

# LinkShorter v2 – Ausführliche Anleitung

## 1. Überblick & Architektur

LinkShorter ist ein selbstgehosteter URL-Shortener, der vollständig in **PHP** geschrieben ist und eine **SQLite**-Datenbank verwendet – es wird also kein MySQL/MariaDB-Server benötigt. Die gesamte Anwendung besteht aus einer einzigen Einstiegsdatei ([index.php](#)), die als **Front-Controller** fungiert: Alle HTTP-Requests werden über die [.htaccess](#) per `mod_rewrite` an diese Datei weitergeleitet.

## Technische Kernkomponenten

Komponente	Datei	Verantwortung
Routing & Controller	<a href="#">index.php</a>	Einziger Entry-Point, wertet <code>\$_GET['route']</code> aus
Datenbank	<a href="#">includes/db.php</a>	SQLite-Verbindung via PDO, Schema-Migration
Authentifizierung	<a href="#">includes/auth.php</a>	Session-basiertes Login
Slug-Generierung	<a href="#">includes/slug.php</a>	Zufällige Kurzlinks via SHA-256
Link-Logik	<a href="#">includes/links.php</a>	CRUD, Click-Recording, Batch-Import
OpenGraph	<a href="#">includes/opengraph.php</a>	Metadaten-Extraktion von Ziel-URLs
QR-Code	<a href="#">includes/qrcode.php</a>	Eigene QR-Code-Implementierung (kein externer Service!)

## Datenfluss bei einem Kurzlink-Aufruf

```
Browser → Apache (.htaccess Rewrite)
        → index.php?route=mein-slug
```

```
→ getLinkBySlug("mein-slug")
→ Prüfung: aktiv? abgelaufen? Passwort? Crawler?
→ recordClick() → HTTP 302 Redirect → Ziel-URL
```

## 2. Installation & Konfiguration

### Voraussetzungen

- PHP  $\geq$  7.4 mit aktivierten Extensions: `pdo_sqlite`, `gd` (für PNG-QR-Codes), `mbstring`
- Apache mit `mod_rewrite` aktiviert
- Schreibrechte auf das Verzeichnis `data/`

### Schritt 1: Dateien hochladen

Laden Sie alle Dateien auf Ihren Webserver hoch. Die Verzeichnisstruktur sollte so aussehen:

```
/
├─ index.php
├─ config.php
├─ .htaccess
├─ assets/
│  └─ style.css
│  └─ app.js
├─ includes/
│  └─ db.php
│  └─ auth.php
│  └─ slug.php
│  └─ links.php
│  └─ opengraph.php
│  └─ qrcode.php
├─ templates/
│  └─ dashboard.php
│  └─ edit.php
│  └─ login.php
│  └─ stats.php
│  └─ settings.php
```

```
| └─ password.php
| └─ og_proxy.php
| └─ unavailable.php
| └─ 404.php
└─ data/          ← wird automatisch erstellt
    └─ linkshorter.db
    └─ qr_cache/
    └─ qr_icon.svg
```

## Schritt 2: Konfiguration anpassen

Öffnen Sie [config.php](#) und passen Sie die Werte an:

```
<?php
define('ADMIN_USERNAME', 'admin');           // Ihr gewünschter Benutzername
define('ADMIN_PASSWORD', 'IhrSicheresPasswort'); // UNBEDINGT ÄNDERN!
define('BASE_URL', 'https://kurz.example.com/'); // Ihre Domain mit abschließendem /
define('DB_PATH', __DIR__ . '/data/linkshorter.db');
define('SITE_TITLE', 'LinkShorter');
define('QR_ICON_PATH', __DIR__ . '/data/qr_icon.svg');
```

**Technischer Hintergrund:** Die Konstanten werden mit `define()` festgelegt und sind damit **global** in allen inkludierten Dateien verfügbar. [ADMIN\\_PASSWORD](#) wird im Klartext gespeichert - es wird **nicht** gehasht, da es bei jedem Login direkt verglichen wird (in [attemptLogin](#)). Dies ist ein bewusster Trade-off für Einfachheit bei einer Single-User-Anwendung.

