

# Website Status Checker

- [URL Health Monitor — A minimalist WordPress plugin](#) [Claude Opus 4.5]
- [Short verdict](#) [ChatGPT 5.4]

# URL Health Monitor — A minimalist WordPress plugin ? [Claude Opus 4.5]

Below is a complete, single-file WordPress plugin that does exactly what you described. It's deliberately minimal while still being production-ready.

---

## Features at a glance

- **Simple backend UI** — a single textarea where you enter URLs (one per line)
  - **WP-Cron powered** — checks every hour by default
  - **Lightweight storage** — results saved in `wp_options`, auto-pruned after 365 days
  - **Frontend shortcode** — `[url_health_monitor]` outputs a clean, responsive status table
  - **Zero dependencies** — pure WordPress APIs, no external libraries
- 

## The plugin code

Create a file called `url-health-monitor.php` inside `wp-content/plugins/url-health-monitor/` and paste the following:

```
<?php
/**
 * Plugin Name: URL Health Monitor
 * Description: A minimalist plugin to monitor URL availability and display results via
shortcode.
 * Version: 1.0.0
 * Author: Your Name
 * License: GPL-2.0+
 */

if ( ! defined( 'ABSPATH' ) ) {
    exit;
```

```

}

class URL_Health_Monitor {

    const OPTION_URLS    = 'uhm_urls';
    const OPTION_RESULTS = 'uhm_results';
    const CRON_HOOK      = 'uhm_check_urls';
    const RETENTION_DAYS = 365;

    public function __construct() {
        // Admin hooks
        add_action( 'admin_menu', [ $this, 'add_admin_menu' ] );
        add_action( 'admin_init', [ $this, 'register_settings' ] );

        // Cron
        add_action( self::CRON_HOOK, [ $this, 'check_all_urls' ] );
        add_filter( 'cron_schedules', [ $this, 'add_cron_interval' ] );

        // Shortcode
        add_shortcode( 'url_health_monitor', [ $this, 'render_shortcode' ] );

        // Activation / Deactivation
        register_activation_hook( __FILE__, [ $this, 'activate' ] );
        register_deactivation_hook( __FILE__, [ $this, 'deactivate' ] );
    }

    /* -----
       Activation & Deactivation
       ----- */

    public function activate() {
        if ( ! wp_next_scheduled( self::CRON_HOOK ) ) {
            wp_schedule_event( time(), 'hourly', self::CRON_HOOK );
        }
    }

    public function deactivate() {
        wp_clear_scheduled_hook( self::CRON_HOOK );
    }
}

```

```

public function add_cron_interval( $schedules ) {
    // You can change 'hourly' above to 'uhm_fifteen_minutes' and uncomment below for
faster checks.
    // $schedules['uhm_fifteen_minutes'] = [
    //     'interval' => 900,
    //     'display' => __( 'Every 15 Minutes' ),
    // ];
    return $schedules;
}

/* -----
Admin Menu & Settings
----- */

public function add_admin_menu() {
    add_options_page(
        'URL Health Monitor',
        'URL Health Monitor',
        'manage_options',
        'url-health-monitor',
        [ $this, 'render_admin_page' ]
    );
}

public function register_settings() {
    register_setting( 'uhm_settings_group', self::OPTION_URLS, [
        'type' => 'string',
        'sanitize_callback' => [ $this, 'sanitize_urls' ],
        'default' => '',
    ] );
}

public function sanitize_urls( $input ) {
    $lines = explode( "\n", $input );
    $clean = [];
    foreach ( $lines as $line ) {
        $url = esc_url_raw( trim( $line ) );
        if ( $url ) {
            $clean[] = $url;
        }
    }
}

```

```

    }
    return implode( "\n", $clean );
}

public function render_admin_page() {
    if ( ! current_user_can( 'manage_options' ) ) {
        return;
    }
    $urls = get_option( self::OPTION_URLS, '' );
    ?>
    <div class="wrap" style="max-width:720px;">
        <h1>URL Health Monitor</h1>
        <form method="post" action="options.php">
            <?php settings_fields( 'uhm_settings_group' ); ?>
            <p>Enter the URLs you want to monitor, <strong>one per line</strong>:</p>
            <textarea name="<?php echo esc_attr( self::OPTION_URLS ); ?>" rows="10"
style="width:100%;font-family:monospace;"><?php echo esc_textarea( $urls ); ?></textarea>
            <?php submit_button( 'Save URLs' ); ?>
        </form>

        <hr>
        <h2>Current status</h2>
        <?php $this->render_admin_status_table(); ?>

        <hr>
        <h2>Shortcode</h2>
        <p>Use <code>[url_health_monitor]</code> to display the status table on the
frontend.</p>
    </div>
    <?php
}

private function render_admin_status_table() {
    $results = $this->get_results();
    if ( empty( $results ) ) {
        echo '<p><em>No data yet. Results will appear after the first scheduled check
(runs hourly).</em></p>';
        echo '<form method="post">';
        wp_nonce_field( 'uhm_manual_check', 'uhm_nonce' );
        echo '<button type="submit" name="uhm_run_now" class="button">Run check

