



Clase 22. Programación Backend

Trabajo con datos: Normalización



OBJETIVOS DE LA CLASE

- Entender el concepto de normalización de datos.
- Comprender el uso de normalizr y su sistema de esquemas.
- Verificar el formato del objeto normalizado.
- Realizar la normalización y desnormalización de una estructura con redundancias.

CRONOGRAMA DEL CURSO

Clase 21



Trabajo con datos: Mocks

Clase 22



Trabajo con datos: Normalización

Clase 22

Clase 23



Cookies, Sesiones y Almacenamientos

Normalización de datos



¿Qué es la normalización de datos?



Es un **proceso** de **estandarización y validación de datos** que consiste en **eliminar** las **redundancias o inconsistencias**, completando datos mediante una serie de reglas que actualizan la información, protegiendo su integridad y favoreciendo la interpretación, para que así sea más fácil de consultar y más útil para quien la gestiona.

SIN NORMALIZACIÓN	CON NORMALIZACIÓN
<ul style="list-style-type: none">✗ Datos erróneos✗ Datos redundantes✗ Datos desactualizados✗ Datos insuficientes✗ Datos nulos	<ul style="list-style-type: none">✓ Datos precisos✓ Datos únicos✓ Datos íntegros✓ Datos completos✓ Datos relevantes

¿Cuándo y cómo se utiliza?



La normalización de datos es útil cuando un repositorio de datos es demasiado grande, contiene redundancias, tiene información profundamente anidada y/o es difícil de usar.

Al normalizar los datos, debemos seguir algunas reglas:

- La estructura de datos debe ser plana.
- Cada entidad debe almacenarse como propiedad de objeto diferente.
- Las relaciones con otras entidades deben crearse basadas en identificadores: 'id'.

Normalizr

Json Node

Json Schema

Normalizer

Result

¿Qué es normalizr?



Es un **paquete** muy útil que utiliza la definición de esquemas personalizados **para crear datos normalizados.**

Se puede instalar desde npm a través de **npm i normalizr**

normalizr **TS**

3.6.1 • Public • Published 5 months ago



Readme



Explore

BETA



0 Dependencies



696 Dependents



36 Versions

normalizr **build passing** **coverage 100%** **npm v3.6.1** **downloads 1.2M/month**

Install

Install from the NPM repository using yarn or npm:

```
yarn add normalizr
```

```
npm install normalizr
```