## Schritt 3: Erster Aufruf

Beim ersten Aufruf von `https://kurz.example.com/` erstellt [getDB](#) automatisch:

- Das `data/`-Verzeichnis
- Die SQLite-Datenbank mit den Tabellen `links`, `clicks` und `settings`

Die Funktion nutzt **WAL-Modus** (`PRAGMA journal_mode=WAL`), was parallele Lese- und Schreibzugriffe ermöglicht und die Performance bei gleichzeitigen Zugriffen deutlich verbessert.

---

# 3. Das Admin-Dashboard

## Login

Rufen Sie `https://kurz.example.com/?page=login` auf. Sie werden zum Login weitergeleitet, da die Route `?page=login` aktiv wird. Die Authentifizierung funktioniert **session-basiert**:

1. `attemptLogin` vergleicht die Eingaben mit den Konstanten aus `config.php`
2. Bei Erfolg wird `$_SESSION['logged_in'] = true` gesetzt
3. Alle geschützten Seiten prüfen via `requireLogin`, ob die Session gültig ist

## Dashboard-Oberfläche

Nach dem Login sehen Sie das Dashboard (`dashboard.php`) mit:

- **Link-Erstellungsformular** (oben)
- **Suchleiste** mit Live-Filterung
- **Link-Tabelle** mit Sortierung nach Slug, URL, Klicks oder Erstellungsdatum

**Technischer Hintergrund zur Sortierung:** Die Funktion `getAllLinks` baut die SQL-Query dynamisch zusammen. Erlaubte Spalten sind in einem Whitelist-Array definiert, um **SQL-Injection** zu verhindern:

```
$allowed = ['slug', 'url', 'clicks', 'created_at', 'active'];  
if (!in_array($sort, $allowed)) $sort = 'created_at';
```

Die Sortierrichtung wird ebenfalls validiert (`ASC` oder `DESC`), bevor sie in die Query eingebaut wird.

# 4. Links erstellen & verwalten

## Einzelnen Link erstellen

1. Geben Sie die **Ziel-URL** ein (Pflichtfeld)
2. Optional: **Custom Slug** – wird automatisch bereinigt (nur `a-zA-Z0-9_-` erlaubt)
3. Optional: **Passwort** – wird mit `password_hash()` als bcrypt-Hash gespeichert
4. Optional: **Ablaufdatum** – als `datetime-local`

5. Optional: **Max Clicks** – nach Erreichen wird der Link deaktiviert

**Technischer Hintergrund zur Slug-Generierung:** Wenn kein Custom Slug angegeben wird, generiert [generateSlug](#) einen 6-Zeichen-Code. Der Algorithmus ist bewusst kryptographisch robust:

```
Eingabe = microtime() + random_bytes(8) + Versuchsnummer
↓
SHA-256 Hash
↓
6 Zeichen aus dem Zeichensatz [a-zA-Z0-9] extrahiert
↓
Kollisionsprüfung gegen Datenbank (max 100 Versuche)
```

Die Kombination aus `microtime()` (Zeitstempel mit Mikrosekunden) und `random_bytes()` (kryptographisch sichere Zufallsbytes) macht Kollisionen extrem unwahrscheinlich. Bei 62 möglichen Zeichen pro Position ergibt ein 6-Zeichen-Slug  $62^6 \approx 56,8$  Milliarden mögliche Kombinationen.

