

Bad Practices:

1. Sensitive information such as database credentials are in the DB class itself.

```
no usages  
private function __construct()  
{  
    $dsn = 'mysql:dbname=phptest;host=127.0.0.1';  
    $user = 'root';  
    $password = 'pass';  
  
    $this->pdo = new \PDO($dsn, $user, $password);  
}
```

2. Common methods in classes can be re-written in a base class to avoid repetition of methods.

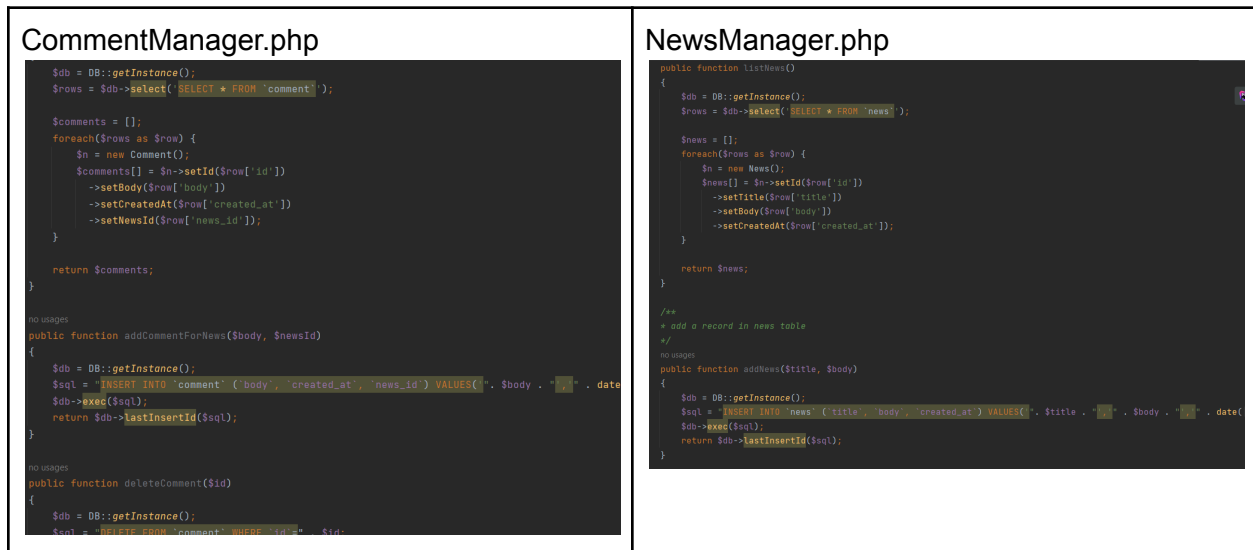
Comment.php

```
class Comment  
{  
    protected $id, $body, $createdAt, $newsId;  
  
    no usages  
    public function setId($id)  
    {  
        $this->id = $id;  
  
        return $this;  
    }  
  
    no usages  
    public function getId()  
    {  
        return $this->id;  
    }  
  
    no usages  
    public function setBody($body)  
    {  
        $this->body = $body;  
  
        return $this;  
    }  
  
    no usages  
    public function getBody()  
    {  
        return $this->body;  
    }  
}
```

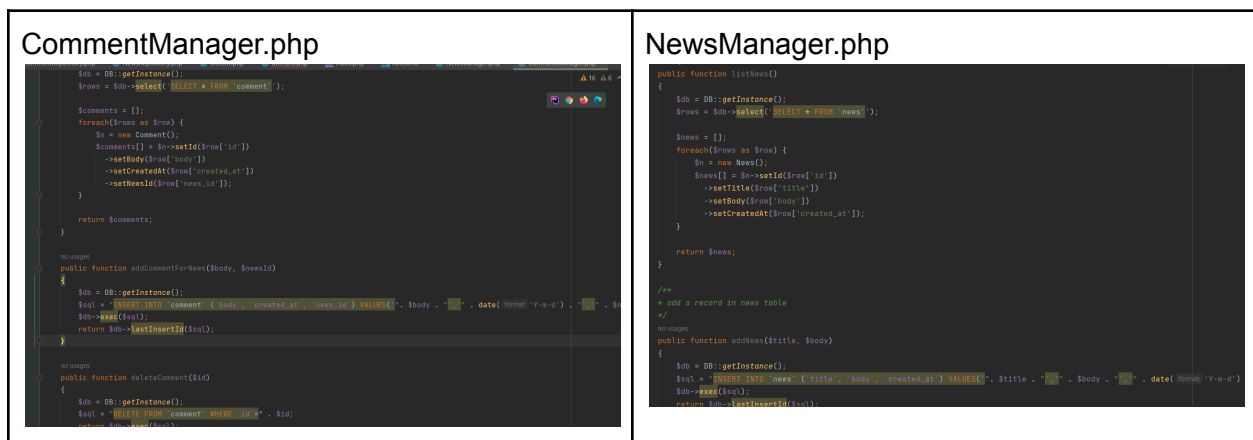
News.php

```
class News  
{  
    protected $id, $title, $body, $createdAt;  
  
    no usages  
    public function setId($id)  
    {  
        $this->id = $id;  
  
        return $this;  
    }  
  
    no usages  
    public function getId()  
    {  
        return $this->id;  
    }  
}
```