Install

```
> npm i normalizr
```

Weekly Downloads

268.855

Version

3.6.1

License

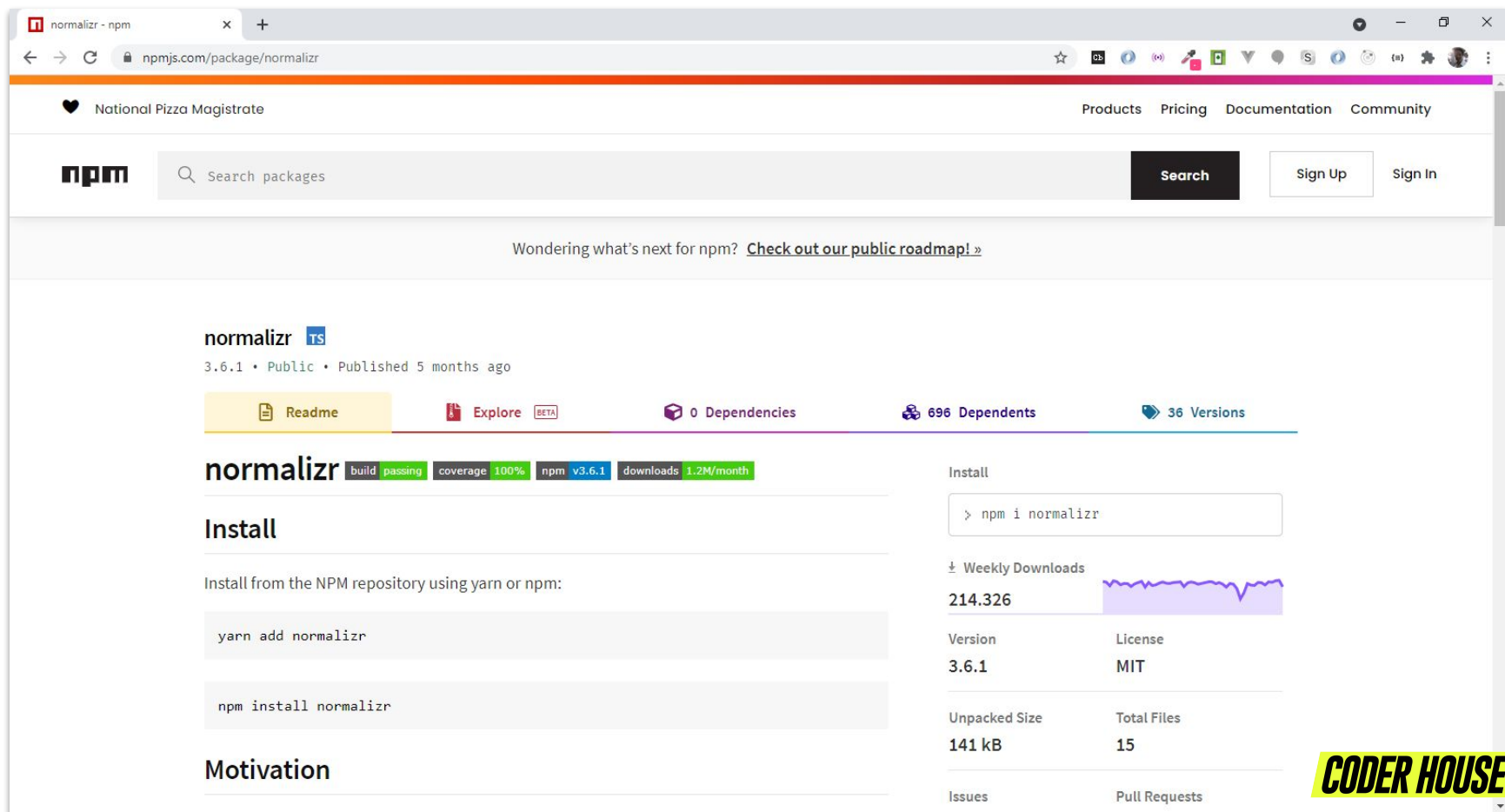
MIT

Unpacked Size


Total Files

CODER HOUSE

Normalizr npm site: <https://www.npmjs.com/package/normalizr>



The screenshot shows the npm package page for 'normalizr'. At the top, there's a navigation bar with the npm logo, a search bar, and links for 'Sign Up' and 'Sign In'. Below this, a banner for 'National Pizza Magistrate' is visible. The main content area features the package name 'normalizr' with a TypeScript icon, version '3.6.1', and publication date '5 months ago'. It includes buttons for 'Readme', 'Explore', 'Dependencies', 'Dependents', and 'Versions'. A status bar shows 'build passing', 'coverage 100%', 'npm v3.6.1', and 'downloads 1.2M/month'. The 'Install' section provides a terminal command and a description of how to install from the NPM repository. A 'Weekly Downloads' chart shows 214,326 downloads. A table lists package details: Version (3.6.1), License (MIT), Unpacked Size (141 kB), Total Files (15), Issues, and Pull Requests. The 'Motivation' section is partially visible at the bottom.

normalizr 

3.6.1 • Public • Published 5 months ago

[Readme](#) [Explore](#) [BETA](#) [0 Dependencies](#) [696 Dependents](#) [36 Versions](#)

normalizr [build passing](#) [coverage 100%](#) [npm v3.6.1](#) [downloads 1.2M/month](#)

Install

Install from the NPM repository using yarn or npm:

```
yarn add normalizr
```

```
npm install normalizr
```

Motivation

Install

```
> npm i normalizr
```

± Weekly Downloads
214.326

Version	License
3.6.1	MIT
Unpacked Size	Total Files
141 kB	15
Issues	Pull Requests

CODER HOUSE

Normalizr github site: <https://github.com/paularmstrong/normalizr>

Search or jump to... Pull requests Issues Marketplace Explore

paularmstrong / normalizr

Sponsor Watch 204 Star 20k Fork 818

<> Code Issues 20 Pull requests 4 Discussions Actions Security Insights

master 40 branches 40 tags Go to file Add file Code

dependabot chore: Bump sri from 8.0.0 to 8.0.1 (#466)	6c57049 3 days ago	452 commits
.github	Create FUNDING.yml	2 months ago
docs	docs: fix example of fallbackStrategy (#439)	11 months ago
examples	chore: remove example yarn.lock file (#456)	5 months ago
src	chore: update eslint/prettier deps	5 months ago
typescript-tests	fix: Add types for fallback strategy (#441)	11 months ago
.babelrc.js	Enable loose transformation for object spread operator to improve per...	15 months ago
.eslintignore	100% test coverage	4 years ago
.eslintrc.js	chore: update eslint/prettier deps	5 months ago
.flowconfig	Add flow	4 years ago
.gitignore	[chore] update eslint	2 years ago
.travis.yml	ci: use node 10, 12, 14	5 months ago
CHANGELOG.md	v3.6.1	5 months ago
CONTRIBUTING.md	add contributing guidelines for issues	4 years ago

About

Normalizes nested JSON according to a schema

redux javascript api flux json reactjs normalize normalizr

Readme MIT License

Releases 40

v3.6.1 Latest on 17 Oct 2020 + 39 releases

Sponsor this project

paularmstrong Paul Armstrong

Sponsor

CODER HOUSE

Ejemplo de uso en un blog



Las publicaciones en un blog pueden tener este formato de datos:

```
const blogpost = {
  id: "1",
  title: "My blog post",
  description: "Short blogpost description",
  content: "Hello world",
  author: {
    id: "1",
    name: "John Doe"
  },
  comments: [
    {
      id: "1",
      author: "Rob",
      content: "Nice post!"
    },
    {
      id: "2",
      author: "Jane",
      content: "I totally agree with you!"
    }
  ]
}
```

Este tipo de estructura parece totalmente correcta, pero cuando almacenemos más publicaciones de blog, **se duplicarán los datos** de los autores. A medida que nuestro blog crece, podemos agregar categorías y relaciones entre comentarios y usuarios.

No es necesario almacenar todo en un objeto.

Solución: Normalizr



```
// Definimos un esquema de usuarios (autores y comentadores)
const authorSchema = new schema.Entity('authors')

// Definimos un esquema de comentadores
const commentSchema = new schema.Entity('comments')

// Definimos un esquema de artículos
const postSchema = new schema.Entity('posts', {
  author: authorSchema,
  comments: [ commentSchema ]
});

const normalizedBlogpost = normalize(blogpost, postSchema);

const denormalizedBlogpost = denormalize(normalizedBlogpost.result, postSchema, normalizedBlogpost.entities);
```

- Normalizr funciona **definiendo esquemas** y luego declarando cómo estos esquemas se representan a través de entidades.
- El único requisito es que cada entidad (publicación, comentario, autor) tenga la **propiedad 'id'**.

Resultado

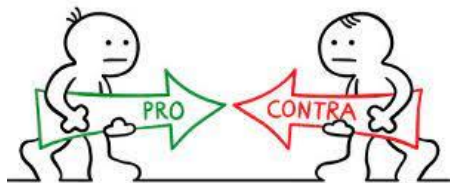


```
{
  entities: {
    authors: { '1': { id: '1', name: 'John Doe' } },
    comments: {
      '1': { id: '1', author: 'Rob', content: 'Nice post!' },
      '2': { id: '2', author: 'Jane', content: 'I totally agree with you!' }
    },
    posts: {
      '1': {
        id: '1',
        title: 'My blog post',
        description: 'Short blogpost description',
        content: 'Hello world',
        author: '1',
        comments: [ '1', '2' ]
      }
    }
  },
  result: '1'
}
```

Este objeto es el resultado del proceso de normalización.

Los **datos** están **agrupados** por ‘entidades’, y ‘result’ es el punto de entrada.

Logramos **desanidar y aplanar la información**. Esto nos va a ayudar a **quitar redundancias**.



Normalizr: Pros y contras

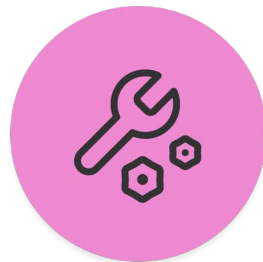


Ventajas de esta solución:

- Actualizar los datos de los comentarios y del autor es muy fácil.
- Posibilidad de mostrar fácilmente todas las publicaciones, autores y comentarios.
- No hay duplicación de datos.

