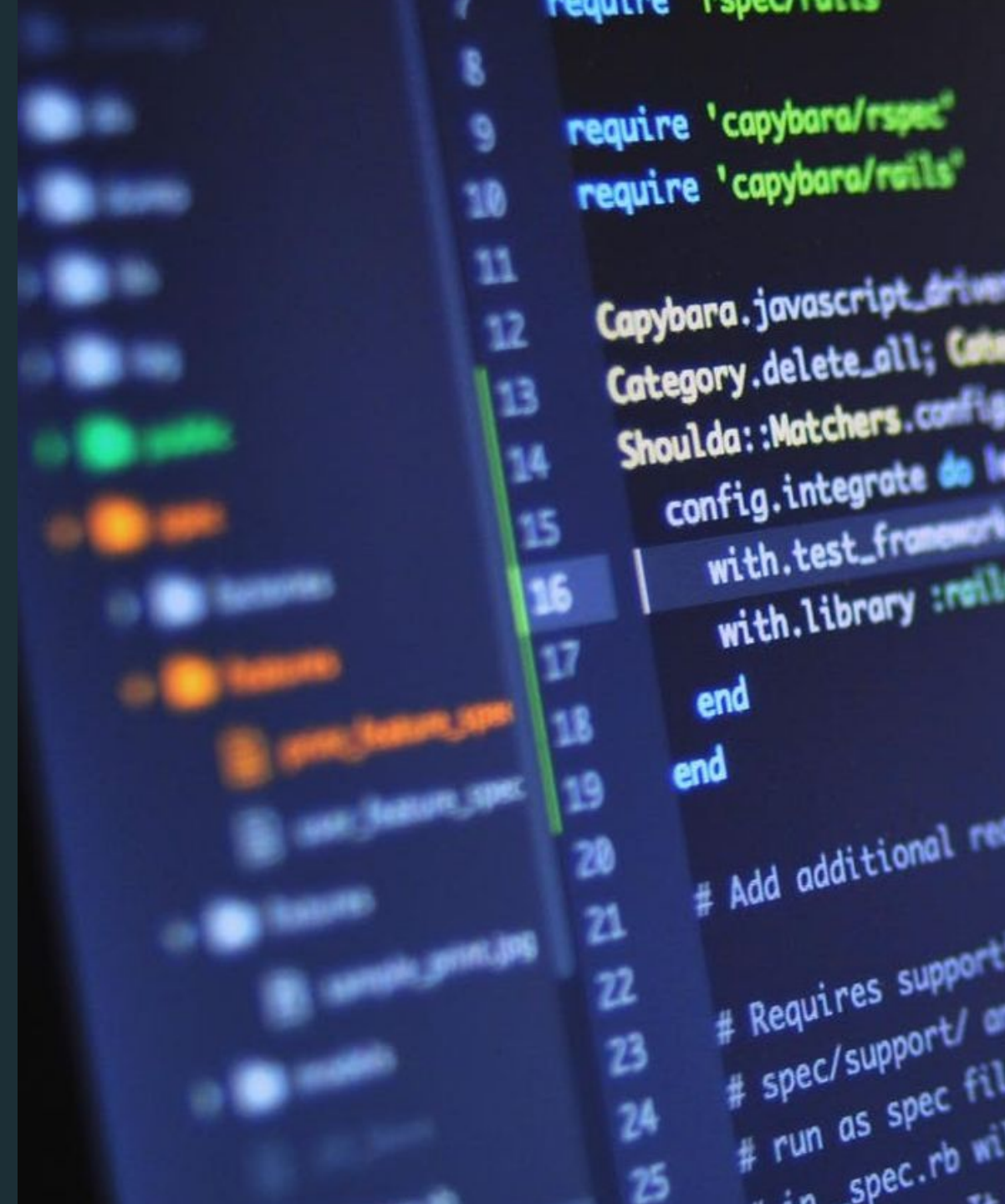


Arquitectura Hexagonal con Java/Quarkus

25-26-27 Y 28 DE JUNIO
ESPAÑA - 2024





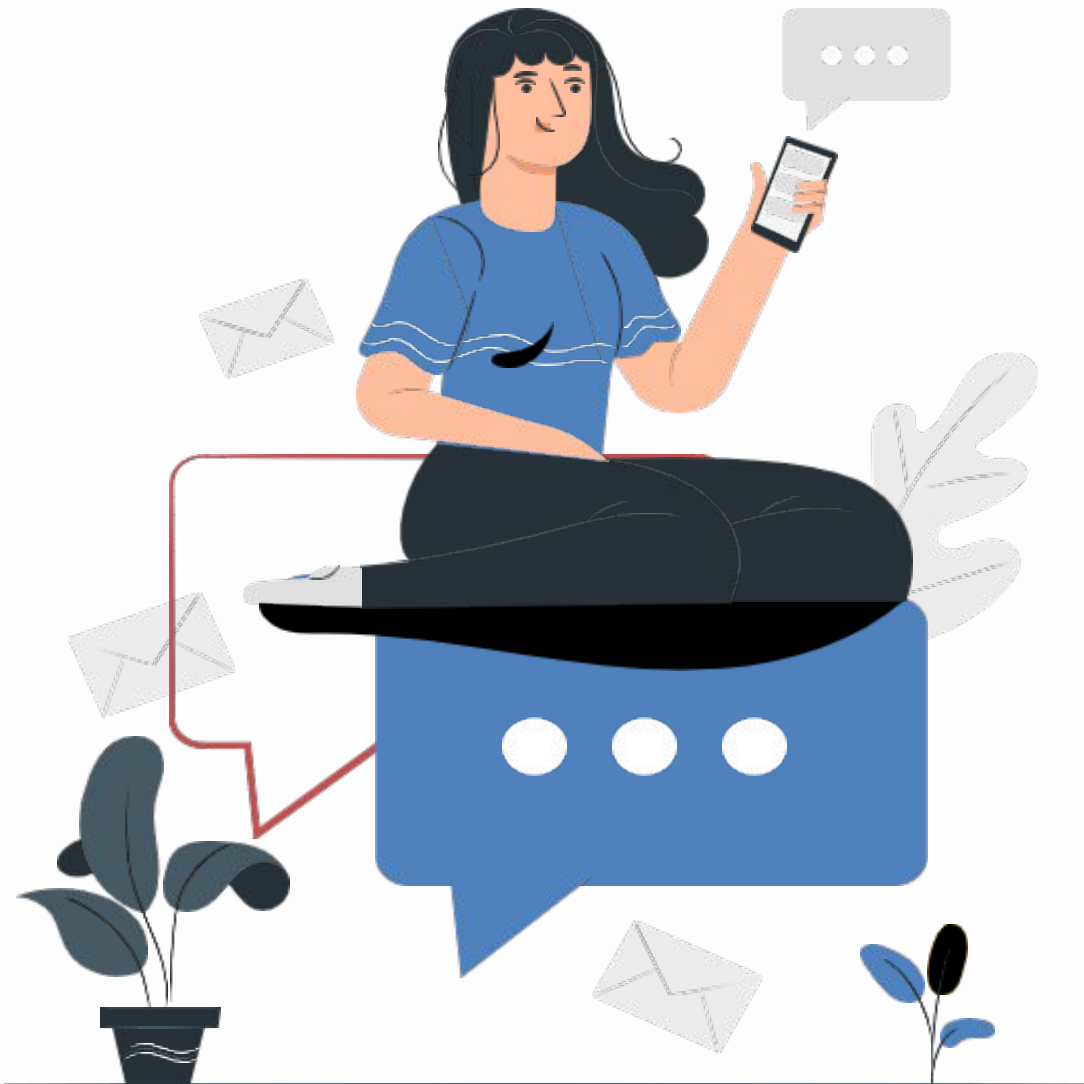
O. Mauricio Sánchez

Technical Team Leader End to End en
#INTERbanking #Fintech...



