submit\_test.php

أضيف قائمة مهام قصيرة لمتابعة العمل (شرح الملف، تحديد مشاكل، واقتراح إصلاحات) وأعلمك أنني أبدأ بشرح السطر الأول الآن. ثم أعود بتحديث التقدم.

Created 3 todos

تقدّم: أنشأت قائمة المهام وبدأت تنفيذ مهمة 1 (شرح الملف). الآن سأشرح الملف submit\_test.php خطوة‑بخطوة (بلغة عربية واضحة)، ثم أذكر المشاكل المقترحة والتحسينات.

## شرح سطري/كتلي لـ submit\_test.php

أشرح هنا السطور بالترتيب المنطقي (كل سطر/تعليمة مع الغاية والتأثير):

1. `<?php`

- بداية ملف PHP. تُمكن المفسر من تفسير ما يلي ككود PHP.

2. `// test/submit\_test.php`

- تعليق يحدد اسم/موقع الملف للمطورين.

3. `include '../config.php';`

- يستورد ملف الإعدادات العام config.php، والذي من المحتمل أن يعرف الاتصال بقاعدة البيانات في متغير مثل `$connexion` و/أو إعدادات أخرى.

- غاية: الحصول على اتصال mysqli وإعدادات التطبيق.

4. `function esc($s){ return htmlspecialchars($s, ENT\_QUOTES | ENT\_SUBSTITUTE, 'UTF-8'); }`

- تعريف دالة مساعدة `esc` لتشفير النصوص عند العرض في HTML لمنع XSS.

- غاية: تعقيم/هروب النصوص قبل وضعها في المخرَج HTML.

5. `// جمع الأسئلة من POST`

- تعليق يشرح أن الكود التالي يجمع إجابات الأسئلة من الطلب POST.

6. `$q=[];`

- تهيئة مصفوفة فارغة لتخزين الإجابات Q1..Q10.

7-10. `for($i=1;$i<=10;$i++){`

`$k='q'.$i;`

`$q[$i] = $\_POST[$k] ?? '';`

`}`

- حلقة تُنشئ المفاتيح `q1`..`q10` ثم تقرأها من `$\_POST`.

- إذا لم تكن قيمة موجودة تُستخدم سلسلة فارغة `''` كقيمة افتراضية.

- غاية: جمع عشرة إجابات محتملة من النموذج المرسل.

11. `$suicide\_q = isset($\_POST['suicide\_q']) ? (int)$\_POST['suicide\_q'] : 0;`

- قراءة الحقل `suicide\_q` من الـ POST، وتحويله إلى عدد (int). إذا لم يكن موجودًا يساوي 0.

- غاية: معرفة ما إذا كان المستخدم أشار إلى سؤال انتحاري (0/1).

12. `// إدخال البيانات فقط id, Q1..Q10, suicide\_q`

- تعليق يشرح أن الإدخال التالي يسجل بيانات الأساس (test\_base) — عبارة توضيحية.

13-18. `$stmt=$connexion->prepare("`

` INSERT INTO test\_base (Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,suicide\_q,timestamp)`

` VALUES (?,?,?,?,?,?,?,?,?,?,?,NOW())`

`");`

- تحضير عبارة SQL مُعدة لإدراج (INSERT) صف في جدول `test\_base` مع أعمدة Q1..Q10 و suicide\_q و timestamp.

- تُستخدم علامات الاستفهام (?) كمعاملات مُعَدّة لتفادي الحقن SQL.

19-25. `$stmt->bind\_param('sssssssssis',`

` $q[1],$q[2],$q[3],$q[4],$q[5],$q[6],$q[7],$q[8],$q[9],$q[10],$suicide\_q`

`);`

- ربط المتغيرات بالعبارة المحضرة. السلسلة الأولى تحدد نوع كل معامل (s = string, i = integer).

- غاية: إدخال قيم Q1..Q10 و `suicide\_q` بأمان في الاستعلام.

- ملاحظة مهمة (مشكلة): نوع الحروف في السلسلة `'sssssssssis'` يبدو غير صحيح بالنسبة للمتغيرات الممرَّرة — التفصيل أسفل في قسم "المشاكل والتحسينات".

26-30. `if(!$stmt->execute()){`

` echo '<div class="alert alert-danger">Erreur insertion: '.esc($connexion->error).'</div>';`

` exit;`

`}`

- تنفيذ العبارة؛ إذا فشل يتم عرض رسالة خطأ (مع تعقيم) ثم إيقاف التنفيذ.

- غاية: التعامل مع خطأ الإدخال وإخبار المستخدم/المطور.

31. `$last\_id = $stmt->insert\_id;`

- الحصول على رقم التعريف (id) للصف الذي تم إدراجه للتو.

- ملاحظة: عادةً يُستخدم `$connexion->insert\_id` (خاصية كائن mysqli). بعض نسخ mysqli\_stmt قد لا تملك insert\_id؛ فمن الأفضل التأكد أو استخدام `$connexion->insert\_id`.

32. `$stmt->close();`

- إغلاق وتحرير موارد التحضير للعبارة.

33. `// جلب السطر بعد trigger`

- تعليق: يبدو أن هناك trigger (مؤشر) في قاعدة البيانات يعبئ أعمدة إضافية (مثل `comment\_msg` أو أعمدة `Display\_...`) بعد الإدراج، لذا الآن يُسترجع الصف الكامل بعد التنفيذ.