Contras:

- Mostrar pocos comentarios en una publicación requiere pasar un objeto con todos los comentarios.
- En aplicaciones pequeñas sin mucha duplicación de datos, es posible que no sea necesario normalizar los datos.



Normalizar JSON

Tiempo: 10 minutos



- 1) Normalizar la estructura del objeto en formato JSON *empresa.json* (disponible en la carpeta de la clase) que describe el organigrama de una empresa. El gerente y el encargado figuran en el array de empleados de la empresa.
- 2) Imprimir por consola el objeto normalizado y la longitud del objeto original y del normalizado. Comparar los resultados.

Nota: En adelante, utilizar la siguiente función 'print' para imprimir el contenido de un objeto:

```
const util = require('util')
function print(objeto) {
  console.log(util.inspect(objeto,false,12,true))
}
```




Desnormalizar JSON

Tiempo: 10 minutos



Desnormalizar JSON

Desnormalizar el objeto del ejercicio anterior, imprimiéndolo por consola junto a su longitud.
Comparar el objeto original con el desnormalizado.

Tiempo: 10 minutos



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

***Proyecto: normalización y
desnormalización de datos en
formato JSON con redundancia***

Pasos a seguir

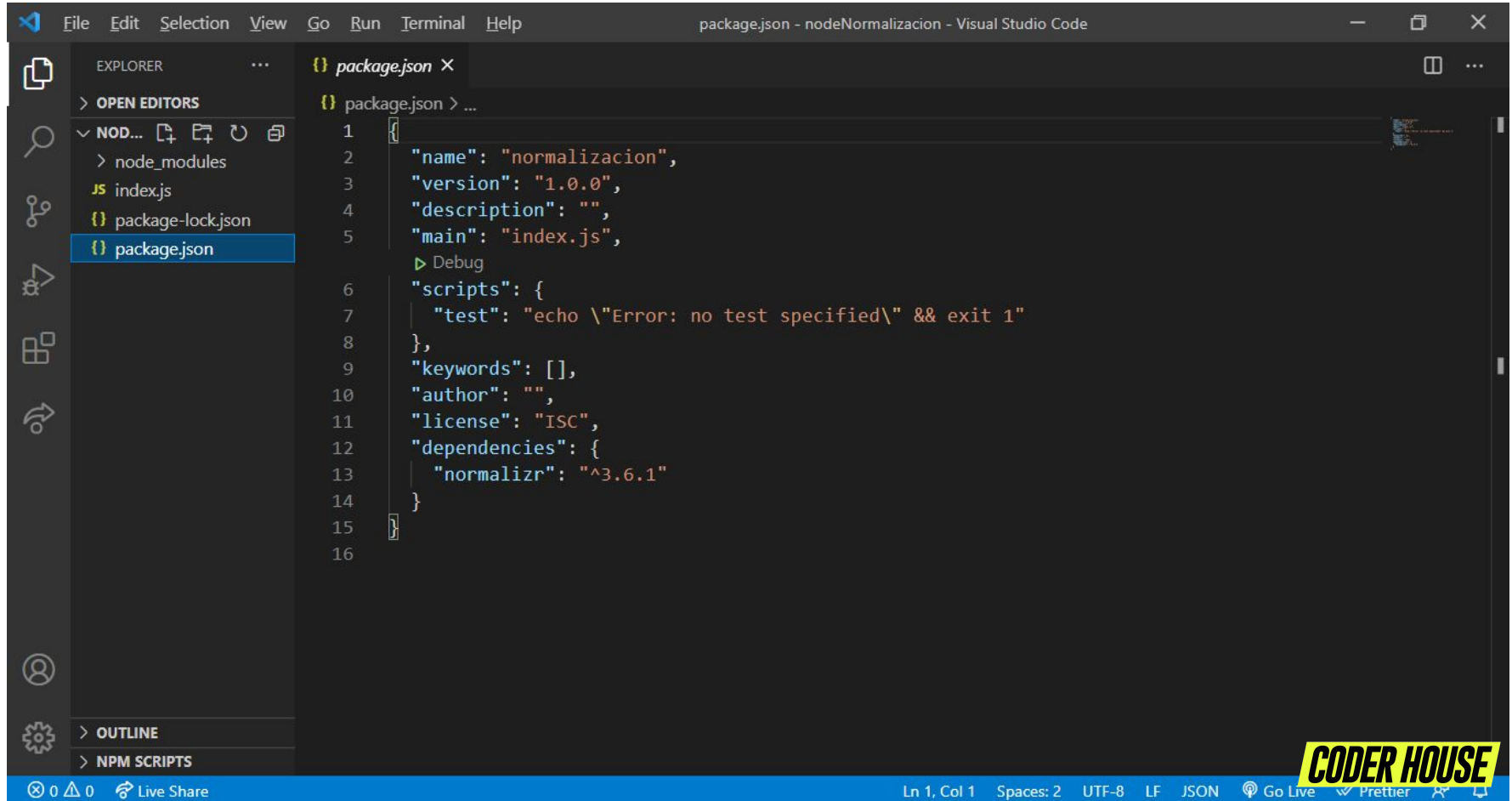


1. Vamos a trabajar con el objeto del blog, pero ahora añadiéndole *información redundante* que va a ser optimizada por **Normalizr**.
2. Dispondremos de un **array de artículos**, donde habrá *autores y comentadores*.
3. El **autor** de un artículo puede ser **comentador** de otro y viceversa.
4. De esta manera habrá *duplicación de información*, lo que producirá que el objeto no posea información centralizada y sea más extenso.
5. Definiremos un **conjunto de esquemas** para quitar esas redundancias.
6. Primero mostraremos el objeto original y su longitud en bytes, luego **normalizaremos** y comprobaremos los datos.
7. Por último **desnormalizaremos**, verificando los datos originales.

