깃에 2nd까지 커밋되어있는 부분은 docker 사용하기 전이라 php 폴더로 안 묶여 있습니다!! 최종은 docker-compose.yml, .env와 php 폴더만 확인하시면 됩니다.

구조는

docker-compose.yml

.env

php/

- Dockerfile
- db.php
- nav.php
- login.php
- register.php
- index.php
- write.php
- list.php
- logout.php
- view.php
- edit.php
- delete.php

이렇게 됩니다..

<Dockerfile>

```
FROM php:8.2-apache

RUN docker-php-ext-install pdo pdo_mysqli

COPY . /var/www/html/

RUN chown -R www-data:www-data /var/www/html/
```

php 8.2가 설치된 apache를 기본 웹서버로 이용한다. PDO와 MySQL을 연결하고, 현재 디렉토리 내 모든 파일을 컨테이너 내부로 복사한다. 또한 복사된 파일들의 소유자를 apache 웹서버 계정으로 설정하였다.

<docker-compose.yml>

```yaml
version: '3.8'

Run All Services
services:
  Run Service
  app:
    build: ./php
    ports:
      - "8080:80"
    volumes:
      - ./php:/var/www/html
    depends_on:
      - db
    environment:
      DB_HOST: ${DB_HOST}
      DB_NAME: ${DB_NAME}
      DB_USER: ${DB_USER}
      DB_PASS: ${DB_PASS}
```

php 폴더 내의 Dockerfile로 이미지를 생성하고 호스트의 8080 포트를 apache 80포트로 연결한다. 또한 로컬 php폴더와 연결하여 수정 사항을 즉각 반영할 수 있게 하였다. app은 db가 먼저 켜진 후 돌아가고, getenv 함수로 불러올 수 있게 환경 변수를 설정한다. (.env에서 가져옴)

```yaml
  Run Service
  db:
    image: mysql:8.0
    container_name: db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: crud
    volumes:
      - mysql_data:/var/lib/mysql
    ports:
      - "3306:3306"

  Run Service
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: phpmyadmin
    restart: always
    ports:
      - "8081:80"
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}

volumes:
  mysql_data:
```

<.env>

```
DB_HOST = db
DB_NAME = crud
DB_USER = root
DB_PASS = yourpassword
MYSQL_ROOT_PASSWORD = yourpassword
```

<db.php>

```php
<?php
$host = getenv(name: 'DB_HOST') ?: 'db';
$db = getenv(name: 'DB_NAME') ?: 'crud';
$user = getenv(name: 'DB_USER') ?: 'root';
$pass = getenv(name: 'DB_PASS') ?: 'yourpassword';

$dsn = "mysql:host=$host;dbname=$db;charset=utf8";

try {
    $pdo = new PDO(dsn: $dsn, username: $user, password: $pass, options: [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);
} catch (PDOException $e) {
    die("DB connect failed: " . $e->getMessage());
}
?>
```

db.php는 중복되는 기본 설정을 담은 파일이다. 환경 변수로부터 DB 접속 정보를 가져오고 (없으면 기본값으로 대체) PDO로 DB 연결(MYSQL)을 시도하는데 실패하면 오류 메시지를 띄운다.

<nav.php>

```php
<h5><u><a href="index.php">home</a></u>
<u><a href="write.php">write</a></u>
<u><a href="list.php">list</a></u>
<u><a href="logout.php">logout</a></u></h5>
```

nav.php는 로그인 후 메뉴가 중복되어서 한 곳에 모은 것이다. index, write, list, logout이 상단에 표시돼고 각자 링크가 연결되어있다.

회원가입을 하기 위해 register.php로 넘어가면

<register.php>

```php
<?php
require_once 'db.php';
```

여기는 db와 연결하기 위해 db.php 파일을 불러오는 부분이다.

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $_POST['username'] ?? '';
    $password = $_POST['password'] ?? '';

    if (!$username || !$password) {
        $message = "Please enter your username and password.";
    } else {
        $hashed = password_hash(password: $password, algo: PASSWORD_DEFAULT);
        $sql = "INSERT INTO users (username, password_hash) VALUES (?, ?)";
        $stmt = $pdo->prepare(query: $sql);
        $stmt->execute(params: [$username, $hashed]);

        header(header: "Location: login.php");
        exit;
    }
}
?>
```

이제 register 폼이 제출되면 username과 password를 변수에 저장한다. 값이 입력되지 않았으면 빈 문자열로 처리하고, 만약 둘 중 하나라도 입력되지 않았으면 입력해달라는 메시지를 띄운다. 그게 아니면 입력된 비밀번호를 해싱하고 sql에 username과 password를 넣는다. 이 바인딩하여 쿼리를 실행한 후 login.php로 이동한다.

