Clase 9

Construcción de APIs

IIC2513 - Tecnologías y Aplicaciones Web

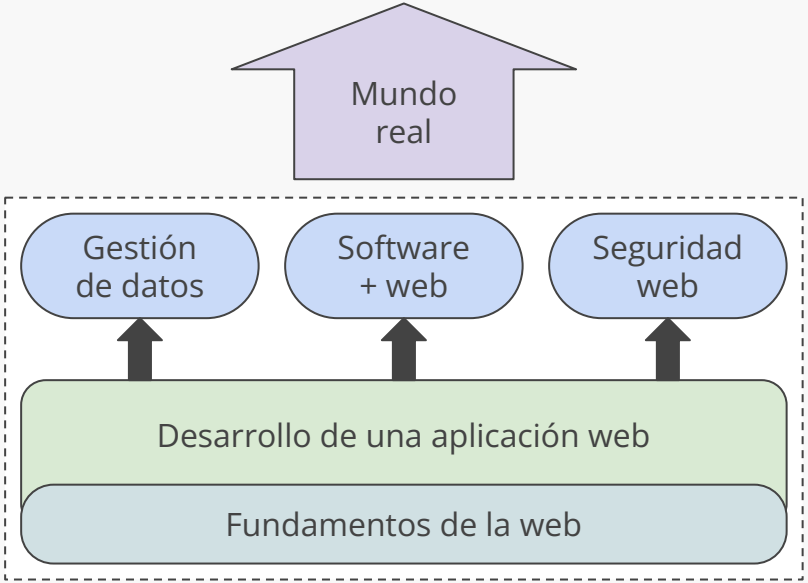
Antonio Ossa Guerra
aaossa@ing.puc.cl

Anuncios

1. Control 2 ya disponible
2. Hay ayudantía de esto el viernes
3. Recuerden responder la ECA

Contenidos del curso

(1) Fundamentos de la web
(2) Desarrollo de una app web
(3) Gestión de datos
(4) Software + web
(5) Seguridad web
(6) El mundo real

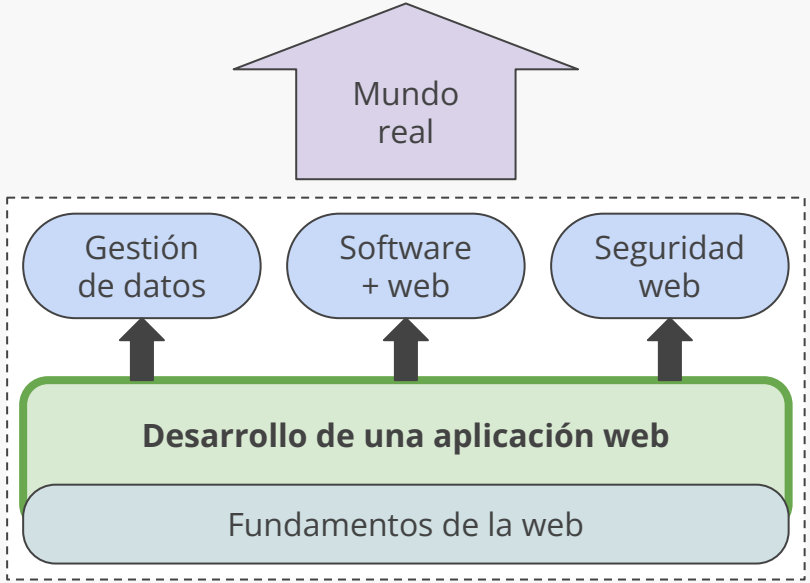


Contenidos del curso

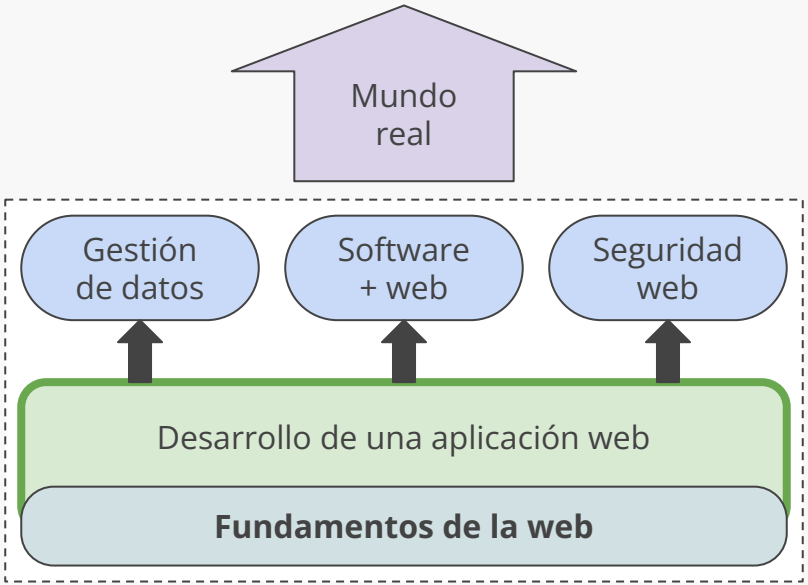
Desarrollo de una aplicación web

Veremos herramientas como React, Node.js, **Koa** y **Express**, y técnicas como CSS avanzado y el uso de *Design Systems*

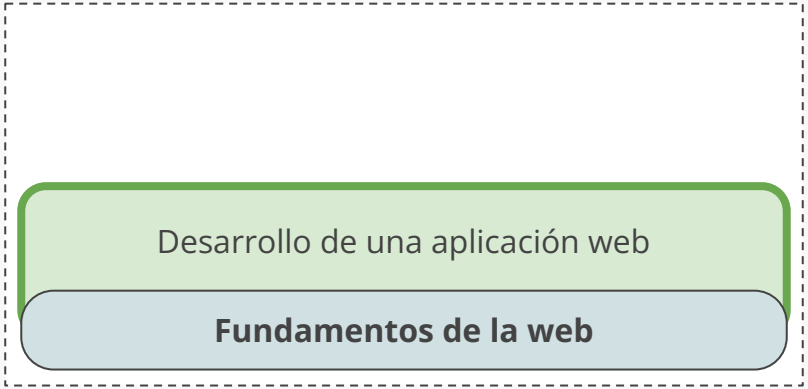
... para desarrollar aplicaciones modernas tanto en el lado del cliente (front-end) como en el lado del **servidor** (back-end)



