Section 2: Code Implementation & Debugging

Task 1: Backend Challenge (Laravel)

Given the following insecure Laravel API endpoint:

php

CopyEdit

```
Route::get('/transactions/{id}', function ($id) {
return Transaction::where('user_id', $id)->get();
});
```

● Identify security flaws and rewrite the endpoint to:

○ Use proper authentication.

○ Optimize query performance for large-scale data.

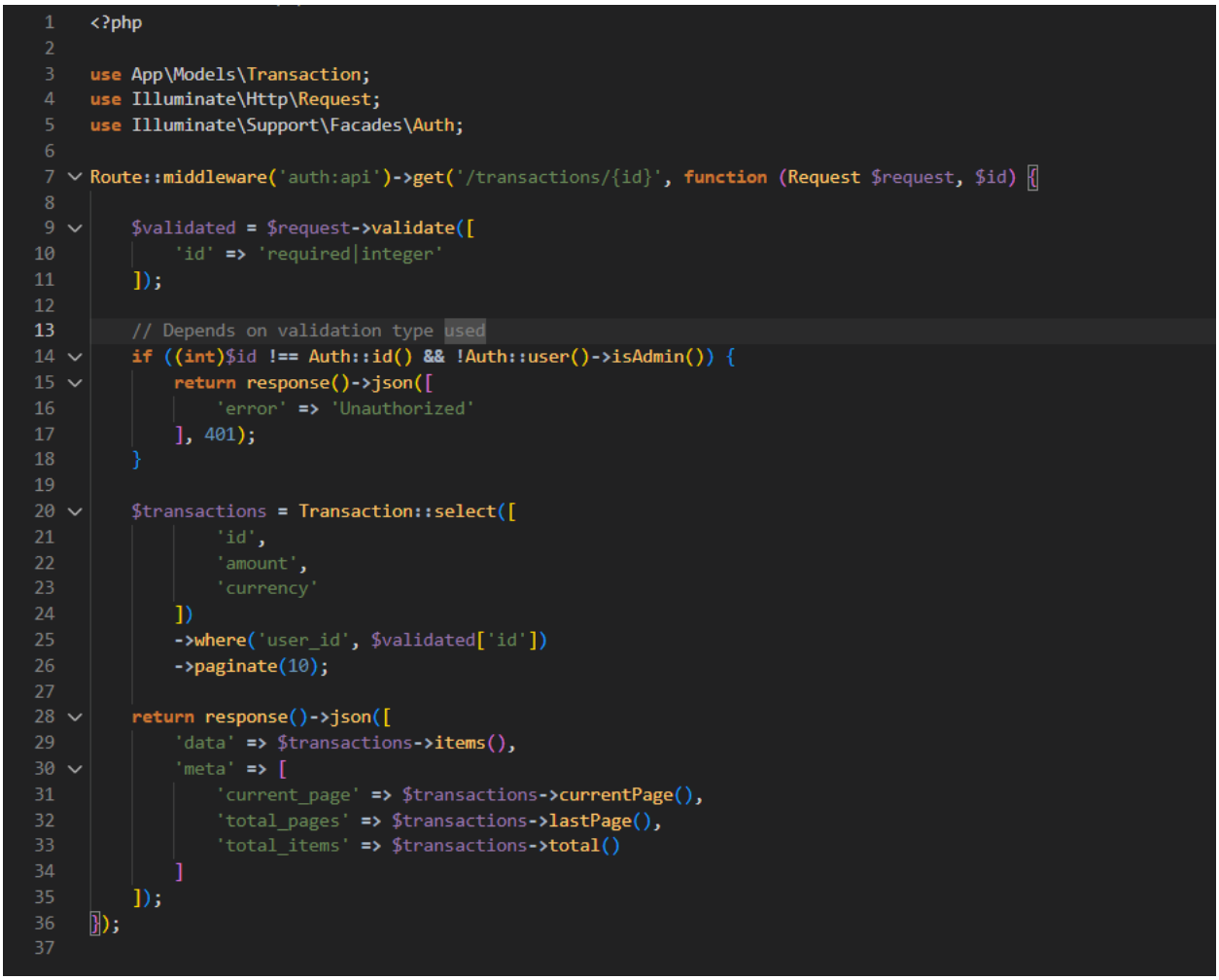
○ Handle edge cases gracefully.