34. `$res = $connexion->query("SELECT \* FROM test\_base WHERE id=".(int)$last\_id);`

- استعلام بسيط لجلب الصف الذي أدخلناه للتو. يتم تحويل `$last\_id` إلى (int) لحماية من حقن.

- غاية: الحصول على القيم التي قد تكون حُشيت بواسطة trigger أو نحسبنا بها لاحقًا.

35. `$row = $res->fetch\_assoc();`

- جلب الصف كصف ترابطي (مصفوفة ارتباطية) في `$row`.

36. `// HTML النتائج`

- تعليق يوضح أن الكود التالي يبني مخرَج HTML لنتائج الاختبار.

37. `$html = '<div id="mainTestResult">';`

- بدء سلسلة HTML لحاوية النتائج.

38. `$html .= '<h4>'.esc($row['comment\_msg']).'</h4>';`

- إضافة عنصر `<h4>` يحتوي نص `comment\_msg` من السجل بعد تعقيمه عبر `esc`.

- غاية: عرض رسالة تعليق/نتيجة موجزة للمستخدم.

39. `// إذا suicide\_q = 1`

- تعليق: سلوك خاص عند وجود مؤشر انتحاري.

40-42. `if($suicide\_q === 1){`

` $html .= '<button class="btn btn-danger my-2" onclick="window.open(\'test/ssrs.php?id='.$last\_id.'\',\'\_blank\')">C-SSRS</button>';`

`}else{`

- إذا كان `suicide\_q` يساوي 1، تُضاف زر أحمر يفتح اختبار/صفحة C-SSRS في نافذة جديدة مع تمرير `id` عبر GET.

- غاية: توجيه المستخدم فوراً لاختبار/استمارة C-SSRS عند وجود مؤشر خطر.

43. ` // الاختبارات المقترحة بناءً على display`

- تعليق: الأسطر التالية تبني قائمة بالاختبارات الفرعية الممكن إظهارها.

44-52. ` $tests=[`

` 'Display\_PHQ9'=>['file'=>'phq.php','label'=>'PHQ-9'],`

` 'Display\_GAD7'=>['file'=>'gad.php','label'=>'GAD-7'],`

` 'Display\_PCL5'=>['file'=>'pcl.php','label'=>'PCL-5'],`

` 'Display\_ISI'=>['file'=>'isi.php','label'=>'ISI'],`

` 'Display\_AUDIT\_C'=>['file'=>'audit\_c.php','label'=>'AUDIT-C'],`

` 'Display\_AUDIT'=>['file'=>'audit.php','label'=>'AUDIT'],`

` 'Display\_DAST10'=>['file'=>'dast.php','label'=>'DAST-10'],`

` ];`

- مصفوفة تعريفية تربط أسماء أعمدة عرض `Display\_...` في جدول `test\_base` بملف الاختبار واسم العرض.

- غاية: تحديد أي اختبارات الفرعية يجب إظهارها بناءً على أعمدة `Display\_\*` الموجودة في `$row`.

53. ` $visibleTests=[];`

- تهيئة مصفوفة ستحتوي الاختبارات التي يجب إظهارها فعلاً.

54. ` foreach($tests as $col=>$info){`

- بدء حلقة على كل اختبار معرف في المصفوفة.

55-57. ` if(!empty($row[$col]) && (int)$row[$col]===1){`

` $visibleTests[$col]=$info;`

- شرط: إذا كانت قيمة العمود `$row[$col]` غير فارغة وتساوي 1 (بعد تحويلها إلى int) نضيف الاختبار إلى قائمة `visibleTests`.

- غاية: قرار إظهار الاختبارات بناءً على بيانات الصف (غالباً تحددها trigger أو منطق سابق).

58. ` // تحديد اسم الجدول الفرعي`

- تعليق: يُستخدم خريطة لتحديد جدول تخزين فرعي لكل اختبار.

59-67. ` $table\_map = [`

` 'Display\_PHQ9'=>'test\_phq',`

` 'Display\_GAD7'=>'test\_gad',`

` 'Display\_PCL5'=>'test\_pcl',`

` 'Display\_ISI'=>'test\_isi',`

` 'Display\_AUDIT\_C'=>'test\_audit',`

` 'Display\_AUDIT'=>'test\_audit',`

` 'Display\_DAST10'=>'test\_dast'`

` ];`

- خارطة ثابتة تربط مفتاح العرض باسم جدول فرعي يُخزن فيه نتائج الاختبار الفرعي (أو صف افتتاحي).

68. ` $table = $table\_map[$col] ?? '';`

- الحصول على اسم الجدول المناسب أو سلسلة فارغة إذا لم يوجد.

69-77. ` if($table){`

` $stmt2 = $connexion->prepare("`

` INSERT INTO $table (id, suicide\_q, comment\_msg, timestamp) `

` VALUES (?, ?, ?, NOW())`

` ");`

` $stmt2->bind\_param('iis', $last\_id, $suicide\_q, $row['comment\_msg']);`

` $stmt2->execute();`

` $stmt2->close();`

` }`

- إذا وجد جدول فرعي، يتم إدخال صف تمهيدي فيه يحتوي `id` (مرجع لصف الأساس)، `suicide\_q` و `comment\_msg` مع طابع زمني NOW().

- تستخدم عناصر Prepared statements للمعاملات لكن اسم الجدول مُضمّن في النص SQL (لا يمكن تمرير اسم جدول كمعامل مُعَد).

- غاية: إنشاء صفوف فرعية ذات صلة بقاعدة الأساس حتى يظهر الاختبار الفرعي بشكل مستقل أو لتخزين حالة مبدئية.