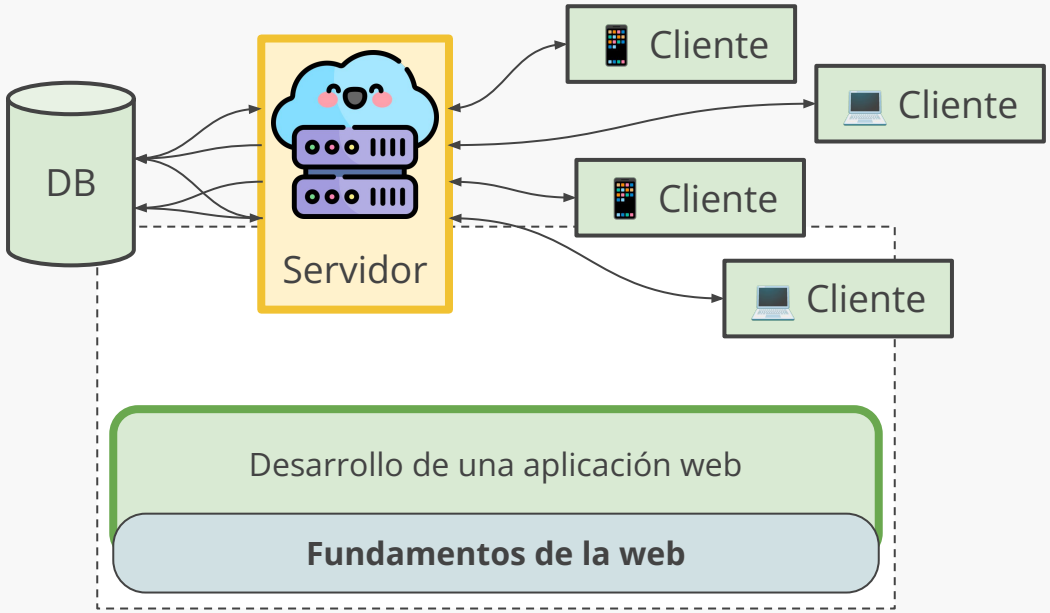
Contenidos del curso



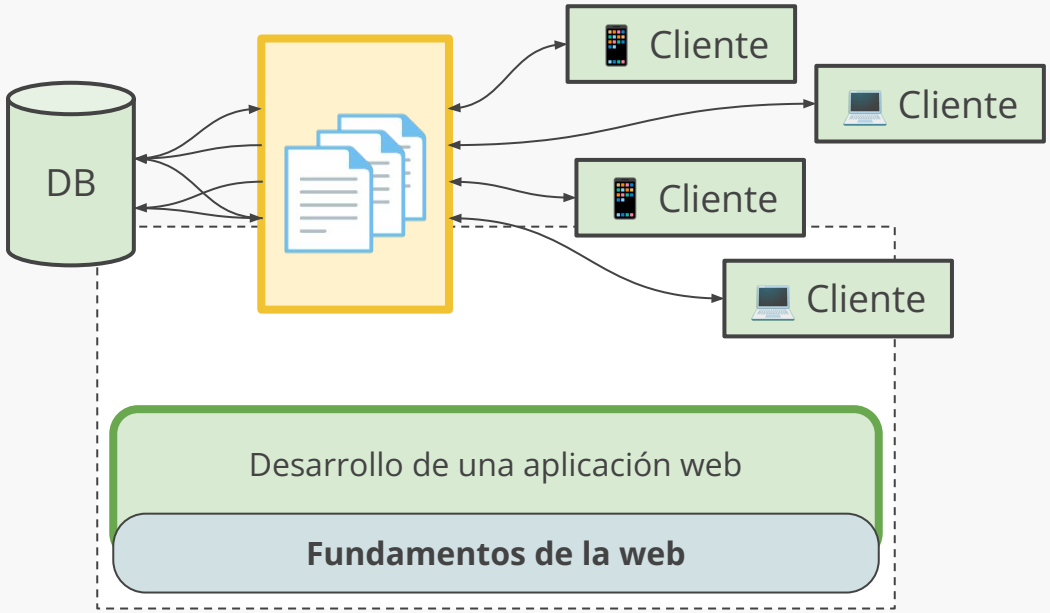
Contenidos del curso



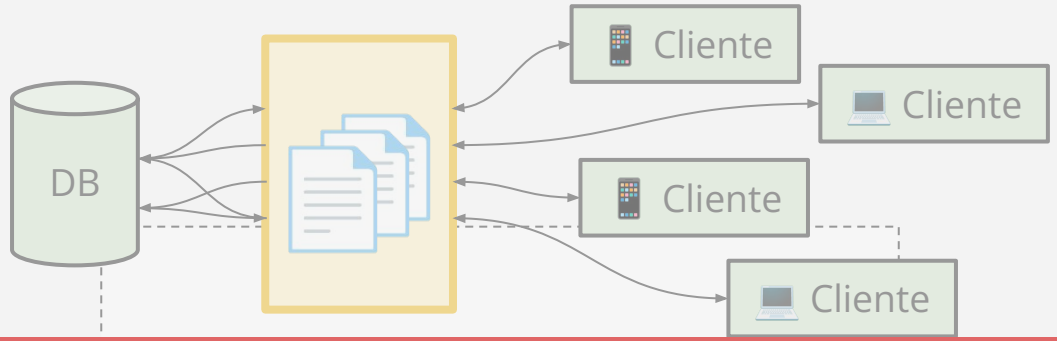
Motivación


Cliente
Cliente
DB
Servidor
Cliente
Cliente
Desarrollo de una aplicación web
Fundamentos de la web

