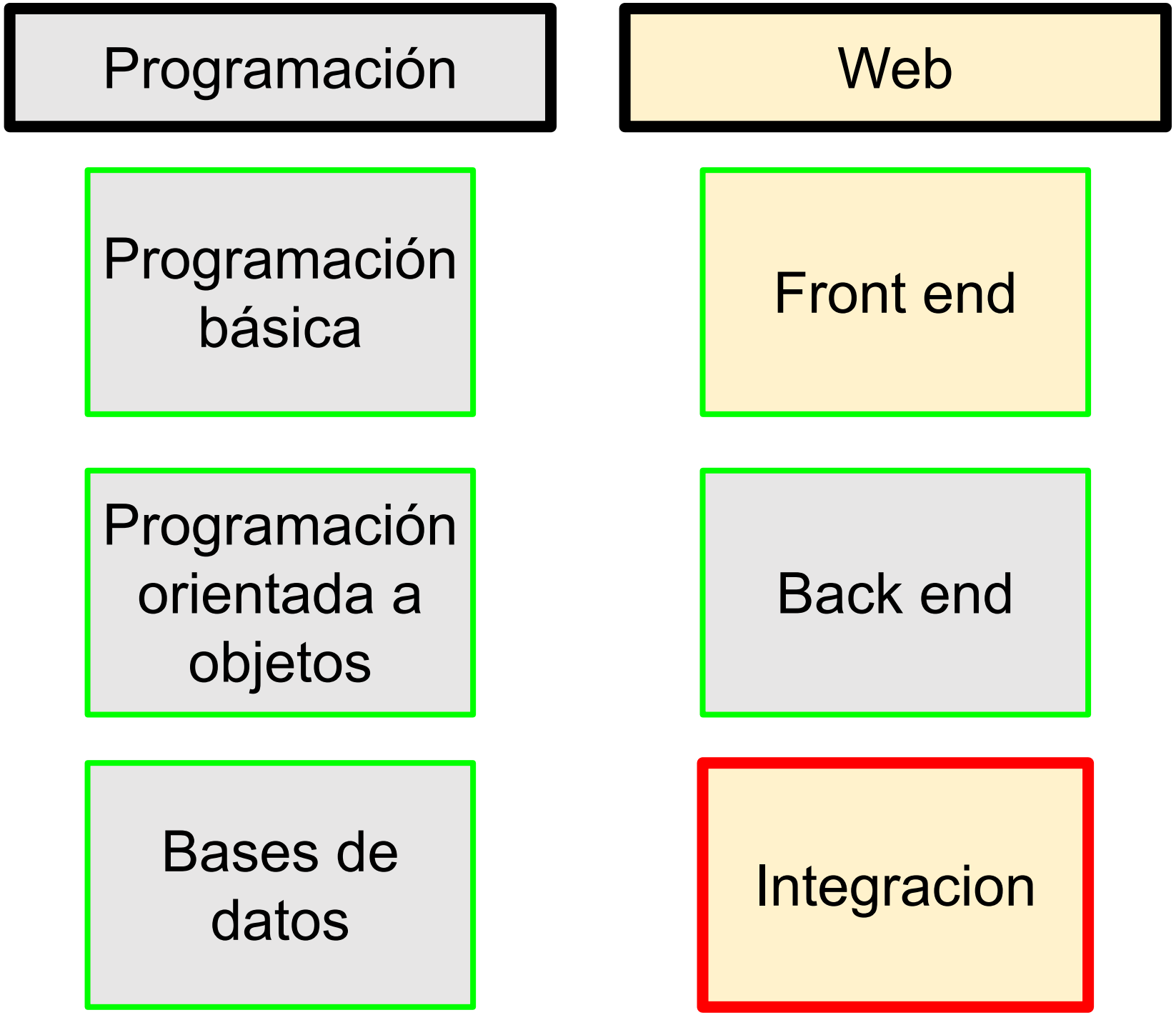
Base de Datos

CFP Programador full-stack

Modelos de Persistencia


Programación
Web
Programación básica
Front end
Programación orientada a objetos
Back end
Bases de datos
Integracion

Modelos de Persistencia

¿Qué pasa con los datos de nuestra aplicación?

Las aplicaciones trabajan con lo que se encuentra cargado en memoria RAM porque:

- La RAM es rápida
- El acceso a RAM es transparente

Pero:

- La RAM es limitada
- La RAM es volátil (una vez apagada la PC, se pierde el contenido que trabajamos en la RAM)

Modelos de Persistencia

¿Cómo guardar y recuperar el estado de la RAM?

1. Utilizar medio de almacenamiento secundario
   - **Archivos planos**
   - **Bases de datos**
2. En Java se traduce a:
   - Manejo de archivos manualmente, archivos mapeados a memoria o **serialización**
   - **Conexión directa a la base de datos**, mapeadores objeto-relacionales
3. Existen otras opciones basadas en bases de datos No-SQL

Mapeo Objeto Relacional (ORM)

- Los ORM tiene como objetivo abstraer al desarrollo de las cuestiones particulares de las bases de datos
- Transforman de forma automática los objetos a tuplas que se persisten en la base de datos cuando llega el momento de almacenarlos
- Transforman de forma automática las tuplas persistidas a objetos. De esta manera se puede operar con los datos
- Evitan el uso de SQL directo. Esto es una ventaja porque las diferentes bases de datos tiene diferencias en cómo está implementado SQL

Paradigmas de Objetos y Relacional

	Programación orientada a objetos	Base de datos relacionales
Elementos	Clases Objetos Atributos Métodos	Tablas Tuplas Atributos Stored procedures, Triggers
Relaciones	Referencias en memoria	Foreign Keys
Identificación	Identificadores de objetos	Primary Keys
Otros aspectos	Encapsulamiento, Herencia, Polimorfismo	Restricciones de integridad, Transacciones
Diseño	Diagrama de clases, secuencia	Diagrama de entidades y relaciones, Esquema relacional
Lenguajes	Java, C++, Python, C# ...	SQL

Curso CFP

CFP Programador full-stack

Conexión API-DB

Agenda

- Conexión con DB
- Uso de Queries Raw
  - Inconvenientes
- Asociación Clase-Tabla
- Operaciones sobre la DB
- Ejercicios

Instalando TypeORM

- La interacción entre nuestra API y MySQL la va a gestionar TypeORM

```
npm i --save @nestjs/typeorm typeorm mysql
```

Referencia
https://docs.nestjs.com/techniques/database

Conexión con MySQL (1)

- Para conectarnos a MySQL necesitamos ciertos datos
  - Host
  - Puerto
  - Usuario
  - Contraseña
- Siempre que queramos conectarnos a una base de datos, vamos a necesitar este tipo de información
- En NestJS tenemos dos formas de dar esta data
  - En app.module.ts
  - Haciendo manejo de conexiones
  - En un archivo externo → ormconfig.json 👍

Conexión con MySQL (2)

- Una forma de acceder a MySQL es mediante manejo de conexiones
  - No suele ser la opción más empleada
  - Normalmente se usan archivos externos para manejo de configuración

```
createConnection({
 type: "mysql",
 host: "localhost",
 port: 3306,
 username: "root",
 password: "root",
 database: "db_test"
}).then(connection => {
      ...
}).catch(error => {
      ...
});
```

Conexión con MySQL (3)

```
{
 "type": "mysql",
 "host": "localhost",
 "port": 3306,
 "username": "root",
 "password": "root",
 "database": "db_test",
 "entities": [
    "src/photo/entities/photo.entity.ts",
    "src/photo/entities/photoalbum.entity.ts"
 ],
 "synchronize": true
}
```

Revisar que está creado CREATE DATABASE IF NOT EXISTS db_test;

ormconfig.json

Conexión con MySQL (4)

- Primero tenemos que crear el ormconfig.json
- Después, en el archivo app.module.ts
  - Importar TypeOrmModule
  - Agregar el módulo a los imports

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { PhotoModule } from './photo/photo.module';
...