```

```

now</button>';
    echo '</form>';

    if ( isset( $_POST['uhm_run_now'] ) && check_admin_referer( 'uhm_manual_check',
'uhm_nonce' ) ) {
        $this->check_all_urls();
        echo '<meta http-equiv="refresh" content="0">';
    }
    return;
}

echo '<form method="post">';
wp_nonce_field( 'uhm_manual_check', 'uhm_nonce' );
echo '<button type="submit" name="uhm_run_now" class="button" style="margin-
bottom:12px;">Run check now</button>';
echo '</form>';

if ( isset( $_POST['uhm_run_now'] ) && check_admin_referer( 'uhm_manual_check',
'uhm_nonce' ) ) {
    $this->check_all_urls();
    echo '<meta http-equiv="refresh" content="0">';
}

$this->render_status_table( $results, false );
}

/* -----
URL Checking Logic
----- */

public function check_all_urls() {
    $raw = get_option( self::OPTION_URLS, '' );
    if ( empty( $raw ) ) {
        return;
    }

    $urls    = array_filter( array_map( 'trim', explode( "\n", $raw ) ) );
    $results = $this->get_results();
    $now     = current_time( 'timestamp' );

```

```

foreach ( $urls as $url ) {
    $response = wp_remote_head( $url, [
        'timeout'      => 15,
        'redirection' => 5,
        'sslverify'    => false,
    ] );

    $status_code = 0;
    $is_up        = false;

    if ( is_wp_error( $response ) ) {
        $status_code = 0;
    } else {
        $status_code = wp_remote_retrieve_response_code( $response );
        $is_up        = ( $status_code >= 200 && $status_code < 400 );
    }

    if ( ! isset( $results[ $url ] ) ) {
        $results[ $url ] = [ 'history' => [] ];
    }

    $results[ $url ]['last_checked'] = $now;
    $results[ $url ]['status_code']  = $status_code;
    $results[ $url ]['is_up']        = $is_up;
    $results[ $url ]['history'][]    = [
        'time'     => $now,
        'status'   => $status_code,
        'is_up'    => $is_up,
    ];
}

// Prune old history & orphan URLs
$results = $this->prune_results( $results, $urls );

update_option( self::OPTION_RESULTS, $results );
}

private function prune_results( $results, $current_urls ) {
    $cutoff = current_time( 'timestamp' ) - ( self::RETENTION_DAYS * DAY_IN_SECONDS );

```

```

foreach ( $results as $url => &$data ) {
    // Remove URLs no longer monitored
    if ( ! in_array( $url, $current_urls, true ) ) {
        unset( $results[ $url ] );
        continue;
    }

    // Remove history entries older than retention period
    $data['history'] = array_filter( $data['history'], function ( $entry ) use (
$cutoff ) {
        return $entry['time'] >= $cutoff;
    } );
    $data['history'] = array_values( $data['history'] ); // reindex
}

return $results;
}

private function get_results() {
    return get_option( self::OPTION_RESULTS, [] );
}

/* -----
Frontend Shortcode
----- */

public function render_shortcode( $atts ) {
    $results = $this->get_results();
    if ( empty( $results ) ) {
        return '<p>No monitoring data available yet.</p>';
    }

    ob_start();
    $this->render_status_table( $results, true );
    return ob_get_clean();
}

private function render_status_table( $results, $is_frontend ) {
    $border_color = '#e0e0e0';
    $sup_color    = '#2e7d32';