Motivación

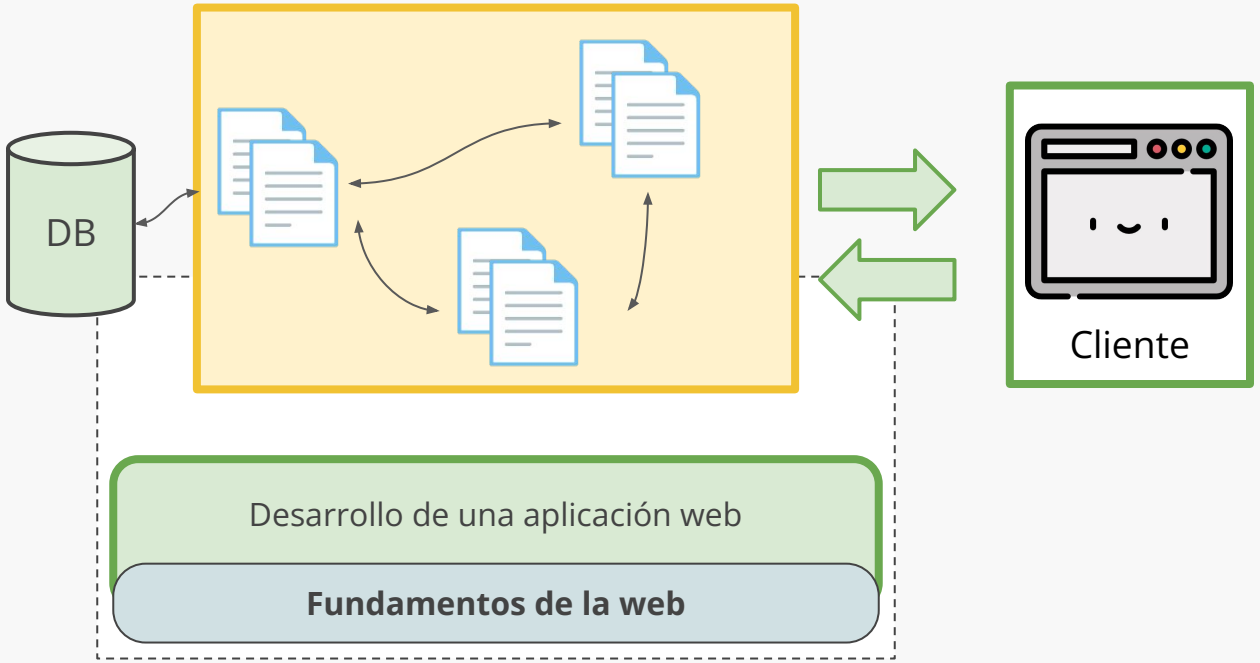

Cliente
Cliente
Cliente
Cliente
DB
Desarrollo de una aplicación web
Fundamentos de la web

Motivación



Pregunta#¿Cómo es el código de un servidor?

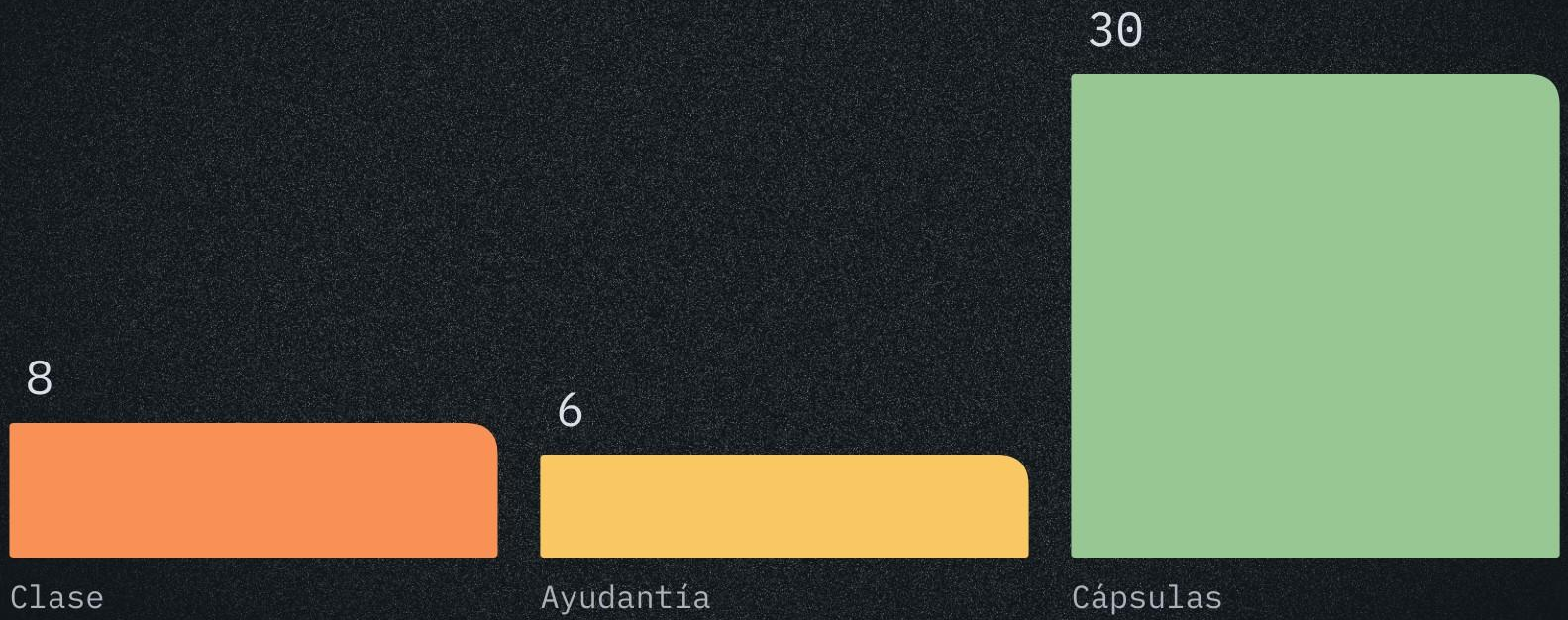
Motivación


DB
Cliente
Desarrollo de una aplicación web
Fundamentos de la web

Menti*#Primero unas preguntas*

Mentimeter

¿Dónde prefieren ver código? 👀



32

Mentimeter

¡Gracias!

Todas son consideradas para hacer que el curso sea una mejor experiencia para ustedes

En el caso del proyecto...

Lenguaje



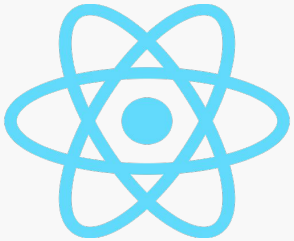


Back-end (servidor)

express

koa

Front-end (cliente)



NEXT.JS

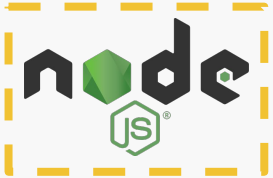


tailwindcss

En el caso del proyecto...

