘 변화에 대비해 해시 열은 VARCHAR(255)처럼 충분한 길이를 둔다. 매번 새 해시를 만들고 문자열끼리 비교하면 같은 비밀번호도 해시가 달라질 수 있다. password_verify를 사용해야 한다. 기존 해시 갱신 필요 여부는 password_needs_rehash로 확인할 수 있다.

잘못하면 어떤 문제가 생기는가

원문 비밀번호를 오류 로그나 GPT 질문에 붙이면 해시 저장을 해도 다른 경로로 유출된다. 비밀번호를 임의로 trim하면 사용자가 정한 값을 바꿀 수 있다. 가입과 로그인에서 같은 정책을 일관되게 적용한다. 실제 계정 기능에는 최대 입력 길이, 알고리즘 제한, 반복 시도 제한, 복구 절차도 필요하다. 이 장은 원리 예제로 실제 서비스 로그인 전체 구현은 아니다.

GPT에게 이렇게 시켜보자

“가입에서는 password_hash, 로그인에서는 password_verify를 써 줘. 해시는 VARCHAR(255)에 저장하고 원문 비밀번호는 출력하거나 로그에 남기지 말아 줘. 가입과 로그인 정책을 일치시켜 줘.”

에러가 났다면

해시가 DB에서 잘리지 않았는지 본다. 새 해시 문자열끼리 비교하고 있지 않은지 본다. 가입과 로그인에서 공백 처리 규칙이 같은지 확인한다. 로그에 비밀번호가 없는지 확인한다.

21장 파일 업로드의 원리

무엇인가

업로드는 브라우저의 파일 내용을 서버로 보내는 작업이다. PHP는 임시 위치에 받은 파일과 그 상태를 \$_FILES로 제공한다. 업로드 파일은 사용자 입력이므로 이름과 확장자, 브라우저가 알려 준 MIME 값을 그대로 믿지 않는다.

왜 필요한가

사진이나 자료를 기록에 첨부할 수 있다. 그러나 파일은 큰 용량을 차지하고 서버에서 코드로 실행되거나 악성 내용을 담을 수 있어 일반 글 입력보다 검사할 일이 많다.

작동 원리

Form에 multipart/form-data를 지정한다. PHP 설정의 크기 제한이 먼저 적용되고 임시 파일이 만들어진다. 서버는 업로드 오류, 크기, 파일 내용 기반 MIME을 검사한다. 안전한 새 이름으로 웹루트 밖에 저장하고 원래 이름은 표시용 정보로 별도 관리한다. 실제 다운로드 기능에도 권한 검사와 적절한 응답 헤더가 필요하다.

코드에서 어떻게 보이는가

코드 24 · html

```
<form action="upload.php" method="post"
  enctype="multipart/form-data">
  <input type="file" name="photo" accept="image/jpeg,image/png">
  <button>보내기</button>
</form>
```

실행 위치 브라우저

입력 선택한 이미지

출력 multipart POST 요청

다음 단계 upload.php의 FILES 배열

수정해도 되는 부분 라벨과 accept 안내

함부로 수정하면 안 되는 부분 accept는 서버 검증을 대신하지 않는다

코드 25 · php

```
<?php
// 검사 흐름을 보여 주는 예제이며 공개 업로드 서비스가 아니다.
$file = $_FILES['photo'] ?? null;
if (!is_array($file)
    || ($file['error'] ?? null) !== UPLOAD_ERR_OK
    || !is_string($file['tmp_name'] ?? null)
    || !is_int($file['size'] ?? null)
    || $file['size'] > 2 * 1024 * 1024) {
    http_response_code(422);
    exit('업로드 상태와 2MB 제한을 확인하세요.');
```

```
}
$mime = (new finfo(FILEINFO_MIME_TYPE))->file($file['tmp_name']);
$extensions = ['image/jpeg' => 'jpg', 'image/png' => 'png'];
if (!isset($extensions[$mime])) {
    http_response_code(422);
    exit('JPEG 또는 PNG만 허용합니다.');
```

```
}
$name = bin2hex(random_bytes(16)) . '.' . $extensions[$mime];
$target = __DIR__ . '/../storage/uploads/' . $name;
if (!move_uploaded_file($file['tmp_name'], $target)) {
    http_response_code(500);
    exit('파일 보관 실패');
```

```
}
echo '업로드 완료';
```

실행 위치 서버의 public/upload.php

입력 photo 업로드와 임시 경로

출력 검사 결과 또는 새 파일

다음 단계 웹루트 밖 storage/uploads

수정해도 되는 부분 크기 제한과 허용 이미지 종류

함부로 수정하면 안 되는 부분 원본 파일명을 저장 경로에 쓰지 않고 검사와 이동 결과를 확인한다

먼저 storage/uploads를 만들고 서버 계정에 쓰기 권한을 준다. fileinfo 확장이 필요하다. 이 예제는 핵심 업로드 흐름만 보여 주며 로그인, CSRF, 이미지 재인코딩, 해상도 제한, 악성 파일 검사, 저장 용량 관리, 다운로드 처리까지 완성한 것은 아니다. 공개 업로드에 그대로 배포하지 않는다. 파일을 웹루트 밖에 두는 이유는 이미지로 보이는 파일을 서버가 실행할 기회를 줄이기 위해서다.

잘못하면 어떤 문제가 생기는가

post_max_size를 넘으면 \$_POST와 \$_FILES가 비어 있을 수 있다. upload_max_filesize, post_max_size, 웹서버 요청 제한을 함께 확인한다. 임시 경로는 요청 이후 영구 보관을 보장하지 않는다. move_uploaded_file이 실패했는데 완료를 표시하면 사용자는 파일이 저장됐다고 오해한다.

GPT에게 이렇게 시켜보자

“이미지 업로드를 로그인과 CSRF 검사 뒤에 처리해 줘. 오류 코드, 크기, 내용 기반 MIME을 검증하고 임의 이름으로 웹루트 밖에 저장해 줘. 공개 서비스에 추가할 검사도 설명해 줘.”

에러가 났다면

enctype와 name을 본다. 업로드 오류 코드를 본다. PHP와 웹서버 크기 제한을 본다. 저장 폴더와 쓰기 권한을 본다. 이동 함수 반환값을 본다.

22장 에러 예외 로그와 GPT 질문법

무엇인가

문법 오류는 코드를 읽을 수 없는 상태다. 실행 중 오류는 특정 값이나 환경 때문에 작업이 실패한 상태다. 논리 오류는 실행은 되지만 결과가 잘못된 상태다. 예외는 실패를 호출 흐름에 전달하는 방식이며 try/catch로 처리할 수 있다.

왜 필요한가

모든 실패를 “안 됨”으로 부르면 해결이 늦어진다. 오류가 브라우저, PHP, DB 중 어디에서 났는지 좁히고 재현 조건을 전달하면 GPT가 정확한 수정을 제안할 수 있다.

작동 원리

try의 작업에서 예외가 발생하면 맞는 catch로 이동한다. 로그는 운영자가 실패 원인을 확인하는 기록이다. 사용자 화면에는 간단한 안내와 문의용 오류 번호를 주고 내부 로그에는 오류 번호와 종류, 위치를 남긴다. 입력 전문, Cookie, 토큰, 비밀번호는 로그에 넣지 않는다.