Objeto JSON de entrada con redundancia

```
const originalData = {
  Id: "999",
  posts: [
    {
      id: "123", author: {id: "1", nombre: "Pablo", apellido: "Perez", DNI: "20442654", direccion: "CABA
123", telefono: "1567876547"}, title: "My awesome blog
post", comments: [{id: "324", commenter: {id: "2", nombre: "Nicole", apellido: "Gonzalez", DNI: "20442638", direccion: "CA
BA
456", telefono: "1567811543"}}, {id: "325", commenter: {id: "3", nombre: "Pedro", apellido: "Mei", DNI: "20446938", direcc
ion: "CABA 789", telefono: "1567291542"}}]
    },
    {
      id: "1123", author: {id: "2", nombre: "Nicole", apellido: "Gonzalez", DNI: "20442638", direccion: "CABA
456", telefono: "1567811543"}, title: "My awesome blog
post", comments: [{id: "1324", commenter: {id: "1", nombre: "Pablo", apellido: "Perez", DNI: "20442654", direccion: "CABA
123", telefono: "1567876547"}}, {id: "1325", commenter: {id: "3", nombre: "Pedro", apellido: "Mei", DNI: "20446938", direc
cion: "CABA 789", telefono: "1567291542"}}]
    },
    {
      id: "2123", author: {id: "3", nombre: "Pedro", apellido: "Mei", DNI: "20446938", direccion: "CABA
789", telefono: "1567291542"}, title: "My awesome blog
post", comments: [{id: "2324", commenter: {id: "2", nombre: "Nicole", apellido: "Gonzalez", DNI: "20442638", direccion: "C
ABA
456", telefono: "1567811543"}}, {id: "2325", commenter: {id: "1", nombre: "Pablo", apellido: "Perez", DNI: "20442654", dir
eccion: "CABA 123", telefono: "1567876547"}}]
    }
  ]
}
```

Blog de artículos: *configuración*



Blog de artículos: *datos JSON con redundancia*

Visual Studio Code interface showing three editor windows for `index.js`, each displaying JSON data for a different article. The data is structured as follows:

ARTÍCULO 1

```
const originalData = {
  id: "999",
  posts: [
    {
      id: "123",
      author: {
        id: "1",
        nombre: "Pablo",
        apellido: "Perez",
        DNI: "28442654",
        direccion: "CABA 123",
        telefono: "1567876547"
      },
      title: "My awesome blog post",
      comments: [
        {
          id: "324",
          commenter: {
            id: "2",
            nombre: "Nicole",
            apellido: "Gonzalez",
            DNI: "28442638",
            direccion: "CABA 456",
            telefono: "1567811543"
          },
          title: "My awesome blog post",
          comments: [
            {
              id: "325",
              commenter: {
                id: "3",
                nombre: "Pedro",
                apellido: "Mei",
                DNI: "28446938",
                direccion: "CABA 789",
                telefono: "1567291542"
              },
              title: "My awesome blog post",
              comments: [
                {
                  id: "2",
                  nombre: "Nicole",
                  apellido: "Gonzalez",
                  DNI: "28442638",
                  direccion: "CABA 456",
                  telefono: "1567811543"
                },
                {
                  id: "1",
                  nombre: "Pablo",
                  apellido: "Perez",
                  DNI: "28442654",
                  direccion: "CABA 123",
                  telefono: "1567876547"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

ARTÍCULO 2

```
{
  id: "1123",
  author: {
    id: "2",
    nombre: "Nicole",
    apellido: "Gonzalez",
    DNI: "28442638",
    direccion: "CABA 456",
    telefono: "1567811543"
  },
  title: "My awesome blog post",
  comments: [
    {
      id: "1324",
      commenter: {
        id: "1",
        nombre: "Pablo",
        apellido: "Perez",
        DNI: "28442654",
        direccion: "CABA 123",
        telefono: "1567876547"
      },
      title: "My awesome blog post",
      comments: [
        {
          id: "1325",
          commenter: {
            id: "3",
            nombre: "Pedro",
            apellido: "Mei",
            DNI: "28446938",
            direccion: "CABA 789",
            telefono: "1567291542"
          },
          title: "My awesome blog post",
          comments: [
            {
              id: "2",
              nombre: "Nicole",
              apellido: "Gonzalez",
              DNI: "28442638",
              direccion: "CABA 456",
              telefono: "1567811543"
            },
            {
              id: "1",
              nombre: "Pablo",
              apellido: "Perez",
              DNI: "28442654",
              direccion: "CABA 123",
              telefono: "1567876547"
            }
          ]
        }
      ]
    }
  ]
}
```

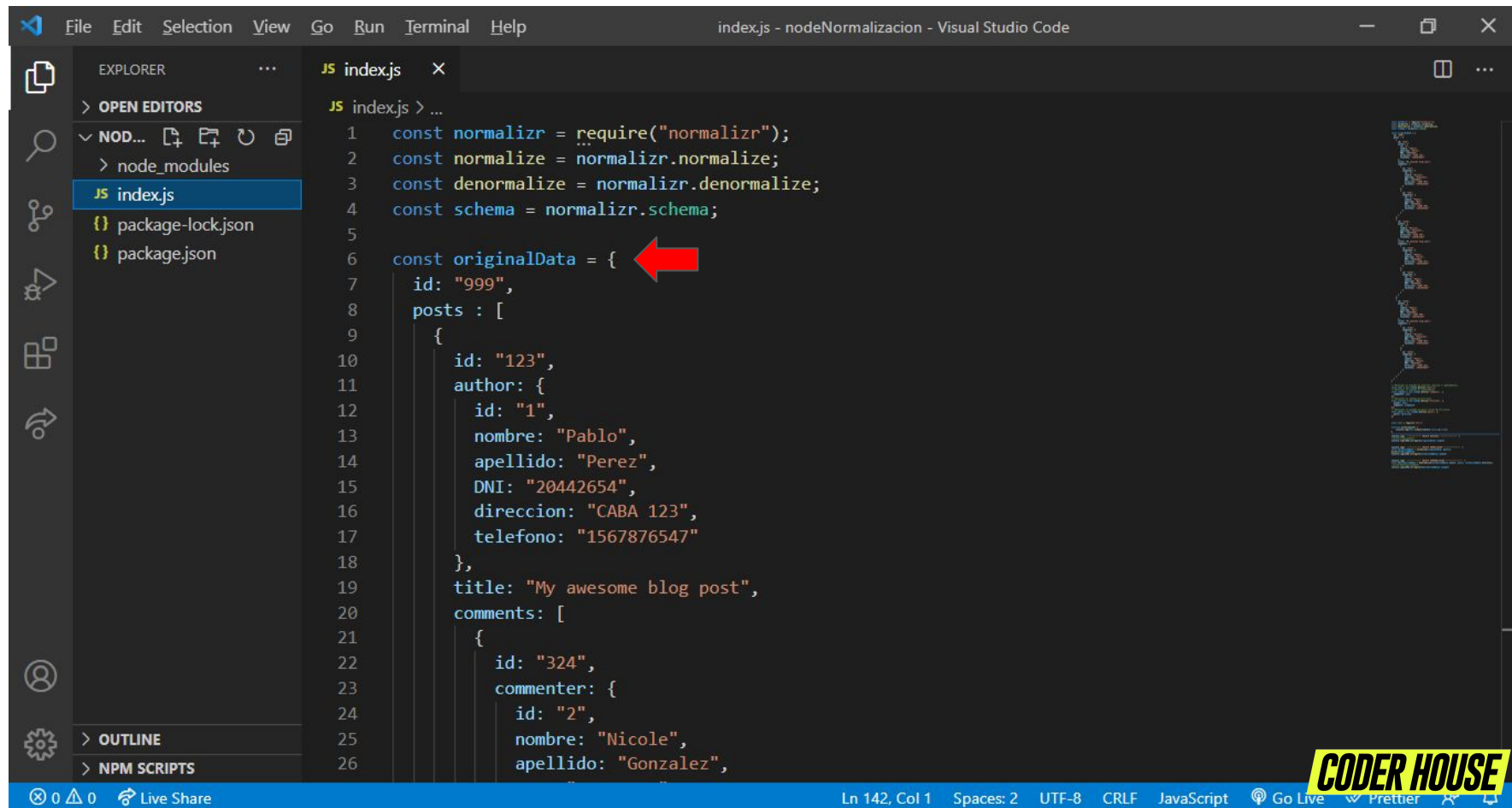
ARTÍCULO 3