## Batch-Import

Klicken Sie auf „**Batch Import**“ im Dashboard. Es öffnet sich ein Modal-Dialog, in dem Sie eine URL pro Zeile eingeben können. [batchCreateLinks](#) verarbeitet jede Zeile einzeln:

1. Trennung nach Zeilenumbrüchen und Trimming
2. URL-Validierung via `filter_var($url, FILTER_VALIDATE_URL)`
3. Automatische Slug-Generierung für jede URL
4. Ergebnisanzeige mit Erfolgs-/Fehlerstatus

## Link bearbeiten

Auf der Edit-Seite ([edit.php](#)) können Sie alle Eigenschaften eines Links ändern. Besonders interessant ist die **Passwort-Verwaltung** mit drei Optionen:

Auswahl	<code>password_action</code>	Verhalten
Behalten / Kein Passwort	<code>keep</code>	<code>password</code> wird aus <code>\$data</code> entfernt
Neues setzen	<code>change</code>	Neues Passwort wird bcrypt-gehasht
Entfernen	<code>remove</code>	<code>password</code> wird auf <code>''</code> gesetzt → in <a href="#">updateLink</a> wird es zu <code>null</code>

**Technischer Hintergrund:** In [updateLink](#) wird die OpenGraph-Daten-Aktualisierung **nur** ausgelöst, wenn sich die URL geändert hat. Das verhindert unnötige HTTP-Requests an die Ziel-URL:

```
$og = ($url !== $link['url']) ? fetchOpenGraph($url) : [  
  'og_title' => $link['og_title'],  
  'og_description' => $link['og_description'],  
  'og_image' => $link['og_image'],  
];
```

## Link löschen

Das Löschen erfolgt via POST-Request mit einer JavaScript-Bestätigung. Dank `ON DELETE CASCADE` in der Datenbank-Schema-Definition werden zugehörige Click-Datensätze automatisch mitgelöscht.

---

# 5. Passwortgeschützte Links

Wenn ein Link mit Passwort versehen ist, zeigt der Server die Seite [password.php](#) an. Der Ablauf:

```
Besucher → /mein-slug  
  → getLinkBySlug() findet Link mit password ≠ null  
  → GET-Request: Zeige Passwort-Formular  
  → POST-Request mit link_password:  
    → password_verify(eingabe, hash) → true: recordClick + Redirect  
    → false: Fehlermeldung
```

**Wichtig:** Das Passwort wird **immer** als bcrypt-Hash gespeichert (`password_hash($password, PASSWORD_DEFAULT)`). Beim Vergleich wird `password_verify()` verwendet, was automatisch den Salt aus dem Hash extrahiert. Das bedeutet: Selbst wenn die Datenbank kompromittiert wird, sind die Link-Passwörter nicht im Klartext einsehbar.

---

## 6. QR-Codes

LinkShorter enthält eine **vollständig eigene QR-Code-Implementierung** – es werden keine externen APIs oder Libraries benötigt.

# Technischer Tiefgang: QR-Code-Generierung

Die Funktion [qrEncode](#) implementiert den kompletten QR-Code-Standard:

- Versionswahl:** Basierend auf der Datenlänge wird die minimale QR-Version (1–40) bestimmt. Version 1 hat 21×21 Module, Version 40 hat 177×177 Module. Die Kapazitätstabelle `$capacityL` enthält die maximale Byte-Kapazität für **Error Correction Level L** (Low, ~7% Fehlerkorrektur).
- Matrix-Aufbau:**
  - [placeFinderPattern](#): Drei 7×7-Erkennungsmuster in den Ecken (oben-links, oben-rechts, unten-links)
  - Timing-Patterns: Abwechselnde Module in Zeile 6 und Spalte 6
  - [placeAlignmentPattern](#): Ab Version 2 werden Ausrichtungsmuster platziert (Positionen aus [getAlignmentPatternPositions](#))
- Datenkodierung** ([encodeData](#)):
  - Mode Indicator: `0100` (Byte-Modus)
  - Zeichenzähler: 8 Bit (Version 1–9) oder 16 Bit (ab Version 10)
  - Daten als 8-Bit-Bytes
  - Padding mit `0xEC 0x11` (abwechselnd)
  - Reed-Solomon-Fehlerkorrektur via [generateECCodewords](#)
- Galois-Feld-Arithmetik:** Für die Fehlerkorrektur werden Berechnungen im **GF(2<sup>8</sup>)** durchgeführt. Die Funktionen [gfMultiply](#), [gfExp](#) und [gfLog](#) implementieren die Multiplikation über das irreduzible Polynom `0x11D` ( $x^8 + x^4 + x^3 + x^2 + 1$ ).
- Data-Interleaving:** Bei mehreren Blöcken werden die Daten- und EC-Codewords interleaved (verschachtelt), um Burst-Fehler besser zu korrigieren.
- Maskierung:** Alle 8 Maskmuster werden getestet. [calculatePenalty](#) berechnet die Strafpunkte nach vier Regeln:
  - Fünf oder mehr gleiche Module in einer Reihe
  - 2×2-Blöcke gleicher Module
  - Spezielle Muster (1:1:3:1:1)
  - Verhältnis dunkler zu heller Module nahe 50%Das Mask-Pattern mit der **niedrigsten Penalty** wird gewählt.