코드에서 어떻게 보이는가

코드 26 · php

```
<?php
try {
    throw new RuntimeException('학습용 실패');
} catch (Throwable $error) {
    $errorId = bin2hex(random_bytes(4));
    error_log(sprintf(
        '[%s] %s %s:%d', $errorId, get_class($error),
        $error->getFile(), $error->getLine()
    ));
    http_response_code(500);
    echo '처리 실패. 오류 번호: ' . $errorId;
}
```

실행 위치 서버

입력 학습용 예외

출력 로그 한 줄과 500 안내

다음 단계 운영자는 로그 사용자는 오류 번호

수정해도 되는 부분 안내 문구

함부로 수정하면 안 되는 부분 내부 경로와 비밀번호를 화면에 보여 주지 않는다

Throwable은 Error와 Exception 계열을 다룬다. 같은 파일의 문법 오류는 실행 전에 발생하므로 그 파일의 try/catch로 모든 문법 오류를 처리할 수는 없다. php -i 파일명으로 실행 전 문법 검사를 한다. 기본 error_log 위치는 PHP와 웹서버 환경마다 달라 확인이 필요하다. 개발에서는 상세 오류를 확인하고 운영에서는 log_errors를 켜고 display_errors는 끈다.

증상	첫 확인 위치	주요 단서
화면 404	URL과 웹서버 경로	파일 위치 라우팅
화면 500	PHP 웹서버 로그	예외 문법 설정
JSON 해석 실패	Network Response	HTML 경고가 섞였는가
DB 연결 실패	PHP 로그	드라이버 인증 주소
저장 후 표가 안 바뀜	DB와 SELECT 조건	LIMIT 정렬 필터
버튼 반응 없음	Console Network	JS 오류 요청 미발생

GPT에게 이렇게 시켜보자

“환경은 PHP 8.x, MySQL, 개발 서버다. POST /save.php에서 날짜 2026-10-04, 항목 테스트, 값 12.50을 보냈다. 기대는 저장 후 목록 이동인데 HTTP 500이다. 오류 번호와 비밀을 지운 로그, 관련 파일, Network의 상태와 응답을 아래에 붙인다. 원인 후보를 좁히고 최소 수정안을 설명해 줘. DB 비밀번호와 Cookie는 제거했다.”

질문에는 사용한 버전, 재현 순서, 기대 결과, 실제 결과, 오류 문구, 관련 코드와 파일 경로를 포함한다. 화면 전체 이미지 하나보다 오류 줄 주변 코드와 텍스트 로그가 유용할 수 있다. GPT가 추측한 해결을 적용한 뒤 같은 입력으로 다시 확인한다.

에러가 났다면

재현 입력 한 건을 정한다. 상태 코드와 응답을 본다. 같은 시각의 로그를 찾는다. 비밀을 지운다. 최소 관련 코드를 함께 전달한다. 수정 후 정상과 실패 입력을 둘 다 시험한다.

23장 config와 개발 서비스 설정

무엇인가

config는 실행 환경별 설정을 모은 곳이다. 환경 변수는 실행 환경에서 프로그램에 전달하는 설정값이다. 같은 코드라도 개발 DB와 실제 서비스 DB를 다르게 연결할 수 있다.

왜 필요한가

DB 비밀번호를 코드 곳곳에 적으면 교체가 어렵다. 개발 화면에 필요한 상세 오류가 운영 화면에 그대로 뜨면 경로와 DB 정보가 노출될 수 있다. 설정과 처리 코드를 분리해야 환경 변경이 작업 규칙 변경으로 번지지 않는다.

작동 원리

공통 설정 파일 한 곳에서 환경 변수 또는 비공개 로컬 설정을 읽는다. 공개 루트 밖에 두고 Git이나 파일 공유에 포함하지 않는다. .env 파일은 PHP가 저절로 읽는 기능이 아니다. 별도 로더나 서버 환경 설정이 필요하다. 우리 실습은 환경 변수 또는 app/config.local.php를 사용한다.

코드에서 어떻게 보이는가

코드 27 · php

```
<?php
return [
    'db_host' => getenv('DB_HOST') ?: '127.0.0.1',
    'db_name' => getenv('DB_NAME') ?: 'record_book',
    'db_user' => getenv('DB_USER') ?: 'record_app',
    'db_password' => getenv('DB_PASSWORD') ?: '',
];
```

실행 위치 서버의 비공개 설정 파일

입력 서버 환경 변수와 기본값

출력 설정 배열

다음 단계 PDO 연결 생성 코드

수정해도 되는 부분 자신의 DB 이름 계정 주소

함부로 수정하면 안 되는 부분 실제 비밀번호를 공개 코드와 배포 ZIP 예제에 넣지 않는다

설정 파일이 return한 배열은 require의 반환값으로 받을 수 있다. 개발과 운영의 차이는 단순 비밀번호 차이만이 아니다. 개발용 내장 서버 대신 운영 웹서버를 사용하고 HTTPS, 접근 권한, 로그 보관, 백업, 오류 노출을 점검한다.

설정	개발	실제 서비스
웹서버	내장 서버 가능	운영 웹서버와 PHP 실행 구성
DB	학습 데이터	최소 권한 서비스 계정
오류 화면	상세 확인 가능	상세 숨기고 내부 로그
HTTPS	localhost HTTP 가능	HTTPS 사용
세션 Secure	HTTP 연습은 false	HTTPS 운영은 true
공개 범위	내 컴퓨터	인증과 권한 확인

설정	개발	실제 서비스
데이터 보호	재현용 데이터	백업과 복원 절차

잘못하면 어떤 문제가 생기는가

환경 변수가 비어 기본값으로 연결되면 다른 DB에 쓸 수 있다. config.local.php를 public 안에 두거나 ZIP에 실제 값을 넣지 않는다. DB 계정은 필요한 SELECT INSERT UPDATE DELETE 권한만 주고 root를 웹앱 계정으로 쓰지 않는다. phpinfo는 내부 정보를 많이 보여 주므로 점검 후 공개 파일을 지운다.

GPT에게 이렇게 시켜보자

“비공개 설정 파일과 환경 변수를 사용하는 구조로 바꿔 줘. 개발과 운영의 오류 표시, HTTPS Cookie, DB 계정을 분리하고 실제 비밀이 공유 파일에 들어가지 않게 해 줘.”

에러가 났다면

실행 중인 PHP가 설정을 읽는지 본다. CLI와 웹서버의 php.ini가 같은지 확인한다. 환경 변수 변경 후 서버 재시작 여부를 본다. 설정값의 실제 비밀번호는 화면에 출력하지 않는다.