78-81. ` }`

`}`

``

- إغلاق شروط وحلقة foreach.

82. ``

- سطر فارغ / فصل منطقي.

83. ` if($visibleTests){`

- إذا توجد اختبارات ظاهرة يتم الآن بناء واجهة عرض لها (Accordion).

84. ` $html .= '<div class="accordion accordion-flush mt-3" id="accordionTests">';`

- بدء عنصر Bootstrap accordion في HTML.

85. ` $first=true;`

- متغيّر للمساعدة في جعل أول عنصر في الـ accordion مفتوحاً افتراضياً.

86. ` foreach($visibleTests as $col=>$info){`

- حلقة لبناء كل عنصر Accordion لكل اختبار مرئي.

87-110. (سلسلة طويلة مُجمَّعة في `$html .= ' ... '` تدرج HTML لكل عنصر)

- المحتوى المضاف يشتمل على:

- `<div class="accordion-item">` ... رأس العنوان الذي يستخدم `esc($col)` كمعرّف.

- زر accordion الذي يكون `collapsed` أو لا بناءًا على `$first` (الأول مفتوح).

- منطقة الانهيار `<div id="collapse...">` مع `show` للعنصر الأول.

- داخل `accordion-body` يوجد:

- `<form action="test/'.esc($info['file']).'" method="POST" class="subTestForm" data-result="result\_...">`

- حقل مخفي `<input type="hidden" name="base\_id" value="'.(int)$last\_id.'">`

- ثم النص `<?php include "test/'.esc($info['file']).'"; ?>` \_ضمن السلسلة\_.

- زر إرسال "Afficher le test".

- عنصر `<div id="result\_...">` فارغ لعرض نتائج الـ AJAX لاحقًا.

- غاية: تمكين المستخدم من إظهار/عرض كل اختبار فرعي داخل Accordion وإرسال/عرض النتيجة. كل نموذج يرسل `base\_id` إلى ملف الاختبار الفرعي.

- ملاحظة حاسمة (سلوك فعلي/مشكلة): الشطر `<?php include "test/'.esc($info['file']).'"; ?>` موجود داخل سلسلة نصية تُبنى في المتغير `$html`. هذا يعني أنه سيُرسَل حرفياً كنص إلى المتصفح ولن يُنفَّذ على الخادم عند بناء `$html`. إذا نية المبرمج كانت تضمين محتوى الملف خادمياً قبل الإرسال، فهذه الطريقة خاطئة — تضمين داخل سلسلة لا ينفذ. سأتناول هذا بالتفصيل في قسم "المشاكل والتحسينات".

111. ` $first=false;`

- بعد أول عنصر، نُعلِم أن العناصر التالية ليست الأولى، فتكون مغلقة افتراضياً.

112. ` }`

- إغلاق حلقة foreach للـ visibleTests.

113. ` $html .= '</div>';`

- إغلاق عنصر الـ accordion في HTML.

114. ` }`

- إغلاق شرط if($visibleTests).

115. `}`

- إغلاق البلوك `else` (الحالة التي ليس فيها `suicide\_q === 1`).

116. ``

- سطر فاصل.

117. `$html .= '</div>';`

- إغلاق الحاوية الرئيسية `#mainTestResult`.

118. `echo $html;`

- إخراج مخرَج HTML النهائي إلى المرسل (المتصفح أو AJAX).

ــ نهاية الملف.

---

## المشاكل/الملاحظات المهمة (محتويات وظيفية وأمنية)

1. خطأ/عدم تطابق في أنواع `bind\_param` للـ INSERT الرئيسي

- السطر: `$stmt->bind\_param('sssssssssis', $q[1], ... $q[10], $suicide\_q);`

- التحليل: سلسلة الأنواع `'sssssssssis'` تحتوي على 11 حرفًا؛ التوزيع يبدو أن العنصر العاشر يتم تعيينه للنوع `i` و الأخير `s` أو الترتيب معكوس. ولكن المتغيرات الممرَّرة هي 11 متغيرًا: Q1..Q10 و `$suicide\_q`. المطلوب هو 10 قيم نصية ثم عدد صحيح واحد. لذلك السلسلة الصحيحة ينبغي أن تكون `'ssssssssssi'` (عشرة `s` ثم `i`).

- أثر: إذا بقي كما هو قد يؤدي ذلك إلى خطأ أو ربط غير متوقع لأن النوع الأخير قد يُستخدم لـ `$suicide\_q` كـ string أو النوع العاشر يُعامل كـ integer. قد يسبب خطأ تنفيذ أو سلوك غير متوقع في قواعد البيانات.

2. استخدام `$stmt->insert\_id` بدلاً من `$connexion->insert\_id`

- السطر: `$last\_id = $stmt->insert\_id;`

- التحليل: عادةً الخاصية `insert\_id` موجودة على كائن mysqli (مثل `$connexion->insert\_id`). بعض إصدارات mysqli\_stmt قد لا تحتوي على `insert\_id`. الأفضل والأكثر موثوقية استخدام `$connexion->insert\_id`.

- اقتراح: استبداله بـ `$last\_id = $connexion->insert\_id;`

3. تضمين PHP داخل سلسلة HTML — لن يُنفَّذ (على الأرجح خطأ)

- السطر داخل `$html .= ' ... <?php include "test/'.esc($info['file']).'"; ?> ... ';`

- التحليل: لأن هذا النص جزء من سلسلة PHP، فإن `<?php ... ?>` يعتبر نصًا عاديًا ولن يُنفَّذ من قبل المفسر. إذا كانت النية إظهار محتوى ملف الاختبار مباشرة في الصفحة (server-side include) فهذه الطريقة لن تعمل.

