The Answer:

1. A. code smell (Duplicated code and logic).

```php
class AuthController
{
    public function login(Request $request)
    {
        // Attempt logging
        if (!Auth::attempt(['email' => $request->email, 'password' => $request->password], true)) {
            return response()->json([
                'error' => 'The credentials provided do not match our records'
            ], 401);
        }

        $user = User::whereEmail($request->email)->first();
        $user = new UserResource($user);

        // code smells (duplicated code and logic)
        $tokenInfo = $this->attemptLogin('password', $request);
        $data = $this->data($tokenInfo, ['user' => $user]);
        return response()->json($data)
        ->withCookie('refresh_token', $tokenInfo->refresh_token, 864000, null, null, false, true);
        // end
    }

    public function refresh(Request $request)
    {
        // code smells (duplicated code and logic)
        $tokenInfo = $this->attemptRefresh('refresh_token', $request);
        $data = $this->data($tokenInfo);
        return response()->json($data)
        ->withCookie('refresh_token', $tokenInfo->refresh_token, 864000, null, null, false, true);
        // end
    }

    //etc
}
```

So, to get rid of this code smell we need to extract into a separate method.

```php
class AuthController
{
    public function login(Request $request)
    {
        // Attempt logging
        if (!Auth::attempt(['email' => $request->email, 'password' => $request->password])) {
            return response()->json([
                'error' => 'The credentials provided do not match our records'
            ], 401);
        }
        $user = User::whereEmail($request->email)->first();
        $user = new UserResource($user);

        // repair
        $attributes['username'] = $request['email'];
        $attributes['password'] = $request['password'];
        return $this->returnToken('password', $attributes, ['user' => $user]);
        // end
    }

    public function refresh()
    {
        // repair
        $attributes['refresh_token'] = Cookie::get('refresh_token');
        return $this->returnToken('refresh_token', $attributes);
        // end
    }

    protected function returnToken($grantType, $attributes, array $data = [])
    {
        $tokenInfo = $this->attemptGrantType($grantType, $attributes);
        $response = $this->data($tokenInfo, $data);

        return response()->json($response)
        ->withCookie('refresh_token', $tokenInfo->refresh_token, 864000, null, null, false, true);
    }

    protected function attemptGrantType($type, $attributes)
    {
        $tokenResponse = $this->proxy($type, $attributes);
        return json_decode($tokenResponse->getContent());
    }

    //etc
}
```

1. B.Dependency Injection
   Dependency injection is a technique whereby ne object supplies the dependencies of another object.
   Why we need Dependency Injection? “Sometimes we need to put application logic somewhere outside”.

2. A.GET
   Should : GET is used to retrieve data from a server at the specified resource.
   API with a /users endpoint, making a get request to that endpoint should return a list of all available users.
   Shouldn’t : A GET should never change data on the server, GET can be cached and

in a browser, refreshed, over and over, this means that if you make the same GET again, you will insert into your database ‘again’.

:

B.POST

Should: POST are used to send data to the API server to create a resource.

Shouldn’t: As POST is not idempotent, major browser will warn you if you send twice the same POST request which is not desirable in GET use cases.