```

```

$down_color = '#c62828';
?>
<style>
    .uhm-table { width:100%; border-collapse:collapse; font-family: -apple-system,
BlinkMacSystemFont, "Segoe UI", Roboto, Oxygen, Ubuntu, sans-serif; font-size:14px; }
    .uhm-table th, .uhm-table td { padding:10px 12px; text-align:left; border-
bottom:1px solid <?php echo $border_color; ?>; }
    .uhm-table th { background:#f7f7f7; font-weight:600; }
    .uhm-table tr:last-child td { border-bottom:none; }
    .uhm-status { display:inline-block; padding:3px 10px; border-radius:4px; font-
weight:500; font-size:12px; }
    .uhm-status.up { background:#e8f5e9; color:<?php echo $up_color; ?>; }
    .uhm-status.down { background:#ffebee; color:<?php echo $down_color; ?>; }
    .uhm-url { word-break:break-all; }
    .uhm-time { color:#666; font-size:12px; }
    .uhm-uptime { font-size:12px; color:#555; }
</style>
<table class="uhm-table">
    <thead>
        <tr>
            <th>URL</th>
            <th>Status</th>
            <th>Uptime (<?php echo self::RETENTION_DAYS; ?>d)</th>
            <th>Last checked</th>
        </tr>
    </thead>
    <tbody>
        <?php foreach ( $results as $url => $data ) :
            $is_up      = ! empty( $data['is_up'] );
            $status_code = isset( $data['status_code'] ) ? $data['status_code'] : '-';
            $last_time  = isset( $data['last_checked'] ) ? human_time_diff(
$data['last_checked'], current_time( 'timestamp' ) ) . ' ago' : '-';

            // Calculate uptime %
            $history = isset( $data['history'] ) ? $data['history'] : [];
            $total   = count( $history );
            $up_count = 0;
            foreach ( $history as $h ) {
                if ( ! empty( $h['is_up'] ) ) {
                    $up_count++;

```

```

    }
}
$uptime = $total > 0 ? round( ( $up_count / $total ) * 100, 1 ) : 0;
?>
<tr>
    <td class="uhm-url"><?php echo esc_html( $url ); ?></td>
    <td>
        <span class="uhm-status <?php echo $is_up ? 'up' : 'down'; ?>">
            <?php echo $is_up ? '✓ Online' : '✗ Offline'; ?>
        </span>
        <span style="margin-left:6px;color:#888;font-size:12px;"><?php echo
esc_html( $status_code ); ?></span>
    </td>
    <td class="uhm-uptime"><?php echo esc_html( $uptime ); ?>%</td>
    <td class="uhm-time"><?php echo esc_html( $last_time ); ?></td>
</tr>
<?php endforeach; ?>
</tbody>
</table>
<?php
}
}
new URL_Health_Monitor();

```

## How to install

1. Create the folder `wp-content/plugins/url-health-monitor/`
2. Save the code above as `url-health-monitor.php` inside that folder
3. Activate the plugin via **Plugins → Installed Plugins**

## How to use

### 1. Configure URLs

Go to **Settings → URL Health Monitor**. Enter the URLs you want to watch (one per line) and click *Save URLs*.

## 2. Wait or trigger manually

The plugin checks URLs every hour via WP-Cron. You can also click *Run check now* in the admin screen to trigger an immediate check.

## 3. Display on the frontend

Insert the shortcode anywhere:

```
[url_health_monitor]
```

It will output a clean status table showing each URL's current state, HTTP status code, uptime percentage, and last-checked time.

# Notes on WP-Cron ??

WordPress cron is *visitor-triggered* by default — it only fires when someone visits your site. If your site has low traffic, consider adding a real server cron job:

```
*/15 * * * * curl -s https://yoursite.com/wp-cron.php?doing_wp_cron > /dev/null 2>&1
```

Then disable the built-in trigger by adding this to `wp-config.php`:

```
define( 'DISABLE_WP_CRON', true );
```

# Customisation ideas ?

Want to change...	Where to look
Check interval	Change <code>'hourly'</code> in <code>activate()</code> to <code>'twicedaily'</code> , <code>'daily'</code> , or register your own schedule via <code>add_cron_interval()</code>
Retention period	Adjust <code>RETENTION_DAYS</code> constant
Styling	Modify the <code>&lt;style&gt;</code> block inside <code>render_status_table()</code>
Timeout / SSL verification	Tweak the <code>wp_remote_head()</code> args in <code>check_all_urls()</code>

That's it — a clean, minimal URL health monitor in under 300 lines of code. ☐☐

# Short verdict [ChatGPT 5.4]

I'd rate it **about 6.5/10** overall ☐☐

It's a **good minimalist prototype** and it does cover the core requirements:

1. backend textarea for URLs
2. periodic checks via WP-Cron
3. saving results
4. frontend shortcode output
5. pruning data older than 365 days

But as a real plugin for production use, it has a few **important weaknesses and a couple of actual bugs / design issues**.

---

## What's good

The plugin already does several things well:

1. **Very simple UX**
    - The admin UI is easy to understand.
    - One URL per line is exactly the right minimalist input format.
  2. **Uses core WordPress APIs**
    - `register_setting()`
    - `wp_schedule_event()`
    - `wp_remote_head()`
    - shortcode API
  3. **Keeps data structure simple**
    - Storing current status plus history per URL is easy to reason about.
  4. **Frontend is lightweight**
    - Clean table output
    - No external CSS/JS dependencies
  5. **Retention logic exists**
    - Old history is pruned, which is important.
- 

## The main issues

# 1. Activation hook inside the class constructor is problematic

This is one of the biggest issues.

You register:

```
register_activation_hook( __FILE__, [ $this, 'activate' ] );  
register_deactivation_hook( __FILE__, [ $this, 'deactivate' ] );
```

inside `__construct()`.

This often *works*, but it's not the cleanest or safest pattern. Activation hooks should ideally be registered in the main plugin file scope, not buried in runtime object construction. It can lead to surprises depending on load timing and plugin structure.

## Better

Use static methods and register them outside the instantiated object:

```
register_activation_hook( __FILE__, [ 'URL_Health_Monitor', 'activate' ] );  
register_deactivation_hook( __FILE__, [ 'URL_Health_Monitor', 'deactivate' ] );
```

and make those methods `public static`.

---

# 2. `sslverify => false` is a bad security practice

This is a clear red flag ☹️

```
'sslverify' => false,
```

Disabling SSL verification means HTTPS checks are less trustworthy and less secure. Unless there is a very specific reason, this should **not** be disabled.

## Better

Just remove it, or explicitly set:

```
'sslverify' => true,
```

### 3. `wp_remote_head()` alone is not reliable

Many websites do **not** respond correctly to `HEAD` requests. Some block them, some misconfigure them, and some return different status codes than for `GET`.

That means the plugin may report false negatives.

## Better approach

1. Try `HEAD` first
2. If that fails or returns an unusable response, fall back to a lightweight `GET`

For example:

```
$response = wp_remote_head( $url, $args );

if ( is_wp_error( $response ) || wp_remote_retrieve_response_code( $response ) === 0 ) {
    $response = wp_remote_get( $url, $args );
}
```

## 4. Manual check form processing is done after output starts

In `render_admin_status_table()`, the form submission is handled after some HTML is already printed, and then a meta refresh is used:

```
echo '<meta http-equiv="refresh" content="0">';
```

This is clunky and not ideal.

# Why this is not great

- not a proper redirect
- mixes display logic with request handling
- can be unreliable
- not very WordPress-like

## Better

Handle the action via:

1. `admin_post_*`
  2. or detect submit at top of admin page and use `wp_safe_redirect()`
- 

# 5. Old data is only pruned when checks run

The requirement says data older than 365 days should be deleted. Technically this happens, but only during `check_all_urls()`.

## Edge case

If cron doesn't run for a while, old data stays longer than intended.

## Better

Prune in more places, for example:

1. before rendering frontend
2. before rendering admin
3. during scheduled checks

That makes retention more consistent.

---

# 6. Inline CSS is output every time the shortcode is used

The `<style>` block is embedded directly in shortcode output.

This is okay for a tiny plugin, but not ideal.

## Downsides

1. duplicate CSS if shortcode appears multiple times
2. less cacheable
3. harder to override in themes

## Better

Register/enqueue a small stylesheet.

---

## 7. No uninstall cleanup

On deactivation, cron is cleared, but plugin options remain forever.

That may be okay for some plugins, but usually it's nice to offer cleanup on uninstall.

## Better

Add `uninstall.php` or `register_uninstall_hook()` to remove:

1. `uhm_urls`
2. `uhm_results`

if complete cleanup is desired.

---

## 8. Storing all history in one option may become inefficient

This is an architectural limitation.

Right now everything is stored in one option:

```
uhm_results
```

For a small number of URLs and hourly checks, this is fine. But over time:

- lots of URLs
- 365 days of history
- repeated writes to one large serialized option

can become inefficient.

## Rough example

If you monitor many URLs hourly for a year, history can get quite large.

## Better options

1. keep only summary stats in options
2. store history in a custom database table
3. limit history granularity

For a minimalist plugin, this is not a deal-breaker, but it is a scalability concern.

---

# 9. Uptime calculation is simplistic

Current uptime is based on:

“ number of successful checks / total checks

That's acceptable, but not a true time-weighted uptime calculation.

## Why it matters

If cron is irregular, the uptime number may be misleading.

For example:

1. one check succeeds
2. site is down for 10 hours
3. next check succeeds

The uptime percentage may still look artificially good.

# For a minimalist plugin

This is okay. But it should be understood as **check success rate**, not perfect uptime monitoring.

You may want to label it more honestly.

---

## 10. Removed URLs are deleted immediately, including history

In `prune_results()`, any URL no longer present in the textarea is removed entirely:

```
if ( ! in_array( $url, $current_urls, true ) ) {  
    unset( $results[ $url ] );  
    continue;  
}
```

That may be intended, but it's worth noting.

## Potential issue

A user may accidentally delete a URL from the textarea, save, and lose all stored history immediately.

## Possible improvement

Keep removed URLs for a grace period, or have an “active/inactive” state.

For a minimalist plugin, immediate deletion may still be acceptable.

---

## Actual bugs or likely bugs

1. `unset()` inside `foreach` by reference is risky

You have:

```
foreach ( $results as $url => &$data ) {
    if ( ! in_array( $url, $current_urls, true ) ) {
        unset( $results[ $url ] );
        continue;
    }
    ...
}
```

This can work, but mixing `foreach` by reference with unsetting array elements is a bit risky and can cause subtle issues.

## Better

Loop without reference and assign back explicitly, or build a new pruned array.

For example:

```
$pruned = [];

foreach ( $results as $url => $data ) {
    if ( ! in_array( $url, $current_urls, true ) ) {
        continue;
    }

    $data['history'] = array_values(array_filter($data['history'], function ($entry) use
($cutoff) {
        return ! empty($entry['time']) && $entry['time'] >= $cutoff;
    }));

    $pruned[$url] = $data;
}

return $pruned;
```

---

## 2. No fallback if cron event already exists with a different schedule

On activation you only schedule if not already scheduled:

```
if ( ! wp_next_scheduled( self::CRON_HOOK ) ) {  
    wp_schedule_event( time(), 'hourly', self::CRON_HOOK );  
}
```

If later you change the interval in code, existing installs may keep the old schedule.

## Better

Unschedule and reschedule when needed, especially on version upgrades.

---

### 3. Status code `0` is displayed for transport errors without message

If a request fails with a `WP_Error`, you store:

```
$status_code = 0;
```

That means the frontend says “Offline (0)”, which is not very informative.

## Better

Store the error message too:

```
'error' => $response->get_error_message()
```

Then display something like:

- **Offline**
- `Timeout`
- `Could not resolve host`
- `SSL certificate problem`

That would make the plugin much more useful.

---

## 4. Potentially no data until cron fires, which confuses users

The plugin says results will appear after the first scheduled check. That's true, but in WordPress, cron may not fire quickly on low-traffic sites.

### Better UX

Run an initial check on activation, or after saving URLs.

That would significantly improve first-run experience.

---

## 5. Processing POST in rendering function is messy

Not exactly a fatal bug, but definitely a code smell.

The admin rendering function both:

1. outputs HTML
2. validates nonce
3. processes POST
4. triggers checks

This should be separated.

---

## Best improvements I'd recommend first

If I were improving this plugin while keeping it minimalist, I'd prioritize these:

1. **Remove** `sslverify => false`
2. **Add HEAD → GET fallback**
3. **Process manual checks cleanly with redirect**
4. **Store error messages, not just status code** `0`
5. **Prune data before display as well**
6. **Move activation/deactivation hooks outside constructor**
7. **Run checks when URLs are saved or on activation**

## 8. Avoid storing large long-term history in one option if scale matters

---

# A cleaner rating by category

Area	Rating	Comment
Simplicity	9/10	Very easy to understand
WordPress integration	7/10	Mostly good, some structural issues
Security	5/10	<code>sslverify =&gt; false</code> hurts the score
Reliability	6/10	HEAD-only checks can misreport status
Scalability	5/10	Single-option history storage is limited
UX	7/10	Good basic UX, but first-run experience could improve
Maintainability	6/10	Reasonably readable, but request handling and hook registration could be cleaner

---

## Overall recommendation

**As a minimal MVP:** yes, it's solid enough.

**As a production-ready plugin:** not yet.

If you want, I can do one of these next:

1. **provide an improved version of the plugin**
2. **rewrite it in a more production-ready way while keeping it minimalist**
3. **convert storage from options to a custom database table**
4. **add nicer frontend cards/table styling without making it bloated**

If you want, I can now give you a **revised and improved plugin version** based on the points above.