24장 데이터 기록 사이트 완성 실습

무엇인가

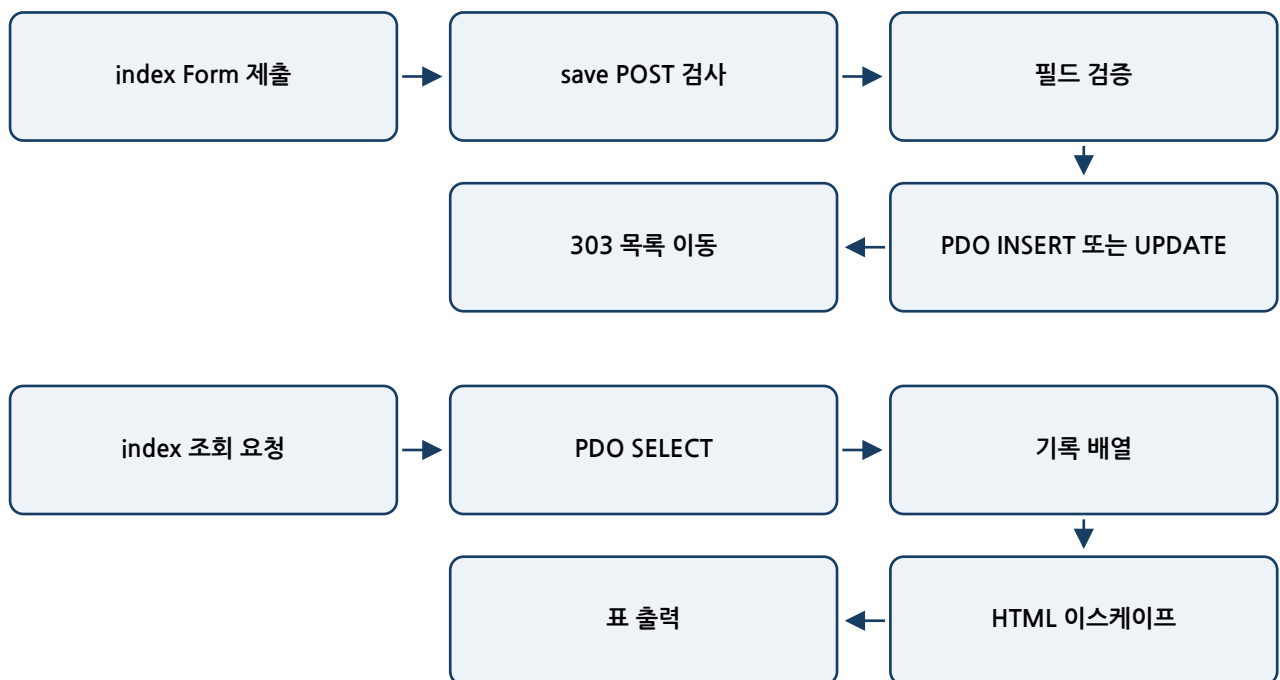
날짜, 항목, 숫자값, 메모를 입력해 DB에 저장하고 표로 조회하는 사이트를 만든다. 등록과 조회 외에 수정과 삭제까지 제공한다. Form으로 변경하고 조회 API로 JSON을 받는다. 같은 검증과 DB 함수가 화면과 API를 지원한다.

왜 필요한가

개별 문법을 이해해도 여러 파일을 어떻게 연결하는지 모르면 직접 사이트를 만들기 어렵다. 이 실습은 입력에서 영구 저장, 조회와 출력까지 이어지는 한 경로를 보여 준다. 로그인 구현은 이 프로젝트에 포함하지 않는다. 개인 컴퓨터 localhost에서만 먼저 실행한다. 공개하려면 18~20장의 원리를 적용해 모든 진입점에 인증과 소유권 검사를 추가해야 한다.

작동 원리

index.php는 입력 Form과 표를 보여 준다. save.php는 POST와 CSRF를 검사하고 필드를 검증한 뒤 INSERT 또는 UPDATE한다. delete.php는 검증된 id를 삭제한다. 성공 후 303 리다이렉트로 index.php에 돌아온다. 이 방식은 POST 후 새로고침 재전송을 줄이지만 더블 클릭이나 네트워크 재시도의 모든 중복 저장을 방지하는 것은 아니다.

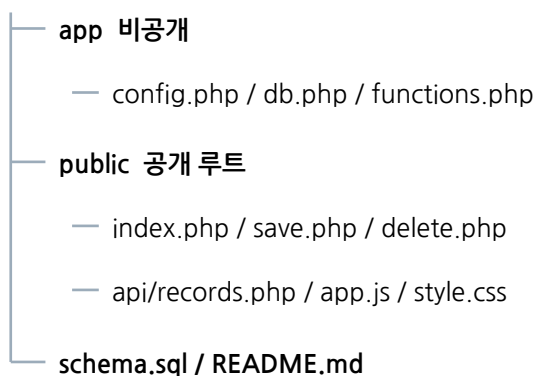


파일	책임	입력과 결과
app/config.php	설정 모으기	환경 값 → 배열
app/config.local.php	개발 비밀값	사용자 편집 설정

파일	책임	입력과 결과
app/bootstrap.php	세션과 오류 처리	요청 → 공통 초기화
app/db.php	PDO 연결	설정 → 연결 객체
app/functions.php	검증과 DB 함수	값 → 결과
public/index.php	Form과 표	GET id → 화면
public/save.php	등록과 수정	POST → DB → 이동
public/delete.php	삭제	POST id → DB → 이동
public/api/records.php	JSON 조회	GET → records 배열
public/app.js	조회 상태 표시	JSON → DOM
schema.sql	최초 테이블 준비	DB 선택 후 실행

폴더를 먼저 준비한다

project



압축 예제 안의 project 폴더를 작업 폴더로 쓴다. 공개 루트는 project/public이다. app과 schema.sql은 public 밖에 있다. 아래 표는 폴더 구조를 보여 준다. config.local.example.php를 복사해 config.local.php로 만들고 실제 설정을 쓴다. 예제 파일은 비밀이 없는 양식이다.

경로	종류	직접 바꿀 부분
project/app/	비공개 처리 폴더	설정과 공통 로직
project/public/	공개 루트	화면과 endpoint
project/public/api/	조회 API 폴더	records.php

경로	종류	직접 바꿀 부분
project/app/config.local.example.php	설정 양식	복사한 뒤 값 입력
project/schema.sql	DB 준비 파일	초기 실습에서 실행
project/README.md	실행 안내	순서 확인

24.1 DB 준비

MySQL 서버를 켜고 관리 도구에서 record_book DB를 만든다. utf8mb4 문자 설정을 사용한다. 관리자 계정으로 DB를 준비한 뒤 앱에는 별도 record_app 계정을 사용한다. 권한은 이 DB의 SELECT INSERT UPDATE DELETE만 준다. 계정의 접속 호스트와 비밀번호 설정 방법은 DB 환경에 따라 달라 관리 도구 안내를 따른다. 공유 호스팅이라면 제공된 DB 이름과 계정을 사용한다.

파일 schema.sql

코드 28 · sql

```
CREATE TABLE IF NOT EXISTS records (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  record_date DATE NOT NULL,
  item VARCHAR(80) NOT NULL,
  value DECIMAL(12,2) NOT NULL,
  memo VARCHAR(1000) NOT NULL DEFAULT '',
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

실행 위치 MySQL 관리 도구의 DB 엔진

입력 선택한 DB와 테이블 정의

출력 records 테이블

다음 단계 PHP의 DB 호출

수정해도 되는 부분 실습 DB 이름과 합의한 열 크기

함부로 수정하면 안 되는 부분 이후 PHP와 검증 규칙도 함께 바꾸고 운영 테이블에 무심코 재실행하지 않는다

id는 한 기록을 식별하는 번호다. record_date는 날짜, item은 항목, value는 소수 2자리 숫자, memo는 메모다. DECIMAL(12,2)는 정수부 최대 10자리와 소수부 2자리이며 부호는 자릿수에 포함하지 않는다. 본 검증은 양수와 음수를 허용하고 범위 안에서 평범한 십진 형식만 받는다. updated_at이나 owner_id는 지금 만들지 않는다. 로그인 확장 시 소유권 열과 쿼리 조건을 함께 설계해야 한다.

24.2 설정 모으기

파일 app/config.local.example.php

코드 29 · php

```
<?php
return [
    'dev' => true,
    'db_host' => '127.0.0.1',
    'db_port' => '3306',
    'db_name' => 'record_book',
    'db_user' => 'record_app',
    'db_password' => 'CHANGE_ME',
];
```

실행 위치 서버의 비공개 설정 파일

입력 복사 후 입력하는 개발 연결 정보

출력 설정 배열

다음 단계 config.php

수정해도 되는 부분 본인의 주소 포트 DB 계정 비밀번호

함부로 수정하면 안 되는 부분 실제 값을 넣은 config.local.php는 공유하지 않는다

위 파일을 config.local.php로 복사한다. db_password 자리에는 본인의 개발 DB 비밀번호를 넣는다. dev=true는 localhost 실습용이다. 운영으로 옮길 때는 false로 바꾸는 것 외에 HTTPS와 로그인, 공개 루트, 백업도 준비해야 한다.

파일 app/config.php

코드 30 · php

```
<?php
$localPath = __DIR__ . '/config.local.php';
$local = is_file($localPath) ? require $localPath : [];
$env = static function (string $key, string $fallback): string {
    $value = getenv($key);
    return $value === false ? $fallback : $value;
};
$appEnv = getenv('APP_ENV');
return [
    'dev' => $appEnv === false
        ? ($local['dev'] ?? true)
        : $appEnv !== 'production',
    'db_host' => $env('DB_HOST', $local['db_host'] ?? '127.0.0.1'),
    'db_port' => $env('DB_PORT', $local['db_port'] ?? '3306'),
    'db_name' => $env('DB_NAME', $local['db_name'] ?? 'record_book'),
    'db_user' => $env('DB_USER', $local['db_user'] ?? 'record_app'),
    'db_password' => $env('DB_PASSWORD', $local['db_password'] ?? ''),
];
```

실행 위치 서버

입력 환경 변수와 비공개 로컬 파일

출력 통합 설정 배열

다음 단계 bootstrap과 db 함수

수정해도 되는 부분 기본 DB 이름과 환경 변수 연결

함부로 수정하면 안 되는 부분 환경 우선순위와 운영 모드 이름을 확인한다

환경 변수 설정이 있다면 그 값을 우선 사용한다. 이 책에서는 복사한 로컬 파일로 시작해도 된다.

APP_ENV=production이면 운영 모드가 되며 display_errors가 꺼지고 세션 Cookie Secure가 켜진다. 다른 APP_ENV 값은 개발 모드로 해석하므로 운영에서 이름을 정확히 설정한다.

24.3 DB 연결과 공통 초기화

파일 app/db.php

코드 31 · php

```
<?php
function db(): PDO
{
    static $pdo = null;
    if ($pdo instanceof PDO) {
        return $pdo;
    }
    $config = require __DIR__ . '/config.php';
    $dsn = 'mysql:host=' . $config['db_host']
        . ';port=' . $config['db_port']
        . ';dbname=' . $config['db_name'] . ';charset=utf8mb4';
    $pdo = new PDO($dsn, $config['db_user'], $config['db_password'], [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES => false,
    ]);
    return $pdo;
}
```

실행 위치 서버 PHP와 DB 연결 과정

입력 비공개 설정

출력 PDO 객체 또는 예외

다음 단계 조회 저장 삭제 함수

수정해도 되는 부분 환경에 맞춘 DSN 구성

함부로 수정하면 안 되는 부분 문자 설정과 예외 옵션과 비공개 비밀번호 구조를 유지한다

db 함수는 같은 요청에서 연결을 다시 만들지 않도록 static 변수로 보관한다. static은 이 예제에서 현재 PHP 요청 중의 함수 호출 사이에 값을 유지하는 용도이며 다음 HTTP 요청의 사용자 상태를 자동으로 보존하는 기능이 아니다.

파일 app/bootstrap.php

코드 32 · php

```
<?php
$config = require __DIR__ . '/config.php';
error_reporting(E_ALL);
ini_set('display_errors', $config['dev'] ? '1' : '0');
ini_set('log_errors', '1');
set_exception_handler(static function (Throwable $error): void {
    $errorId = bin2hex(random_bytes(4));
    $driverCode = $error instanceof PDOException
        ? ($error->errorInfo[1] ?? '') : '';
    error_log(sprintf('[%s] %s code=%s driver=%s %s:%d',
        $errorId, get_class($error), (string) $error->getCode(),
        (string) $driverCode, $error->getFile(), $error->getLine()));
    http_response_code(500);
    $path = parse_url($_SERVER['REQUEST_URI'] ?? '/', PHP_URL_PATH);
    if (is_string($path) && str_starts_with($path, '/api/')) {
        header('Content-Type: application/json; charset=utf-8');
        echo json_encode(['error' => '서버 처리 실패', 'error_id' => $errorId,
            JSON_UNESCAPED_UNICODE | JSON_THROW_ON_ERROR]);
    } else {
        header('Content-Type: text/plain; charset=utf-8');
        echo '서버 처리 실패. 오류 번호: ' . $errorId;
    }
});
ini_set('session.use_strict_mode', '1');
ini_set('session.use_only_cookies', '1');
session_set_cookie_params([
    'httponly' => true,
    'secure' => !$config['dev'],
    'samesite' => 'Lax',
    'path' => '/',
]);
session_start();
require_once __DIR__ . '/db.php';
require_once __DIR__ . '/functions.php';
if (!isset($_SESSION['csrf'])) {
    $_SESSION['csrf'] = bin2hex(random_bytes(32));
}
```

실행 위치 모든 요청의 서버 진입 직후

입력 설정 요청 경로 세션 Cookie

출력 초기화된 세션과 공통 함수

다음 단계 화면과 처리 endpoint

수정해도 되는 부분 안내 문구와 로그 위치 정책

함부로 수정하면 안 되는 부분 출력 전 세션 설정과 비밀을 숨긴 예외 처리를 유지한다

공통 초기화는 설정, 로그 정책, 세션, 함수 로드를 담당한다. 예외 처리기는 API 경로이면 JSON 오류를, 화면이면 글 오류를 반환한다. 앱 예외 로그에는 오류 종류, SQLSTATE와 DB 드라이버 오류 번호, 파일, 줄과 문 의용 번호를 남긴다. 구체적인 DB 오류 메시지는 운영 화면에 노출하지 않는다. 개발에서는 별도 디버깅으로

원인을 확인하되 비밀값을 공유하지 않는다.

24.4 입력 검증과 DB 함수를 한 곳에 모으기

파일 app/functions.php

코드 33 · php

