APPENDIX: Live Code Modifications & Teaching Comment Code Listings

Part A: Live Modification Tutorials (On-Demand)

Tutorial 1: Add Failed Login Counter (5-10 minutes live)

What to say before you start:

> "I'll add a feature to lock accounts after 5 failed login attempts. This demonstrates my understanding of database migrations, model attributes, and controller logic."

Step 1: Create Migration (1 minute)

Command:

```
php artisan make:migration add_failed_attempts_to_users_table
```

Edit the migration file (database/migrations/YYYY_MM_DD_HHMMSS_add_failed_attempts_to_users_table.php):

```
public function up()
{
    Schema::table('user', function (Blueprint $table) {
        $table->integer('failed_attempts')->default(0);
        $table->timestamp('locked_until')->nullable();
    });
}

public function down()
{
    Schema::table('user', function (Blueprint $table) {
        $table->dropColumn(['failed_attempts', 'locked_until']);
    });
}
```

Run migration:

```
php artisan migrate
```

Explain:

> "I added two columns:
>
> - `failed_attempts` : Counter (default 0)
> - `locked_until` : Timestamp when account unlocks (nullable)
>
> The `down()` method allows rollback if needed."

Step 2: Update User Model (1 minute)

Edit `app/Models/User.php` :

Add to `$fillable` array:

```
protected $fillable = [
    'first_name',
    'last_name',
    'email',
    'password',
    'failed_attempts',  // ADD THIS
    'locked_until',      // ADD THIS
];
```

Add to `$casts` array (if it doesn't exist, create it):

```
protected $casts = [
    'locked_until' => 'datetime',
];
```

Explain:

> "I made the columns fillable so we can mass-assign them. The cast to 'datetime' makes `locked_until` a Carbon object for easy date manipulation."

Step 3: Modify Login Method (3-5 minutes)

Edit `app/Http/Controllers/AuthController.php`:

Before the `Auth::attempt()` call, add:

```
// Check if account is locked
$user = User::where('email', strtolower($credentials['Email']))->first();

if ($user && $user->locked_until && $user->locked_until->isFuture()) {
    return back()->withErrors([
        'Email' => 'Account locked due to too many failed attempts. Try again at ' .
                   $user->locked_until->format('Y-m-d H:i:s')
    ])->onlyInput('Email');
}
```

In the `else` block (failed login), replace with:

```
else {
    $user = User::where('email', strtolower($credentials['Email']))->first();

    if ($user) {
        $user->failed_attempts += 1;

        if ($user->failed_attempts >= 5) {
            $user->locked_until = now()->addMinutes(15);
            $user->save();

            return back()->withErrors([
                'Email' => 'Too many failed attempts. Account locked for 15 minutes.'
            ])->onlyInput('Email');
        }

        $user->save();
    }

    return back()->withErrors([
        'Email' => 'The provided credentials do not match our records.'
    ])->onlyInput('Email');
}
```

In the success block (after `Auth::attempt()` returns true), add:

```
$user = Auth::user();
$user->failed_attempts = 0;
$user->locked_until = null;
$user->save();
```

Step 4: Test Live (1 minute)

Try logging in with wrong password 5 times, then show the lockout message.

Tutorial 2: Add Email Verification (10-15 minutes live)

What to say before you start:

> "I'll implement email verification for new registrations."

Step 1: Modify User Model (2 minutes)

Edit `app/Models/User.php`:

```php
use Illuminate\Contracts\Auth\MustVerifyEmail;

class User extends Authenticatable implements MustVerifyEmail
{
    // ... existing code
}
```

Step 2: Update Registration Controller (3 minutes)

After creating the user in register method:

```php
$user->sendEmailVerificationNotification();
return redirect('/email/verify')->with('message', 'Check your email to verify your account.');
```

Step 3: Add Verification Routes (2 minutes)

In `routes/web.php`:

```php
Route::get('/email/verify', function () {
    return view('auth.verify-email');
})->middleware('auth')->name('verification.notice');

Route::get('/email/verify/{id}/{hash}', function (EmailVerificationRequest $request) {
    $request->fulfill();
    return redirect('/home');
})->middleware(['auth', 'signed'])->name('verification.verify');
```

Step 4: Test

Show email in `storage/logs/laravel.log` and click verification link.

Part B: Teaching Comment Code Locations

Summary Table

File	Teaching Comments	Priority
`app/Http/Controllers/AuthController.php`	Lines 1-850	⭐⭐⭐ CRITICAL
`app/Http/Controllers/AdminAuthController.php`	Lines 1-720	⭐⭐ HIGH
`resources/views/auth/login.blade.php`	Lines 1-400	⭐⭐ HIGH
`app/Models/User.php`	Lines 45-50 (password hashing)	⭐⭐⭐ CRITICAL
`routes/web.php`	Scattered throughout	⭐ MEDIUM

Most Critical Code (Study First)

1. Login Method (AuthController.php, ~lines 95-130)

- Input validation
- Auth::attempt() usage
- Session regeneration
- Redirect behavior

2. Password Hashing (User.php, lines 45-50)

- Mutator pattern
- Hash::make() usage
- Security reasoning

3. Guard Usage (AdminAuthController.php, ~line 115)

- Auth::guard('admin')→attempt()
- Dual authentication system
- Guard configuration

4. CSRF Protection (login.blade.php, ~line 80)

- @csrf directive
- Form submission
- Security explanation

5. Validation (AuthController.php, ~line 100)

- Request validation rules
- Error handling
- Input preservation