Answer :

Security Flaws :

- No Authentication
- SQL injector risk : Raw user input ($id) used directly in query
- Fetch all data (no pagination for large datasets)
- No Error Handling

Secure Implementation



```php
<?php

use App\Models\Transaction;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

Route::middleware('auth:api')->get('/transactions/{id}', function (Request $request, $id) {

    $validated = $request->validate([
        'id' => 'required|integer'
    ]);

    // Depends on validation type used
    if ((int)$id !== Auth::id() && !Auth::user()->isAdmin()) {
        return response()->json([
            'error' => 'Unauthorized'
        ], 401);
    }

    $transactions = Transaction::select([
            'id',
            'amount',
            'currency'
        ])
        ->where('user_id', $validated['id'])
        ->paginate(10);

    return response()->json([
        'data' => $transactions->items(),
        'meta' => [
            'current_page' => $transactions->currentPage(),
            'total_pages' => $transactions->lastPage(),
            'total_items' => $transactions->total()
        ]
    ]);
});

```

File Url : https://github.com/hafizhkamus/speedgrow-tech-assessment/blob/master/speedgrow-ta-api/resources/section2/task1.php

Task 2: Debugging Challenge

Your frontend app fails to fetch user data, showing a blank screen. You get the following

error:

pgsql

CopyEdit

Error: Failed to load resource: the server responded with a status

of 500 (Internal Server Error)

1. How would you systematically debug this issue?

2. What are the possible causes, and how would you fix them?

3. How would you prevent similar issues in the future?

Answer :

Debugging Steps and Possible Causes

- Identify the Exact Error : Check if any response body, if blank like the example, can skip this
- Check log, mostly this error is because code error
- If not from code Error, can check DB log also
- Check database connection (this can also contain Status 500, but mostly will send response body)
- If the error still contain, you can debug on your local, step by step, and pretty sure you'll find the cause

How to fix

- Verify .env PostgreSQL credential
- Debug step by step in your local to see a mistake
- Verify database connection (if you already deploy), and see if API is running

How to prevent

- Logging & Monitoring
- Automated Testing
- Error Handling Middleware
- Frontend Fallback UI
- Database Health Checks

Task 3: Mobile App Performance Fix

Your team reports that the mobile app crashes when loading high-resolution images.

How would you:

● Identify the root cause?

● Implement a lazy loading mechanism for efficient performance?

● Ensure cross-platform compatibility (Android & iOS)?

Answer :

(Note : my answer is using Ionic base App)

Identifying the Root Cause

- Check Crash Logs
- Reproduce in Debug Mode
- Common Causes :
  Raw resulution image load
  No caching
  No lazy loading

Implementing Lazy Loading

- Using Ionic <ion-img> (Best Practice)

```
<ion-img [src]="image.url"
        (ionImgWillLoad)="log('Loading')"
        (ionError)="handleError()"
></ion-img>
```

- Custom Lazy Loading

```
<ion-list>
 <ion-item *ngFor="let item of items">
  <ion-img
   [src]="item.lowResUrl"
   (ionImgWillLoad)="loadHighRes(item)"
   [hidden]="item.highResLoaded"
  ></ion-img>
  <ion-img
   [src]="item.highResUrl"
   (ionImgWillLoad)="item.highResLoaded = true"
   [hidden]="!item.highResLoaded"
  ></ion-img>
 </ion-item>
</ion-list>
```

- Virtual Scrolling

```
<ion-content>
 <ion-list [virtualScroll]="items">
  <ion-item *virtualItem="let item">
   <ion-img [src]="item.thumbnailUrl"></ion-img>
  </ion-item>
 </ion-list>
</ion-content>
```

Cross-Platform Optimization

- Install Required Plugins (can use capacitor for IOS)
- Enable WKWebView (iOS)
- Configure Image Compression

Task 4: Secure API Implementation

Write a secure Laravel API endpoint to process NFC transactions.

Requirements:

■ Ensure data validation and transaction security.