```
{
  telefono: "1567291542",
  id: "2123",
  author: {
    id: "3",
    nombre: "Pedro",
    apellido: "Mei",
    DNI: "28446938",
    direccion: "CABA 789",
    telefono: "1567291542"
  },
  title: "My awesome blog post",
  comments: [
    {
      id: "2324",
      commenter: {
        id: "2",
        nombre: "Nicole",
        apellido: "Gonzalez",
        DNI: "28442638",
        direccion: "CABA 456",
        telefono: "1567811543"
      },
      title: "My awesome blog post",
      comments: [
        {
          id: "2325",
          commenter: {
            id: "1",
            nombre: "Pablo",
            apellido: "Perez",
            DNI: "28442654",
            direccion: "CABA 123",
            telefono: "1567876547"
          },
          title: "My awesome blog post",
          comments: [
            {
              id: "2",
              nombre: "Nicole",
              apellido: "Gonzalez",
              DNI: "28442638",
              direccion: "CABA 456",
              telefono: "1567811543"
            },
            {
              id: "1",
              nombre: "Pablo",
              apellido: "Perez",
              DNI: "28442654",
              direccion: "CABA 123",
              telefono: "1567876547"
            }
          ]
        }
      ]
    }
  ]
}
```

The image illustrates data redundancy in JSON, where the same nested object structure is repeated across different articles. A red arrow points to the first article's data, and colored boxes highlight the author and comment objects within each article's data structure.