- إصلاحين محتملين:

- تنفيذ include وقت الإنشاء: استخدم تجميع المحتوى عبر output buffering:

```

ob\_start();

include \_\_DIR\_\_ . '/'.$info['file'];

$test\_html = ob\_get\_clean();

$html .= $test\_html;

```

- أو إذا الهدف هو تقديم كود PHP ليتم تنفيذه لاحقًا على العميل عبر طلب منفصل، لا تضع `<?php include ... ?>` كجزء من سلسلة، بل ضع رابطًا/زرًا/إجراءًا يقوم بطلب الملف (`test/phq.php`) عبر AJAX أو فتحه في نافذة جديدة (كما هو معمول في جزء SSRS).

- تلميح: وضع include حرفي في string قد يكون محاولة خاطئة لعمل include عبر AJAX لاحقًا — يجب توضيح النية.

4. إدخال اسم الجدول مباشرة في SQL `$table` (خطر انبعاث SQL) لكن الخريطة ثابتة

- السطر: `INSERT INTO $table ...`

- التحليل: لا يمكن تمرير اسم جدول كمعامل Prepared، لذا يتم تضمينه في النص SQL. هذا يشكل خطرًا إذا كان `$table` مشتقًا من مدخلات المستخدم. هنا `$table` مأخوذ من `$table\_map` الثابت، لذا آمن. مع ذلك يجب التنبيه: لا تسمح بتمرير أسماء جداول من المستخدم مباشرةً دون تحقق صارم.

5. غياب تحقق/تنقية المدخلات قبل الإدراج (خاصة لأسئلة Q)

- حاليًا تَستخدم prepared statements لذلك تحمي من SQL injection، لكن لا يوجد تحقق على نوع الإجابات (رقمية/نصية) أو طولها. من الأفضل التحقق من الحدود وتطهير المحتويات قبل الإدراج.

6. عدم وجود CSRF protection

- النماذج لا تحتوي توكن CSRF، مما يجعلها عرضة لهجمات Cross-Site Request Forgery إذا كانت الجلسة تعتمد على ملفات تعريف الارتباط للمصادقة.

7. التعامل مع الأخطاء محدود (لا تسجيل/لا إعادة محاولة)

- في حالة فشل إدخال فرعي `$stmt2->execute()` لا يتم التحقق أو تسجيل الخطأ. من الأفضل التحقق وإجراء تسجيل أخطاء.

8. إمكانية تسريب خطأ `$connexion->error` إلى المستخدم النهائي

- يُعرض خطأ قاعدة البيانات في HTML. هذا مفيد للتطوير لكن قد يكشف تفاصيل حساسة في بيئة الإنتاج. الأفضل عرض رسالة عامة وتسجيل الرسالة التفصيلية في لوق خاص.

9. الأداء: يتم استدعاء INSERT لكل جدول فرعي داخل حلقة — إذا كان هناك عدة جداول هذا مقبول لكن يمكن تحسينه بتجميع أو ضمان العمليات ضمن معاملة (transaction) إذا كان التناسق مهمًا.

---

## اقتراحات إصلاح سريعة وموثوقة

1. تصحيح سلسلة أنواع bind\_param للـ INSERT الرئيسي:

- استبدل:

```

$stmt->bind\_param('sssssssssis', $q[1], ..., $q[10], $suicide\_q);

```

بـ:

```

$stmt->bind\_param('ssssssssssi', $q[1], ..., $q[10], $suicide\_q);

```

(عشرة `s` ثم `i`).

2. استخدام `$connexion->insert\_id`:

- استبدل `$last\_id = $stmt->insert\_id;` بـ `$last\_id = $connexion->insert\_id;`

3. إذا كنت تريد تضمين محتوى الملف الفرعي server-side داخل `$html` فعّل `include` فعليًا عبر output buffering:

- مثال داخل الحلقة عند بناء HTML:

```

$file = \_\_DIR\_\_ . '/' . $info['file']; // تأكد من المسار الصحيح

ob\_start();

include $file;

$test\_html = ob\_get\_clean();

$html .= $test\_html;

```

- هذا يضمن تنفيذ PHP الموجود في الملف المراد تضمينه وضم الناتج في `$html`.

4. إن كان القصد عرض النموذج فقط ثم تحميل/ملء تفاصيل عبر AJAX: لا تضع `<?php include ... ?>` في السلسلة. بدلاً من ذلك ضع مكانًا للتحميل الديناميكي (div) واستخدم JavaScript لجلب `test/phq.php?base\_id=...` وحقنه داخل الـ div.

5. التحقق من القيم القادمة (validation):

- التحقق من أن كل Qx تقع ضمن النطاق المتوقع (مثلاً أرقام 0..3 أو قيم مقيدة).

- التحقق من أن `$suicide\_q` هو 0 أو 1.

6. حماية CSRF:

- أضف توكن CSRF في النماذج وتحقق منه على الخادم قبل معالجة الإدخال.

7. تحسن في التعامل مع الأخطاء:

- سجل الأخطاء المفصلة في ملف لوق (error log) بدلاً من عرضها مباشرة.

- تحقق من نتائج تنفيذ `$stmt2->execute()` وأجب عند خطأ.

