# Llenguatge de marques i sistemes de gestió de la informació

R.A 4: Estableix mecanismes de validació de documents per a l'intercanvi d'informació utilitzant mètodes per definir-ne la sintaxi i l'estructura.

# JSON

**JSON** (JavaScript Object Notation) és un format lleuger d'intercanvi de dades que és fàcil de llegir i escriure tant per a humans com per a màquines. És una alternativa popular a XML a causa de la seva simplicitat i compatibilitat amb molts llenguatges de programació.

JSON està basat en dues estructures:

- **Objectes**: Una col·lecció de parells clau-valor tancats entre claus `{}`

```
{
  "nom": "Joan",
  "edat": 30,
  "casat": true
}
```

- **Arrays**: Una llista ordenada de valors tancats entre claudàtors `[]`.

```
[
  "poma",
  "plàtan",
  "cirera"
]
```

# JSON - Avantatges i Sintaxis

Avantatges:

- **Simplicitat**: JSON és fàcil d'entendre i escriure.
- **Lleuger**: Menys sobrecàrrega en comparació amb XML.
- **Compatibilitat**: Compatible amb la majoria dels llenguatges de programació.
- **Flexibilitat**: Pot representar estructures de dades complexes d'una forma més senzilla.

**Perquè és important?**

- **Interoperabilitat**: JSON és un format que es pot enviar i rebre fàcilment entre diferents sistemes, com un client JavaScript i un servidor Python o Node.js.
- **Serialització**: Convertir un objecte a JSON és una forma de serialització, on es converteixen les dades a un format que es pot emmagatzemar o transmetre.

Sintaxis, JSON soporta el següent tipus de dades:

- **Cadenes(String)**: "text"
- **Números**: 123, 45.67
- **Booleans**: true, false
- **Nuls**: null
- **Objectes**: {"clave": "valor"}
- **Arrays(taules)**: ["valor1", "valor2"]

# JSON - Manipulació JSON amb Javascript (I)

**1. Convertir un objecte JavaScript a JSON**

Quan tens un objecte en JavaScript que vols enviar a un servidor, emmagatzemar en un fitxer, o transferir a través d'una API, normalment necessites convertir aquest objecte en una cadena JSON. Això es fa amb la funció `JSON.stringify`.

```
const obj = { nom: "Joan", edat: 30 };
const jsonString = JSON.stringify(obj);
console.log(jsonString);
```

**Explicació**:

- `JSON.stringify(obj)`: Aquesta funció converteix un objecte JavaScript (`obj`) en una cadena JSON.
- El resultat és una cadena de text que representa l'objecte en format JSON, amb les claus i valors correctament escapats i formats.

**Perquè és important?**

- **Interoperabilitat**: JSON és un format que es pot enviar i rebre fàcilment entre diferents sistemes, com un client JavaScript i un servidor Python o Node.js.
- **Serialització**: Convertir un objecte a JSON és una forma de serialització, on es converteixen les dades a un format que es pot emmagatzemar o transmetre.

# JSON - Manipulació JSON amb Javascript (II)

**2. Convertir una cadena JSON a un objecte JavaScript**

Quan reps una cadena JSON d'un servidor, base de dades o fitxer, normalment vols convertir-la de nou en un objecte JavaScript per poder treballar amb ella dins del teu codi. Això es fa amb la funció `JSON.parse`.

```
const jsonString = '{"nom": "Joan", "edat": 30}';
const obj = JSON.parse(jsonString);
console.log(obj);
```

**Explicació**:

- `JSON.parse(jsonString)`: Aquesta funció pren una cadena en format JSON (`jsonString`) i la converteix en un objecte JavaScript.
- El resultat és un objecte JavaScript que pots manipular com qualsevol altre objecte dins del teu codi.

**Perquè és important?**

- **Deserialització**: Convertir una cadena JSON a un objecte JavaScript és una forma de deserialització, on es recupera l'estat de l'objecte des d'una representació textual.
- **Manipulació de dades**: Un cop tens l'objecte en format JavaScript, pots accedir i manipular les seves propietats directament, la qual cosa és essencial per a la majoria de les aplicacions web modernes.

# JSON - JSON SCHEMA

**JSON Schema** és una especificació que defineix una estructura per validar el contingut i l'estructura de documents JSON. JSON Schema permet assegurar que un document JSON segueixi un patró o estructura predefinida, la qual cosa és útil per garantir la integritat de les dades, especialment en sistemes grans on es comparteixen o intercanvien dades entre múltiples aplicacions o serveis.

Amb **JSON Schema**, es pot validar automàticament que les dades compleixin una estructura específica, ajudant a prevenir errors i assegurar la coherència. **JSON Schema** també serveix com a documentació formal sobre la forma que haurien de tenir les dades, la qual cosa facilita la col·laboració entre equips.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Persona",
  "type": "object",
  "properties": {
    "nom": {
      "type": "string"
    },
    "edat": {
      "type": "integer",
      "minimum": 0
    }
  },
  "required": ["nom", "edat"]
}
```

`$schema`: Aquesta clau especifica la versió de l'esquema que s'està utilitzant. En aquest cas, es fa referència al "draft-07" de JSON Schema.
`title`: Un títol descriptiu per l'esquema, que en aquest exemple és "Persona".
`type`: Indica el tipus de l'element que s'està definint. En aquest cas, es defineix un objecte.
`properties`: Defineix les propietats que pot tenir l'objecte. Cada propietat té un nom (per exemple, `nom` o `edat`) i un tipus (`string` o `integer`).
`required`: Llista de propietats que són obligatòries en l'objecte. En aquest exemple, tant `nom` com `edat` són necessàries.

# JSON - JSON SCHEMA - Execució

Per poder executar JSON SCHEMA hem d'instal·lar la llibreria **ajv** de **Node.js,** amb l'instrucció:

npm install ajv

hem de tenir el nostre schema.json, com per exemple:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Usuari",
  "type": "object",
  "properties": {
        "nom": { "type": "string" },
        "edat": { "type": "integer" },
        "email": { "type": "string" }
  },
  "required": ["nom", "edat"]
}
```

també hem de tenir una data.json, com per exemple:

```
{
  "nom": "Joan",
  "edat": 25
  "email":"hola@hola.com"
}
```

I ho podem comprovar amb un codi com el següent en el nostre javascript:

```
const Ajv = require('ajv');
const ajv = new Ajv();
const schema = require('./schema.json');
const data = require('./data.json');

const validate = ajv.compile(schema);
const valid = validate(data);

if (valid) {
        console.log('El JSON és vàlid!');
} else {
        console.error('Errors de validació:', validate.errors);
}
```

# JSON - JSONPATH

**JSONPath** és un llenguatge de consulta dissenyat per extreure dades de documents JSON. Funciona de manera similar a com XPath s'utilitza per a XML, permetent accedir fàcilment a parts específiques de dades en estructures JSON complexes. JSONPath permet accedir ràpidament a dades específiques dins de documents JSON complexos, sense la necessitat de processar manualment tot l'objecte.

```
{
  "store": {
    "book": [
      { "category": "fiction", "title": "The Great Gatsby", "price":10.99 },
      { "category": "fantasy", "title": "The Hobbit", "price": 5.99 }
    ]
  }
}
```

Consulta JSONPath:

```
$.store.book[*].title
```

Resultat de la consulta:

```
["The Great Gatsby", "The Hobbit"]
```

`$`: Representa l'element arrel del document JSON.
`store.book[*].title`: Aquesta part de la consulta navega a través de l'estructura JSON per accedir a la propietat `title` de cada element dins de l'array `book` dins de `store`.
`[*]`: Selecciona tots els elements de l'array `book`.

# JSON i les APIs Web

**REST (Representational State Transfer)** és un estil arquitectònic per a la creació de serveis web que utilitzen els mètodes HTTP per a operar sobre recursos. En les APIs RESTful, **JSON** és el format principal utilitzat per intercanviar dades entre el client (normalment una aplicació web o mòbil) i el servidor. RESTful APIs són senzilles d'entendre i implementar, i JSON és fàcilment llegible per humans i màquines, cosa que el fa ideal per a l'intercanvi de dades en aquest context. L'ús de JSON permet que APIs RESTful siguin accessibles des de qualsevol llenguatge de programació que pugui manejar peticions HTTP.

**Principals mètodes HTTP en una API RESTful:**

- **GET**: Utilitzat per obtenir dades del servidor. Per exemple, per recuperar informació sobre un usuari o llista de productes.
- **POST**: Utilitzat per enviar dades al servidor. Normalment s'utilitza per crear nous recursos, com ara un nou usuari o un nou element en una base de dades.
- **PUT**: Utilitzat per actualitzar dades existents al servidor. Es fa servir quan es volen modificar els detalls d'un recurs existent.
- **DELETE**: Utilitzat per eliminar dades del servidor. S'utilitza per eliminar un recurs específic.

**Exemple de cicle de vida d'una API RESTful:**

1. Un client fa una petició **GET** per obtenir una llista de productes.
2. El servidor respon amb un objecte JSON que conté la llista de productes.
3. El client fa una petició **POST** per afegir un nou producte, enviant les dades en format JSON.
4. El servidor afegeix el producte a la base de dades i retorna un objecte JSON amb els detalls del nou producte.

# JSON i les APIs Web - FetchAPI

En les aplicacions web modernes, el consum de dades d'una API RESTful es fa sovint utilitzant la **Fetch API**, una interfície nativa de JavaScript per fer peticions HTTP. Fetch API treballa de **manera asincrònica**, permetent que les aplicacions no es bloquegin mentre esperen una resposta del servidor. És una manera senzilla i neta de treballar amb dades JSON en aplicacions web modernes.

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .cath(error => console.log('Error:', error));
```

`fetch('https://api.example.com/data')`: Realitza una petició GET a l'URL especificat.
`response.json()`: Converteix la resposta de la petició a un objecte JSON.
`then(data => console.log(data))`: Manipula les dades JSON, en aquest cas, les mostra a la consola.
`catch(error => console.error('Error:', error))`: Captura i mostra qualsevol error que es produeixi durant la petició o la conversió.

# JSON i les APIs Web - Node.js i Express

Quan creem APIs, és habitual que aquestes retornin dades en format JSON. **Node.js** i **Express** són eines molt populars per crear APIs en l'entorn de desenvolupament JavaScript.

Amb **Node.js i Express**, és fàcil crear **APIs** que poden enviar i rebre JSON, cosa que permet una ràpida desenvolupament d'aplicacions. Són **escalables**, es pot utilitzar per crear APIs robustes i altament disponibles.

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

app.get('/data', (req, res) => {
  res.json({ nom: "Joan", edat: 30 });
});

app.listen(3000, () => {
  console.log('Servidor executant-se en el port 3000');
});
```

`app.get('/data', (req, res) => { ... })`: Defineix una ruta HTTP GET que respondrà amb dades JSON.
`res.json({ nombre: "Juan", edad: 30 })`: Utilitza el mètode `json` d'Express per enviar una resposta en format JSON.
`app.listen(3000, () => { ... })`: Inicia el servidor a l'escolta del port 3000.

# JSON i les APIs Web - JWT

Quan les dades sensibles s'envien a través d'una API, és crucial implementar mecanismes d'autenticació i autorització. Un mètode comú per fer-ho és utilitzant **JSON Web Tokens (JWT)**.

Els tokens JWT permeten que les APIs autentiquen usuaris de manera segura, ja que cada petició del client ha d'incloure un token vàlid.

JWT no requereix emmagatzemar sessions al servidor, ja que la informació necessària per a l'autenticació està continguda dins del propi token, cosa que el fa ideal per a aplicacions distribuïdes.

```
const jwt = require('jsonwebtoken');
const token = jwt.sign({ userId: 1 }, 'secretKey', { expiresIn: '1h'});
console.log(token);
```

`jwt.sign({ userId: 1 }, 'secretKey', { expiresIn: '1h' })`: Genera un token JWT. Aquest token conté una càrrega útil amb el `userId` (identificador de l'usuari) i està signat amb una `secretKey`. El token expira en una hora (`expiresIn: '1h'`).

`token`: El token generat és una cadena de text que es pot enviar al client per a la seva utilització en futures peticions.

# JSON Exemples d'ús

1. Configuració d'aplicacions mitjançant JSON

```json
{
  "app": {
    "name": "App",
    "version": "1.0.0",
    "port": 3000
  }
}
```

2. JSON per bases de dades (NO SQL): Bases de dades com MongoDB emmagatzemen dades en format JSON (BSON).

```json
{
  "_id": "507f191e810c19729de860ea",
  "nom": "Joan",
  "edad": 30,
  "hobbies": ["llegir", "viatjar", "córrer"]
}
```

3. Integració de JSON en eines de desenvolupament: Eines com ESLint i Prettier poden utilitzar-se per validar i formatejar JSON.