3. Database instances were called multiple times.



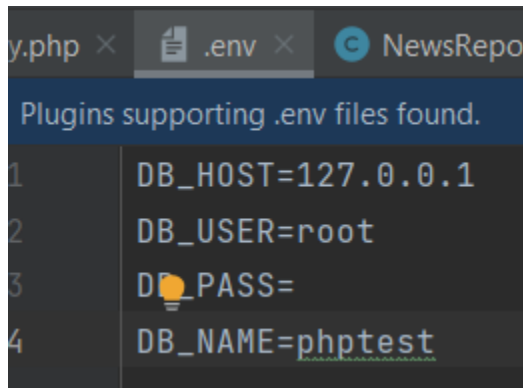
4. Database query were not secure thus susceptible to attacks.



5. Missing PHPDoc on classes. This will help the team identify who last touched a file, either the whole class or the methods within it.
6. Class name on utils for both Comment and News should be repository and not manager. Since both classes main responsibility is to handle data storage and retrieval operations for the entity it represents, it is better to rename the class as **CommentRepository** and **NewsRepository**. If the class contains business logic, the manager, as class name, can be used.

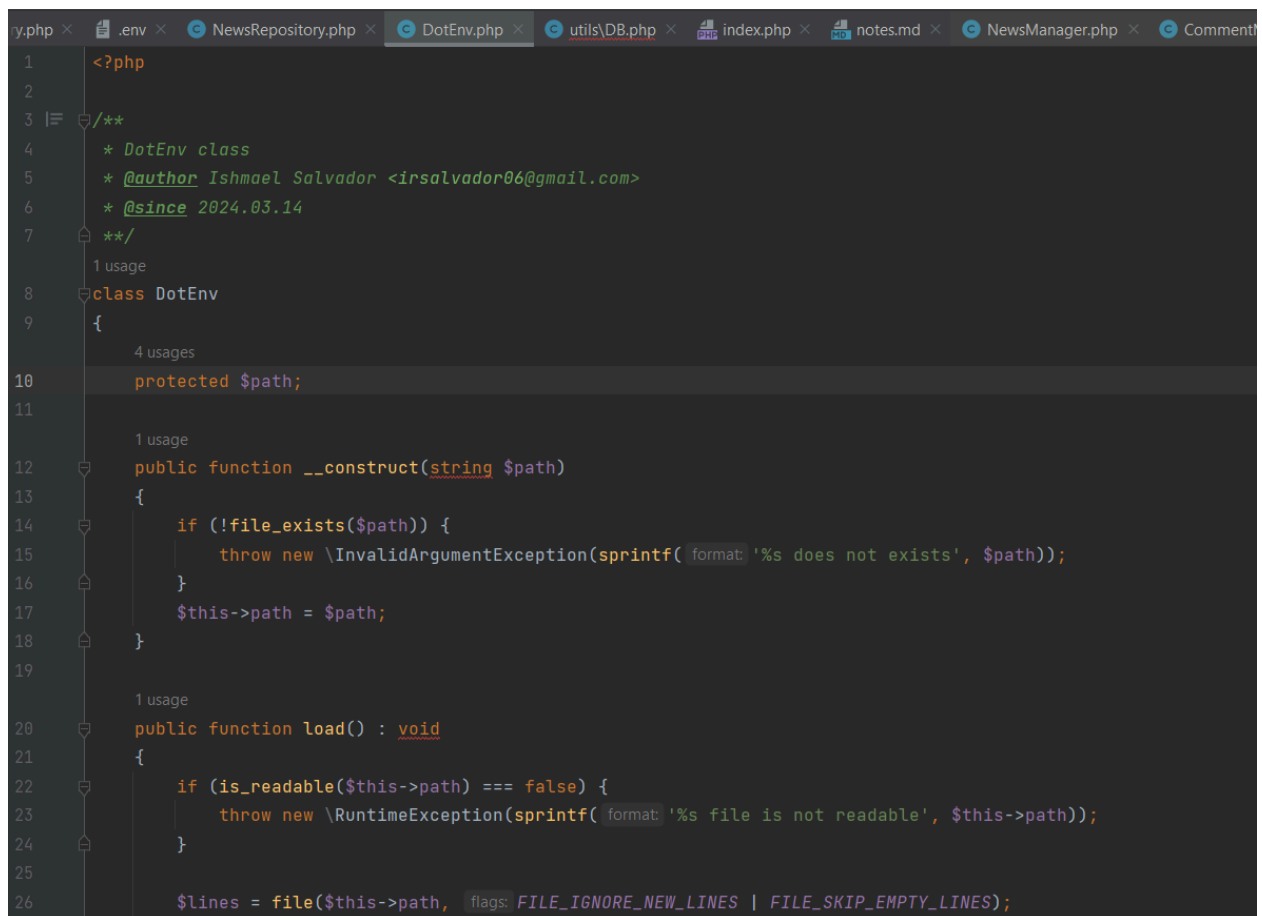
REFACTORING:

1. Implement a .env file to handle sensitive information like database credentials.



```
y.php x .env x NewsRepo
Plugins supporting .env files found.
1 DB_HOST=127.0.0.1
2 DB_USER=root
3 DB_PASS=
4 DB_NAME=phptest
```

2. Create a new class that will parse the .env file.



```
y.php x .env x NewsRepository.php x DotEnv.php x utils\DB.php x index.php x notes.md x NewsManager.php x Comment
1 <?php
2
3 /**
4  * DotEnv class
5  * @author Ishmael Salvador <irsalvador06@gmail.com>
6  * @since 2024.03.14
7  */
8
9 1 usage
10 class DotEnv
11 {
12     4 usages
13     protected $path;
14
15     1 usage
16     public function __construct(string $path)
17     {
18         if (!file_exists($path)) {
19             throw new \InvalidArgumentException(sprintf('format: %s does not exists', $path));
20         }
21         $this->path = $path;
22     }
23
24     1 usage
25     public function load() : void
26     {
27         if (is_readable($this->path) === false) {
28             throw new \RuntimeException(sprintf('format: %s file is not readable', $this->path));
29         }
30
31         $lines = file($this->path, flags: FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
```

3. Use the parser of the .env file to DB class.

```
1 usage
public static function getInstance()
{
    if (self::$instance === null) {
        try {
            (new DotEnv( path: __DIR__ . '/../.env'))->load();
            $dsn = 'mysql:dbname=' . getenv( varname: 'DB_NAME') . ';host=' . getenv( varname: 'DB_HOST');
            $user = getenv( varname: 'DB_USER');
            $password = getenv( varname: 'DB_PASS');
            $pdo = new \PDO($dsn, $user, $password);
            self::$instance = new self($pdo);
        } catch(PDOExcetion $e) {
            $error = $e->getMessage();
            echo $error;
        }
    }
    return self::$instance;
}
```

4. Apply the Singleton pattern to the DB class to ensure a single instance is used throughout the application.

```
1 usage
public static function getInstance()
{
    if (self::$instance === null) {
        try {
            (new DotEnv( path: __DIR__ . '/../.env'))->load();
            $dsn = 'mysql:dbname=' . getenv( varname: 'DB_NAME') . ';host=' . getenv( varname: 'DB_HOST');
            $user = getenv( varname: 'DB_USER');
            $password = getenv( varname: 'DB_PASS');
            $pdo = new \PDO($dsn, $user, $password);
            self::$instance = new self($pdo);
        } catch(PDOExcetion $e) {
            $error = $e->getMessage();
            echo $error;
        }
    }
    return self::$instance;
}

9 usages
public function getPdo()
{
    return $this->pdo;
}
```

5. Use the PDO prepare() and execute() in the query builder. These will help us prevent SQL injection attacks by eliminating the need to manually quote and escape the parameters.

CommentRepository.php

```
1 usage
public function listComments()
{
    $sql = 'SELECT * FROM `comment`';
    $sth = $this->db->getPdo()->query($sql);
    $rows = $sth->fetchAll();

    $comments = [];
    foreach ($rows as $row) {
        $c = new Comment();
        $comments[] = $c->setId($row['id'])
            ->setBody($row['body'])
            ->setCreatedAt($row['created_at'])
            ->setNewsId($row['news_id']);
    }

    return $comments;
}

1 usage
public function listCommentsByNewsId($newsId)
{
    $sql = 'SELECT * FROM `comment` WHERE `news_id` = ?';
    $stmt = $this->db->getPdo()->prepare($sql);
    $stmt->execute([$newsId]);
    $rows = $stmt->fetchAll();
}
```

NewsRepository.php

```
no usages
public function addNews($title, $body)
{
    $sql = 'INSERT INTO `news` (`title`, `body`, `created_at`) VALUES(?, ?, ?)';
    $stmt = $this->db->getPdo()->prepare($sql);
    $stmt->execute([$title, $body, date('Y-m-d')]);
    return $this->db->getPdo()->lastInsertId();
}

no usages
public function deleteNews($id)
{
    $commentRepository = new CommentRepository($this->db);
    $comments = $commentRepository->listCommentsByNewsId($id);

    foreach ($comments as $comment) {
        $commentRepository->deleteComment($comment->getId());
    }

    $sql = 'DELETE FROM `news` WHERE `id` = ?';
    $stmt = $this->db->getPdo()->prepare($sql);
    return $stmt->execute([$id]);
}
```

6. Introduce the Repository pattern for the data access logic.

CommentRepository.php

```
<?php

include_once('BaseRepository.php');

/**
 * CommentRepository class
 * @modified Ishmael Salvador <irsalvador86@gmail.com>
 * @since 2024.03.14
 */
2 usages
class CommentRepository extends BaseRepository
{
    2 usages
    public function __construct(DB $db)
    {
        parent::__construct($db);
        require_once(ROOT . '/class/Comment.php');
    }

    1 usage
    public function listComments()
    {
        $sql = 'SELECT * FROM `comment`';
        $sth = $this->db->getPdo()->query($sql);
        $rows = $sth->fetchAll();

        $comments = [];
        foreach ($rows as $row) {
            $c = new Comment();
            $comments[] = $c->setId($row['id'])
                ->setBody($row['body'])
                ->setCreatedAt($row['created_at'])
                ->setNewsId($row['news_id']);
        }

        return $comments;
    }
}
```

NewsRepository.php

```
<?php

include_once('BaseRepository.php');

/**
 * NewsRepository class
 * @modified Ishmael Salvador <irsalvador86@gmail.com>
 * @since 2024.03.14
 */
1 usage
class NewsRepository extends BaseRepository
{
    1 usage
    public function __construct(DB $db)
    {
        parent::__construct($db);
        require_once(ROOT . '/class/News.php');
    }

    1 usage
    public function listNews()
    {
        $sql = 'SELECT * FROM `news`';
        $sth = $this->db->getPdo()->query($sql);
        $rows = $sth->fetchAll();

        $news = [];
        foreach ($rows as $row) {
            $n = new News();
            $news[] = $n->setId($row['id'])
                ->setTitle($row['title'])
                ->setBody($row['body'])
                ->setCreatedAt($row['created_at']);
        }

        return $news;
    }
}
```

7. Create a base class for BaseRepository to handle the injecting the DB dependency into the subclasses and provides a common \$db property that can be used by both repositories(CommentRepository and NewsRepository).

```
1 <?php
2
3 BaseRepository class
4 Since: 2024.03.14
5 Author: Ishmael Salvador irsalvador06@gmail.com
6 2 usages 2 inheritors
7
8 class BaseRepository
9 {
10     11 usages
11     protected $db;
12
13     2 usages 2 overrides
14     public function __construct(DB $db)
15     {
16         $this->db = $db;
17     }
18 }
```

8. Retain the Dependency Injection pattern for the class dependencies on Comment and News classes by encapsulating it with getters and setters methods. This allows for better control over the access and modification of the class properties.

9. Create a base class for BaseClass since it has a common structure and functionality for setting and getting the id property.

```
<?php

BaseClass Common base class to hold common functionality between classes
Since: 2024.03.14
Author: Ishmael Salvador irsalvador06@gmail.com

2 usages 2 inheritors
class BaseClass
{
    protected $id, $body, $createdAt;

    3 usages
    public function setId($id)
    {
        $this->id = $id;
        return $this;
    }

    no usages
    public function getId()
    {
        return $this->id;
    }

    3 usages
    public function setBody($body)
    {
        $this->body = $body;
        return $this;
    }
}
```

10. Apply inheritance by extending both News and Comment class to BaseClass in order to inherit the get and set methods of id, body and create_at.

News.php

```
<?php

include_once('BaseClass.php');

Comment class
Since: 2024.03.14
Modified: Ishmael Salvador <irsalvador06@gmail.com>

1 page
class News extends BaseClass
{
    2 usages
```

Comment.php

```
<?php

include_once('BaseClass.php');

Comment class
Since: 2024.03.14
Modified: Ishmael Salvador <irsalvador06@gmail.com>

2 pages
class Comment extends BaseClass
{
    1 usage
```