Mauricio Sánchez

Technical Team Leader End to End en #INTERbanking
#Fintech // Sr.Developer ☕ // #Speaker // // #Ambassador IB
// Co-Owner #JavaBsAs #JUG #Community // 🧑🏫 Docente



01

Presentación DE ALUMNOS



Maurisandev

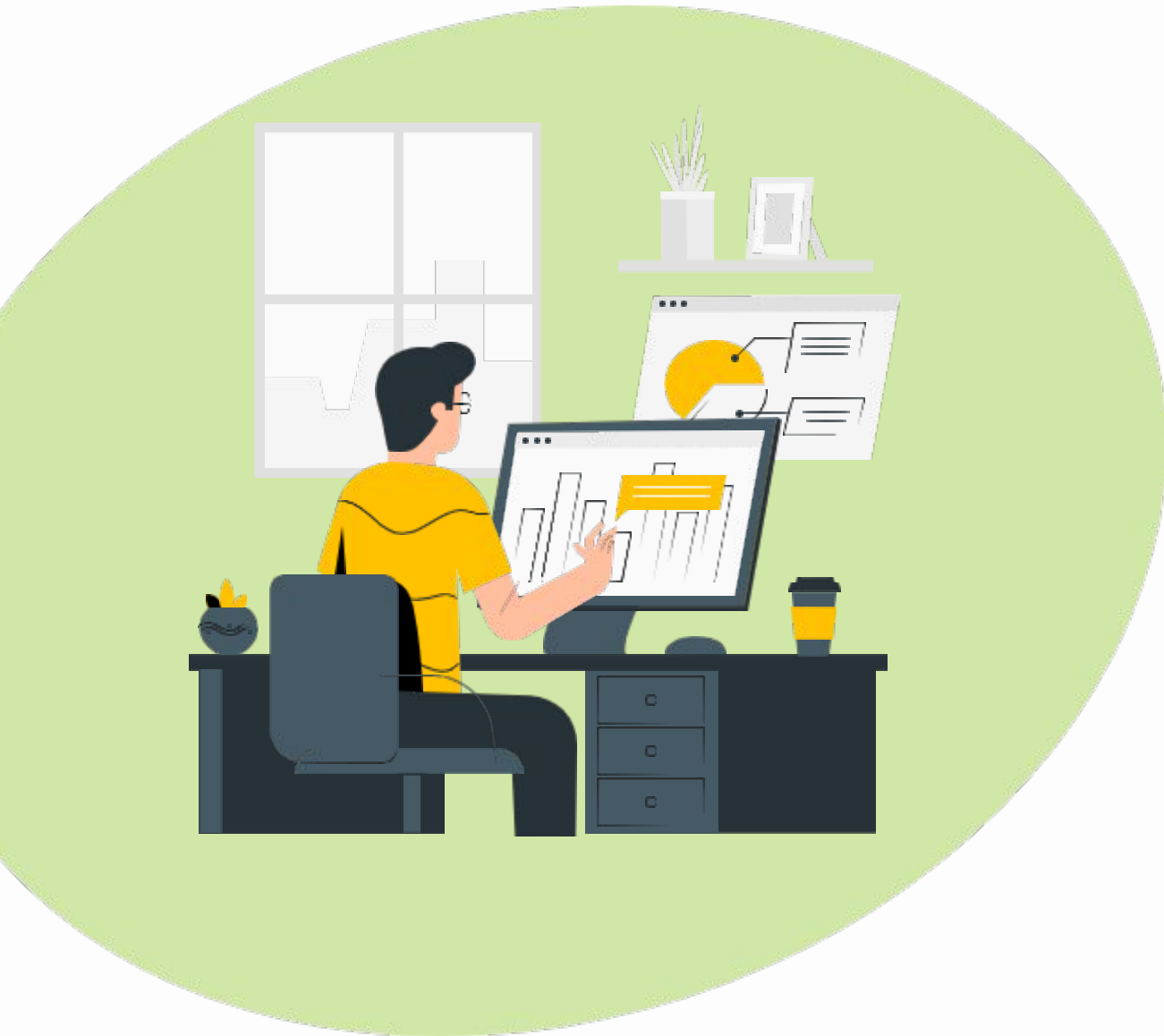


@MauriDeveloper



maurisan4011@gmail.com





02

OBJETIVOS DEL CURSO



Maurisandev



@MauriDeveloper



maurisan4011@gmail.com





03

DISTRIBUCIÓN DE CLASES



Maurisandev



@MauriDeveloper



maurisan4011@gmail.com



Clase 01

Introducción a la Arquitectura Hexagonal



- Introducción a la Arquitectura Hexagonal

Conceptos Básicos , Estructura General

- Ejercicio Práctico con Java

Setup del Entorno , Creación del Proyecto Base



Maurisandev



@MauriDeveloper



maurisan4011@gmail.com

Clase 02

Diseño del Núcleo

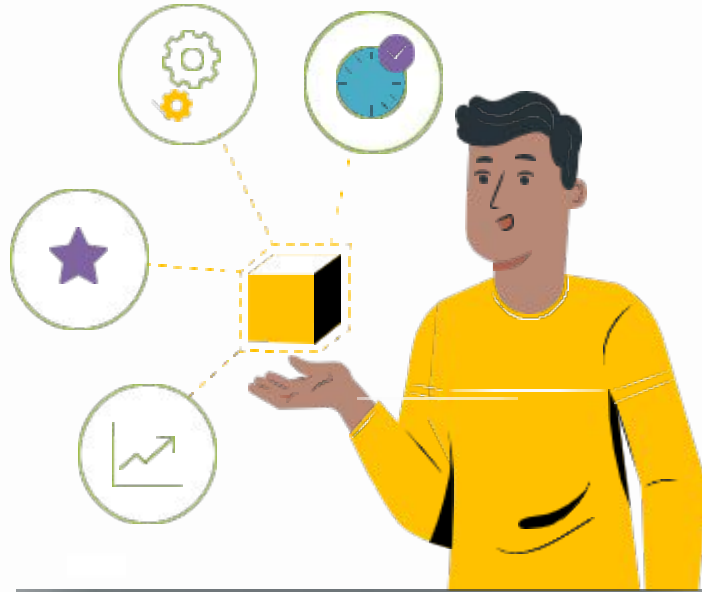


- **Diseño del Núcleo (Dominio)**
Modelado del Dominio, Reglas de Negocio
- **Ejercicio Práctico con Java/Quarkus**

