# What is needed to run the service?

1. **Node.JS (v18 is used)**
2. **Express**
3. **UUID library**
4. **valid-url library**
5. **Nodemon (to restart server upon save)**
6. **Supertest library**
7. **JEST Framework for testing**
8. **Docker Desktop (if containerizing)**

# How can we run the service?

1. Ensure the dependencies needed are installed, open a terminal, navigate to project’s root directory and run **npm install ,** this will install the dependencies in node\_modules folder.
2. Open a terminal, navigate to project’s root directory and type **npm run devStart**
3. The web service will now be running on localhost:3000, depending on the port set in **server.js file**
4. CURL commands can now be made, or install [REST Client VS Code extension](https://marketplace.visualstudio.com/items?itemName=humao.rest-client) to run the sample requests setup in **route.rest file**, by clicking the “Send Request” button in VS Code.

## Automated Test Cases

4 Assertion Test Cases written using JEST Framework, located at **\_\_test\_\_/api.test.js** :

1. POST returns a generated UUID when providing a valid URL in request body.
2. GET returns the original URL using the passing generated UUID in request parameter.
3. POST returns an error 404 “Invalid URL” when providing an invalid URL.
4. POST returns an error 404 “URL is not accessible” when providing an inaccessible URL.

To run the automated tests, navigate to root directory of folder, open a terminal and type:

**npm run test**

# Evaluation

## Assumptions

* The JSON request provided has the following format of { “url”: “{website.name} }“
* If a URL has already been provided and exists in-memory database, the server will return the original UUID associated instead of generating new ones each time.
* Edge Case of some different URL format is not handled, for example [www.google.com.com](http://www.google.com.com)

## Difficulties encountered & Improvements made

**Learning to use JEST with ESM Modules**

* The initial design used to validate URL was to use an existing library **“url-exist”**, however it uses dynamic import ESM modules instead of CommonJS (“require”).
* This results in various intermittent issues when incorporating JEST testing framework, as ES modules requires enabling experimental features on NODE\_OPTIONS to enable dynamic import statements like **“import url-exist from url-exist”** to work. This also requires additional configuration in jest.config.js and jest.setup.js files and may also require Babel configuration.
* The final design is to use **“valid-url”** library to validate URL format (though can be done with regex) and then use **fetch** to check if a url is accessible.
* I believe this is also a good approach to avoid mixing CommonJS and ES modules and avoid unnecessary node.js libraries if necessary.
* Some noticeable speed performance when running JEST tests as well, from 800-1000ms to 350ms after removing the url-exist library.

## Improvements can be made

* Cover the edge case of various different URL format, for example [www.google.com.com](http://www.google.com.com)
* Apply Caching in Middleware Router

# Stretch Goals

## Containerizing with Docker

1. Requires Docker Desktop to be installed
2. To check any running docker containers, use **docker ps**
3. Build image using Dockerfile instructions, for example **docker build -t jarrel/test-server:1.0 .**
4. Run **docker run -p 5000:3000 {image.id}** to port forward to 5000
5. Now we can send requests to localhost:5000/shorten\_url as the container is running.

Text

Description automatically generated

## Tracking unique visits

* To capture unique visits, some session information is needed, such as cookies, or JSON Web Tokens (JWT); possibly AWS Cognito’s Authentication?
* Dynamic QR Code can also be used to include tracking and analytics after associating a unique QR code to the specified endpoint (e.g POST), then each time it is scanned it can be tracked; possibly Google Analytics?

## Scaling service for higher number of requests

* Use a dedicated DB (e.g MongoDB) to hold data, e.g in a JSON model schema structure.
* The Middleware Router can be used to apply Caching to check existing URL that already has a UUID generated, this eliminates the overhead before querying the DB.
* Redis can be used as a Cache option, or directly as modern multi-model in-memory primary DB system that also acts as a cache, e.g RedisJSON and RediSearch. The advantage of using it as a primary DB system is that it reduces the overall complexity of the design, as it now goes from: Client -> Server -> Redis

# Resources used:

**“url-exist” library:** <https://stackoverflow.com/questions/26007187/node-js-check-if-a-remote-url-exists>

**Tracking unique visits read:** <https://blog.beaconstac.com/2021/10/qr-code-tracking-google-analytics/>

**Scaling service requests read:** <https://betterprogramming.pub/scaling-web-service-b391557a1134>

Fireship’s tutorial on YouTube:

* **Learn Docker in 7 Easy Steps – Full Beginner’s Tutorial:** <https://youtu.be/gAkwW2tuIqE>
* **Redis in 100 Seconds:** <https://youtu.be/G1rOthIU-uo>