## Ausgabeformate

- **PNG** ([getQRCodePNG](#)): Erzeugt via GD-Library (`imagecreatetruecolor`), mit optionalem Center-Icon via ImageMagick (`convert`-Befehl)
- **SVG** ([getQRCodeSVG](#)): Reine XML-Generierung, Icon wird als eingebettetes SVG eingefügt

# Caching

Beide Formate werden im Verzeichnis `data/qr_cache/` gecacht (24 Stunden TTL). Der Cache-Key ist ein MD5-Hash aus `Daten + Größe + Format`.

## Custom Icon

Unter **Settings** können Sie ein SVG-Icon hochladen ([admin/action mit action=upload\\_icon](#)), das im Zentrum aller QR-Codes angezeigt wird. Beim Upload wird der QR-Cache geleert, damit die neuen QR-Codes das Icon enthalten.

## 7. OpenGraph-Proxying

Wenn ein Kurzlink in sozialen Netzwerken geteilt wird, erkennt LinkShorter den Crawler anhand des User-Agents:

```
$isCrawler = preg_match(
    '/facebookexternalhit|Twitterbot|LinkedInBot|WhatsApp|Slackbot|TelegramBot|Discordbot|bot|crawler|spider/i',
    $ua
);
```

Statt den Crawler weiterzuleiten, wird [og\\_proxy.php](#) ausgeliefert – eine minimale HTML-Seite mit den **OpenGraph-Meta-Tags** der Ziel-URL. Das bewirkt, dass in der Vorschau (z.B. auf Facebook, Twitter, Slack) das **Bild, der Titel und die Beschreibung der Original-Seite** angezeigt werden, obwohl der geteilte Link eine kurze URL ist.

**Technischer Hintergrund:** [fetchOpenGraph](#) extrahiert beim Erstellen eines Links die Meta-Daten:

1. HTTP-Request an die Ziel-URL mit 10-Sekunden-Timeout
2. HTML-Parsing via `DOMDocument`
3. Extraktion von `og:title`, `og:description`, `og:image`
4. Fallback auf `<title>`-Tag, wenn kein `og:title` vorhanden

Die SSL-Verifikation ist bewusst deaktiviert (`verify_peer => false`), um Probleme mit selbstsignierten Zertifikaten zu vermeiden. In Produktionsumgebungen sollte das ggf. angepasst werden.

**Wichtig:** Das OpenGraph-Proxying funktioniert auch für passwortgeschützte Links – Crawler erhalten die Vorschau, ohne ein Passwort eingeben zu müssen. Normale Benutzer werden weiterhin nach dem Passwort gefragt.

---

## 8. Statistiken & Click-Tracking

### Was wird erfasst?

Bei jedem erfolgreichen Redirect (auch nach Passworteingabe) ruft `recordClick` zwei Datenbankoperationen aus:

1. **Inkrement des Klickzählers** in der `links`-Tabelle
2. **Detaillierter Click-Eintrag** in der `clicks`-Tabelle:
  - `ip`: IP-Adresse des Besuchers (`$_SERVER['REMOTE_ADDR']`)
  - `user_agent`: Browser-Kennung
  - `referer`: Woher der Besucher kam
  - `clicked_at`: Zeitstempel (automatisch via SQLite `datetime('now')`)

### Stats-Seite

Die Stats-Seite ([stats.php](#)) zeigt:

- **Gesamtklicks** als große Zahl
- **Max Clicks** (Limit oder  $\infty$ )
- **Status** (aktiv/inaktiv)
- **Letzte 100 Klicks** als Tabelle mit IP, Referer, User-Agent und Zeitstempel

Die Abfrage in `getClickStats` ist auf 100 Einträge limitiert (`LIMIT 100`), um bei viel-geklickten Links die Performance zu gewährleisten.