8. استخدام معاملة (transaction) إن كانت الكتابات إلى الجداول المتعددة يجب أن تكون متسقة:

- `BEGIN`, ثم جميع الإدراجات، ثم `COMMIT` أو `ROLLBACK` عند الخطأ.

---

## خاتمة قصيرة وملخّص التحقق

- شرحت محتوى `test/submit\_test.php` سطراً بسطر تقريبًا مع تبسيط منقطع (شرحت التعابير المهمة كتجمعات منطقية).

- أبرزت أخطاء/مخاطر رئيسية:

- خطأ في ترتيب/أنواع `bind\_param`.

- `include` داخل سلسلة — لن يُنفَّذ.

- استخدام `$stmt->insert\_id` قد لا يكون موثوقًا.

- غياب CSRF وValidation.

- قدمت إصلاحات واضحة وقابلة للتطبيق (تصحيح bind types، استخدام `ob\_start()`+`include` إن لزم، استخدام `$connexion->insert\_id`, إضافة تحقق وCSRF).

هل تريد أن أطبق أحد هذه الإصلاحات مباشرة على الملف (`test/submit\_test.php`)؟ أخبرني أيّ إصلاح تفضّل (مثلاً: تصحيح bind types + استخدام `$connexion->insert\_id` + تصحيح تضمين الملفات عبر output buffering)، وسأقوم بتعديل الملف وتشغيل فحص سريع/تحقق. - هذا يضمن تنفيذ PHP الموجود في الملف المراد تضمينه وضم الناتج في `$html`.

4. إن كان القصد عرض النموذج فقط ثم تحميل/ملء تفاصيل عبر AJAX: لا تضع `<?php include ... ?>` في السلسلة. بدلاً من ذلك ضع مكانًا للتحميل الديناميكي (div) واستخدم JavaScript لجلب `test/phq.php?base\_id=...` وحقنه داخل الـ div.

5. التحقق من القيم القادمة (validation):

- التحقق من أن كل Qx تقع ضمن النطاق المتوقع (مثلاً أرقام 0..3 أو قيم مقيدة).

- التحقق من أن `$suicide\_q` هو 0 أو 1.

6. حماية CSRF:

- أضف توكن CSRF في النماذج وتحقق منه على الخادم قبل معالجة الإدخال.

7. تحسن في التعامل مع الأخطاء:

- سجل الأخطاء المفصلة في ملف لوق (error log) بدلاً من عرضها مباشرة.

- تحقق من نتائج تنفيذ `$stmt2->execute()` وأجب عند خطأ.

8. استخدام معاملة (transaction) إن كانت الكتابات إلى الجداول المتعددة يجب أن تكون متسقة:

- `BEGIN`, ثم جميع الإدراجات، ثم `COMMIT` أو `ROLLBACK` عند الخطأ.

---

## خاتمة قصيرة وملخّص التحقق

- شرحت محتوى `test/submit\_test.php` سطراً بسطر تقريبًا مع تبسيط منقطع (شرحت التعابير المهمة كتجمعات منطقية).

- أبرزت أخطاء/مخاطر رئيسية:

- خطأ في ترتيب/أنواع `bind\_param`.

- `include` داخل سلسلة — لن يُنفَّذ.

- استخدام `$stmt->insert\_id` قد لا يكون موثوقًا.

- غياب CSRF وValidation.

- قدمت إصلاحات واضحة وقابلة للتطبيق (تصحيح bind types، استخدام `ob\_start()`+`include` إن لزم، استخدام `$connexion->insert\_id`, إضافة تحقق وCSRF).

هل تريد أن أطبق أحد هذه الإصلاحات مباشرة على الملف (`test/submit\_test.php`)؟ أخبرني أيّ إصلاح تفضّل (مثلاً: تصحيح bind types + استخدام `$connexion->insert\_id` + تصحيح تضمين الملفات عبر output buffering)، وسأقوم بتعديل الملف وتشغيل فحص سريع/تحقق.

submit\_test\_filiale.php

أنت تريد شرح مبسّط وواضح للفِيلَة submit\_test\_filiale.php بعد التعديلات — فيما يلي شرح عربي منظّم يوضّح ماذا يفعل كل جزء ولماذا، مع ملاحظات أمان وسلوك نهائي ومثال طلب.

## لمحة عامة سريعة

الملف يستقبل طلب POST من صفحات/نماذج الاختبارات الفرعية (PHQ‑9, GAD‑7, PCL‑5, ...). هدفه:

- التأكد أن هناك صف (row) موجود في جدول الاختبار الفرعي لـ `id` المعطى (صفٌ مبدئي يُنشأ سابقًا في `submit\_test.php`).

- تحديث أعمدة الأسئلة (Q1..Q20) فقط للقيم المرسلة، لكن فقط في الحالة الأولى (يمنع تعديل بعد أول إرسال: شرط Q1 فارغ).

- بعد التحديث يسترجع وينصّف `comment\_msg\_test` (الذي قد ينتجه TRIGGER أو تُحسب قيمه لاحقًا).

- يعيد استجابة JSON بنتيجة (`comment\_msg\_test`) أو خطأ.

## الحقول المتوقعة في POST

المفتاح الأساسي: `id` (رقم الـ id من `test\_base`).

اختياري/متوقع:

- `file` — اسم الملف الفرعي المرسل (مثال: `phq.php`) لتحديد أي جدول فرعي نعمل عليه.

- `suicide\_q` — 0 أو 1 (اختياري).

- `comment\_msg` — نص حاشية مبدئية (اختياري).

- `q1`, `q2`, ... `q20` — إجابات الاختبار الفرعي (قد يكون عدد أقل بحسب الاختبار).

الناتج: JSON مثل

{ "comment\_msg\_test": "نصٍ..." } أو { "error": "Server error" }.

## شرح خطوة‑بخطوة مع تفسير السطور المهمة

1. include وإعداد الهيدر

- `include '../config.php';`

يجلب إعداد الاتصال بقاعدة البيانات. بعد التعديل الكود يستخدم `$connexion` كاتصال mysqli.

- `header('Content-Type: application/json');`

كل الاستجابات تتم بصيغة JSON لذلك نضبط الهيدر.

2. دالة مساعدة للتعقيم (غير مستخدمة كثيراً هنا لكن مفيدة)

- `function esc($s) { return htmlspecialchars(...); }`

تستعمل لتعقيم نصوص عند الطباعة في HTML، لكن هنا الاستجابة JSON.

3. التحقق من وجود `id`

- إذا لم يُرسل `id` يرد خطأ JSON ويوقف التنفيذ.

4. قراءة المتغيرات من POST

- ` $id = (int)$\_POST['id'];` وقراءة `suicide\_q` و`comment\_msg`.

- ثم جمع Q1..Q20 في مصفوفة `$Q` (القيم غير المرسلة تُصبح null).

5. تحديد الجدول الفرعي عبر خريطة أسماء الملفات

- هناك مصفوفة `$mapping` تربط `'phq.php' => 'test\_phq'` الخ.

- إذا لم يُرسل `file` أو إنْ لم يكن موجودًا في الخريطة يرد خطأ JSON.

6. التأكد أن الصف موجود (INSERT إذا لم يوجد)

- يُحضّر استعلام: `SELECT id FROM $table WHERE id = ?`

- إذا لم يوجد صفّ لِـ `id`، يقوم بإدراج صف مبدئي:

`INSERT INTO $table (id, suicide\_q, comment\_msg, timestamp) VALUES (?, ?, ?, NOW())`

- التعامل يلغي تسريب أخطاء DB للمستخدم: الأخطاء تُسجل في لوق سيرفر، والعميل يحصل على `{'error':'Server error'}`.

7. بناء وتطبيق UPDATE للحقول المرسلة

- يبني مصفوفة `updateSet` عبر المرور على `$Q` ويضيف لكل مفتاح مُرسَل شرط `"qN = ?"`.

- قبل إضافة أي عمود يتأكد بواسطة regex أن اسم العمود صالح: `preg\_match('/^q[1-9][0-9]\*$/i', $col)` (يحمي من حقن أسماء أعمدة).

- يضيف دائمًا `timestamp = NOW()` لتحديث الطابع الزمني.

- بناء جملة UPDATE:

`UPDATE $table SET q1=?, q2=?, ..., timestamp=NOW() WHERE id = ? AND (Q1 IS NULL OR Q1 = '')`

الشروط الأخيرة -> تمنع التعديل إذا كانت Q1 ليست فارغة (أي أنه تم إرسال الاختبار مسبقًا).

- ثم يربط المعاملات بطريقة آمنة لـ `bind\_param` (يحوّل مصفوفة القيم إلى مراجع ثم يستخدم `call\_user\_func\_array`) وينفّذ.

8. جلب `comment\_msg\_test` للرد

- بعد التحديث/trigger يجري:

`SELECT comment\_msg\_test FROM $table WHERE id = ? LIMIT 1`

ويرد JSON يحتوي `comment\_msg\_test` (أو سلسلة فارغة إن لم تتوفر).

## سلوكيات/قرارات مهمة في الكود

- الأسماء الثابتة في `$mapping` تحمي من حقن اسم جدول؛ لا يُستخدم اسم جدول من المستخدم مباشرةً بدون تحقق.

- إدراج صف مبدئي إذا لم يكن موجودًا يتيح أن `submit\_test.php` لا يكون ضروريًا دائماً — لكن نظامك السابق أنشأ الصف في `submit\_test.php` لذلك عادة لن يدخل الفرع.

- منع التعديل بعد الإرسال: الشرط `(Q1 IS NULL OR Q1 = '')` يضمن أن التحديث يتم مرة واحدة فقط — هذا سلوك متعمد (حالة B).

- الأخطاء التفصيلية لا تُعاد للعميل، بل تُسجَّل في لوق السيرفر لتجنب كشف معلومات داخلية.

## ملاحظات أمان وتحسينات مقترحة

- حالياً لا يوجد CSRF token: من الأفضل إضافة حماية CSRF للنماذج.

- تحقّق محدود على قيم الـ Q: ينبغي التحقق من نطاق/نوع القيم (مثلاً 0..3 أو قيم مقيدة) قبل الحفظ.

- قد تحتاج إلى تغليف INSERT+UPDATE داخل معاملة (transaction) إن أردت ضمان الاتساق إذا كانت عمليات متعددة مرتبطة.

- إذا كنت تريد حساب النتيجة/المستوى وكتابة `comment\_msg\_test` في نفس الملف بدل trigger، أضيف لك الدالة المناسبة لحساب نقاط PHQ‑9 / GAD‑7 إلخ هنا.

- سجّلات الأخطاء تُرسل لـ `error\_log()` — تأكد أن لوق السيرفر متاح للـ debugging في التطوير.