Lenguaje



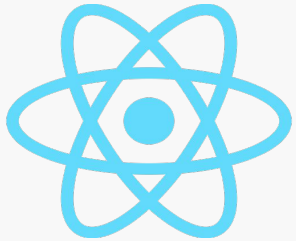


Back-end (servidor)





Front-end (cliente)

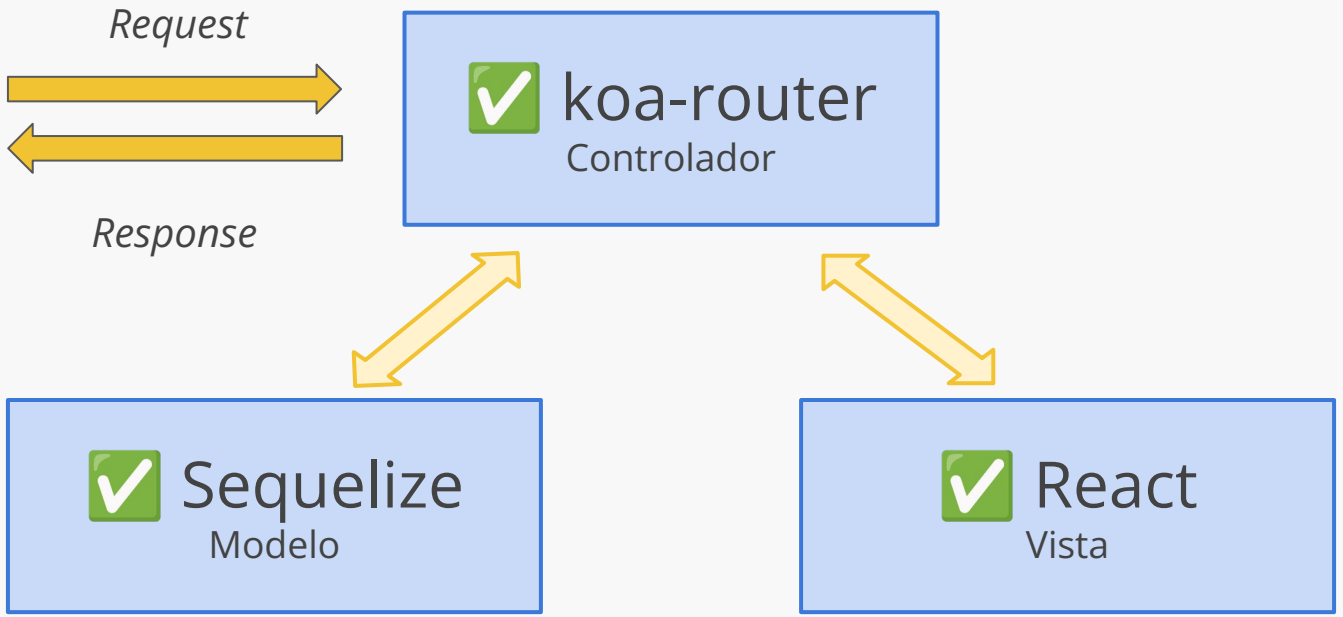


NEXT.JS



tailwindcss

¿Cómo se ve el proyecto del curso?



Agenda

✨ **Estructura de un proyecto**

“Hello world”

package.json

El archivo package.json es un archivo de configuración para npm que se ubica en la raíz de un proyecto y contiene metadata del *package* o proyecto.

- Lista las dependencias del proyecto
- Especifica las versiones del proyecto
- Permite que la ejecución sea reproducible

```
{
  "name": "demo-iic2513",
  "description": "",
  "version": "1.0.0",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/IIC2513/demo-iic2513.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/IIC2513/demo-iic2513/issues"
  },
  "homepage": "https://github.com/IIC2513/demo-iic2513"
}

```

Fuente: [Creating a package.json file | npm Docs]

`package.json`

El archivo, al menos, deberá contener metadata del *package* o proyecto, por ejemplo:

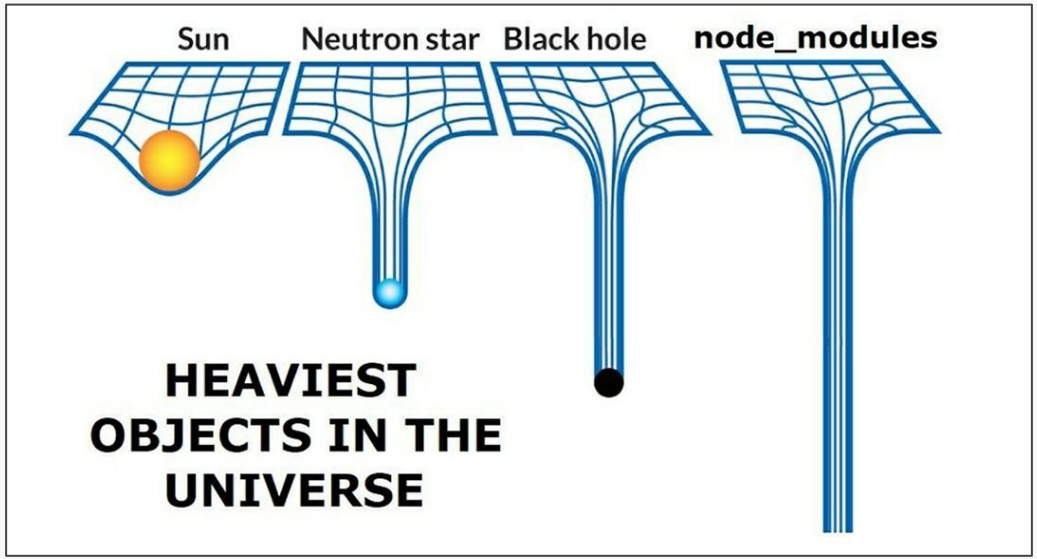
- Nombre y versión (requeridos)
- Autor
- Descripción
- Scripts
- *Keywords*
- Licencia
- *Bugs*
- *Homepage*