```
<?php
function e(string $text): string
{
    return htmlspecialchars($text, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
}
function stop_page(int $status, string $message): never
{
    http_response_code($status);
    header('Content-Type: text/html; charset=utf-8');
    echo '<!doctype html><html lang="ko"><meta charset="utf-8">';
    echo '<title>처리 안내</title><h1>처리 안내</h1><p>' . e($message) . '</p>';
    echo '<p>입력 중이었다면 브라우저 뒤로 가기로 돌아가세요.</p>';
    echo '<a href="index.php">목록으로</a></html>';
    exit;
}
function post_only(): void
{
    if (($_SERVER['REQUEST_METHOD'] ?? '') !== 'POST') {
        header('Allow: POST');
        stop_page(405, 'POST 요청만 허용합니다.');
```

```
}
```

```
function csrf_check(): void
```

```
{
```

```
    $token = $_POST['csrf'] ?? null;
```

```
    if (!is_string($token))
```

```
        || !hash_equals($_SESSION['csrf'], $token)) {
```

```
        stop_page(403, '요청 확인 실패. 목록을 다시 열고 제출하세요.');
```

```
    }
```

```
}
```

```
function text_field(array $input, string $key): string
```

```
{
```

```
    $value = $input[$key] ?? null;
```

```
    if (!is_string($value)) {
```

```
        throw new InvalidArgumentException($key . ': 글 입력이 필요합니다.');
```

```
    }
```

```
    return trim($value);
```

```
}
```

```
function positive_id(mixed $value): int
```

```
{
```

```
    if (!is_string($value) || !preg_match('/^[1-9][0-9]{0,9}$/D', $value)) {
```

```
        throw new InvalidArgumentException('기록 번호가 잘못되었습니다.');
```

```
    }
```

```
    $id = filter_var($value, FILTER_VALIDATE_INT,
```

```
        ['options' => ['min_range' => 1, 'max_range' => 2147483647]]);
```

```
    if ($id === false) {
```

```
        throw new InvalidArgumentException('기록 번호 범위 오류');
```

```

    }
    return $id;
}
function validate_record(array $input): array
{
    $date = text_field($input, 'record_date');
    $item = text_field($input, 'item');
    $value = text_field($input, 'value');
    $memo = text_field($input, 'memo');
    $parsed = DateTimeImmutable::createFromFormat('!Y-m-d', $date);
    if (!$parsed || $parsed->format('Y-m-d') !== $date
        || substr($date, 0, 4) < '1000'
        || substr($date, 0, 4) > '9999') {
        throw new InvalidArgumentException('실제 날짜를 YYYY-MM-DD로 입력하세요.');
```

```

    }
    if ($item === '' || mb_strlen($item, 'UTF-8') > 80) {
        throw new InvalidArgumentException('항목은 1~80글자로 입력하세요.');
```

```

    }
    if (mb_strlen($memo, 'UTF-8') > 1000) {
        throw new InvalidArgumentException('메모는 1000글자 이하입니다.');
```

```

    }
    if (!preg_match('/^-(?:0|[1-9][0-9]{0,9})(?:\.[0-9]{1,2})?$/D', $value)) {
        throw new InvalidArgumentException(
            '숫자는 정수부 10자리, 소수부 2자리 이내입니다. 쉼표 없이 입력하세요.'
        );
    }
    return ['record_date' => $date, 'item' => $item,
        'value' => $value, 'memo' => $memo];
}
function find_record(int $id): array|false
{
    $stmt = db()->prepare('SELECT id, record_date, item, value, memo
        FROM records WHERE id = :id');
    $stmt->execute(['id' => $id]);
    return $stmt->fetch();
}
function list_records(): array
{
    return db()->query('SELECT id, record_date, item, value, memo
        FROM records ORDER BY record_date DESC, id DESC LIMIT 200')->fetchAll();
}
function write_record(array $record, ?int $id): void
{
    if ($id === null) {
        $sql = 'INSERT INTO records (record_date, item, value, memo)
            VALUES (:record_date, :item, :value, :memo)';
    } else {
        $sql = 'UPDATE records SET record_date = :record_date,
            item = :item, value = :value, memo = :memo WHERE id = :id';
        $record['id'] = $id;
    }
    $stmt = db()->prepare($sql);
    $stmt->execute($record);
}
function go_list(): never
{

```

```

    header('Location: index.php', true, 303);
    exit;
}

```

실행 위치 서버 PHP와 필요한 DB 호출

입력 Form 값 토큰 id와 기록 배열

출력 검증된 값 조회 배열 또는 오류

다음 단계 화면 저장 삭제 처리

수정해도 되는 부분 업무상 길이 범위와 안내 문구

함부로 수정하면 안 되는 부분 변경 시 DB 정의도 맞추고 CSRF 바인딩 출력 처리를 유지한다

이 파일을 처음부터 모두 외울 필요는 없다. e는 HTML 출력, csrf_check는 요청의 토큰 검사, text_field는 글 입력 검사, validate_record는 한 기록 검증, find_record는 한 건 조회, list_records는 목록, write_record는 등록·수정이다. 이름을 보고 목적을 찾아가는 연습을 한다.

숫자 검증은 값이 문자열인지, 부호와 숫자와 최대 2자리 소수인지, 정수부가 10자리 이내인지 확인한다. 0과 음수를 허용한다. 십표, 지수, 앞자리 0이 붙은 012는 거절한다. 사용자에게 허용 형식을 안내한 뒤 필요하면 업무 규칙을 바꾼다. 날짜는 실제 날짜와 DB 연도 범위를 검사한다. mb_strlen으로 한글을 글자 단위로 세므로 mbstring이 필요하다.

24.5 화면 만들기

파일 public/index.php

코드 34 · php

```

<?php
require_once __DIR__ . '/../app/bootstrap.php';
$editing = false;
$record = ['record_date' => '', 'item' => '', 'value' => '', 'memo' => ''];
if (isset($_GET['id'])) {
    try {
        $id = positive_id($_GET['id']);
    } catch (InvalidArgumentException $error) {
        stop_page(400, $error->getMessage());
    }
    $record = find_record($id);
    if ($record === false) {
        stop_page(404, '기록이 없습니다.');
```

