shorten-a-railway (server & frontend)

frontend -> Submit new link to shorten.
-> reporting a link.
-> See how many links exist.
-> QR code generator?

Backend -> Stored in an sqlite for portability.
-> schema

ID, url_original, date; (email?) } (*)
auto-incr. [0], google.com, System_date; }

Api -> Create_new_shortened_link } (*)
-> follow_redirect_to_original_link. }
-> if you want it removed
↳ Email_us. (corvid_co@protonmail.com)

2 Ends -> 1) front end 2) Backend

Shorten-A-Railway-Server
-〃- -A-〃- -client

The steps of the functionality:
-> Backup every 30 minutes (Max: 10) (*)
1) Boot 2) Scan db for link (url) count (A)
3) Open end listeners (A) 4) listen on ports.
Everything has logs!
* past 5-hrs + 1 Everyboot ≈ 11

- Url routing: /products -> show all products
  /products/:id
  /products/1 -> show product wil ID 1

Diffrent Url routes => Diffrent Url req handlers

get -> read the data
post -> write the data.

```
res.send("text");
res.json({ID:1, name: "Choky milk 1Ltr"});
```

Middleware -> Piece of code that handles data before or after a request

- Request processing:
  (1) pre-processing -> code to run before request is processed
  (2) processing -> process request // return resources
  (3) post-processing -> (optional) run ~~code after~~ (ex. logging) process.

Ex: `app.use((req, res, next) => { });`

data in — data_out — middleware

Global middleware

```
app.use((req, res, next) => {
    console.log("Middleware");
    next();
});
```

```
function isAuthorised(req, res, next) {
    const authHeader = req.headers.authorisation;
    if (!authHeader || authHeader !== "passwd") {
        return res.status(401).send("Unauthorised!!");
    }
    next();
}

app.get("/re", isAuthorised, (req, res) => {
    res.json();
});

headers: {
    authorization: "password"
},
```

Route manager:

Url conforming: scheme:[//authority]path[?query][#frag]

scheme => https
Authority => google.com
Path => /search/
query => arg=apple&page=1
fragment => sort=asc

Handlers:

```
app.get("/products/:id"), (req, res) => {
    //body
});
```

Incoming data:

route param.	/products/:id
Query param.	/products?page=1&page_size=20
request body	post /products

Action	Method	Data
Create	Post	Request body
Read	GET	route & query param.
Update	PUT	Request body
Delete	DELETE	route & query param.

Spells "CRUD"

The 4 basic operations that can be performed on Data

if url "/products?page=1&page_size=20"

req.query.page_size

req.query.page

in an app.get!!

Write data:

Method	Description
Post	Create a new resource
PUT	Update All of a resource
PATCH	Update a part of a resource

As per earlier notes: shasten a railway will have the following endpoints that may also deliver a static html page for the UI

As per "CRUD" functionality, the api will emplemend:
1) Create 2) Read

$ Not:
1) Update 2) Delete®

Deletion will require: 1) request by original Creator.
2) reporting the link.

-11-

Endpoints:
(get) / -> Index (aka main)
(post) /new -> Create a new link
(get) /v -> follow short link.
(post) /report/: ! soon will implement!

Query:
None for "/" is just a get (sendFile("index.html");
/new/:link/:email ~> err, checking, etc
/v/:id -> redirect.

Needs:
① rate limiting in /new. 10/10 Needs!
② make alphabetical the id (back & forth)
↑ Not that much Needed