MIDDLEWARE *and more!*

Code 301

MIDDLEWARE – WHAT IS IT?

- Middleware is software that provides services or components to other software
- Makes it easier for developers to communicate between different parts of an application
- We are using **page.js** as a middleware component to intercept our routes and control the functionality and views of our app

MIDDLEWARE

What part of this delicious meal represents middleware?

PAGE.JS – WHY ARE WE USING IT?

- Page.js offers us a series of helpers to handle certain functionality based on the routes defined in our app
- The most notable helpers are the **Context (ctx)** object and the **next()** function
  - The **context (ctx)** object allows us to manage **state** and persist interactions/data between multiple routes
  - The **next()** function, which is based off of and commonly used in Express.js, allows us to move on to the next callback defined in a route

CONTEXT OBJECT?? - WHY DO WE NEED THAT?

- Routes are passed a **Context (ctx)** object, which allows us to share information between our routes
- This object gives us the ability to share an arbitrarily created state and/or the history state provided by the pushState API
- By assigning arbitrary properties to our **Context** object (**ctx**), we can track, manage, and share a specific **state** of our application, resource, or functionality
  - for example: ctx.user = 'brian' will assign the user property to the page.js ctx object, allowing us to reference this in our logic

WORKING WITH URL PARAMS

- Using page.js, we can access URL params that meet the criteria defined in our routes
  - for example, if we have defined our route as:
    - ‘/user/:id’
  - we can then access the url params, automatically, on the ctx object by referencing ctx.params.id
    - this will give us the result of any url that meets the requirements defined in our route
      - lets say we have a route of “/user/29345”
        - then we will receive a result of “29345” if we console.log ctx.params.id

WORKING WITH MULTIPLE CALLBACKS IN PAGE.JS

- As we have discussed, page.js gives us the ability to work with multiple callbacks for any given route
- These callbacks can be invoked by calling the “next()” method within the current callback and must **be passed as a callback** into the current method
  - **example:**

```
function load(ctx, next){
  var id = ctx.params.id
  $.getJSON('/user/' + id + '.json', function(user){
    ctx.user = user;
    next()
  })
}
```

USING PAGE.JS TO INTERACT WITH THE PUSHSTATE API

- You can use page.js to interact with the browsers history state by using the ctx object
  - This can be done by executing the save() method
    - This is an abstraction layer built on top of the pushState API and hooks into the functionality that the native pushState method, replaceState(), gives us access to
- Example:

```
function show(ctx){
  if (ctx.state.images) {
    displayImages(ctx.state.images)
  } else {
    $.getJSON('/photos', function(images){
      ctx.state.images = images
      ctx.save()
      displayImages(images)
    })
  }
}
```