Tipo	Campo	Descripción
STRING	`name`	Nombre del proyecto, librería o paquete. Se recomienda que coincida con el repositorio.
STRING	`version`	Versión del paquete. Generalmente se utiliza **semver** (lo veremos más adelante).
STRING	`description`	Descripción breve del paquete o proyecto.
STRING	`type`	Tipo de sistema de módulos a utilizar. Con `module` usa ESM, en caso contrario usa CommonJS.
STRING	`main`	**Punto de entrada** del proyecto. Suele ser `index.js` (node) o `index.html` (browser).
STRING	`module`	Idem al anterior, pero respecto a ES Modules en lugar de **CommonJS**.
OBJECT	`scripts`	Colección de scripts del proyecto (lo veremos más adelante).
ARRAY	`keywords`	ARRAY de STRING con palabras clave relacionadas con el proyecto. Util en búsquedas.
STRING	`author`	Nombre del autor del paquete o un ARRAY con `name`, `email` y/o `url`.
STRING	`license`	Tipo de licencia del paquete o proyecto. Por defecto, ISC.
OBJECT	`dependencies`	Colección de paquetes para producción y la versión instalada.
OBJECT	`devDependencies`	Colección de paquetes para desarrollo y la versión instalada.
STRING	`homepage`	URL de la página principal del paquete.
OBJECT	`repository`	URL del repositorio. Se debe indicar `type` (git, svn...) y `url` (ruta).
OBJECT	`bugs`	Objeto con campo `url` con la URL de la página de issues del proyecto.

Fuente: package.json | npm Docs Tabla: El archivo package.json - Javascript en español - Lenguaje JS

`node_modules`

Cualquier paquete que se instale dentro de un proyecto, quedará en la carpeta `node_modules`, cada uno junto a todo su contenido y sus dependencias respectivas dentro de su propio `node_modules`



Fuente: La carpeta node_modules - Javascript en español - Lenguaje JS

Algunas alternativas para iniciar un proyecto



Usando npm:

`npm init [-y/--yes]`



Usando Yarn:

`yarn init [-y/--yes]`



Usando Vite:

`npm create vite@latest`

`yarn create vite@latest`

Agenda

Estructura de un proyecto

✨ **“Hello world”**

Importante:

Todo el código de esta clase se encuentra implementado en [IIC2513/Mi-primera-API]

Se recomienda seguir los pasos descritos de manera local y solo consultar la implementación al final o en caso de dudas (*hint*: ver los *commits*)

Agenda

- Estructura de un proyecto
- ✨ **“Hello world”**
  - El primer *router*
  - Configurando Sequelize
  - Configurando Postgres
  - Creando el primer recurso
  - Eliminando la base de datos

Koa - next generation web framework for node.js:
Página principal de Koa, un framework web minimalista para Node.js



next generation web framework for node.js

“Hello world” (Koa)

koa

1. Crear proyecto:
   `yarn init`
2. Agregar dependencias de Koa y similares:
   `yarn add koa koa-logger koa-router`
3. **Crear archivo `src/index.js` con código base**
4. Agregar dependencia dev Nodemon:
   `yarn add nodemon --dev`
5. Agregar script en package.json:
   ```
   "scripts": {
       "dev": "nodemon src/index.js"
   }
   ```
6. Probar servidor, que debería mostrar el mensaje en toda ruta de `localhost:3000`:
   `yarn dev`

```
1  // src/index.js
2  const Koa = require('koa');
3  const Logger = require('koa-logger');
4
5  const app = new Koa();
6
7  app.use(Logger());
8
9  app.use((ctx, next) => {
10   ctx.body = 'Hola mundo';
11 });
12
13 app.listen(3000);
14
```

Extra

Express - Node.js web application framework:
Página principal de Express, un framework para aplicaciones web en JavaScript bastante popular

express | search | Home | Getting started | Guide | API reference | Advanced topics | Resources | Support | Blog | English

Express 5.1.0

Fast, unopinionated, minimalist web framework for Node.js

```
$ npm install express --save
```

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Express@5.1.0: Now the Default on npm with LTS Timeline

Express 5.1.0 is now the default on npm, and we're introducing an official LTS schedule for the v4 and v5 release lines. Check out our latest blog for more information.

“Hello world” (Express)

express

1. Crear proyecto:
   `yarn init`
2. Agregar Express como dependencia:
   `yarn add express`
3. **Crear archivo `src/app.js` con código base**
4. Agregar dependencia dev Nodemon:
   `yarn add nodemon --dev`
5. Agregar script en package.json:
   ```
   "scripts": {
       "dev": "nodemon src/app.js"
   }
   ```
6. Probar servidor, que debería mostrar el mensaje en toda ruta de `localhost:3000`:
   `yarn dev`

```
1  // src/app.js
2  const express = require('express');
3  const app = express();
4  const port = 3000;
5
6  app.get('/', (req, res) => {
7    res.send('Hello World!')
8  });
9
10 app.listen(port, () => {
11   console.log(`Listening on port ${port}`);
12 });
13
```

nodemon - reload, automatically:
Librería que monitorea cambios en archivos de un servidor para reiniciar automáticamente



nodemon reload, automatically.

Nodemon is a utility **depended on about 3 million projects**, that will monitor for any changes in your source and automatically restart your server. Perfect for development.

Swap **nodemon** instead of **node** to run your code, and now your process will automatically restart when your code changes. To install, get Node.js, then from your terminal run:

```
npm install -g nodemon
```

Features

Agenda

Estructura de un proyecto

“Hello world”

✨ **El primer *router***

Configurando Sequelize

Configurando Postgres

Creando el primer recurso

Eliminando la base de datos

koajs/router: Router middleware for Koa:
Código y documentación de koa-router disponible en el repositorio oficial en GitHub

README | Code of conduct | MIT license | Security

@koa/router

> Router middleware for Koa. Maintained by Forward Email and Lad.

CI passing | code style XO | styled with prettier | made with lass | license MIT

Table of Contents

- Features
- Migrating to 7 / Koa 2
- Install
- Typescript Support
- API Reference
- Contributors
- License

Features

- Express-style routing (`app.get` , `app.put` , `app.post` , etc.)
- Named URL parameters
- Named routes with URL generation
- Match routes with specific host

El primer *router* (Koa)

koa

1. Crear carpeta `src/routes/` con archivos
2. **Creamos nuestro primer router en `src/routes/movies.js` con código base**
3. Creamos archivo `src/routes/index.js` donde se conectarán los routers
4. Modificamos `src/index.js` para utilizar el router que reúne a todos los demás
5. Probar ruta `localhost:3000/movies`, que debería mostrar el mensaje:
   `yarn dev`

```
1  // src/routes/movies.js
2  const Router = require('koa-router');
3  const router = new Router();
4
5  router.get('/', (ctx) ⇒ {
6    ctx.body = 'GET /movies';
7  });
8
9  module.exports = router;
10
```

El primer *router* (Koa)

koa

1. Crear carpeta `src/routes/` con archivos
2. Creamos nuestro primer router en `src/routes/movies.js` con código base
3. **Creamos archivo `src/routes/index.js` donde se conectarán los routers**
4. Modificamos `src/index.js` para utilizar el router que reúne a todos los demás
5. Probar ruta `localhost:3000/movies`, que debería mostrar el mensaje:
   `yarn dev`

```
1 // src/routes/index.js
2 const Router = require('koa-router');
3 const movies = require('./movies');
4
5 const router = new Router();
6 router.use('/movies', movies.routes());
7
8 module.exports = router;
9
```

El primer *router* (Koa)

koa

1. Crear carpeta `src/routes/` con archivos
2. Creamos nuestro primer router en `src/routes/movies.js` con código base
3. Creamos archivo `src/routes/index.js` donde se conectarán los routers
4. **Modificamos `src/index.js` para utilizar el router que reúne a todos los demás**
5. Probar ruta `localhost:3000/movies`, que debería mostrar el mensaje:
   `yarn dev`

```
1  // src/index.js
2  const Koa = require('koa');
3  const Logger = require('koa-logger');
4  const router = require('./routes');
5
6  const app = new Koa();
7
8  app.use(Logger());
9
10 app.use(router.routes());
11
12 app.listen(3000);
13
```

Extra

Express routing: Guía en la documentación de Express sobre *routing*

express search Home Getting started **Guide** API reference Advanced topics Resources Support Blog English

Routing

Routing refers to how an application's endpoints (URIs) respond to client requests. For an introduction to routing, see Basic routing.

You define routing using methods of the Express `app` object that correspond to HTTP methods; for example, `app.get()` to handle GET requests and `app.post` to handle POST requests. For a full list, see app.METHOD. You can also use app.all() to handle all HTTP methods and app.use() to specify middleware as the callback function (See Using middleware for details).

These routing methods specify a callback function (sometimes called "handler functions") called when the application receives a request to the specified route (endpoint) and HTTP method. In other words, the application "listens" for requests that match the specified route(s) and method(s), and when it detects a match, it calls the specified callback function.

In fact, the routing methods can have more than one callback function as arguments. With multiple callback functions, it is important to provide `next` as an argument to the callback function and then call `next()` within the body of the function to hand off control to the next callback.

The following code is an example of a very basic route.

```
const express = require('express')
const app = express()

// respond with "hello world" when a GET request is made to the homepage
app.get('/', (req, res) => {
  res.send('hello world')
})
```

Route methods

El primer *router* (Express)

express

1. Crear carpeta `src/routes/` con archivos
2. **Creamos nuestro primer router en `src/routes/movies.js` con código base**
3. Modificamos `src/app.js` para conectar el que acabamos de crear
4. Probar ruta `localhost:3000/movies`, que debería mostrar el mensaje:
   `yarn dev`

```
// src/routes/movies.js
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.send('GET /movies');
})

module.exports = router;

```

El primer *router* (Express)

express

1. Crear carpeta `src/routes/` con archivos
2. Creamos nuestro primer router en `src/routes/movies.js` con código base
3. **Modificamos `src/app.js` para conectar el que acabamos de crear**
4. Probar ruta `localhost:3000/movies`, que debería mostrar el mensaje:
   `yarn dev`

```
1  // src/app.js
2  const express = require('express');
3  const movies = require('./routes/movies');
4  const app = express();
5  const port = 3000;
6
7  app.use('/movies', movies);
8
9  app.listen(port, () => {
10   console.log(`Listening on port ${port}`);
11 });
12
```

Agenda

Estructura de un proyecto

“Hello world”

El primer *router*

✨ **Configurando Sequelize**

Configurando Postgres

Creando el primer recurso

Eliminando la base de datos

Sequelize | Feature-rich ORM for modern TypeScript & JavaScript:
Página principal y documentación de Sequelize

Sequelize v6 - stable API References Slack GitHub Security Search CTRL K



Sequelize

Sequelize is a modern TypeScript and Node.js ORM for Oracle, Postgres, MySQL, MariaDB, SQLite and SQL Server, and more. Featuring solid transaction support, relations, eager and lazy loading, read replication and more.