```

<link rel="stylesheet" href="style.css">
<script src="app.js" defer></script>
</head>
<body>
<main>
<h1>데이터 기록장</h1>
<p>날짜 · 항목 · 숫자값 · 메모를 기록합니다. localhost 학습용입니다.</p>
<h2><?= $editing ? '기록 수정' : '새 기록' ?></h2>
<form action="save.php" method="post" class="entry">
  <input type="hidden" name="csrf" value="<?= e($_SESSION['csrf']) ?>">
  <?php if ($editing): ?>
    <input type="hidden" name="id" value="<?= e((string) $record['id']) ?>">
  <?php endif; ?>
  <label>날짜
    <input type="date" name="record_date" required
      value="<?= e($record['record_date']) ?>">
  </label>
  <label>항목
    <input name="item" maxlength="80" required value="<?= e($record['item']) ?>">
  </label>
  <label>숫자값 (유효숫자 없이, 소수 2자리까지)
    <input name="value" inputmode="decimal" required value="<?= e($record['value']) ?>">
  </label>
  <label>메모
    <textarea name="memo" maxlength="1000"><?= e($record['memo']) ?></textarea>
  </label>
  <button type="submit"><?= $editing ? '수정 저장' : '등록' ?></button>
  <?php if ($editing): ?><a href="index.php">수정 취소</a><?php endif; ?>
</form>
<h2>최근 기록</h2>
<p>날짜와 번호의 내림차순으로 최대 200건을 보여 줍니다.</p>
<p id="status" role="status">JSON 조회 준비 중</p>
<button type="button" id="reload">JSON 조회 확인</button>
<?php if ($records === []): ?>
  <p>아직 기록이 없습니다.</p>
<?php else: ?>
  <div class="table-wrap">
    <table>
      <thead><tr><th>날짜</th><th>항목</th><th>값</th><th>메모</th><th>관리</th></tr></thead>
      <tbody>
        <?php foreach ($records as $row): ?>
          <tr>
            <td><?= e($row['record_date']) ?></td>
            <td><?= e($row['item']) ?></td>
            <td><?= e($row['value']) ?></td>
            <td class="memo"><?= e($row['memo']) ?></td>
            <td>
              <a href="?id=<?= e((string) $row['id']) ?>">수정</a>
              <form action="delete.php" method="post" class="delete-form">
                <input type="hidden" name="csrf" value="<?= e($_SESSION['csrf']) ?>">
                <input type="hidden" name="id" value="<?= e((string) $row['id']) ?>">
                <button type="submit">삭제</button>
              </form>
            </td>
          </tr>
        <?php endforeach; ?>

```

```

</tbody>
</table>
</div>
<?php endif; ?>
</main>
</body>
</html>

```

실행 위치 PHP는 서버 HTML은 브라우저

입력 GET id와 DB 기록

출력 입력 Form과 기록 표

다음 단계 save delete endpoint와 브라우저

수정해도 되는 부분 라벨 제목 화면 순서

함부로 수정하면 안 되는 부분 동적 값의 e와 name 토큰 id 전달을 유지한다

GET id가 있으면 수정할 기록을 읽어 Form에 채운다. 없는 id면 새 기록 Form이다. 표의 수정은 GET으로 해당 편집 화면을 열고, 삭제는 CSRF 토큰이 들어 있는 POST Form으로 실행한다. 모든 기록 문자열은 e로 출력한다. 메모에 태그를 입력해도 실행하지 않고 글로 표시한다.

파일 public/style.css

코드 35 · css

```

body { margin: 0; background: #f4f6f8; color: #182331;
  font-family: system-ui, sans-serif; line-height: 1.6; }
main { max-width: 980px; margin: auto; padding: 24px; }
h1, h2 { line-height: 1.3; }
.entry { display: grid; gap: 12px; max-width: 560px; }
label { display: grid; gap: 4px; }
input, textarea, button { font: inherit; padding: 9px;
  border: 1px solid #aab4bf; border-radius: 5px; }
textarea { min-height: 85px; }
button { cursor: pointer; background: #173e62; color: white; }
a { color: #174c7a; }
.table-wrap { overflow-x: auto; margin-top: 16px; }
table { width: 100%; border-collapse: collapse; background: white; }
th, td { padding: 12px; border: 1px solid #d4dce3; text-align: left; }
th { background: #e8edf2; }
.memo { white-space: pre-wrap; overflow-wrap: anywhere; min-width: 160px; }
.delete-form { display: inline-block; margin: 5px; }
@media (max-width: 600px) { main { padding: 16px; } }

```

실행 위치 브라우저

입력 HTML 요소와 클래스

출력 읽기 편한 화면

다음 단계 사용자 화면

수정해도 되는 부분 색 간격 폭

함부로 수정하면 안 되는 부분 PHP 필드 이름과 무관하며 CSS만으로 검증을 만들 수 없다

CSS는 처리 규칙을 바꾸지 않는다. 화면의 폭과 간격, 모바일에서 표를 가로로 스크롤하는 방식만 조정한다. 데이터 입력의 name이나 PHP 키는 CSS 클래스와 별개다.

24.6 등록과 수정 처리

파일 public/save.php

코드 36 · php

```
<?php
require_once __DIR__ . '/../app/bootstrap.php';
post_only();
csrf_check();
try {
    $record = validate_record($_POST);
    $id = array_key_exists('id', $_POST) ? positive_id($_POST['id']) : null;
} catch (InvalidArgumentException $error) {
    stop_page(422, $error->getMessage());
}
if ($id !== null && find_record($id) === false) {
    stop_page(404, '수정할 기록이 없습니다.');
```

실행 위치 서버

입력 POST Form의 필드와 토큰 선택적 id

출력 INSERT 또는 UPDATE 뒤 303 이동

다음 단계 index.php

수정해도 되는 부분 안내 문구

함부로 수정하면 안 되는 부분 검증 전에 쓰지 않고 id 조건과 CSRF를 유지한다

id가 없으면 등록, 양의 정수 id가 있으면 수정이다. 검증 오류가 생기면 422 화면을 보여 주고 사용자가 뒤로 돌아갈 수 있게 한다. 이 작은 예제는 실패 입력값을 서버에 보관해 Form에 자동 재표시하는 기능은 넣지 않았다. 브라우저 뒤로 가기의 값 복원은 환경에 따라 다를 수 있다. 사용성을 확장하려면 검증 오류와 안전하게 이스케이프한 입력값을 다시 Form에 채운다.

24.7 삭제 처리

파일 public/delete.php

코드 37 · php

```
<?php
require_once __DIR__ . '/../app/bootstrap.php';
post_only();
csrf_check();
try {
    $id = positive_id($_POST['id'] ?? null);
} catch (InvalidArgumentException $error) {
```

```

    stop_page(422, $error->getMessage());
}
if (find_record($id) === false) {
    stop_page(404, '삭제할 기록이 없습니다.');
```

```

}
$stmt = db()->prepare('DELETE FROM records WHERE id = :id');
$stmt->execute(['id' => $id]);
go_list();
```

실행 위치 서버

입력 POST id와 CSRF 토큰

출력 대상 삭제 뒤 303 이동

다음 단계 index.php

수정해도 되는 부분 안내 문구

함부로 수정하면 안 되는 부분 DELETE의 WHERE와 토큰 검사를 빼지 않는다

삭제 전에 존재 여부를 조회한다. 실제 다중 사용자 서비스에서는 이 SELECT와 DELETE에 소유권 조건이 필요하다. 사용자 확인창은 실수 방지 도구일 뿐 서버 권한 검사를 대신하지 않는다. 데이터 변경은 최신 200건 표 밖의 기록에도 id를 알면 적용될 수 있으므로 공개 시 반드시 서버 권한을 추가한다.

24.8 JSON 조회와 브라우저 연결

파일 public/api/records.php

코드 38 · php

```

<?php
require_once __DIR__ . '/../app/bootstrap.php';
header('Content-Type: application/json; charset=utf-8');
header('Cache-Control: no-store');
if (($_SERVER['REQUEST_METHOD'] ?? '') !== 'GET') {
    header('Allow: GET');
    http_response_code(405);
    echo json_encode(['error' => 'GET 요청만 허용합니다.'],
        JSON_UNESCAPED_UNICODE | JSON_THROW_ON_ERROR);
    exit;
}
echo json_encode(['records' => list_records()],
    JSON_UNESCAPED_UNICODE | JSON_THROW_ON_ERROR);
```

실행 위치 서버

입력 GET 요청

출력 records를 담은 JSON 응답

다음 단계 fetch의 response.json

수정해도 되는 부분 응답에 추가할 합의된 필드

함부로 수정하면 안 되는 부분 JSON 외 출력 금지와 메서드 상태 코드 계약을 유지한다

파일 public/app.js

코드 39 · js