## مثال سريع لنداء endpoint (PowerShell / curl) — اختياري

(افتراضي: الملف على المسار `http://localhost/PSY/test/submit\_test\_filiale.php`)

```powershell

curl.exe -Method POST -Uri 'http://localhost/PSY/test/submit\_test\_filiale.php' `

-Body @{

id = 123

file = 'phq.php'

q1 = '1'

q2 = '0'

q3 = '2'

suicide\_q = 0

}

```

الرد المتوقَّع (JSON):

{ "comment\_msg\_test": "..." }

## الخلاصة المختصرة

- الملف يتحقق من وجود الصف، ينشئ صفًا مبدئيًا إن لم يوجد، يحدث الحقول المرسلة بطريقة آمنة فقط في أول إرسال، ثم يعيد `comment\_msg\_test` في JSON.

- الكود الآن أكثر أمانًا (prepared statements، فحص أسماء الأعمدة، تسجيل أخطاء داخلية)، لكن يبقى مطلوبًا إضافة التحقق على القيم وCSRF وحساب النتيجة إن أردت أن لا تعتمد على TRIGGER.

هل تريد أن أضيف الآن حساب نتيجة PHQ‑9 أو GAD‑7 داخل هذا الملف (أو أن أضع مثال دالة حسابية لكل اختبار)؟ أي اختبار بداية تفضّل أن أطبّق له الحساب؟

المشكل الذي اواجهه دائما هو {"error":"Invalid subtest file"} عندما اضغط afficher le resultat de test و لا يتم ادراج اية ادخالات في السطر المضاف بواسطة submit\_test.php في جدول الاختبار المعني

قصّة المشكلة باختصار وبوضوح — سبب الخطأ الذي تراه ("Invalid subtest file") هو أن `$\_POST['file']` غير موجود أو قيمته لا تطابق مفاتيح الخريطة `$mapping` في `submit\_test\_filiale.php`. وهذا يؤدي إلى رد JSON هذا، ولذلك لا يجرى أي إدراج/تحديث في جدول الاختبار.

إليك الأسباب الأكثر احتمالاً وما الذي يجب فحصه أولاً، ثم تصحيح عملي جاهز للتطبيق.

1) لماذا يحصل ذلك عمليًا (جذر المشكلة)

- في الكود الذي يولّد واجهة الاختبارات (الـ accordion) — الملف submit\_test.php — المكوّن `<form>` الذي يُفترض أن يرسل بيانات الاختبار يبدو أنه لا يرسل حقلًا مخفيًا باسم `file` (أو يرسله باسم آخر). في الملف الذي أرسلت سابقًا يوجد السطر داخل سلسلة:

`<?php include "test/'.esc($info['file']).'"; ?>`

وهذا داخل سلسلة `$html`، لذلك لا يُنفّذ INCLUDE على الخادم؛ يتحوّل إلى نص في الـ HTML وليس إلى استمارة فعلية للـ subtest. النتيجة: نماذج الاختبار الفرعي أو الحقول المخفية لن تُدرج في الصفحة، وبالتالي عند الضغط على "Afficher le test" لن يتم إرسال `file` إلى `submit\_test\_filiale.php`.

- نقطة إضافية: في `submit\_test.php` يبدو أن `form` كانت تشير في `action` إلى `test/<subfile>.php` بدلاً من الإشارة إلى `submit\_test\_filiale.php`. فإذا كان الزر يرسل إلى ملف الاختبار الفرعي بدلاً من إلى ملف التجميع/الحفظ (`submit\_test\_filiale.php`) فالتوقع أن `submit\_test\_filiale.php` لن يتلقى أبدًا الطلب المناسب مع `file`.

الخلاصة: المشكلة غالبًا في الواجهة (HTML / تشكيل النموذج) — النموذج لا يرسل المعطيات المتوقعة (خاصة `file`)، أو يرسلها باسم/مسار مختلف.

2) خطوات فحص سريعة (لتتحقق بنفسك)

- افتح أدوات المطور (DevTools) -> تبويب Network -> راقب الطلب عند الضغط على زر "Afficher le test".

- انظر Request Payload أو Form Data: هل هناك حقل `file`؟ ما قيمته؟ إلى أي عنوان (URL) تُرسل البيانات؟ (هل إلى submit\_test\_filiale.php أم إلى phq.php إلخ)

- إن لم ترَ `file` أو إن كان الاسم مختلفًا، فهذه هي المشكلة.

- خيار سريع بالـ curl لاختبار endpoint مباشرة (مثال):

curl (PowerShell):

```powershell

curl.exe -Method POST -Uri 'http://localhost/PSY/test/submit\_test\_filiale.php' -Body @{

id = 123

file = 'phq.php'

q1 = '1'

q2 = '0'

suicide\_q = 0

} | ConvertFrom-Json

```

لو أعاد هذا `{ "comment\_msg\_test": ... }` فمعنى ذلك أن endpoint يعمل — إذًا المشكلة في المتصفح/النموذج.

3) حلّان عمليّان (الأفضل: تعديل توليد النموذج في `submit\_test.php`)

A — أفضل إصلاح (أوصي به)

- عدّل `submit\_test.php` بحيث:

1. يبني الـ `<form>` ليُرسل إلى `submit\_test\_filiale.php` (أو المسار الصحيح) وليس إلى ملف الاختبار الفرعي مباشرة.

2. يُدرج حقل مخفي `<input type="hidden" name="file" value="phq.php">` (أي اسم الملف كما هو في الخريطة).

3. إذا أردت تضمين نموذج الاختبار داخل الصفحة مباشرة (ليعرض الحقول q1..qN) فقم بعمل include على الخادم عند بناء الـ HTML (باستخدام output buffering) بدلاً من وضع `<?php include ... ?>` داخل سلسلة نصية.

