**URL Shortener – Milestone 1 Documentation**

**Project Overview**

The URL Shortener is a web service that converts long URLs into short, easy-to-share links.  
**Milestone 1 (MVP)** focuses on building the **core functionality**: shortening URLs, storing them in a database, and redirecting users to the original URL.

**Tech Stack:**

* **Backend:** Node.js, Express
* **Database:** Supabase (PostgreSQL)
* **Hosting:** Local / Vercel / Render / Heroku
* **Environment Variables:** Managed via .env

**Features Implemented (Milestone 1)**

1. **POST /shorten API**
   * Accepts a valid URL in JSON format.
   * Generates a **unique Base62 short ID**.
   * Stores short\_id and target\_url in Supabase.
   * Returns the shortened URL.
2. **GET /:short\_id API**
   * Accepts a short ID in the path.
   * Fetches the corresponding target URL from the database.
   * Redirects the user to the original URL.
   * Returns 404 if the short ID is not found.
3. **Input Validation**
   * Checks that submitted URLs are **valid** (http:// or https://).
   * Ensures URL length is within acceptable limits.
4. **Database Integration**
   * Supabase table urls stores:
     + id (primary key)
     + short\_id (unique, Base62)
     + target\_url (original URL)
     + created\_at (timestamp)
5. **Deployment Ready**
   * Local testing with node index.js.
   * Can be deployed to **Vercel/Render/Heroku**.
   * Environment variables are managed in .env.

**Project Structure**

url-shortener/

├── index.js # Express server and APIs

├── utils/ # Helper functions

│ └── idGenerator.js # Base62 short ID generator

├── package.json

├── package-lock.json

├── node\_modules/

└── .env # Environment variables

**API Usage**

**1. Shorten a URL**

**Request**

POST /shorten

Content-Type: application/json

{

"url": "https://youtube.com"

}

**Response**

{

"short\_url": "http://localhost:3000/Ab3kLpQ"

}

**2. Redirect**

**Request**

GET /Ab3kLpQ

**Behavior**

* Redirects to https://youtube.com.
* Returns **404 Not Found** if the short ID does not exist.

**How to Test Locally**

1. Run the server:

node index.js

1. Test shortening a URL using curl:

curl -X POST http://localhost:3000/shorten \

-H "Content-Type: application/json" \

-d '{"url":"https://youtube.com"}'

1. Test redirection:

curl -i [http://localhost:3000/<short\_id](http://localhost:3000/%3cshort_id)>

CREATE TABLE urls (

id BIGSERIAL PRIMARY KEY,

short\_id VARCHAR(10) UNIQUE NOT NULL,

target\_url TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT NOW()

);

**Milestone 2: Performance Testing & Optimization**

**Objective:**  
Ensure the URL shortener can handle high concurrent requests efficiently and demonstrate performance improvements with caching.

**1️⃣ Setup**

* **Backend:** Node.js with Express
* **Database:** Supabase (PostgreSQL)
* **Cache:** Redis
* **Load Testing Tool:** k6
* **Test Data:** 100+ short URLs stored in short\_urls.txt

**2️⃣ Test Approach**

1. **Load Generation:**
   * Used k6 to simulate multiple virtual users (VUs).
   * URLs were randomly selected from the pre-generated short URLs file.
   * Automatic redirects disabled to validate 302 Found responses and Location headers.
2. **Test Scenarios:**
   * **With caching:** Redis cache is checked before querying Supabase.
   * **Without caching:** Direct queries to Supabase, bypassing Redis.
3. **Metrics Monitored:**
   * Response time (average, median, 95th percentile)
   * HTTP request success rate (302 + Location)
   * Throughput (requests per second)
4. **Load Profile:**
   * Gradual ramp-up, sustain, and ramp-down of virtual users to simulate real-world traffic.

**3️⃣ Sample k6 Script**

import http from 'k6/http';

import { check, sleep } from 'k6';

import { SharedArray } from 'k6/data';

const urls = new SharedArray('short urls', function() {

return open('short\_urls.txt').split('\n').filter(u => u.trim() !== '');

});

export const options = {

stages: [

{ duration: '1m', target: 100 }, // ramp-up to 100 VUs

{ duration: '3m', target: 100 }, // sustain load

{ duration: '1m', target: 0 }, // ramp-down

],

thresholds: {

http\_req\_failed: ['rate<0.01'],

http\_req\_duration: ['p(95)<500'],

},

};

export default function () {

const url = urls[Math.floor(Math.random() \* urls.length)];

const res = http.get(url, { redirects: 0 });

check(res, {

'status is 302': (r) => r.status === 302,

'has Location header': (r) => r.headers['Location'] !== undefined,

});

sleep(0.1);

}

**4️⃣ Results**

| **Scenario** | **Avg Response Time** | **p95 Response Time** | **Success Rate** | **Notes** |
| --- | --- | --- | --- | --- |
| Without caching | 128 ms | 146 ms | 99.99% | Direct Supabase queries |
| With caching (Redis) | 60 ms | 82 ms | 100% | Significant improvement in latency |

* **Conclusion:** Caching improves performance ~2x.
* The system successfully handled hundreds of thousands of requests in load testing.

**5️⃣ Key Takeaways**

* Implemented realistic performance tests simulating concurrent users.
* Validated 302 redirects and location headers for all short URLs.
* Demonstrated the benefit of Redis caching in reducing database load and latency.
* Generated performance metrics suitable for inclusion in a **GitHub README** or resume bullet points.

If you want, I can also **draft 3–4 ATS-friendly bullet points from this milestone** that you can directly paste into your resume to highlight backend performance expertise.