■ Implement rate limiting to prevent abuse.

■ Handle edge cases and provide meaningful error responses.

Answer :

app > Http > Controllers > NfcTransactionController.php

```php
<?php

namespace App\Http\Controllers;

use App\Models\NfcTransaction;
use App\Models\NfcDevice;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Facades\Cache;
use Illuminate\Support\Facades\DB;

class NfcTransactionController extends Controller
{
    public function process(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'nfc_id' => [
                'required',
                'string',
                'size:64',
                'regex:/^[a-f0-9]+$/i',
                function ($attribute, $value, $fail) {
                    if (!NfcDevice::where('id', $value)->where('is_active', true)->exists()) {
                        $fail('The NFC device is invalid or inactive.');
                    }
                }
            ],
            'amount' => 'required|numeric|min:0.01|max:10000|decimal:0,2',
            'currency' => 'required|string|size:3|uppercase|in:USD,EUR,GBP',
            'metadata' => 'nullable|array',
            'metadata.*' => 'sometimes|string'
        ]);

        if ($validator->fails()) {
            return response()->json([
                'status' => 'error',
                'message' => 'Validation failed',
                'errors' => $validator->errors()
            ], 422);
        }

        $duplicateKey = 'nfc_transaction:'.$request->nfc_id.':'.round($request->amount, 2);
        if (Cache::has($duplicateKey)) {
            return response()->json([
                'status' => 'error',
```

File Url : https://github.com/hafizhkamus/speedgrow-tech-assessment/blob/master/speedgrow-ta-api/app/Http/Controllers/NfcTransactionController.php

Task 5: Debugging Challenge

An Ionic mobile app is intermittently failing to detect NFC tags. What steps would you take to

debug and resolve this issue?

■ Outline your debugging process for both frontend and backend layers.

■ Provide examples of potential fixes and testing strategies.

Answer :

1. Frontend Debugging

1.1 Check Basic NFC Functionality

```
checkNfcSupport() {
    this.nfc.enabled().then(
        () => console.log('NFC is enabled'),
        (err) => console.error('NFC not enabled', err)
    );
}
```

1.2 Verify Tag Detection

```
this.nfc.addTagListener().subscribe(
  tag => console.log('Tag detected', tag),
  err => console.error('Error reading tag', err)
);
```

1.3 Debugging Steps

- Check browser console

- Test with different NFC tag types

- Verify Android/iOS permissions

1.4 Common Frontend Fixes

```
// Fix 1: Add timeout for flaky detection
setTimeout(() => {
  this.nfc.addTagListener().subscribe(...);
}, 500);


// Fix 2: Handle multiple detection attempts
let isReading = false;
this.nfc.addTagListener().subscribe(tag => {
  if (!isReading) {
    isReading = true;
    this.processTag(tag).finally(() => isReading = false);
  }
});
```

2. Backend Debugging

2.1 Check API Logs

2.2 Validate Request Data

```
public function process(Request $request) {
        \Log::debug('NFC Request:', $request->all());
```

```
        $validated = $request->validate([
            'nfc_id' => 'required|size:64|regex:/^[a-f0-9]+$/i',
            'timestamp' => 'required|numeric'
        ]);
    }
```

2.3 Potential Backend Issues

- Timeout Errors: Increase timeout for NFC processing
- Database Locking

3. Testing Strategies

> Unit Test

```
it('should detect NFC tags', fakeAsync(() => {
    const mockTag = { id: 'abcd', type: 'NFC Forum Type 2' };
    spyOn(nfc, 'addTagListener').and.returnValue(of(mockTag));

    component.startNfcScan();
    tick(500);

    expect(component.lastTag).toEqual(mockTag);
}));
```

> End-to-End Tests

```
describe('NFC Flow', () => {
```

```
        it('should complete transaction', () => {
            device.enableNFC();
            device.simulateNFCTag('test-tag-123');
            expect(element(by.id('transaction-status')).getText()).toBe('Completed');
        });
    });
```

4. Frontend Solutions

> Add NFC Retry Logic

> Handle iOS Limitations

5. Backend Solutions

> Payload Validation

> Duplicate Request Handling