```php
<!DOCTYPE html>
<html>

<head>
    <title>Register</title>
</head>

<body>
    <h5><u><a href="login.php">home</a></u>
    <u><a href="register.php">register</a></u>
    <u><a href="login.php">login</a></u></h5>

    <h3><strong>register</strong></h3>

    <form action="register.php" method="POST">
        username <input type="text" name="username" required><br><br>
        password <input type="password" name="password" required><br><br>
        <button type="submit">register</button>
    </form>

</body>

</html>
```

html 부분은 전반적인 구조를 담당하는데, 우선 상단에 home, regist, login 버튼을 띄우고 각자 링크가 연결돼 있다. register 폼을 띄운다. 값을 입력하고 register 버튼을 누르면 POST 방식으로 서버에 요청을 보낸다.

<login.php>

```php
<?php
session_start();
require_once 'db.php';
```

로그인 상태를 유지하기 위해 session start를 하고 db.php를 불러온다.

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $_POST['username'] ?? '';
    $password = $_POST['password'] ?? '';

    $sql = "SELECT * FROM users WHERE username = ?";
    $stmt = $pdo->prepare(query: $sql);
    $stmt->execute(params: [$username]);
    $user = $stmt->fetch();

    if ($user && password_verify(password: $password, hash: $user['password_hash'])) {
        $_SESSION['username'] = $username;
        header(header: "Location: index.php");
        exit;
    } else {
        $message = "You've got wrong username or password.";
    }
}
?>
```

만약 로그인 폼이 제출되면 위 코드를 실행하게 된다. 폼으로 전달받은 username과 password를 변수에 저장한다. PDO의 prepared statement로 SQL 쿼리를 실행해 사용자를 조회한다. 조회한 사용자 정보가 있고, 입력 비밀번호와 DB에 저장된 해시 비밀번호가 일치한다면 세션에 사용자 이름을 저장해 로그인을 유지하고, index.php로 이동한다. 만약 입력 정보가 DB 저장값과 일치하지 않는다면 틀렸다는 메시지를 저장한다.

이렇게 php부분이 끝난다....

```php
<!DOCTYPE html>
<html>

<head>
    <title>Login</title>
</head>

<body>
    <h5><u><a href="login.php">home</a></u>
    <u><a href="register.php">register</a></u>
    <u><a href="login.php">login</a></u></h5>

    <h3><strong>login</strong></h3>

    <form action="login.php" method="POST">
        username <input type="text" name="username" required><br><br>
        password <input type="password" name="password" required><br><br>
        <button type="submit">login</button>
    </form>

    <?php if (isset($message)) echo "<p>$message</p>"; ?>

</body>

</html>
```

상단에 home, register, login을 띄우고 버튼을 누르면 각자 연결된 페이지로 이동한다. (이건 로그인 폼을 제출하기 전의 디자인) 정보가 submit되면 POST 방식으로 서버에 요청을 보낸다. 만약 로그인 정보가 일치하지 않으면 위에서 받은 메시지를 띄운다.

<index.php>

```php
<?php
session_start();

if (!isset($_SESSION['username'])) {
    header(header: "Location: login.php");
    exit;
}
?>
```

만약 로그인 상태가 아니라면 login.php 페이지로 가도록 만들었다.

```html
<!DOCTYPE html>
<html>

<head>
    <title>Home</title>
</head>

<body>
    <?php require_once 'nav.php'; ?>

    <h3><strong>home</strong></h3>

    <h4>You are logged in as a user.</h4>

</body>

</html>
```

로그인 후 메뉴를 담은 nav.php를 불러오고, 로그인이 되었다는 메시지를 띄운다. write를 눌러 write.php에 들어가게 되면

<write.php>

```php
<?php
session_start();
if (!isset($_SESSION['username'])) {
    header(header: "Location: login.php");
    exit;
}

require_once 'db.php';
```

session start를 하고 username이 없으면 login 페이지로 돌아가게 된다.

마찬가지로 db.php를 불러온다.

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $title = $_POST['title'] ?? '';
    $content = $_POST['content'] ?? '';
    $username = $_SESSION['username'];

    $stmt = $pdo->prepare(query: "INSERT INTO posts (title, content, username) VALUES (?, ?, ?)");
    $stmt->execute(params: [$title, $content, $username]);

    header(header: "Location: list.php");
    exit;
}
?>
```

만약 write 폼이 제출되면 사용자가 작성한 title, content 내용을 가져온다. title, content, user를 posts 테이블에 저장하고 이 작업이 끝나면 list.php로 이동한다.

```php
<!DOCTYPE html>
<html>

<head>
    <title>Write</title>
</head>

<body>
    <?php require_once 'nav.php'; ?>

    <h3><strong>write</strong></h3>
    <form method="POST">
    Title <input type="text" name="title" required><br><br>
    Content <br>
    <textarea name="content" rows="10" cols="50" required></textarea><br><br>
    <button type="submit">submit</button>
</form>

</body>

</html>
```

nav.php를 불러 메뉴를 구성하고, required로 title과 content가 필수로 입력되게 했다. content는 사이즈를 지정하여 여러 줄이 입력될 수 있게 하였고, submit 버튼을 누르면 입력 내용이 POST로 서버에 전송된다.

<list.php>

```php
<?php
require_once 'db.php';

$stmt = $pdo->query(query: "SELECT * FROM posts ORDER BY created_at DESC");
$posts = $stmt->fetchAll();
?>
```

db.php를 불러오고 posts 테이블로부터 쿼리를 가져온다.