@Module({
 imports: [
   TypeOrmModule.forRoot(),
   PhotoModule,
 ],
 ...
})
export class AppModule {
}
```

Asociación Clase-Tabla (1)

- El siguiente paso es asociar una clase con una tabla
- Cada clase puede verse como una tabla
- Cada variable de una clase, se puede asociar a una columna de la tabla
- Trabajamos con anotaciones
- Cada clase asociada a una tabla, la vamos a llamar *entity*
  - Tener en cuenta que hay que agregar la ruta al archivo con la entity en ormconfig.json

```
...
"database": "db_test",
"entities": [
   "src/photo/entities/photo.entity.ts"
],
...
```

Asociación Clase-Tabla (2)

```
import { Column, Entity, PrimaryGeneratedColumn } from 'typeorm';

@Entity('photo')
export class Photo {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({
      length: 255
    })
    name: string;

    @Column()
    description: string;

    @Column()
    views: number;

    @Column({
      default: false
    })
    published: boolean;
}
```

Nombre que va a tener la tabla que se cree al levantar la API

Definimos PK

Diferentes formas de definir columnas

TypeORM - Repository

- La interacción con MySQL la hacemos a través de una variable de tipo Repository
  - La inyectamos en el constructor de nuestro servicio
- A través de esa variable hacemos todas las operaciones

```
constructor(
   @InjectRepository(Photo)
    private readonly photoRepository: Repository<Photo>
) {}
```

Tener en cuenta que necesitamos pasar el tipo de la entity que armamos → Photo

TypeORM - Consultas Raw

```
const result = await this.photoRepository.query('select * from photo;');

const photo = new Photo();
photo.id = result[0]['id'];
photo.name = result[0]['name'];
photo.description = result[0]['description'];
photo.filename = result[0]['filename'];
photo.views = result[0]['views'];
photo.published = result[0]['published'];
photo.albumId = result[0]['albumId'];
```

Acá van las queries que venían haciendo directo sobre MySQL

Obtenemos la primer fila
Accedemos a cada columna

Consultas Raw - Inconvenientes

- Si bien es un recurso válido escribir las queries directo, tiene sus inconvenientes
- El ejemplo anterior parece fácil porque es una sola tabla
  - Qué pasaría si queremos hacer una query que haga un JOIN entre varias tablas?
  - Leer a mano el resultado de ese tipo de queries no suele ser una tarea muy agradable que digamos
- Suponer el caso del INSERT o el UPDATE
  - Habría que intercalar todos los valores necesarios en el string que define a la query 😵

ORM: Object Relational Mapping

Los ORM hacen un mapeo entre

Modelo de Objetos <-> Modelo Relacional

La idea es que podamos pensar en términos de objetos y preocuparnos lo menos posible por cómo se almacenan en la tabla

TypeORM - Operaciones Básicas

- Las ventajas que proveen los ORMs es que podemos interactuar con la DB de esta manera

```
const myPhoto = await this.photoRepository.findOne(id);
```

Obtenemos una foto por ID

```
const photo = new Photo();
photo.name = dto.name;
photo.description = dto.description;
photo.filename = dto.filename;
photo.published = dto.published;
photo.views = dto.views;
photo.album = album;
```

Creamos una variable de tipo Photo

```
await this.photoRepository.save(photo);
```

La guardamos directo en la DB

Notar que en ningún momento hicimos una query → es la idea de los ORMs

TypeORM - Consideraciones

- En caso de que la API no levante por no encontrar el esquema, crearlo directo
  - create database if not exists db_test;
- Cuando levantan la API, se crean las tablas automáticamente, en función de las entities que tengamos
  - Tener cuidado de no modificar las entities estando las tablas ya creadas
- La creación de las tablas normalmente se hace mediante el mecanismo de *migraciones*
  - Es un tema más avanzado que queda por fuera del alcance de este curso

Demo en Vivo

- Levantar una API
- Mostrar las cosas de la conexión con MySQL
- Pegarle a un endpoint y mostrar los datos en el Workbench
- Ver que pasa cuando borramos la tabla, y levantamos la API de vuelta

Curso CFP

CFP Programador full-stack

Ejercicios

Ejercicios

- Levantar MySQL
- Crear un endpoint de prueba con una entity básica
- Implementar un POST que agregue una row de la entity creada
- Agregar un GET que obtenga todos los registros de la tabla