```

async function loadRecords() {
  const status = document.querySelector('#status');
  status.textContent = '조회 중';
  try {
    const response = await fetch('api/records.php', { cache: 'no-store' });
    const data = await response.json();
    if (!response.ok) {
      throw new Error(data.error ?? '조회 실패');
    }
    if (!Array.isArray(data.records)) {
      throw new Error('응답의 records 배열을 확인하세요.');
```

실행 위치 브라우저

입력 조회 버튼 클릭과 JSON 응답

출력 상태 문구와 삭제 확인

다음 단계 화면 DOM과 삭제 Form

수정해도 되는 부분 상태 문구와 확인 문구

함부로 수정하면 안 되는 부분 response.ok와 catch와 textContent를 유지한다

API는 목록을 반환하고 JavaScript는 받은 건수를 표시한다. records의 숫자 열은 PDO에서 문자열로 반환될 수 있다. 정확한 십진 저장 표현이므로 문자열을 그대로 전송한다. 차트에서 계산하려면 단위와 범위를 고려해 Number로 변환해야 한다. DB의 기록이 영구 저장되었다는 사실과 화면에 최신 200건만 나타난다는 사실은 다르다.

24.9 실행 순서

PHP 버전과 pdo_mysql, mbstring을 확인한다. MySQL을 실행한다. DB와 앱 계정을 준비한다. 선택한 DB에서 schema.sql을 한 번 실행한다. config.local.example.php를 config.local.php로 복사하고 설정을 입력한다. 터미널에서 project 폴더로 이동한다. 다음 명령을 실행한다. 브라우저로 127.0.0.1:8000에 접속한다.

코드 40 · sh

```
php -S 127.0.0.1:8000 -t public
```

실행 위치 개발 터미널

입력 project 폴더와 public 경로

출력 localhost 개발 서버

다음 단계 브라우저 요청

수정해도 되는 부분 비어 있는 다른 포트

함부로 수정하면 안 되는 부분 내장 서버를 공개 운영 서버로 쓰지 않는다

등록 예시는 날짜 2026-10-04, 항목 걸음 수, 값 20000, 메모 현장 근무다. 등록 후 표가 보이면 숫자값을 21000으로 수정하고 메모도 바꾼다. API 주소 /api/records.php를 열어 같은 값을 확인한다. 마지막으로 이 테스트 기록을 삭제한다.

한 건의 데이터가 이동하는 길을 따라가자

등록 Form에서 항목에 걸음 수, 값에 20000을 적었다고 하자. 브라우저는 item=걸음 수, value=20000이라는 이름과 값의 관계를 POST 본문으로 보낸다. 숫자 입력도 아직 글 형태다. PHP는 \$_POST에서 읽고 csrf_check로 이 화면에서 만든 토큰이 맞는지 확인한다. validate_record는 날짜와 글 길이, 십진 형식을 검사해 정돈한 배열을 돌려준다.

write_record는 id가 없다는 사실을 보고 INSERT를 선택한다. SQL에는 :item과 :value가 들어 있고 execute 배열에는 실제 걸음 수와 20000이 들어 있다. PDO가 둘을 분리해 DB에 전달한다. DB는 DECIMAL 규칙에 따라 20000.00으로 보관한다. 이때부터 PHP 요청이 끝나도 기록은 남는다.

save.php는 303과 Location을 보낸다. 브라우저는 index.php를 새 GET으로 요청한다. 새 요청에서는 이전 일반 변수가 없다. list_records가 DB에서 다시 읽어 목록을 만든다. foreach가 한 행을 꺼내 e를 거쳐 HTML 표에 넣는다. 별도로 app.js가 조회 API를 요청하면 같은 목록이 JSON으로 표현된다. Form, DB, HTML, JSON은 같은 데이터의 서로 다른 표현이다.

수정할 때는 표의 수정 링크가 ?id=17 같은 GET을 만든다. 서버는 17번을 찾아 입력창에 넣는다. 제출할 때 hidden id가 POST에 포함되어 UPDATE를 선택한다. id는 대상 선택값이며 사용자 신분 증명값은 아니다. 로그인 사이트로 확장하면 확인된 세션 사용자 id와 소유권 조건을 별도로 넣는다.

24.10 기대 결과를 확인한다

시험	기대 결과	확인하는 규칙
정상 기록 등록	목록에 새 기록	INSERT와 리다이렉트
값 0 입력	저장 성공	0을 빈 값으로 취급하지 않음
값 -12.50 입력	저장 성공	음수와 소수 2자리
날짜 2026-02-30	422 거절	실제 날짜 검사

시험	기대 결과	확인하는 규칙
값 1,000 또는 1e3	422 거절	십진 입력 형식
소수 12.345	422 거절	소수 자릿수
항목 공백만	422 거절	trim과 필수 검사
항목에 script 태그	표에서 글로 표시	HTML 이스케이프
수정 화면의 없는 id	404	대상 존재 검사
save.php를 GET으로 열기	405	메서드 제한
토큰 삭제 후 POST	403	CSRF 검사
새로고침	POST 재전송 없이 목록	303 처리
새 브라우저로 다시 조회	DB 기록이 남음	영구 저장

최신 200건 제한은 학습용이다. 모든 기록을 내려받거나 오래된 기록을 탐색하려면 페이지 나누기와 조회 조건을 추가한다. 입력 Form에 max와 maxlength를 더할 수 있지만 서버 검증도 유지한다.

잘못하면 어떤 문제가 생기는가

웹루트를 project 전체로 잡으면 비공개 설정과 SQL 파일이 접근 가능해질 수 있다. root 계정을 앱 연결에 쓰지 않는다. 이 프로젝트는 로그인 없는 로컬 실습이므로 비밀번호 한 칸을 JavaScript로 만들어 인터넷에 공개해서는 안 된다. 저장을 여러 번 클릭하면 중복 행이 생길 수 있다. 중복 방지가 필요하면 버튼 상태뿐 아니라 서버의 중복 요청 식별과 DB 유일성 규칙도 설계한다.

GPT에게 이렇게 시켜보자

“이 기록장에 항목 검색을 추가해 줘. SQL 값은 바인딩하고 결과 HTML은 이스케이프해 줘. 기존 날짜·숫자 검증과 CSRF를 유지하며 변경한 파일과 입력·출력 흐름을 설명해 줘. 제3권을 보기 전이므로 복잡한 DB 설계는 피하고 검색 한 기능만 완성해 줘.”

에러가 났다면

index만 안 열리면 개발 서버와 공개 루트를 본다. 모든 화면이 실패하면 config와 PDO 연결을 본다. 등록만 실패하면 POST 필드, 토큰, 숫자 규칙을 본다. 표만 비면 선택한 DB와 SELECT 결과를 본다. JSON만 실패하면 Response에 오류 글이 섞였는지 본다. 수정한 파일은 php -l로 문법 검사 후 같은 입력으로 재확인한다.

25장 제3권으로 이어지는 복습

무엇인가

이번 권에서는 PHP가 요청을 받아 DB에 명령을 보내고 결과를 응답하는 구조를 배웠다. 제3권의 주인공은 데이터를 어떤 표에 넣고 어떤 SQL로 꺼낼지다.

왜 필요한가

PHP 연결이 정상이어도 DB 구조가 목적과 맞지 않으면 검색과 분석이 어려워진다. 항목명이 제각각이거나 숫자를 메모 칸에 넣으면 나중에 합계를 내기 어렵다. 데이터를 저장하는 규칙은 화면을 만들기 전에 생각할 가치가 있다.

작동 원리

지금 프로젝트의 흐름을 한 문장씩 말해 본다. Form은 name과 값을 보낸다. 서버는 타입과 규칙을 검사한다. PDO는 자리표시자에 값을 전달한다. DB는 행을 저장한다. 조회는 배열로 받는다. HTML은 이스케이프하고 JSON은 구조를 인코딩해 반환한다.



코드에서 어떻게 보이는가

코드 41 · php

