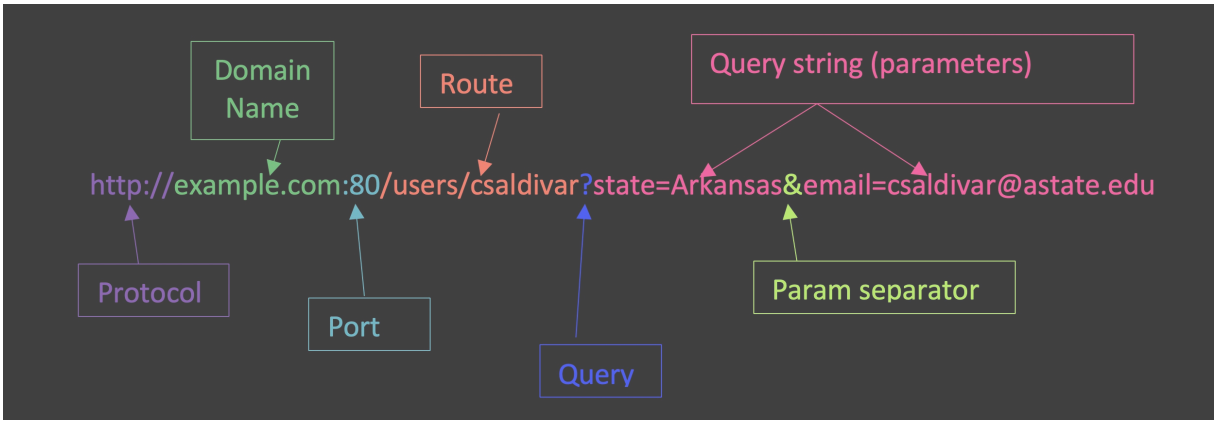
# URLs, HTTP and Routes

## URLs

A lot of the documentation you'll be reading talk about URIs. For our purposes URLs and URIs are effectively the same thing. All URLs are URIs but not all URIs are URLs.

The `protocol` portion of the URL makes it a URL. Without the `protocol` means it is just a URI.




### Protocol

This will default to `http://` if you do not specify in the the browser's URL bar. In your code you should always explicitly include the `protocol`.

### Domain Name

This is the website's domain name or in our case the IP address. Remember that DNS will convert domain names into IP addresses.

### Route

This is the route to a resource. We will specify these in our server code by creating routes with express. These are sometimes referred to as **api endpoints**. We often think of these as **nouns** (where HTTP methods are often thought of as **verbs**).

### Query (?)

This is optional and it denotes the beginning of the **query string**. We can use this to pass data to the server using only the url rather than passing data in the body.

### Query Parameters

The query parameters follow the `?` symbol. They are `key-value` fields with the following syntax:

`key=value`
Note that there is no space on either side of the `=` sign.

So the following query parameter could be represented with this javascript object:
`state=Arkansas`

```
{
 "state": "Arkansas"
}
```

Use the `&` to separate multiple parameters.

`state=Arkansas&email=csaldivar@astate.edu`

```
{
 "state": "Arkansas",
 "email": "csaldivar@astate.edu"
}
```

These are accessible from in express by using the request object's `query` attribute.

These are necessary when using GET requests that need to pass data to the server. The GET request cannot have a request body so if you want to send some data to the server you could use query parameters in the URL to pass data along.

## HTTP

We will use HTTP as our communication protocol for sending requests to our servers. HTTP is built on the concept of **methods** (also known as **verbs**). Which have semantic meaning. Although the verbs themselves don't do anything it is up to you to implement their behavior correctly.

When deciding which HTTP method to use you have to consider what behavior you are after. The are 2 major concepts to consider, whether a request is **safe** and or **idempotent**.

Safe requests do not alter the state of the server. GET requests are safe since they should be read-only. The just GET a resource and don't do anything. They are also idempotent.

Idempotent requests are those that result in the same server state regardless of how many times you make them. PUT requests are idempotent but not safe. For example say you place an order and you accidentally hit the submit button twice. Ideally the server wouldn't place 2 orders. If implemented correctly the server should only place 1 order and when it gets the second request it should perform a check to ensure it does not place a second order. This is the fundamental difference between PUT and POST. PUT is idempotent

and POST is not.

We can specify which HTTP verb we are implementing in express by using express' application object. For example:

| HTTP Method | Express syntax |
|---|---|
| GET | app.get() |
| PUT | app.put() |
| POST | app.post() |
| DELETE | app.delete() |

These are the methods you'll use most often; however, you can check the documentation for others.

## Routes

These are the "paths" to resources on our server. They can resources like HTML/CSS/JavaScript files or they can just be API endpoints that perform some action. For example we can have a POST endpoint that is used to add a new order to our database. This functionality is a resource even though it isn't a tangible "thing".

Express lets use use named URL segments to capture the values at specific locations in a route. You just place a `:` before the name of the field you want to capture.

```
app.get('/users/:username/emails/:email', (req, res) => {
    res.send(JSON.stringify({
        body: req.body,
        query: req.query,
        params: req.params
    }))
});
```

The route above captures the values at the $2^{nd}$ and $4^{th}$ fields in the route. Express will then put this in a javascript object in the `req.params` attribute. The objects keys will be the names we used in the route itself.

So the following URL:

`http://example.com/users/chris/emails/csaldivar@astate.edu`

results in this object:

```
{
 "username": "chris",
 "email": "csaldivar@astate.edu"
}
```

## HTTPS

This is just HTTP over TLS. This will encrypt all requests. You do need to configure your server to use a SSL cert and listen on port 443 (default for HTTPS) but all of the routes, HTTP methods and urls work essentially the same. Just replace `http://` with `https://`. We'll go over how to configure our servers to use HTTPS later.