### Link-Deaktivierung

Ein Link wird automatisch als „nicht verfügbar“ angezeigt, wenn:

- `active = 0` (manuell deaktiviert)
- `max_clicks` erreicht wurde (`clicks >= max_clicks`)
- `expires_at` in der Vergangenheit liegt

In allen drei Fällen wird [unavailable.php](#) angezeigt.

---

## 9. Die REST-API

LinkShorter bietet zwei API-Endpunkte, die mit **HTTP Basic Authentication** geschützt sind.

### POST /api/shorten

Erstellt einen neuen Kurzlink.

#### Request:

```
POST /api/shorten HTTP/1.1
Authorization: Basic base64(username:password)
Content-Type: application/json

{
  "url": "https://example.com/sehr-lange-url",
  "slug": "custom",           // optional
  "password": "geheim",     // optional
  "expires_at": "2025-12-31T23:59", // optional
  "max_clicks": 100        // optional
}
```

#### Response (Erfolg):

```
{
  "short_url": "https://kurz.example.com/custom",
  "slug": "custom"
}
```

#### Response (Fehler):

```
{
  "error": "Slug already exists"
}
```

## GET /api/check

Prüft die Erreichbarkeit und Authentifizierung einer Instanz. Wird von der Chrome-Extension beim Hinzufügen einer neuen Instanz verwendet.

### Response:

```
{
  "status": "ok",
  "title": "LinkShorter"
}
```

## Technischer Hintergrund

Die Authentifizierung wird im [index.php](#) inline geprüft:

```
$authHeader = $_SERVER['HTTP_AUTHORIZATION'] ?? '';
if (preg_match('/^Basic\s+(.+)$/i', $authHeader, $m)) {
    $decoded = base64_decode($m[1]);
    list($user, $pass) = explode(':', $decoded, 2);
    // Vergleich mit ADMIN_USERNAME und ADMIN_PASSWORD
}
```

**Hinweis:** Apache kann den `Authorization`-Header manchmal nicht an PHP weiterleiten. In diesem Fall muss in der `.htaccess` folgende Zeile ergänzt werden:

```
SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1
```

# 10. Die Chrome-Extension

## Überblick

Die Chrome-Extension ermöglicht es, die **aktuelle Tab-URL** mit einem Klick zu kürzen, ohne das Dashboard öffnen zu müssen. Sie unterstützt **mehrere LinkShorter-Instanzen**, was nützlich ist, wenn man verschiedene Domains für verschiedene Zwecke nutzt.

# Installation

1. Navigieren Sie in Chrome zu `chrome://extensions/`
2. Aktivieren Sie den **Entwicklermodus** (oben rechts)
3. Klicken Sie **„Entpackte Erweiterung laden“**
4. Wählen Sie den Ordner `chrome-extension/`

# Dateien der Extension

Datei	Funktion
<a href="#">manifest.json</a>	Extension-Konfiguration (Manifest V3)
<a href="#">popup.html</a>	UI-Struktur
<a href="#">popup.css</a>	Styling
<a href="#">popup.js</a>	Gesamte Logik

# Instanz hinzufügen

1. Klicken Sie auf das **⚙️-Symbol** (Settings)
2. Geben Sie ein:
  - **Name:** Anzeigename (z.B. „Mein Server“)
  - **URL:** Die BASE\_URL Ihrer LinkShorter-Installation
  - **Username/Password:** Wie in [config.php](#) konfiguriert
3. Klicken Sie **„Add & Verify“**

**Technischer Hintergrund:** Beim Hinzufügen wird zunächst ein `GET /api/check` ausgeführt, um die Verbindung und Authentifizierung zu testen. Erst wenn `{"status": "ok"}` zurückkommt, wird die Instanz gespeichert. Die Instanzen werden in `chrome.storage.sync` gespeichert, was bedeutet, dass sie über mehrere Chrome-Installationen hinweg **synchronisiert** werden (wenn der Nutzer in Chrome eingeloggt ist).