Desarrollo del núcleo , Integracion con Quarkus

Clase03

Adaptadores



- Diseño e Implementacion de Adaptadores

Adaptadores Primarios y Secundarios, Patrones de Diseño

- Ejercicio Práctico con Java/Quarkus

Implementación de Adaptadores , Pruebas de adaptadores



Maurisande



@MauriDevelope



maurisan4011@gmail.com

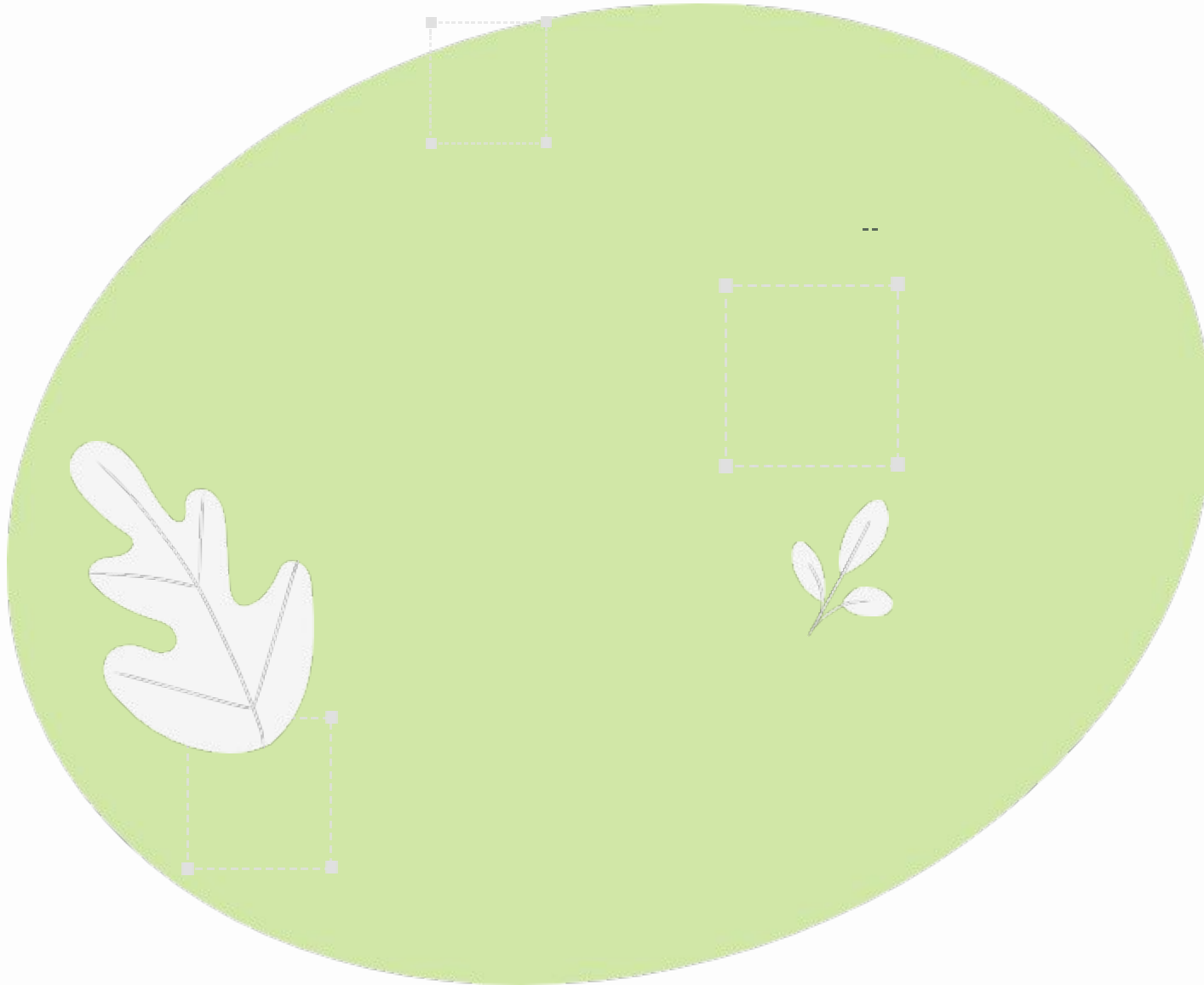
Clase 04

Interfaces de Usuario, Pruebas y Casos de Estudio



- **Interfaces de Usuario**
Diseño de Interfaces de Usuario, Frameworks y Herramientas.
- **Pruebas de Arquitectura Hexagonal**
Tipos de Pruebas , Estrategias de pruebas
- **Casos de Estudio y Ejemplos Prácticos**

Revision de casos de Estudio, Ejemplos prácticos



04

HERRAMIENTAS

QUE UTILIZAMOS



Maurisandev



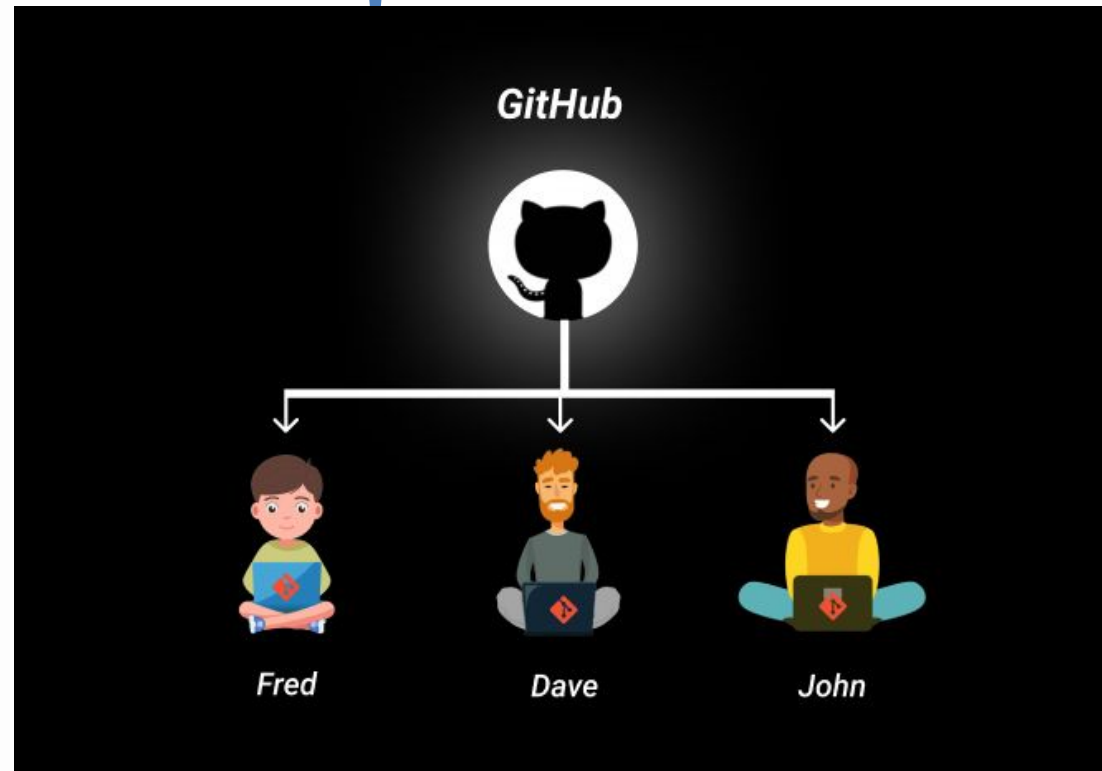
@MauriDeveloper



maurisan4011@gmail.com

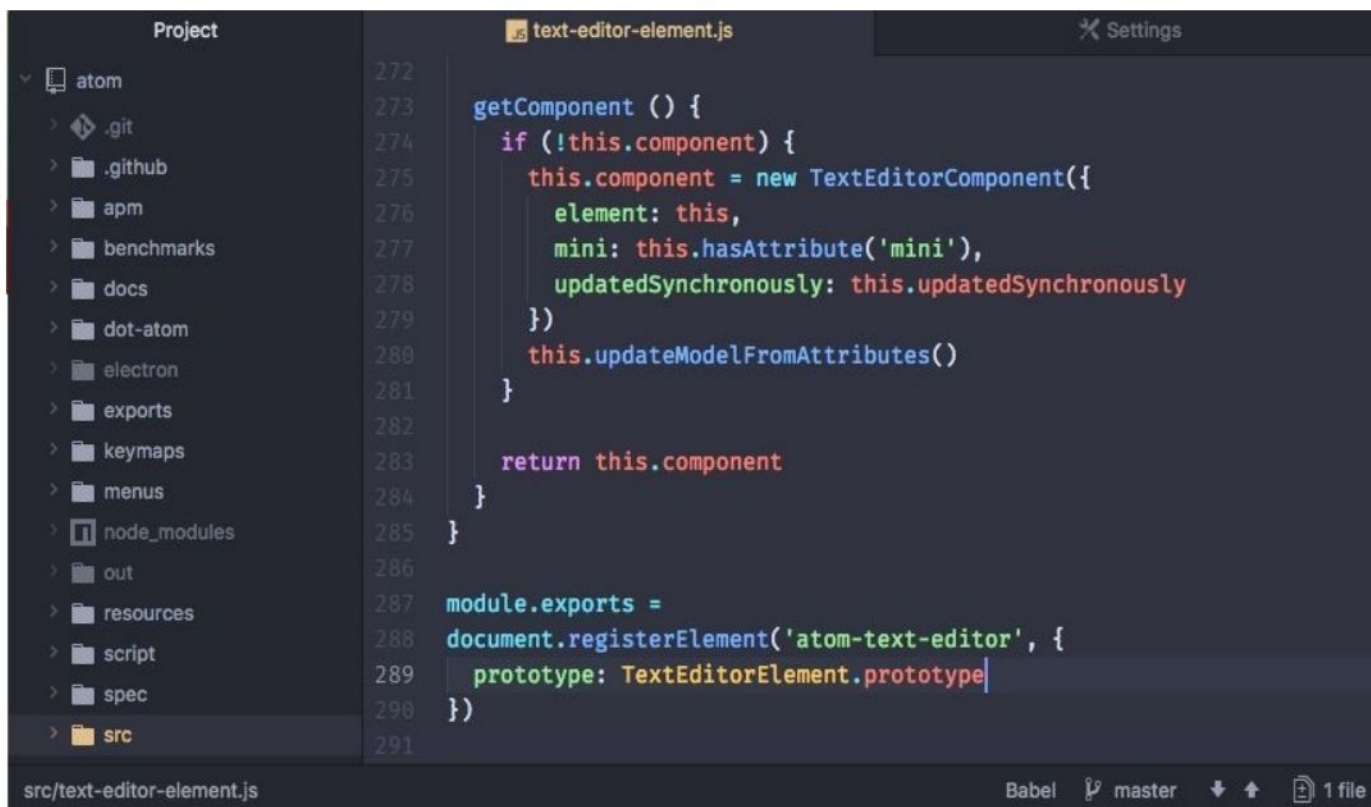


GIT



IDE

ATOM



The screenshot displays the Atom IDE interface. On the left is the 'Project' sidebar showing a file tree for the 'atom' project, including folders like '.git', '.github', 'apm', 'benchmarks', 'docs', 'dot-atom', 'electron', 'exports', 'keymaps', 'menus', 'node_modules', 'out', 'resources', 'script', 'spec', and 'src'. The main editor area shows the file 'text-editor-element.js' with the following JavaScript code:

```
272  
273  
274  getComponent () {  
275    if (!this.component) {  
276      this.component = new TextEditorComponent({  
277        element: this,  
278        mini: this.hasAttribute('mini'),  
279        updatedSynchronously: this.updatedSynchronously  
280      })  
281      this.updateModelFromAttributes()  
282    }  
283    return this.component  
284  }  
285 }  
286  
287 module.exports =  
288 document.registerElement('atom-text-editor', {  
289   prototype: TextEditorElement.prototype  
290 })  
291
```

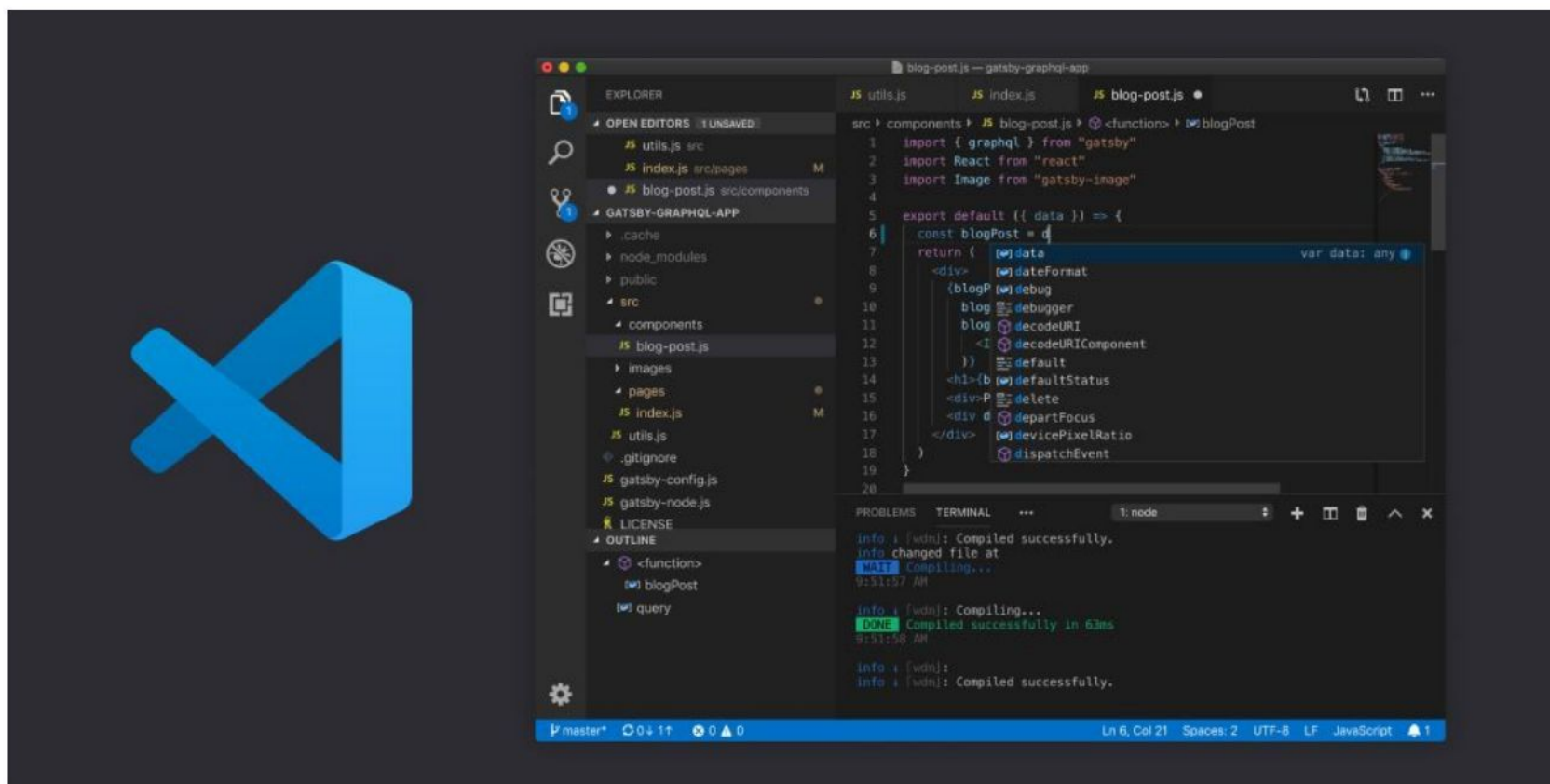
The status bar at the bottom indicates 'src/text-editor-element.js', 'Babel', 'master', and '1 file'.

<http://atom.i>

IDE



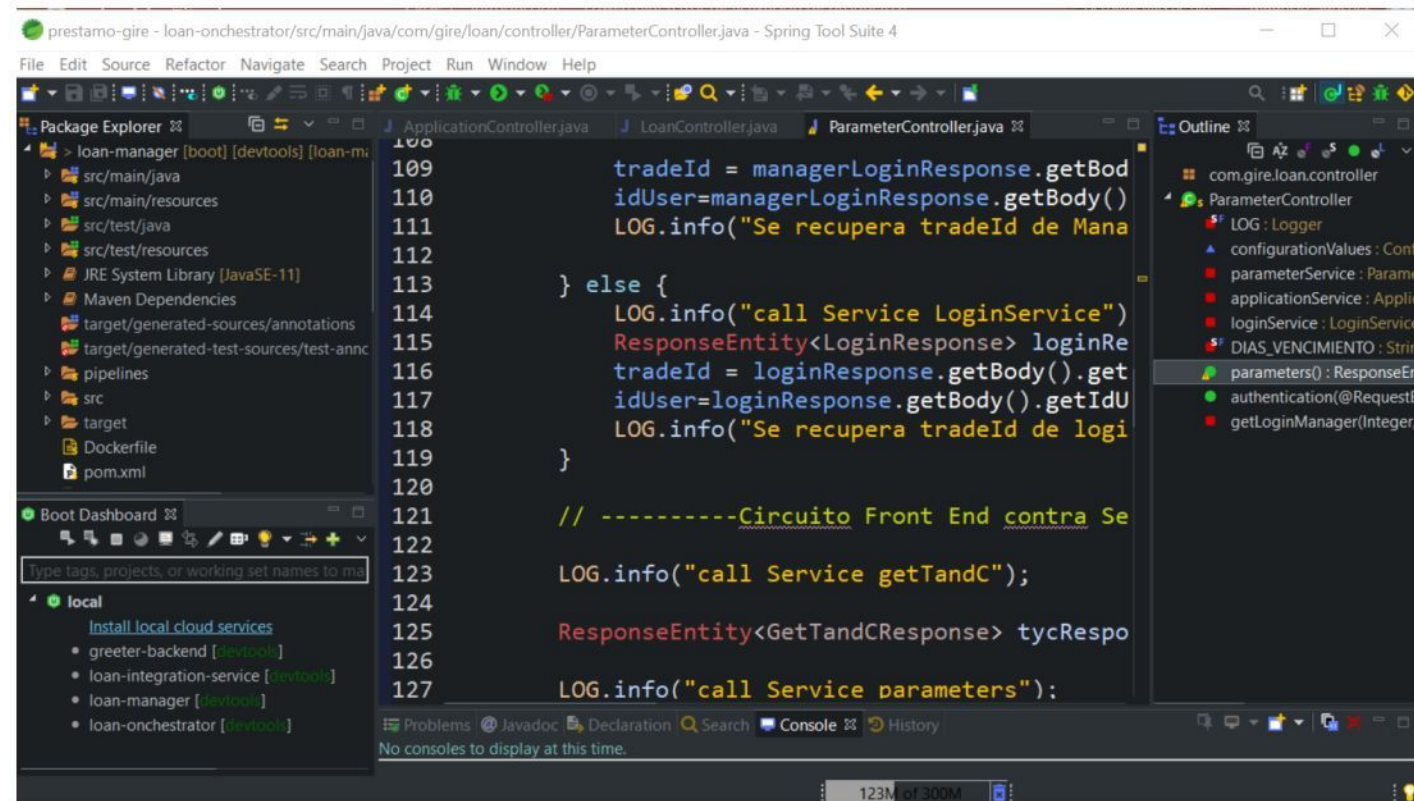
VS Code



<https://code.visualstudio.com/download>

Spring Tool Suite 4

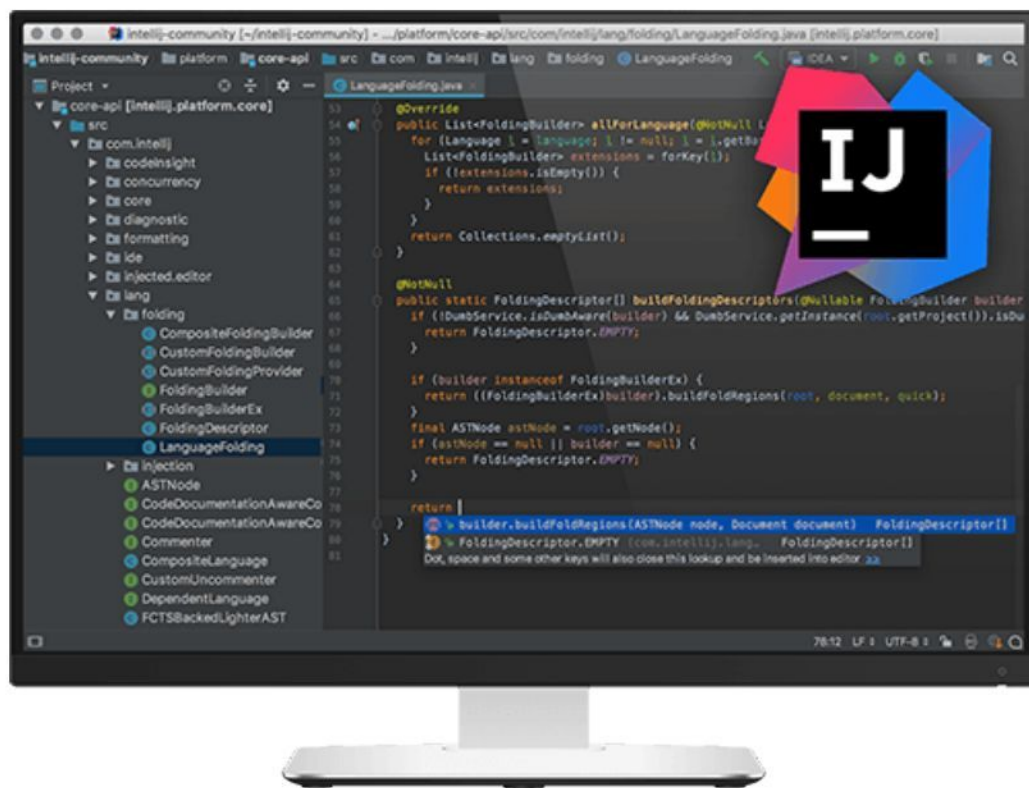
IDE



<https://spring.io/tools>

IntelliJ IDEA

IDE



<https://www.jetbrains.com/es-es/idea/>

Agenda

- **Introducción a la Arquitectura**

- **Hexagonal** Conceptos básicos

- Definición y propósito de la Arquitectura Hexagonal. Principios fundamentales y beneficios. Comparación con otras arquitecturas (Monolítica, Microservicios, etc.).

- **Estructura General:** Capas de la arquitectura: Núcleo (Dominio), Adaptadores, Interfaces. Comunicación entre capas y principios de inversión de dependencias.

- **Ejercicio Práctico con Java** Setup del

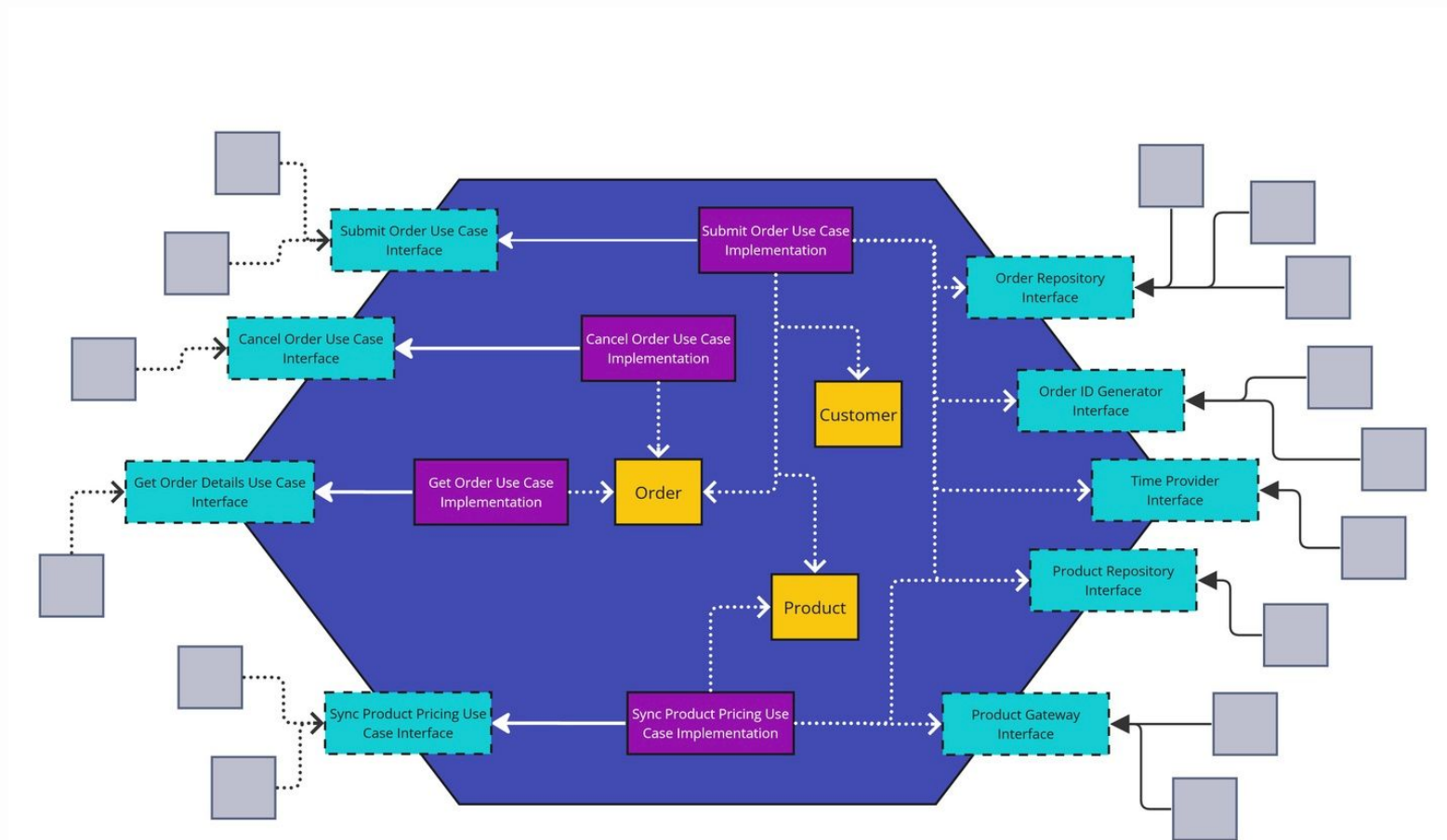
- **Entorno:** Instalación y configuración de Java y un IDE (Eclipse/IntelliJ). Introducción a Quarkus y configuración inicial del proyecto.

- **Creación del Proyecto Base:** Estructuración del proyecto siguiendo los principios de la Arquitectura Hexagonal. Implementación de un ejemplo sencillo que muestre la separación de capas.



Arquitectura Hexagonal: Separación de Preocupaciones y Flexibilidad

Introducción a los conceptos clave de la Arquitectura Hexagonal y cómo esta arquitectura promueve la separación de intereses y la flexibilidad en el desarrollo de aplicaciones



Introducción a la Arquitectura Hexagonal



Definición

La Arquitectura Hexagonal, también conocida como Puertos y Adaptadores, es un patrón de diseño arquitectónico que separa la lógica de negocio del código de infraestructura, permitiendo una mayor flexibilidad y mantenibilidad del sistema.



Propósito

El objetivo principal de la Arquitectura Hexagonal es aislar la lógica de negocio de las dependencias externas, lo que facilita la prueba, la implementación y la migración a diferentes entornos y tecnologías.



Ventajas

Algunos de los principales beneficios de la Arquitectura Hexagonal incluyen una mayor independencia de las tecnologías, una mejor organización del código y una mayor facilidad para probar y mantener el sistema.

La Arquitectura Hexagonal es un enfoque arquitectónico que permite separar la lógica de negocio de las dependencias externas, lo que se traduce en una mayor flexibilidad, testabilidad y mantenibilidad del sistema.

Principios Fundamentales

- Separación de Preocupaciones

La Arquitectura Hexagonal separa claramente las responsabilidades entre el núcleo de la aplicación (dominio) y los detalles de implementación (adaptadores). Esto mejora la mantenibilidad y la testabilidad del sistema.

- Portabilidad y Reusabilidad

Al separar claramente las responsabilidades, el núcleo de la aplicación se vuelve portable y puede ser reutilizado en diferentes contextos o proyectos, sin depender de detalles específicos de implementación.

- Flexibilidad e Independencia

El núcleo de la aplicación es independiente de las tecnologías y frameworks utilizados en los adaptadores. Esto permite cambiar o reemplazar fácilmente los detalles de implementación sin afectar el núcleo de la aplicación.

- Fácil Adaptabilidad

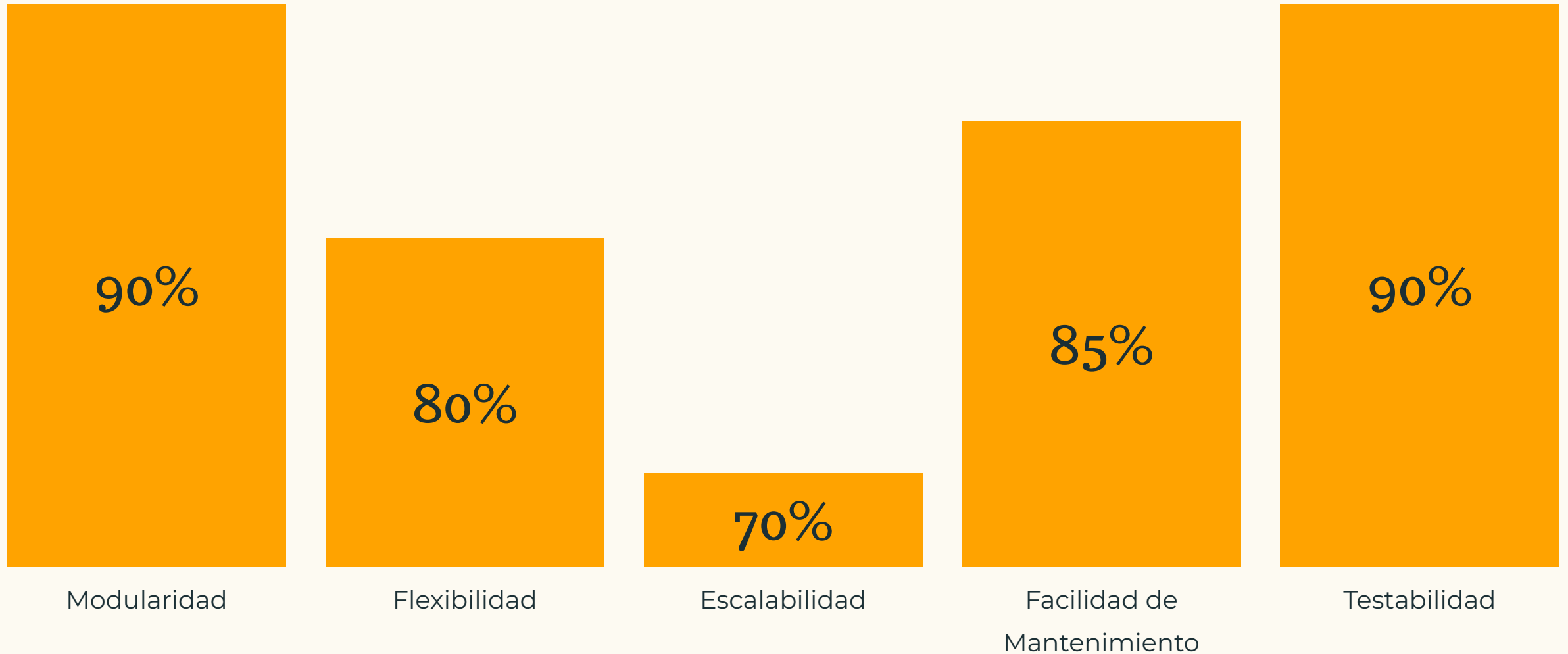
La Arquitectura Hexagonal permite adaptar fácilmente la aplicación a nuevos requisitos o cambios en el entorno, sin necesidad de modificar el núcleo de la aplicación.

- Inversión de Dependencias

Las capas de la arquitectura dependen de interfaces en lugar de implementaciones concretas. Esto facilita la sustitución de implementaciones y la inyección de dependencias, mejorando la modularidad y la capacidad de prueba.

Comparación con otras Arquitecturas

Escala de 0 a 100, donde 100 representa el mejor rendimiento



Comparación con otras Arquitecturas

Evaluación de la complejidad de la implementación en una escala de 0 a 100



Arquitectura Monolítica



Arquitectura de
Microservicios



Arquitectura de Capas
Tradicionales



Arquitectura Hexagonal

Estructura General

Núcleo (Dominio)

Esta capa contiene la lógica de negocio, reglas de dominio y casos de uso de la aplicación. Es independiente de cualquier tecnología o interfaz externa.

Adaptadores

Los adaptadores actúan como intermediarios entre el núcleo y el mundo exterior. Convierten las entradas y salidas de la aplicación a un formato que el núcleo pueda entender y procesar.

Interfaces

Las interfaces definen los contratos entre el núcleo y los adaptadores. Estas interfaces permiten desacoplar el núcleo de las implementaciones concretas de los adaptadores.

Comunicación entre Capas

La comunicación entre las capas se realiza a través de las interfaces siguiendo el principio de inversión de dependencias. Esto permite que el núcleo no dependa directamente de las implementaciones de los adaptadores.

Beneficios

La separación en capas y el uso de interfaces promueve la modularidad, flexibilidad y testabilidad de la aplicación. Permite hacer cambios en las implementaciones de los adaptadores sin afectar al núcleo.

Comunicación entre Capas



Principio de Inversión de Dependencias

Comunicación
Unidireccional entre Capas

Acoplamiento Débil entre Capas

Interfaces como Puntos de Integración

Entorno de Desarrollo

Instalación de Java

Descarga e instalación de la última versión de Java Development Kit (JDK) en el sistema operativo utilizado. Asegurarse de que la variable de entorno JAVA_HOME esté correctamente configurada.

Selección del IDE

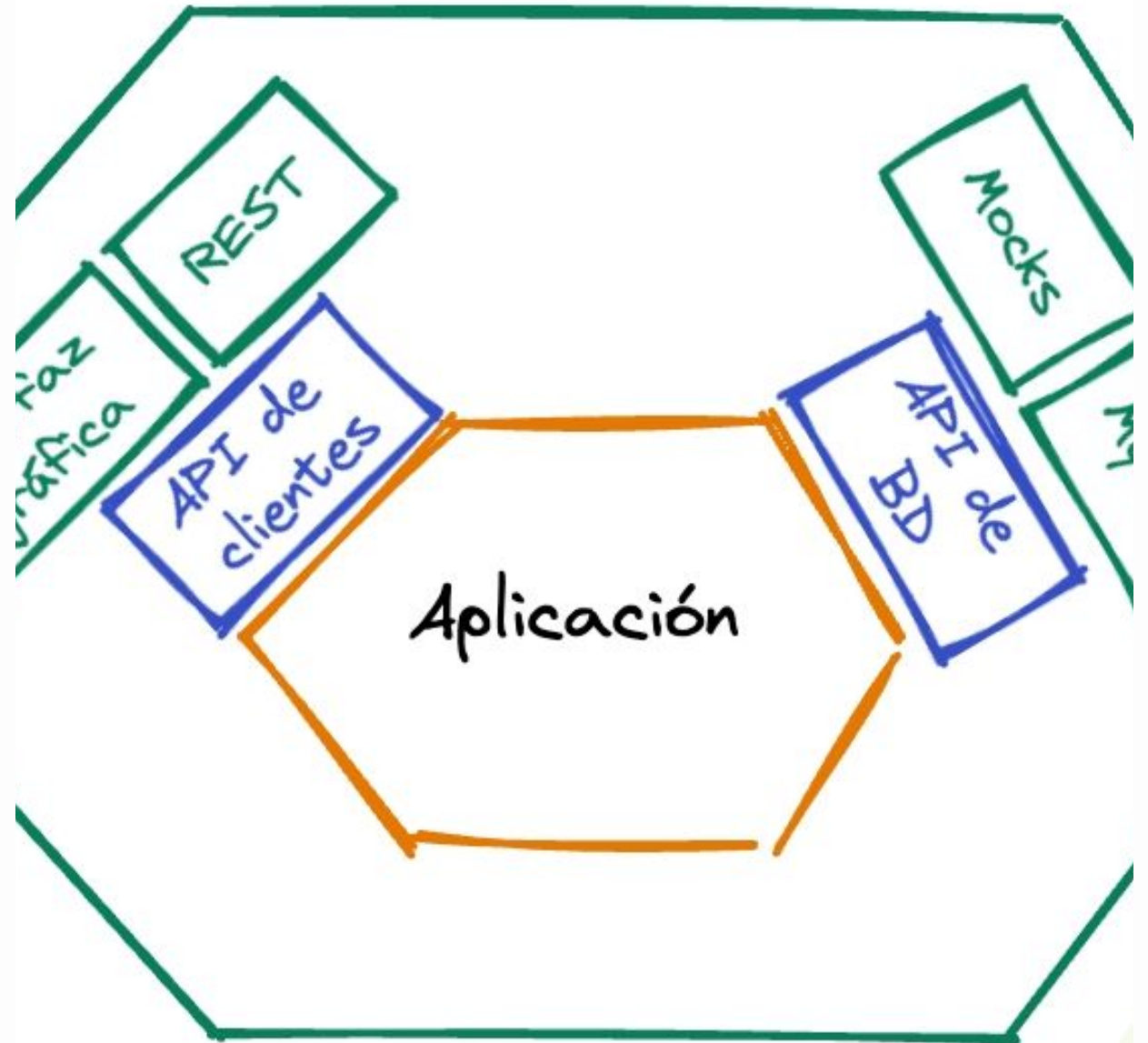
Elección de un Integrated Development Environment (IDE) como Eclipse o IntelliJ IDEA. Instalación y configuración del IDE seleccionado, incluyendo la integración con Java.

Instalación de Quarkus

Descarga e instalación de Quarkus, un framework de desarrollo de aplicaciones Java nativas de la nube, a través de herramientas como Maven o Gradle. Configuración inicial del proyecto Quarkus.

Creación del Proyecto Base

La estructuración del proyecto siguiendo los principios de la Arquitectura Hexagonal implica la separación del proyecto en tres capas principales: el Núcleo (Dominio), los Adaptadores y las Interfaces. Esta división permite mantener una arquitectura flexible y desacoplada, facilitando la modificación y sustitución de componentes sin afectar al resto del sistema.



Ejemplo Práctico

```
// Paso 1: Crear el proyecto base con Quarkus
// En la terminal, ejecutar:
mvn io.quarkus:quarkus-maven-plugin:create \
  -DgroupId=com.example \
  -DprojectArtifactId=hexagonal-architecture \
  -DclassName=com.example.hexagonal.GreetingResource" \
  -Dpath="/hello"
cd hexagonal-architecture

// Paso 2: Definir la estructura de paquetes
mkdir -p src/main/java/com/example/domain
mkdir -p src/main/java/com/example/application
mkdir -p src/main/java/com/example/infrastructure

// Paso 3: Crear una entidad en el dominio
package com.example.domain;

public class Product {
    private int id;
    private String name;
    private double price;
    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
    // Getters Setters
}

// Paso 4: Crear un servicio de dominio
package com.example.application;

import jakarta.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class ProductService {
    public String getProducts() {
        // Implementación de obtención de productos
        return "Listado de productos";
    }
}

// Paso 5: Crear un controlador REST para interactuar con el servicio
package com.example.infrastructure;

import com.example.application.ProductService;
import jakarta.inject.Inject;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

@Path("/products")
public class ProductResource {

    private final ProductService productService;

    @Inject
    public ProductResource(ProductService productService) {
        this.productService = productService;
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public String getProducts() {
        return productService.getProducts();
    }
}
```

```
// Paso 1: Crear el proyecto base con Quarkus
// En la terminal, ejecutar:
mvn io.quarkus:quarkus-maven-plugin:create \
    -DprojectGroupId=com.example \
    -DprojectArtifactId=hexagonal-architecture \
    -DclassName="com.example.hexagonal.GreetingResource" \
    -Dpath="/hello"
cd hexagonal-architecture

// Paso 2: Definir la estructura de paquetes
mkdir -p src/main/java/com/example/domain
mkdir -p src/main/java/com/example/application
mkdir -p src/main/java/com/example/infrastructure

// Paso 3: Crear una entidad en el dominio
package com.example.domain;

public class Product {
    private int id;
    private String name;
    private double price;
    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
    // Getters Setters
}

// Paso 4: Crear un servicio de dominio
package com.example.application;

import jakarta.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class ProductService {
    public String getProducts() {
        // Implementación de obtención de productos
        return "Listado de productos";
    }
}

// Paso 5: Crear un controlador REST para interactuar con el servicio
package com.example.infrastructure;

import com.example.application.ProductService;
import jakarta.inject.Inject;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

@Path("/products")
public class ProductResource {

    private final ProductService productService;

    @Inject
    public ProductResource(ProductService productService) {
        this.productService = productService;
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public String getProducts() {
        return productService.getProducts();
    }
}
```

“Muchas Gracias hasta
la siguiente clase”