Getting Started | API Reference | Upgrade to v6 | Support us

1. INSTALL DEPENDENCIES

```
npm install sequelize sqlite3
# or
yarn add sequelize sqlite3
```

Configurando Sequelize

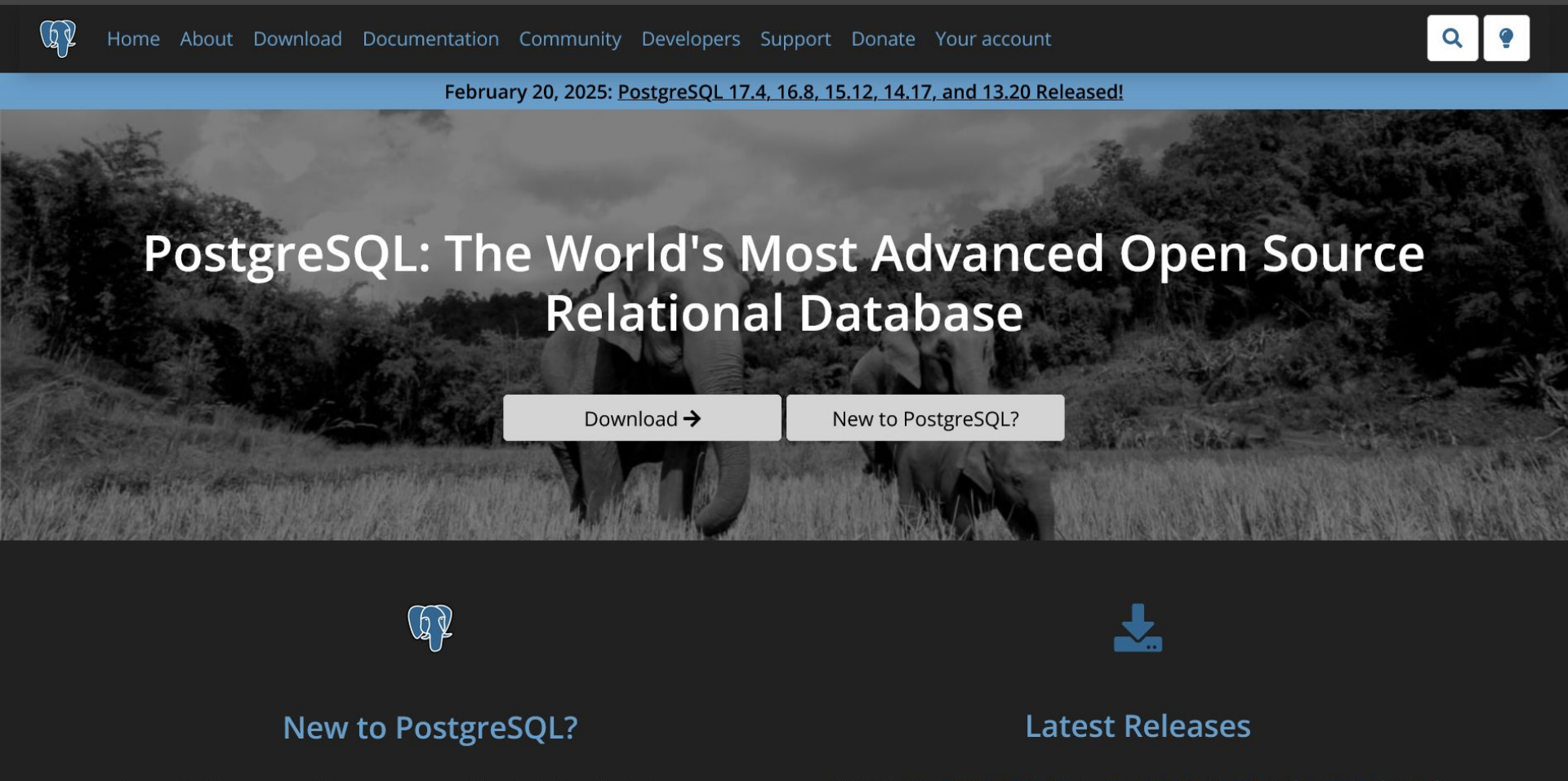
1. Agregar dependencias de Sequelize y Postgres:
   `yarn add sequelize pg pg-hstore`
2. Agregar dependencia dev Sequelize CLI:
   `yarn add sequelize-cli –dev`
3. Carpetas base para Sequelize:
   `yarn sequelize-cli init`
4. Mover carpetas creadas por el comando (`config`, `migrations`, `models`, `seeders`) dentro de `src/`
5. Crear archivo .sequelizerc para indicar nueva ubicación de archivos relevantes para Sequelize

```
1  // .sequelizerc
2  const path = require('path');
3
4  module.exports = {
5    'config': path.resolve('src', 'config', 'config.json'),
6    'models-path': path.resolve('src', 'models'),
7    'seeders-path': path.resolve('src', 'seeders'),
8    'migrations-path': path.resolve('src', 'migrations')
9  }
10
```

Agenda

- Estructura de un proyecto
- “Hello world”
- El primer *router*
- Configurando Sequelize
- ✨ **Configurando Postgres**
- Creando el primer recurso
- Eliminando la base de datos

PostgreSQL: The world's most advanced open source database:
Página principal y documentación de PostgreSQL



Configurando Postgres

1. Instalar, configurar e iniciar Postgres
   `sudo service postgresql start`

2. Ingresar al usuario de Postgres:
   `sudo -i -u postgres`

3. Crear base de datos (lo opuesto es usar `dropdb`):
   `createdb demo_dev`

4. Agregar usuario (lo opuesto es usar `dropuser`):
   `createuser demo_user`

5. Crear credenciales y dar permisos a usuario sobre la base de datos:
   ```
   psql  # iniciará la consola de postgres
   alter user demo_user with encrypted password 'pwd';
   grant all privileges on database demo_dev to demo_user;
   alter user demo_user createdb;
   exit  # cierra la consola de postgres
   ```

6. Salimos del usuario de postgres (distinto del `exit` anterior):
   `exit`

7. **Actualizar credenciales y datos en archivo `src/config/config.json`**

8. De ser necesario, crear base de datos con Sequelize CLI (da error si ya la creamos):
   `yarn sequelize-cli db:create`

```
1  # src/config/config.json
2  {
3    "development": {
4      "username": "aaossa",
5      "password": "pwd",
6      "database": "demo_dev",
7      "host": "127.0.0.1",
8      "dialect": "postgres"
9    },
10   ...
11 }
12
```

Agenda

Estructura de un proyecto

“Hello world”

- El primer *router*
- Configurando Sequelize
- Configurando Postgres
- ✨ **Creando el primer recurso**
- Eliminando la base de datos

Creando el primer recurso

1. Creamos el modelo para la clase Movie (y su migración) con sus atributos usando Sequelize CLI
   `yarn sequelize-cli model:generate --name Movie --attributes title:string,genre:string,description:string,rating:float`
2. Ejecutar migración para crear la clase:
   `yarn sequelize-cli db:migrate`
3. Creamos una *seed* para la tabla de películas:
   `yarn sequelize-cli seed:generate --name first-movies`
4. **Definimos las *seeds* en el archivo creado en src/seeders**
5. Agregamos las *seeds* correspondientes a películas:
   `yarn sequelize-cli db:seed:all`
6. Actualizamos la ruta en el controlador de películas para cargar todas las películas disponibles

```
// src/seeders/*-first-movies.js
'use strict';

module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.bulkInsert('Movies', [{
      title: 'Arrival',
      genre: 'Sci-fi/Thriller',
      description: 'Louise Banks, a linguistics expert,
        along with her team, must interpret the language
        of aliens who have come to Earth in a mysterious
        spaceship.',
      rating: 7.9,
      createdAt: new Date(),
      updatedAt: new Date()
    }, {
      title: 'Gone Girl',
      genre: 'Thriller/Mystery',
      description: 'Nick Dunne discovers that the entire
        media focus has shifted on him when his wife,
        Amy Dunne, mysteriously disappears on the day of
        their fifth wedding anniversary.',
      rating: 8.1,
      createdAt: new Date(),
      updatedAt: new Date()
    }], {});
  },

  async down(queryInterface, Sequelize) {
    await queryInterface.bulkDelete('Movies', null, {});
  }
};