```

<?php
// PDO 연결이 있고 records 테이블이 준비된 뒤 실행한다.
$stmt = $pdo->prepare(
    'SELECT record_date, item, value FROM records
    WHERE item = :item ORDER BY record_date DESC'
);
$stmt->execute(['item' => '걸음 수']);
$rows = $stmt->fetchAll();
  
```

실행 위치 서버와 DB

입력 항목 걸음 수

출력 날짜순 기록 배열

다음 단계 향후 표 또는 차트

수정해도 되는 부분 조회할 항목과 고정 정렬 규칙

함부로 수정하면 안 되는 부분 외부 항목은 바인딩하고 외부 SQL 조각은 그대로 붙이지 않는다

이 예제를 보면 PHP 쪽 단계는 이미 익숙하다. 다음 권에서는 WHERE가 어떤 행을 고르는지, ORDER BY가 순서를 어떻게 만드는지, 같은 항목의 날짜별 합계와 평균은 어떻게 조회하는지 배우게 된다. 여기서는 SQL 문법을 더 넓히지 않고 역할 경계를 확인한다.

잘못하면 어떤 문제가 생기는가

날짜와 숫자를 전부 문자열 메모에 넣으면 정렬과 계산이 어려워진다. 여러 단위의 값을 한 항목처럼 합치면 합계가 무의미해진다. 단위와 항목 기준을 정하는 일은 문법 암기보다 먼저다.

스스로 답할 질문	답의 핵심
PHP 소스가 브라우저에서 실행되는가	이 책 구조에서는 서버에서 실행
POST이면 자동으로 안전한가	아니다 HTTPS 검증 권한 등이 필요
Form 숫자는 어떤 형태로 받는가	보통 문자열이며 배열 입력도 검사
JSON POST는 POST 배열에 들어가는가	자동으로 들어가지 않음
DB 저장은 어느 줄에서 일어나는가	INSERT를 execute한 시점
Prepared Statement는 무엇을 분리하는가	SQL 구조와 데이터 값
로그인과 권한은 같은가	인증과 접근 허용은 별도
세션과 DB는 같은 저장소인가	목적과 수명이 다름
XSS는 어디에서 실행되는 문제인가	브라우저 출력 문맥
운영 오류를 화면에 보여 주는가	상세는 숨기고 로그에서 확인

GPT에게 이렇게 시켜보자

“이 기록장의 DB를 제3권 학습 대상으로 삼겠다. id, 날짜, 항목, 숫자, 메모의 데이터 타입과 제약이 왜 필요한지 중고등학생 눈높이로 설명해 줘. 연결 PHP 코드는 반복하지 말고 DB 구조와 SELECT 원리에 집중해 줘.”

에러가 났다면

PHP 입력과 검증이 정상인지 먼저 본다. DB에 실제 저장된 행을 본다. SQL이 어떤 조건으로 행을 고르는지 본다. 결과를 받은 뒤 출력에서 누락됐는지 확인한다. 이 순서로 문제의 영역을 구분하면 GPT에게도 정확히 질문할 수 있다.

부록 코드 읽기 사전

표현	읽는 방법
\$name	name이라는 변수
=	오른쪽 결과를 왼쪽에 보관
.	문자열 연결
[]	배열 생성 또는 키로 값 읽기
=>	배열의 키와 값 연결
??	없거나 null이면 기본값
===	값과 타입을 비교
!	참거짓 반전
function	작업 정의
return	결과를 호출자에게 전달
echo	응답 본문에 출력
exit	현재 처리를 끝냄
->	객체의 메서드나 속성 사용
::	클래스의 상수나 정적 요소
new	객체 생성
try catch	예외 처리 구간
__DIR__	현재 파일의 폴더
:item	SQL 값 자리표시자
header	응답 헤더 설정
303	변경 후 다른 조회 페이지로 이동
400 401 403	잘못된 요청 인증 필요 접근 거절
404 405 422 500	대상 없음 메서드 오류 검증 실패 서버 오류

공식 참고 자료와 검증 범위

이 책은 원리 설명을 위해 직접 작성한 교재이며 공식 문서를 그대로 번역한 책이 아니다. 기본 예제는 PHP 8.2 이상 문법을 기준으로 작성했다. 실행 환경별 설치법과 지원 버전은 사용 시 확인해야 한다. 제공 ZIP과 교재의 프로젝트 코드는 동일하다. 이 제작 환경에서는 PHP와 MySQL 실행기가 준비되지 않아 DB 연동 실행 검증은 수행하지 못했다. 따라서 24장의 확인표를 사용자의 개발 환경에서 실행해 확인한다. 문서의 한글, 코드 줄, 도식과 페이지 배치는 렌더링으로 점검한다.

PHP 언어와 HTTP 입력의 전체 동작은 아래 공식 문서를 참고한다. 보안 예제는 개념을 이해하기 위한 범위이며 공개 서비스의 계정·업로드·운영 설계 전체를 대신하지 않는다.

1. PHP Manual · Superglobals

<https://www.php.net/manual/en/language.variables.superglobals.php>

2. PHP Manual · PDO prepare

<https://www.php.net/manual/en/pdo.prepare.php>

3. PHP Manual · SQL Injection

<https://www.php.net/manual/en/security.database.sql-injection.php>

4. PHP Manual · password_hash

<https://www.php.net/manual/en/function.password-hash.php>

5. PHP Manual · Securing Session INI Settings

<https://www.php.net/manual/en/session.security.ini.php>

6. PHP Manual · Handling file uploads

<https://www.php.net/manual/en/features.file-upload.php>

7. PHP Manual · json_encode

<https://www.php.net/manual/en/function.json-encode.php>

8. MDN · Using the Fetch API

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

코드 검토 시 핵심 질문은 여섯 가지다. 어디에서 실행되는가. 무엇을 받는가. 무엇을 내놓는가. 어디로 전달하는가. 내가 무엇을 바꿀 수 있는가. 무엇을 바꾸면 계약과 보안 규칙이 깨지는가. 이 질문을 습관으로 만들면 GPT가 만든 코드를 이해하고 수정 요청을 구체화할 수 있다.