# URL kürzen

1. Navigieren Sie zur gewünschten Webseite
2. Klicken Sie auf das LinkShorter-Icon in der Toolbar
3. Die **aktuelle Tab-URL** wird automatisch eingetragen (via `chrome.tabs.query`)
4. Optional: Wählen Sie eine andere Instanz oder geben Sie einen Custom Slug ein
5. Klicken Sie **„Shorten“**

6. Die verkürzte URL erscheint mit **Copy-Button**

# Berechtigungen

Die Extension benötigt nur zwei Berechtigungen ([manifest.json](#)):

- `activeTab`: Zugriff auf die URL des aktiven Tabs
- `storage`: Speichern der Instanz-Konfigurationen

Es werden **keine** Host-Permissions benötigt, da `fetch()` in Manifest V3 standardmäßig CORS-Requests durchführen darf (die API-Endpunkte müssen allerdings CORS erlauben oder die Extension muss die Requests direkt an die URL senden).

---

# 11. Automatische Link-Expiration (Cron)

## Einrichtung

Der Endpunkt `/cron` deaktiviert alle abgelaufenen Links. Die Funktion [expireLinks](#) führt folgendes SQL aus:

```
UPDATE links SET active = 0
WHERE expires_at IS NOT NULL
      AND expires_at <= datetime('now')
      AND active = 1
```

## Cron-Job einrichten

Fügen Sie in Ihrer Crontab folgenden Eintrag hinzu (z.B. alle 5 Minuten):

```
*/5 * * * * curl -s https://kurz.example.com/cron > /dev/null 2>&1
```

**Alternativ via PHP-CLI:**

```
*/5 * * * * php /var/www/html/index.php route=cron > /dev/null 2>&1
```

Die Antwort ist ein JSON-Objekt: `{"expired": 3}` – die Anzahl der gerade deaktivierten Links.

**Hinweis:** Der Cron-Endpoint ist **nicht authentifiziert**. Er führt jedoch nur eine Statusänderung durch (aktiv → inaktiv) und gibt keine sensiblen Daten zurück. Wenn Sie das absichern möchten, können Sie einen API-Key prüfen oder den Zugriff per `.htaccess` einschränken.

**Wichtig:** Auch ohne Cron werden abgelaufene Links beim Aufrufen als „nicht verfügbar“ angezeigt, da die Prüfung `strtotime($link['expires_at']) <= time()` direkt in der Routing-Logik stattfindet. Der Cron-Job sorgt lediglich dafür, dass der `active`-Status in der Datenbank korrekt gesetzt wird (relevant für die Dashboard-Anzeige).

---

## 12. Sicherheitshinweise

### Passwort in config.php

Das Admin-Passwort in [config.php](#) steht im **Klartext**. Stellen Sie sicher, dass:

- Die Datei nicht über den Webserver direkt abrufbar ist
- Ein starkes Passwort verwendet wird (nicht das Standard-`hackme123`!)

### Datenbankschutz

Die [.htaccess](#) blockiert den direkten Zugriff auf `.db`- und `.sqlite`-Dateien:

```
<FilesMatch "\.db$">
  Require all denied
</FilesMatch>
```

### XSS-Schutz

Alle Benutzereingaben werden in den Templates mit `htmlspecialchars()` escaped, z.B.:

```
<?= htmlspecialchars($link['slug']) ?>
```

### SQL-Injection-Schutz

Alle Datenbankabfragen verwenden **Prepared Statements** mit Parameter-Binding:

```
$stmt = $db->prepare('SELECT * FROM links WHERE slug = ?');  
$stmt->execute([$slug]);
```

Die einzige Ausnahme ist die dynamische `ORDER BY`-Klausel in [getAllLinks](#), die aber über ein Whitelist-Array abgesichert ist.