CODER HOUSE

Blog de artículos: *dependencias*



Visual Studio Code interface showing a JavaScript file named `index.js` in the Explorer sidebar. The file content is as follows:

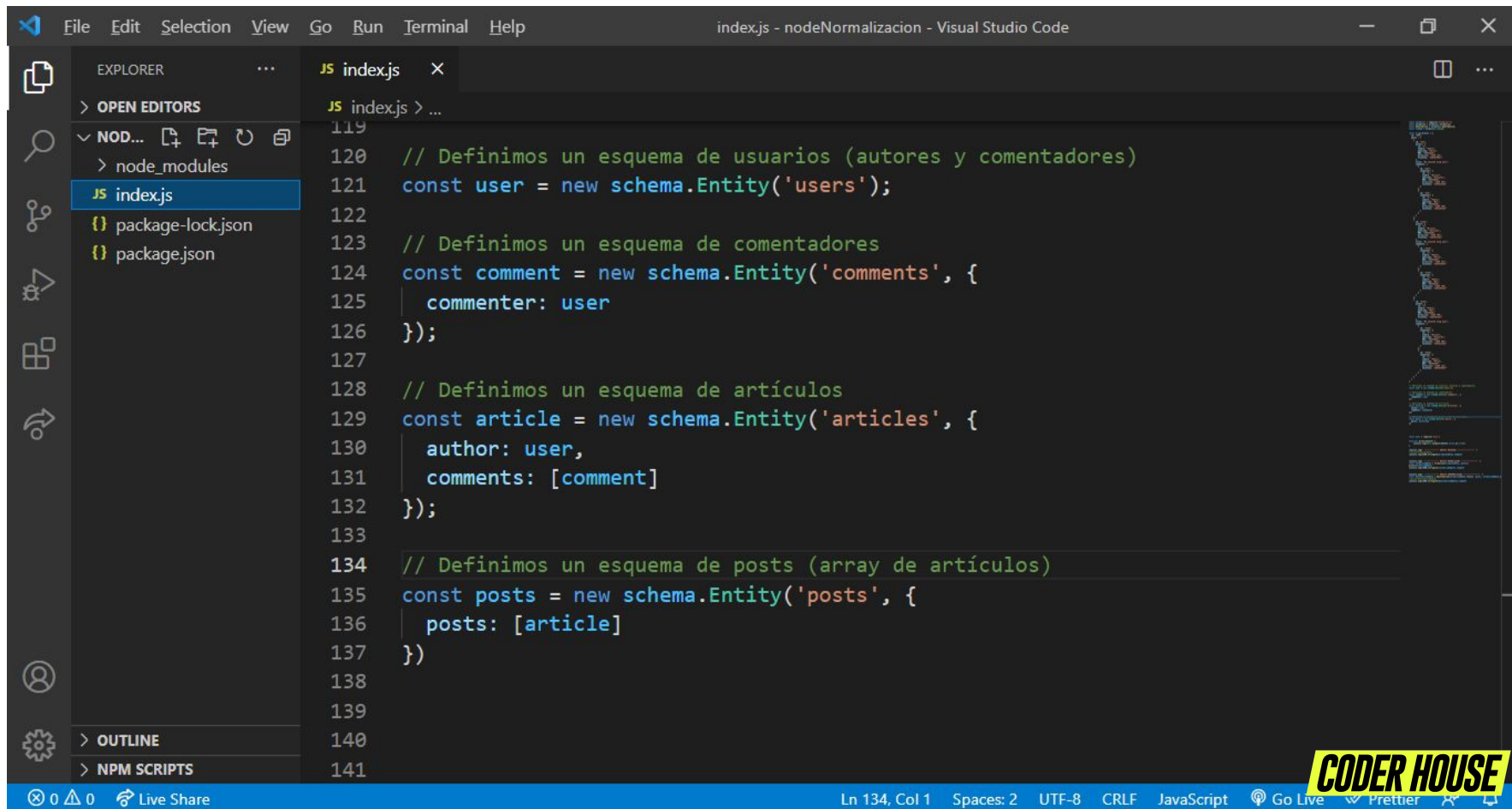
```
1 const normalizr = require("normalizr");
2 const normalize = normalizr.normalize;
3 const denormalize = normalizr.denormalize;
4 const schema = normalizr.schema;
5
6 const originalData = {
7   id: "999",
8   posts: [
9     {
10      id: "123",
11      author: {
12        id: "1",
13        nombre: "Pablo",
14        apellido: "Perez",
15        DNI: "20442654",
16        direccion: "CABA 123",
17        telefono: "1567876547"
18      },
19      title: "My awesome blog post",
20      comments: [
21        {
22          id: "324",
23          commenter: {
24            id: "2",
25            nombre: "Nicole",
26            apellido: "Gonzalez",
```

A red arrow points to the `originalData` object definition on line 6.

Visual Studio Code status bar at the bottom shows: Ln 142, Col 1 Spaces: 2 UTF-8 CRLF JavaScript Go Live Prettier

CODER HOUSE

Blog de artículos: *esquemas*



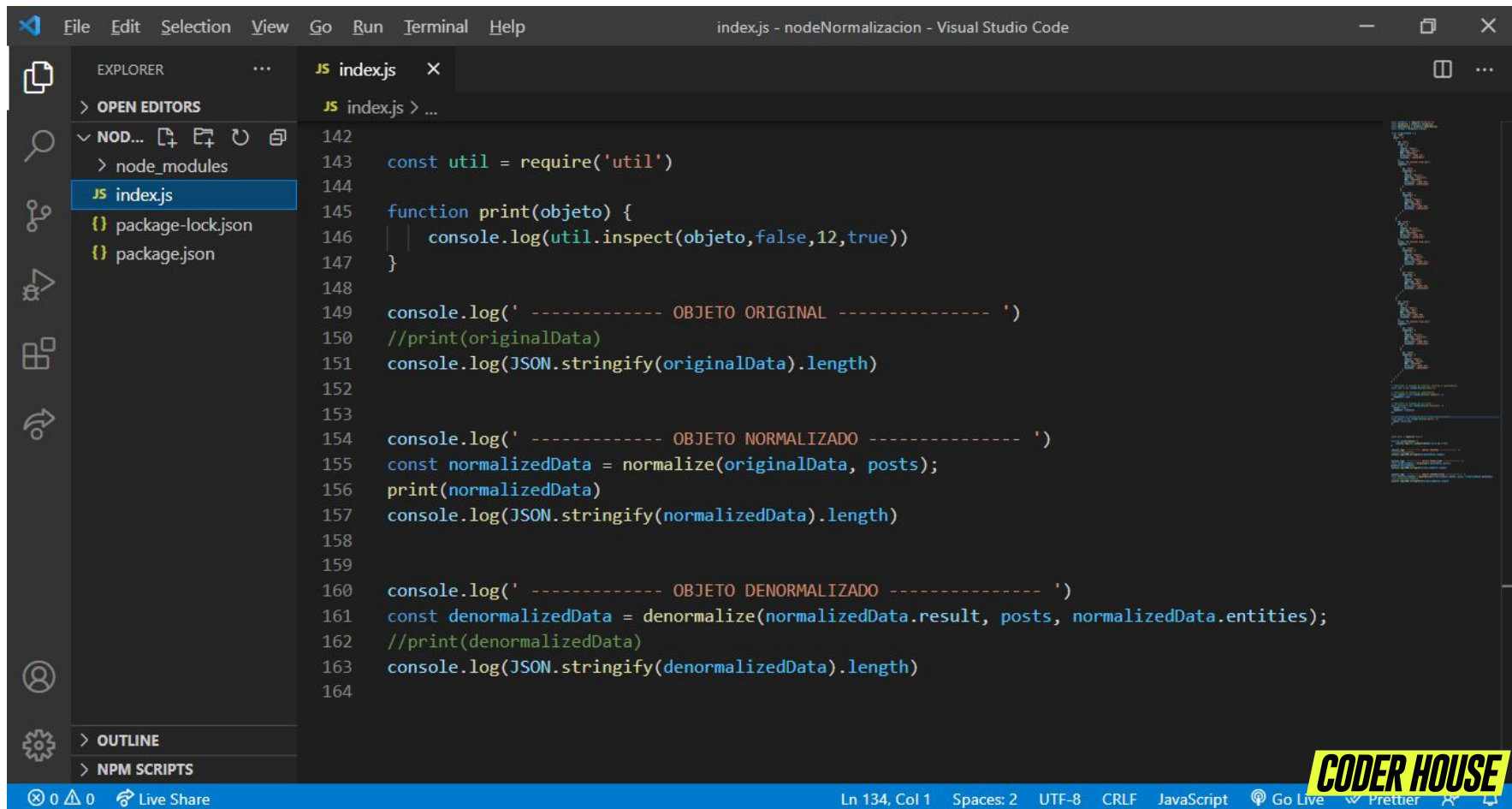
The image shows a screenshot of the Visual Studio Code editor interface. The title bar indicates the file is 'index.js - nodeNormalizacion - Visual Studio Code'. The Explorer sidebar on the left shows the project structure with 'index.js' selected. The main editor area displays the following JavaScript code:

```
119
120 // Definimos un esquema de usuarios (autores y comentadores)
121 const user = new schema.Entity('users');
122
123 // Definimos un esquema de comentadores
124 const comment = new schema.Entity('comments', {
125   commenter: user
126 });
127
128 // Definimos un esquema de artículos
129 const article = new schema.Entity('articles', {
130   author: user,
131   comments: [comment]
132 });
133
134 // Definimos un esquema de posts (array de artículos)
135 const posts = new schema.Entity('posts', {
136   posts: [article]
137 });
138
139
140
141
```

The status bar at the bottom shows 'Ln 134, Col 1', 'Spaces: 2', 'UTF-8', 'CRLF', 'JavaScript', and 'Go Live'.

CODER HOUSE

Blog de artículos: *procesos y comprobaciones*



Visual Studio Code interface showing a JavaScript file named `index.js` with the following code:

```
142
143  const util = require('util')
144
145  function print(objeto) {
146    |   console.log(util.inspect(objeto,false,12,true))
147  }
148
149  console.log(' ----- OBJETO ORIGINAL ----- ')
150  //print(originalData)
151  console.log(JSON.stringify(originalData).length)
152
153
154  console.log(' ----- OBJETO NORMALIZADO ----- ')
155  const normalizedData = normalize(originalData, posts);
156  print(normalizedData)
157  console.log(JSON.stringify(normalizedData).length)
158
159
160  console.log(' ----- OBJETO DENORMALIZADO ----- ')
161  const denormalizedData = denormalize(normalizedData.result, posts, normalizedData.entities);
162  //print(denormalizedData)
163  console.log(JSON.stringify(denormalizedData).length)
164
```

The Explorer sidebar shows the file structure:

- EXPLORED
- OPEN EDITORS
- NOD...
 - node_modules
 - JS index.js
 - package-lock.json
 - package.json
- OUTLINE
- NPM SCRIPTS

The status bar at the bottom indicates: Ln 134, Col 1, Spaces: 2, UTF-8, CRLF, JavaScript, Go Live, Prettier.

util.inspect

- **Node.js** proporciona una función **inspect** provista en el módulo **util** con fines de depuración. Esta devuelve una representación de cadena de un objeto que puede ser grande, complejo y con un alto nivel de anidamiento.
- Formato: `util.inspect(object[, showHidden[, depth[, colors]])`
- Ejemplo: `util.inspect(myObj, true, 7, true)`
 - El primer parámetro es el **objeto** a inspeccionar.
 - El segundo parámetro muestra todas las **propiedades** ocultas y no ocultas.
 - El tercer parámetro indica hasta qué **profundidad** es analizado el objeto.
 - El cuarto parámetro colorea la **salida**.



Blog de artículos: *resultados en consola*

```
λ node .
----- OBJETO ORIGINAL -----
1381
----- OBJETO NORMALIZADO -----
{
  entities: {
    users: {
      '1': {
        id: '1',
        nombre: 'Pablo',
        apellido: 'Perez',
        DNI: '20442654',
        direccion: 'CABA 123',
        telefono: '1567876547'
      },
      '2': {
        id: '2',
        nombre: 'Nicole',
        apellido: 'Gonzalez',
        DNI: '20442638',
        direccion: 'CABA 456',
        telefono: '1567811543'
      },
      '3': {
        id: '3',
        nombre: 'Pedro',
        apellido: 'Mei',
        DNI: '20446938',
        direccion: 'CABA 789',
        telefono: '1567291542'
      }
    },
    comments: {
      '324': { id: '324', commenter: '2' },
      '325': { id: '325', commenter: '3' },
      '1324': { id: '1324', commenter: '1' }
    }
  },
  articles: {
    '123': {
      id: '123',
      author: '1',
      title: 'My awesome blog post',
      comments: [ '324', '325' ]
    },
    '1123': {
      id: '1123',
      author: '2',
      title: 'My awesome blog post',
      comments: [ '1324', '1325' ]
    },
    '2123': {
      id: '2123',
      author: '3',
      title: 'My awesome blog post',
      comments: [ '2324', '2325' ]
    }
  },
  posts: { '999': { id: '999', posts: [ '123', '1123', '2123' ] } },
  result: '999'
}
961
----- OBJETO DENORMALIZADO -----
1381
```

961 < 1381

🎉

CODER HOUSE

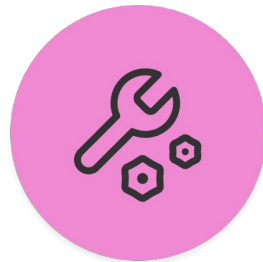
Objeto JSON de salida sin redundancia