مثال (مقتطف تغييري لـ `submit\_test.php`) — الفكرة الأساسية:

- عند بناء كل عنصر من الـ accordion استبدل الجزء الذي كان:

action="test/'.esc($info['file']).'"

وداخل السلسلة: `<?php include "test/'.esc($info['file']).'"; ?>`

- بــ:

- action: `"submit\_test\_filiale.php"` (أو `"test/submit\_test\_filiale.php"` حسب المسار).

- أضف: `<input type="hidden" name="file" value="'.esc($info['file']).'">`

- لإظهار محتوى الملف الفرعي داخل الـ form (server-side include) استخدم:

```

ob\_start();

include \_\_DIR\_\_ . '/' . $info['file']; // \_\_DIR\_\_ داخل test/ ينتج المسار الصحيح

$test\_html = ob\_get\_clean();

$html .= $test\_html;

```

هذا يضمن أن الحقول `q1..qN` موجودة داخل النموذج، وكذلك الحقل المخفي `file`.

B — حل بديل (أخف تعديلًا)

- إبقاء `submit\_test.php` كما هو، لكن تعديل `submit\_test\_filiale.php` لكي يقبل حالات غياب `file` ويحاول استنتاج الجدول من عنوان الـ Referer أو من `base\_id` — هذا أقل أمانًا ومعرض للأخطاء. لا أنصح به كحل أولي.

4) قطعة كود جاهزة لتضمين في `submit\_test.php`

(أعرض لك مثال واضح: كيف تبني الـ form بشكل صحيح)

- داخل الحلقة حيث تبني الـ HTML لكل اختبار استبدل الجزء الحالي بهذا النمط (pseudocode قصير):

```

$action = 'submit\_test\_filiale.php'; // أو 'test/submit\_test\_filiale.php' إذا يلزم

$html .= '<form action="'.$action.'" method="POST" class="subTestForm" data-result="result\_'.esc($col).'">';

$html .= '<input type="hidden" name="id" value="'.(int)$last\_id.'">';

$html .= '<input type="hidden" name="file" value="'.esc($info['file']).'">';

// الآن نريد تضمين النموذج الفرعي فعليًا:

ob\_start();

include \_\_DIR\_\_ . '/' . $info['file']; // يفترض أن phq.php ... في نفس مجلد test/

$test\_inner = ob\_get\_clean();

$html .= $test\_inner;

$html .= '<button type="submit" class="btn btn-success my-3">Afficher le test</button>';

$html .= '</form>';

```

5) هل أطبّق التصحيح مباشرة على ملف `submit\_test.php` الموجود في المشروع؟

- أقدر أن أطبق التعديل مباشرة الآن: أغير الـ action، أضيف الحقل المخفي `file`، وأستبدل include الحرفي بطريقة `ob\_start()` لضم النموذج الفرعي server-side. هذا سيُصلح المشكلة فورًا لأن الطلب من المتصفح سيحتوي على `file` و`id` وحقول qN، وسيصل إلى `submit\_test\_filiale.php` بشكل صحيح.

- أحتاج موافقتك لأطبّق التعديل على الملف أو أقدر أن أقدّم لك ملف التعديل الكامل هنا لتنسخه.

6) خلاصة سريعة وخطوات تنفيذ مقترحة الآن (مختصرة)

- تحقق أولًا في DevTools → Network: هل الطلب يحتوي على `file` وُما هو الـ URL الذي يُرسل إليه؟

- إن وجدت أن `file` مفقود أو أن الـ action يرسل إلى ملف فرعي بدل `submit\_test\_filiale.php` فطبّق التعديلات أعلاه في `submit\_test.php`.

- أستطيع أجرِّي التعديل الآن وأعيد اختبار الواجهة.

أخبرني: تريدي أن أطبق التعديل المقترح على `test/submit\_test.php` الآن؟ أطبّقه مباشرة (وأختبر بنفسي ما يمكن فحصه) أو أرسل لك نص التعديل لتلصقه؟5) هل أطبّق التصحيح مباشرة على ملف `submit\_test.php` الموجود في المشروع؟

- أقدر أن أطبق التعديل مباشرة الآن: أغير الـ action، أضيف الحقل المخفي `file`، وأستبدل include الحرفي بطريقة `ob\_start()` لضم النموذج الفرعي server-side. هذا سيُصلح المشكلة فورًا لأن الطلب من المتصفح سيحتوي على `file` و`id` وحقول qN، وسيصل إلى `submit\_test\_filiale.php` بشكل صحيح.

- أحتاج موافقتك لأطبّق التعديل على الملف أو أقدر أن أقدّم لك ملف التعديل الكامل هنا لتنسخه.

6) خلاصة سريعة وخطوات تنفيذ مقترحة الآن (مختصرة)

- تحقق أولًا في DevTools → Network: هل الطلب يحتوي على `file` وُما هو الـ URL الذي يُرسل إليه؟

- إن وجدت أن `file` مفقود أو أن الـ action يرسل إلى ملف فرعي بدل `submit\_test\_filiale.php` فطبّق التعديلات أعلاه في `submit\_test.php`.

- أستطيع أجرِّي التعديل الآن وأعيد اختبار الواجهة.

أخبرني: تريدي أن أطبق التعديل المقترح على `test/submit\_test.php` الآن؟ أطبّقه مباشرة (وأختبر بنفسي ما يمكن فحصه) أو أرسل لك نص التعديل لتلصقه؟