## CSRF-Schutz

Aktuell gibt es **keinen CSRF-Token-Schutz**. Da die Anwendung nur einen einzigen Admin-User hat, ist das Risiko begrenzt, aber bei einer Erweiterung sollte ein Token-System implementiert werden.

# 13. Technische Architektur im Detail

## Routing-System

Das Routing in [index.php](#) funktioniert als **Kaskade von if-Statements**:

```
Eingang: $_GET['route'] (via .htaccess Rewrite)  
↓  
1. Exakte Routen: 'cron', 'api/shorten', 'api/check', 'admin/action'  
↓  
2. QR-Routen: 'qr/png', 'qr/svg', 'qr/png/download', 'qr/svg/download'  
↓  
3. Slug-Auflösung: Beliebiger Pfad → getLinkBySlug()  
↓  
4. Seiten-Routing: $_GET['page'] → login, dashboard, edit, stats, settings
```

## Datenbank-Schema

```

links (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  slug TEXT UNIQUE NOT NULL,          -- Der Kurzlink-Code
  url TEXT NOT NULL,                  -- Ziel-URL
  password TEXT DEFAULT NULL,         -- bcrypt-Hash oder NULL
  expires_at TEXT DEFAULT NULL,       -- ISO-8601 Ablaufzeit
  max_clicks INTEGER DEFAULT NULL,    -- Klick-Limit
  clicks INTEGER DEFAULT 0,           -- Aktueller Zähler
  active INTEGER DEFAULT 1,           -- 1 = aktiv, 0 = deaktiviert
  og_title TEXT,                      -- OpenGraph-Cache
  og_description TEXT,
  og_image TEXT,
  created_at TEXT,                    -- datetime('now')
  updated_at TEXT
)

clicks (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  link_id INTEGER NOT NULL,           -- FK → links.id
  ip TEXT,
  user_agent TEXT,
  referer TEXT,
  clicked_at TEXT                     -- datetime('now')
  FOREIGN KEY (link_id) REFERENCES links(id) ON DELETE CASCADE
)

settings (
  key TEXT PRIMARY KEY,
  value TEXT                          -- Aktuell nicht aktiv genutzt
)

```

# Frontend-JavaScript

[app.js](#) ist bewusst minimal und framework-frei:

- **Batch-Modal:** Toggle der CSS-Klasse `active` auf dem Overlay
- **QR-Modal:** Dynamisches Setzen der `src`- und `href`-Attribute basierend auf dem Slug
- **Slug-Preview:** Live-Aktualisierung beim Tippen mit Regex-Filterung
- **Copy-Button:** Via `navigator.clipboard.writeText()` mit visueller Bestätigung

- **Delete-Confirmation:** Nativer `confirm()`-Dialog

# Performance-Überlegungen

- **SQLite WAL-Modus:** Ermöglicht gleichzeitige Leser während ein Schreibvorgang läuft
  - **QR-Code-Cache:** Verhindert wiederholte Berechnung (24h TTL)
  - **OpenGraph-Cache in DB:** Meta-Daten werden nur beim Erstellen/Ändern der URL abgerufen
  - **Statische Assets:** CSS und JS werden direkt ausgeliefert (kein Build-Prozess nötig)
- 

# Zusammenfassung

LinkShorter v2 ist eine **schlanke, selbstgehostete Lösung** ohne externe Abhängigkeiten. Die bemerkenswerteste technische Leistung ist die vollständige **QR-Code-Implementierung in reinem PHP**, inklusive Reed-Solomon-Fehlerkorrektur und Galois-Feld-Arithmetik. Die Chrome-Extension ergänzt das System um einen **komfortablen Workflow** direkt aus dem Browser heraus, wobei die Multi-Instanz-Unterstützung besonders für Nutzer mit mehreren Domains nützlich ist.

---

Revision #3

Created 2026-06-10 16:45:39 UTC by art10m

Updated 2026-06-10 16:57:59 UTC by art10m