```

Creando el primer recurso (Koa)

koa

1. Creamos el modelo para la clase Movie (y su migración) con sus atributos usando Sequelize CLI
   `yarn sequelize-cli model:generate --name Movie --attributes title:string,genre:string,description:string,rating:float`
2. Ejecutar migración para crear la clase:
   `yarn sequelize-cli db:migrate`
3. Creamos una *seed* para la tabla de películas:
   `yarn sequelize-cli seed:generate --name first-movies`
4. Definimos las *seeds* en el archivo creado en src/seeders
5. Agregamos las *seeds* correspondientes a películas:
   `yarn sequelize-cli db:seed:all`
6. **Actualizamos la ruta en el controlador de películas para cargar todas las películas disponibles**

```
1  // src/routes/movies.js
2  const Router = require('koa-router');
3  const router = new Router();
4  const { Movie } = require('../models');
5
6  router.get('/', async (ctx) => {
7    try {
8      const movies = await Movie.findAll();
9      ctx.body = movies;
10   } catch (error) {
11     console.log(error);
12     ctx.throw(404);
13   }
14 });
15
16 module.exports = router;
17
```

Creando el primer recurso (Express)

express

1. Creamos el modelo para la clase Movie (y su migración) con sus atributos usando Sequelize CLI
   `yarn sequelize-cli model:generate --name Movie --attributes title:string,genre:string,description:string,rating:float`
2. Ejecutar migración para crear la clase:
   `yarn sequelize-cli db:migrate`
3. Creamos una *seed* para la tabla de películas:
   `yarn sequelize-cli seed:generate --name first-movies`
4. Definimos las *seeds* en el archivo creado en src/seeders
5. Agregamos las *seeds* correspondientes a películas:
   `yarn sequelize-cli db:seed:all`
6. **Actualizamos la ruta en el controlador de películas para cargar todas las películas disponibles**

```
1  // src/routes/movies.js
2  const express = require('express');
3  const router = express.Router();
4  const { Movie } = require('../models');
5
6  router.get('/', async (req, res) => {
7    try {
8      const movies = await Movie.findAll();
9      res.json(movies);
10   } catch (error) {
11     console.error(error);
12     res.status(404).send('Not Found');
13   }
14 });
15
16 module.exports = router;
17
```

Próximos pasos

Esta API no está completa, queda bastante que se puede hacer. ¿Cómo continuar?

1. Implementar hacer GET de una película particular (`/movies/1`)
2. Completar CRUD usando GET, POST, DELETE y PUT cuando corresponda
3. Usar variables de entorno para las credenciales de la DB (`dotenv` o `dotenvx`)
4. Agregar un nuevo recurso como Users (`/users`) con su CRUD
5. Crear un nuevo modelo que una Movies y Users como Likes y la relación apropiada
6. Consumir API desde un front-end con React

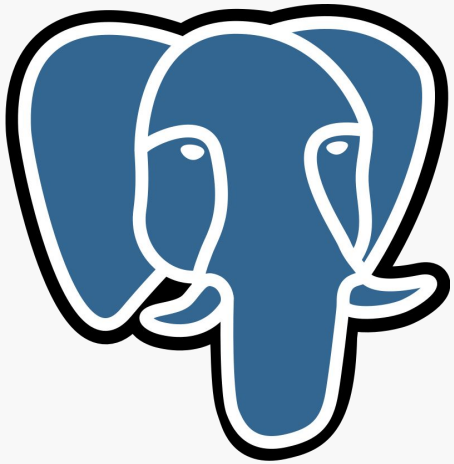
(Recursos adicionales de Express y Sequelize CLI)

Agenda

- Estructura de un proyecto
- “Hello world”
  - El primer *router*
  - Configurando Sequelize
  - Configurando Postgres
  - Creando el primer recurso
  - ✨ **Eliminando la base de datos**

Eliminando la base de datos

1. Ingresar al usuario de postgres:
   `sudo -i -u postgres`
2. Eliminamos la base de datos:
   `dropdb demo_dev`
3. Eliminamos el usuario:
   `dropuser demo_user`
4. Salimos del usuario de postgres:
   `exit`
5. Detenemos el servicio de Postgres:
   `sudo service postgresql stop`



Agenda

Estructura de un proyecto

“Hello world”

✨ **¿Qué se viene pronto?**

Próximas fechas importantes: clases y ayudantías

	Clases y ayudantías						
	Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Sábado**	**Domingo**
Semana 5 (31/3 - 06/4)		Node.js		Desarrollando servidores	Ayudantía: Consumo de APIs		
Semana 6 (07/4 - 13/4)		HOY		QA, CI & CD	Ayudantía: Construcción de APIs		
Semana 7 (14/4 - 20/4)		Fundamentos de front-end		Jueves Santo	Viernes Santo	Sábado Santo	Domingo de Resurrección
Semana 8 (21/4 - 27/4)		Charla Front-end (Florencia Valladares)		Diseño web en la práctica			
Semana 9 (28/4 - 04/5)	UC: Semana de Receso	UC: Semana de Receso	UC: Semana de Receso	UC: Semana de Receso	UC: Semana de Receso	UC: Semana de Receso	

Próximas fechas importantes: evaluaciones

	Evaluaciones						
	Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Sábado**	**Domingo**
Semana 5 (31/3 - 06/4)					Fin T1 (React)		
Semana 6 (07/4 - 13/4)	Control 2	HOY			Inicio T2 API		
Semana 7 (14/4 - 20/4)				Jueves Santo	Viernes Santo	Sábado Santo	Domingo de Resurrección
Semana 8 (21/4 - 27/4)	UC: Sin eval.	UC: Sin eval.	Fin T2 (API)				
Semana 9 (28/4 - 04/5)	UC: Semana de Receso	UC: Semana de Receso	UC: Semana de Receso	UC: Semana de Receso	UC: Semana de Receso	UC: Semana de Receso	UC: Sin eval.

Clase 9

Construcción de APIs

IIC2513 - Tecnologías y Aplicaciones Web

Antonio Ossa Guerra
aaossa@ing.puc.cl