```php
<!DOCTYPE html>
<html>

<head>
    <title>List</title>
</head>

<body>
    <?php require_once 'nav.php'; ?>

    <h3><strong>list</strong></h3>
    <a href="write.php">[write]</a><br><br>
</form>

</body>

</html>
```

write 버튼도 만들어봤다.

```php
<?php foreach ($posts as $post): ?>
    <div style="border:1px solid #ccc; padding:10px; margin-bottom:10px;">
        <a href="view.php?id=<?= $post['id'] ?>">
            <strong><?= htmlspecialchars(string: $post['title']) ?></strong>
        </a><br>
        <small>writer : <?= htmlspecialchars(string: $post['username']) ?> | <?= $post['created_at'] ?></small>
    </div>
<?php endforeach; ?>
```

posts 배열에 있는 게시글 데이터를 하나씩 가져오며 반복문이 돈다. 리스트에는 각 글들마다 writer와 작성 일자와 시간이 표기되게 했다. 글 제목을 클릭하면 view.php로 이동하며 해당하는 글의 id가 전달된다. 그리고 htmlspecialchars로 보안처리가 되게 하였다.

<view.php>

```php
<?php
session_start();
require_once 'db.php';

$id = $_GET['id'] ?? null;
if (!$id) die("Invalid post ID");

$stmt = $pdo->prepare(query: "SELECT * FROM posts WHERE id = ?");
$stmt->execute(params: [$id]);
$post = $stmt->fetch();

if (!$post) {
    die("Post not found");
}
?>
```

전달받은 id에 대해 해당하는 글 데이터를 가져온다.

```php
<!DOCTYPE html>
<html>
<head><title>View Post</title></head>
<body>
    <h2><?= htmlspecialchars(string: $post['title']) ?></h2>
    <p><?= nl2br(string: htmlspecialchars(string: $post['content'])) ?></p>
    <small>writer : <?= htmlspecialchars(string: $post['username']) ?> | <?= $post['created_at'] ?></small><br><br>

    <a href="list.php">Back to list</a><br><br>

    <?php if (isset($_SESSION['username']) && $_SESSION['username'] === $post['username']): ?>
        <a href="edit.php?id=<?= $post['id'] ?>">[edit]</a>
        <a href="delete.php?id=<?= $post['id'] ?>" onclick="return confirm('Are you sure?')">[delete]</a>
    <?php endif; ?>

</body>
</html>
```

view 페이지에서도 마찬가지로 writer로 작성 일자와 시간을 볼 수 있는데, 추가적인 것은 back to list, edit, delete 기능이다. back to list는 유저가 해당 작성자인지 아닌지 상관

없이 가능하고, 만약 해당 작성자라면 edit, delete 버튼이 뜬다. edit을 누르면 edit.php로 넘어가고, delete는 한 번 누르면 Are you sure? 이라는 문구가 뜬다.

<edit.php>

```php
<?php
session_start();
require_once 'db.php';

$id = $_GET['id'] ?? null;
if (!$id) die("Invalid post ID");

$stmt = $pdo->prepare(query: "SELECT * FROM posts WHERE id = ?");
$stmt->execute(params: [$id]);
$post = $stmt->fetch();

if (!$post) die("Post not found");
```

```php
if (!isset($_SESSION['username']) || $_SESSION['username'] !== $post['username']) {
    die("You don't have permission.");
}

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $title = $_POST['title'] ?? '';
    $content = $_POST['content'] ?? '';

    $updateStmt = $pdo->prepare(query: "UPDATE posts SET title = ?, content = ? WHERE id = ?");
    $updateStmt->execute(params: [$title, $content, $id]);

    header(header: "Location: view.php?id=$id");
    exit;
}
```

앞에서 해당 유저가 아니라면 edit, delete 버튼이 뜨지 않도록 하긴 했지만 한 번 더 체크를 하게 하였다. (user가 아니라면 permission 없다고 뜸) edit 페이지에서 수정된 폼이 제출되면 또 title과 content를 받고 updateStmt로 새로운 데이터를 실행하고 view 페이지로 넘어간다

<delete.php>

```php
<?php
session_start();
require_once 'db.php';

$id = $_GET['id'] ?? null;
if (!$id) die("Invalid post ID");

$stmt = $pdo->prepare(query: "SELECT * FROM posts WHERE id = ?");
$stmt->execute(params: [$id]);
$post = $stmt->fetch();
```

```php
if (!$post) die("Post not found");

if (!isset($_SESSION['username']) || $_SESSION['username'] !== $post['username']) {
    die("You don't have permission.");
}

$deleteStmt = $pdo->prepare(query: "DELETE FROM posts WHERE id = ?");
$deleteStmt->execute(params: [$id]);

header(header: "Location: list.php");
exit;
?>
```

전달받은 id에 해당하는 글을 찾아 지우고 list.php로 이동한다.

<logout.php>

```php
<?php

session_start();
session_unset();
session_destroy();
header(header: "Location: login.php");
exit;

?>
```

session을 unset하기 위해 start하고 DB를 다 유지하면서 세션을 제거한다. 이후 로그인 페이지로 이동한다.