```
const denormalizedData = {
  Entities: {
    Users: {
      1: {id: "1", nombre: "Pablo", apellido: "Perez", DNI: "20442654", direccion: "CABA
123", telefono: "1567876547"},
      2: {id: "2", nombre: "Nicole", apellido: "Gonzalez", DNI: "20442638", direccion: "CABA
456", telefono: "1567811543"},
      3: {id: "3", nombre: "Pedro", apellido: "Mei", DNI: "20446938", direccion: "CABA
789", telefono: "1567291542"}
    },
    Comments: {
      324: {id: "324", commenter: "2"},
      325: {id: "325", commenter: "3"},
      1324: {id: "1324", commenter: "1"},
      1325: {id: "1325", commenter: "3"},
      2324: {id: "2324", commenter: "2"},
      2325: {id: "2325", commenter: "1"}
    },
    Articles: {
      123: {id: "123", author: "1", title: "My awesome blog post", comments: ["324", "325"]},
      1123: {id: "1123", author: "2", title: "My awesome blog post", comments: ["1324", "1325"]},
      2123: {id: "2123", author: "3", title: "My awesome blog post", comments: ["2324", "2325"]}
    },
    Posts: {
      999: {id: "999", posts: ["123", "1123", "2123"]}
    }
  },
  result: "999"
}
```

Conclusiones



- **normalizedData** es es objeto resultante del proceso de normalización.
- Al revisar su estructura, se puede comprobar que las redundancias fueron eliminadas y su tamaño es menor que el del objeto original.
- Luego hacemos el proceso inverso con **denormalize**
- Verificamos la estructura del objeto desnormalizado y vemos que hemos recuperado los datos originales y el tamaño del objeto corresponde.
- Para el ejemplo dado, logramos una reducción de tamaño de **1381 bytes de los datos originales a 961 bytes normalizados**, lo que representa un **30% de compresión** de la información.
- Le eliminación de las redundancias implica una disminución del tamaño de la estructura que contiene nuestros datos.



Normalización y desnormalización con redundancia

Tiempo: 15 minutos



Dado el objeto en formato JSON *holding.json* (disponible en la carpeta de la clase) que representa la información correspondiente a un grupo de empresas:

1. Definir el esquema de normalización.
2. Obtener el objeto normalizado e imprimirlo por consola.
3. Desnormalizar el objeto obtenido en el punto anterior.
4. Imprimir la longitud del objeto original, del normalizado y del desnormalizado
5. Imprimir el porcentaje de compresión del proceso de normalización.

Comparar y analizar los resultados.

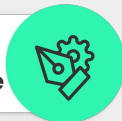


MOCKS Y NORMALIZACIÓN

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Desafío
entregable



>> Consigna 1:

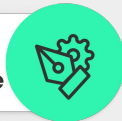
Sobre el desafío entregable de la clase 8 (sql y node: nuestra primera base de datos), crear una vista en forma de tabla que consuma desde la ruta '/api/productos-test' del servidor una lista con 5 **productos** generados al azar utilizando **Faker.js** como generador de información aleatoria de test (en lugar de tomarse desde la base de datos). Elegir apropiadamente los temas para conformar el objeto 'producto' (nombre, precio y foto).

NORMALIZACIÓN

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Desafío
entregable



>> Consigna 2:

Ahora, vamos a **reformatar el formato de los mensajes** y la forma de comunicación del chat (centro de mensajes).

El nuevo formato de mensaje será:

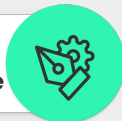
```
{
  author: {
    id: 'mail del usuario',
    nombre: 'nombre del usuario',
    apellido: 'apellido del usuario',
    edad: 'edad del usuario',
    alias: 'alias del usuario',
    avatar: 'url avatar (foto, logo) del usuario'
  },
  text: 'mensaje del usuario'
}
```

NORMALIZACIÓN

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Desafío
entregable



>> Aspectos a incluir en el entregable:

1. Modificar la persistencia de los mensajes para que utilicen un contenedor que permita guardar objetos anidados (archivos, mongodb, firebase).
2. El mensaje se envía del frontend hacia el backend, el cual lo almacenará en la base de datos elegida. Luego cuando el cliente se conecte o envíe un mensaje, recibirá un **array de mensajes** a representar en su vista.
3. El array que se devuelve debe estar **normalizado con normalizr**, conteniendo una entidad de autores. Considerar que el array tiene sus autores con su correspondiente id (mail del usuario), pero necesita incluir para el proceso de normalización un **id para todo el array** en su conjunto (podemos asignarle nosotros un valor fijo).
Ejemplo: { id: 'mensajes', mensajes: [] }
4. El frontend debería poseer el **mismo esquema de normalización** que el backend, para que este pueda desnormalizar y presentar la información adecuada en la vista.

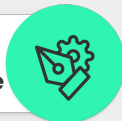
CODER HOUSE

NORMALIZACIÓN

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Desafío
entregable



5. Considerar que se puede **cambiar el nombre del id** que usa normalizr, agregando un tercer parametro a la función schema.Entity, por ejemplo:

```
const schemaAuthor = new schema.Entity('author', {...}, {idAttribute: 'email'});
```

En este schema cambia el nombre del id con que se normaliza el nombre de los autores a 'email'. Más info en la [web oficial](#).

6. Presentar en el frontend (a modo de test) el **porcentaje de compresión** de los mensajes recibidos. Puede ser en el título del centro de mensajes.

>> Nota: incluir en el frontend el script de normalizr de la siguiente cdn:

<https://cdn.jsdelivr.net/npm/normalizr@3.6.1/dist/normalizr.browser.min.js>

Así podremos utilizar los mismos métodos de normalizr que en el backend. Por ejemplo: new normalizr.schema.Entity , normalizr.denormalize(...,...,...)



Centro de Mensajes (Compresión: 62%)

d@s [28/3/2021 12:43:34] : holis!!!



g@s [28/3/2021 12:44:16] : Holaaaaaaaaaaaa



g@s [28/3/2021 12:44:33] : Como están!



c@s [28/3/2021 12:45:05] : heyyyyyyyyyy



Application Network Console Elements Sources >> ⚙️ ⋮ ✕

top Filter Default levels ⚙️

- ▶ {entities: {...}, result: "mensajes"} 7343 chat.js:20
- ▶ {id: "mensajes", mensajes: Array(44)} 11644 chat.js:25
- Porcentaje de compresión 63% chat.js:28
- ▼ {entities: {...}, result: "mensajes"} ⓘ chat.js:20
 - ▼ entities:
 - ▼ author:
 - ▶ c@s: {email: "c@s", nombre: "Cecilia", apellido: "Sanchez", edad: ...}
 - ▶ d@s: {email: "d@s", nombre: "Daniel", apellido: "Sanchez", edad: ...}
 - ▶ g@s: {email: "g@s", nombre: "Gabriela", apellido: "Sanchez", edad: ...}
 - ▶ __proto__: Object
 - ▶ post: {6060a426412d552b8cae776d: {...}, 6060a450412d552b8cae776e: {...}, ...}
 - ▼ posts:
 - ▶ mensajes: {id: "mensajes", mensajes: Array(45)}
 - ▶ __proto__: Object
 - ▶ __proto__: Object
 - result: "mensajes"
 - ▶ __proto__: Object
- 7497 chat.js:25
- ▼ {id: "mensajes", mensajes: Array(45)} ⓘ
 - id: "mensajes"
 - ▶ mensajes: (45) [...], [...], [...], [...], [...], [...], [...], [...], [...], [...], ...
 - ▶ __proto__: Object
- 11939
- Porcentaje de compresión 62% chat.js:28

¿PREGUNTAS?



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Normalización y Desnormalización
- Normalizr.js
- Ejemplo: Blog de Artículos



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE