Sobre MVC y otras cosas...

Tópicos Java...

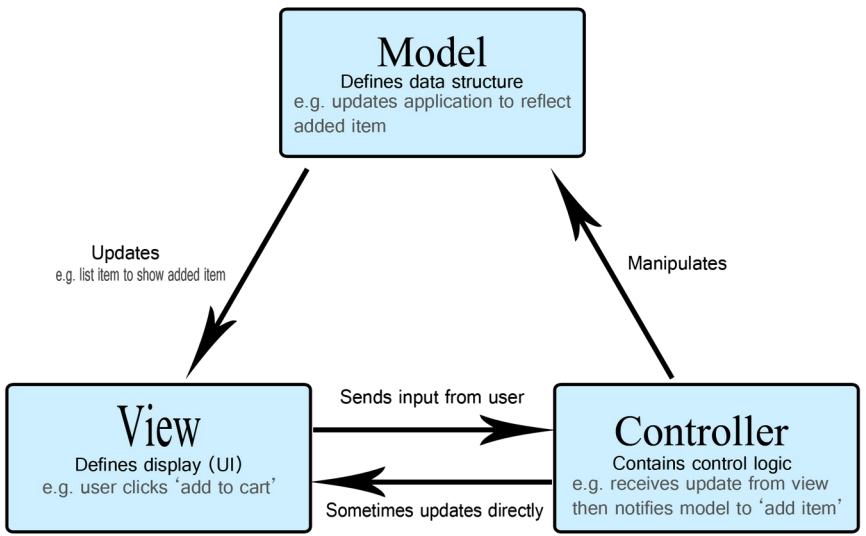
MVC es...

...un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica empresarial y la visualización del software. Esta "separación de preocupaciones (responsabilidades)" permite una mejor división del trabajo y un mejor mantenimiento. Algunos otros patrones de diseño se basan en MVC, como MVVM (Model-View-Viewmodel),
MVP (Model-View-Presenter), y MVW (Model-View-Whatever). [1]

MVC es...

Las tres partes del patrón de diseño de software MVC se pueden describir de la siguiente manera:

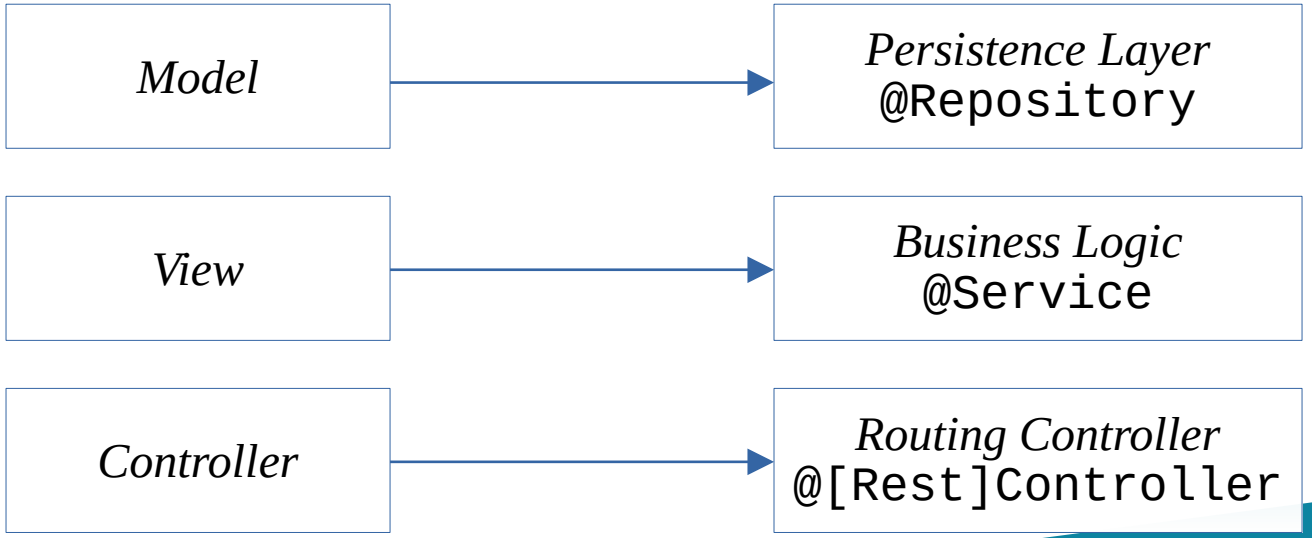
- **Modelo**: Define los datos de la aplicación, su modelado en entidades y su persistencia en un medio interno o externo (BBDD). (Capa de Persistencia)
- **Vista**: Se encarga del diseño y presentación (acondicionamiento) de los datos de la capa del modelo. (Capa de lógica de negocio)
- **Controlador**: Enruta comandos(peticiones) a los modelos y vistas. (Capa de Routing. EntryPoint/endpoint)



MVC vs. Spring...

¿Porqué MVC?

Por el Paralelismo con las capas "naturales" de Spring...



Modelo: @Repository

Definición desde la especificacion EJB 3.X [9]

- Capa de Persistencia de Datos mediante la implementación (Spring Data) de la especificación JPA (Java Persistence API).
  - Gestión automática de los recursos de datos (DataSources).
  - Manejo simplificado de EntityManagerFactory & EntityManager.
  - Contexto de Persistencia, puede crear y gestionar el contexto de persistencia manualmente o mediante inyección de forma autiomática.
  - Objetos de entidad, Clase Java simple que representa una fila de una tabla de base de datos en su forma más simple. (ORM)
  - JPQL (JPA Query Language), abstraciones para manipular consultas de datos mediante notacion de objetos java.
  - @Procedure, @NamedProcedures
-

**** Revisar Compatibilidades entre JPA 2.2, 3.0, 3.2***

Modelo: @Repository

- @Procedure (spring-data), Soporte a Procedimientos Almacenados. [10]
- @Repository, Abstracción (Interface) a un Origen de Datos.
- Operaciones comunes: QueryMethods, Mappings
- Alternativas: IBM WebSphere Liberty, Hibernate, Apache OpenJPA, Eclipse Link (Jakarta)*

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {


    @Query(nativeQuery = true, value = "call get_employees")
    List<Employee> getAllEmployees();

    @Query(nativeQuery = true, value = "call get_employee_id(:emp_id)")
    Employee getEmployeeById(@Param("emp_id") Long emp_id);

    @Transactional
    @Modifying
    @Query(nativeQuery = true, value = "call delete_emp_id(:emp_id)")
    void deleteEmployeeById(@Param("emp_id") Long emp_id);
```

**** Revisar Compatibilidades entre JPA 2.2, 3.0, 3.2***

Vista: @Service

- Cuando se anota una clase, nos permite, por convención, manejar, por inyección de dependencias, uno o mas repositorios para encapsular la lógica de negocio.
- *“Marcamos los beans con* `@Service` *para indicar que contienen la lógica empresarial. Además de usarse en la capa de servicio, esta anotación no tiene ningún otro uso especial.”* [5]

Controlador: @RestController

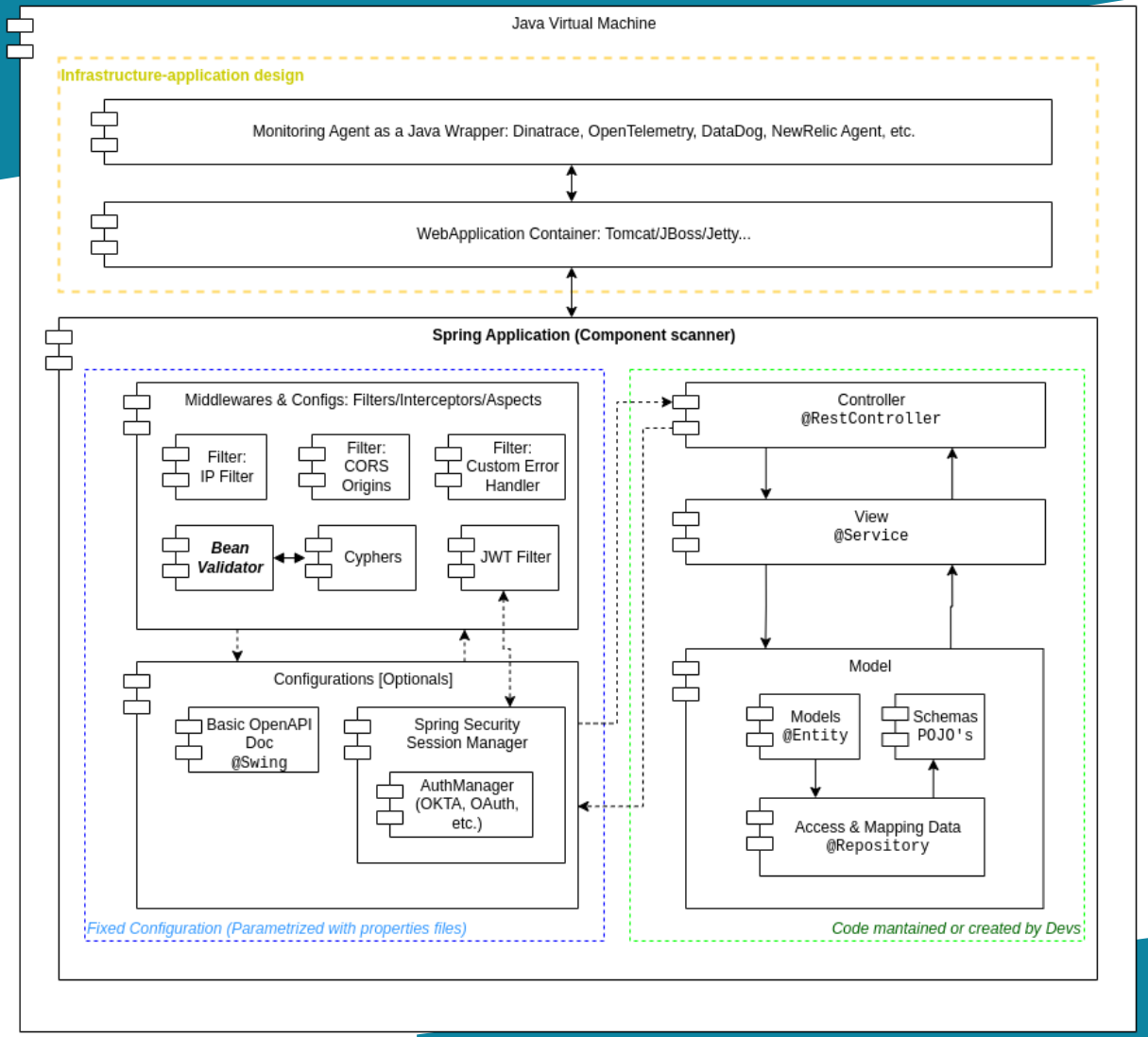
- Nos permite definir en una clase una ruta de exposición del servicio (endpoint) y encapsular y asociar sus métodos a sub-rutas o a métodos HTTP siguiendo la convencion RESTful.
- Anotaciones para enlazar automáticamente parámetros: @RequestBody, @RequestParam, @PathParam [12]
- Una diferencia clave entre un controlador MVC tradicional y el controlador de servicios web RESTful es la forma en que se crea el cuerpo de la respuesta HTTP. En lugar de depender de una tecnología de visualización para realizar la representación del lado del servidor de los datos en HTML, el controlador de servicio web REST completa y devuelve un objeto como JSON directamente en la respuesta HTTP. [11]

```java
@RestController
@RequestMapping("books-rest")
public class SimpleBookRestController {

    @GetMapping("/{id}", produces = "application/json")
    public Book getBook(@PathVariable int id) {
        return findBookById(id);
    }

    private Book findBookById(int id) {
        // ...
    }
}
```

Propuesta de Componentes:



Ventajas

- Uso de Abstraciones(interfaces) muy bien diseñadas y depuradas por la comunidad Spring.
- Basado en Principios y Convenciones comunes: Breans, POJOs, etc. *Convención antes de configuración*.
- Facilita la inyección y gestión de dependencias. (@Autowired, et al)
- Patrón de Diseño facil aprender y programar. Modulos y capas claros y bien definidos.
- Desarrollo y mantenimiento ágil.
- Genera aplicaciones livainas, ideal para RESTful y Microservicios. (facilidad de escalamiento)
- Es altamente robusto y confiable: @AsyncEnable, @Transactional, spring-batch, spring-security, ...
- Usar abstracciones (frameworks), aumenta la productividad y reduce errores en runtime. D.Y.R.
- ...

Desventajas

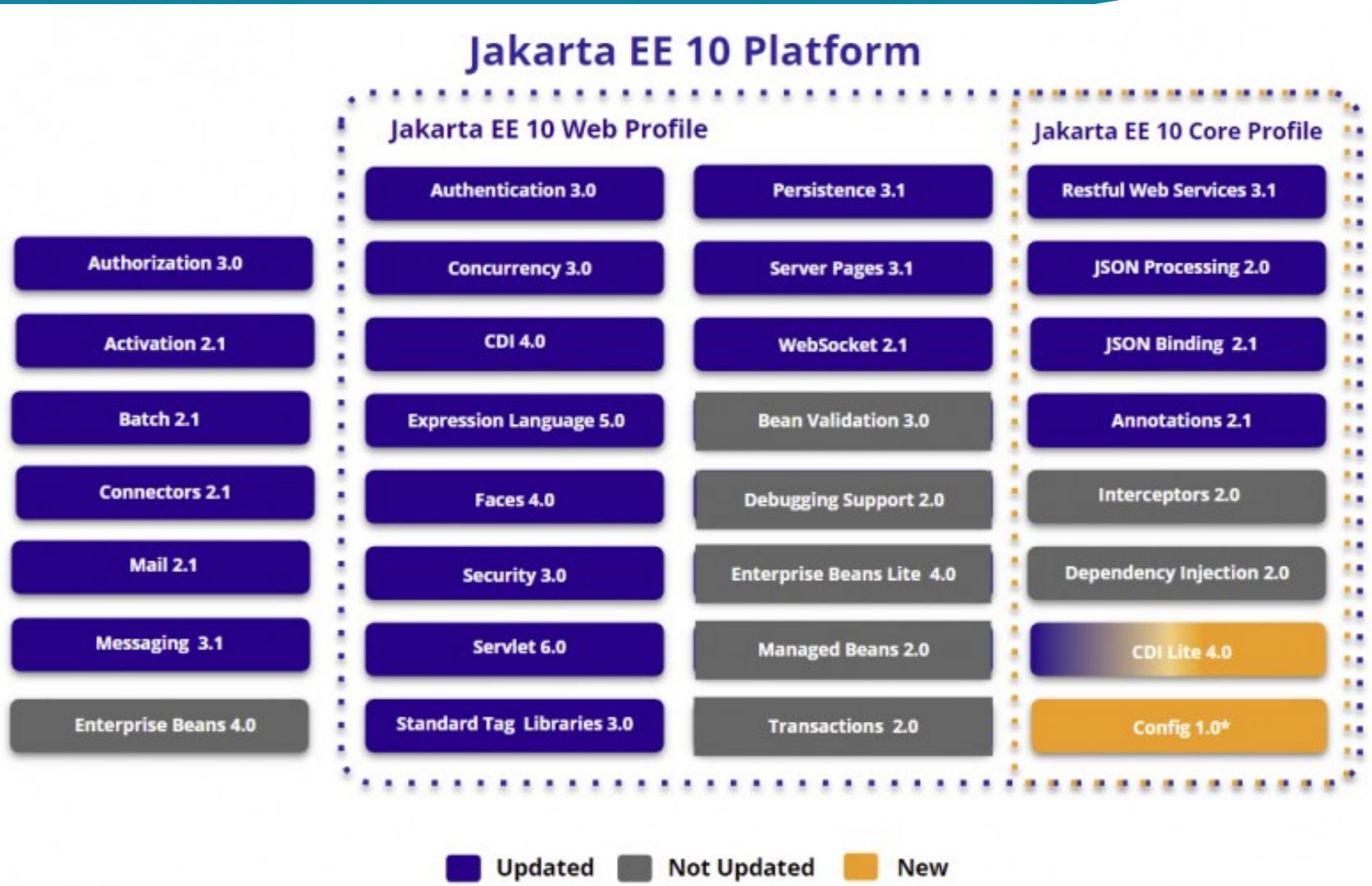
- Altamente Dependiente del Framework Spring y sus dependencias.
- Algunas actualizaciones (spring) en configuración han sido muy drásticas. DeprecatedHell.
- Para programadores noveles, la intervención en algunos módulos suele ser compleja (manejo de sesiones, gestión de beans).
- Configuración custom compleja.
- La semantica de funcionalidades avanzada, suele ser complicada (a veces, fea).

Otras Opciones Java...

Alternativas Java: Java EE & Jakarta

- Es una plataforma de programación (parte de la Plataforma Java) para desarrollar y ejecutar software con arquitectura de N capas distribuidas y que se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.
- La plataforma Java EE está definida por un conjunto de especificaciones administradas por el JCP (Java Community Process), que exteienden la version estandard de Java (Java SE).
- La ultima versión de JavaEE mantenida por Oracle fué la 8. El mantenimiento de la especicficación pasa a la Eclipse Fundation, cambiando el nombre por Jakarta EE.
- La versión mas actual es la Jakarta EE 10.

Alternativas Java: Java EE & Jakarta



Alternativas Java: Java EE & Jakarta

```java
@EJB
@Stateful
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public class EmployeeExtendedDAO implements IEmployeeDAO{

    @TransactionAttribute(TransactionAttributeType.NEVER)
    public Employee getEmployeeById(Long empId){
        //Se consulta el empleado mediante el ID
        return manager.find(Employee.class, empId);
    }
}
```

```java
@Path("multiparts")
@RequestScoped
public class MultipartResource {
    private static final Logger LOGGER = Logger.getLogger(MultipartResource.class.getName());

    java.nio.file.Path uploadedPath;

    @PostConstruct
    public void init() {
        try {
            uploadedPath = Files.createTempDirectory(Paths.get("/temp"), "uploads_");
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    @Path("simple")
    @POST
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    public Response uploadFile(@FormParam("name") String name,
                               @FormParam("part") EntityPart part) {
        LOGGER.log(Level.INFO, "name: {0} ", name);
```

```java
@Path("greeting")
@RequestScoped
public class GreetingResource {
@GET
    public String hello(@QueryParam("name") String name) {
        return "Say 'Hello' to " + (name == null ? "World" : name) + " at " + LocalDateTime.now();
    }
}
```

Alternativas Java: Quarkus

- Es un *framework* Java diseñado para Kubernetes. Componentes clave: OpenJDK HotSpot y GraalVM.
- El objetivo de Quarkus es ser una plataforma líder en Kubernetes y entornos *serverless.*
- Ofrece un entorno de programación imperativo y reactivo tratando de optimizar el desarrollo de arquitecturas de aplicaciones distribuidas.

Alternativas Java: Quarkus

```java
package org.acme.rest.json;

import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.Set;

import jakarta.ws.rs.DELETE;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;

@Path("/fruits")
public class FruitResource {

    private Set<Fruit> fruits = Collections.newSetFro

    public FruitResource() {
        fruits.add(new Fruit("Apple", "Winter fruit")
        fruits.add(new Fruit("Pineapple", "Tropical f
    }

    @GET
    public Set<Fruit> list() {
        return fruits;
    }

    @POST
    public Set<Fruit> add(Fruit fruit) {
        fruits.add(fruit);
        return fruits;
    }

    @DELETE
    public Set<Fruit> delete(Fruit fruit) {
        fruits.removeIf(existingFruit -> existingFrui
        return fruits;
    }
}
```

```java
@Path("/customers")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@Tag(name = "customer", description = "Customer Operations")
@AllArgsConstructor
@Slf4j
public class CustomerResource {

    private final CustomerService customerService;

    @GET
    @APIResponse(
            responseCode = "200",
            description = "Get All Customers",
            content = @Content(
                    mediaType = MediaType.APPLICATION_JSON,
                    schema = @Schema(type = SchemaType.ARRAY, implementation = Customer.class)
            ))
    public Response get() {
        return Response.ok(customerService.findAll()).build();
    }

    @GET
    @Path("/{customerId}")
    @APIResponse(
            responseCode = "200",
            description = "Get Customer by customerId",
            content = @Content(
                    mediaType = MediaType.APPLICATION_JSON,
                    schema = @Schema(type = SchemaType.OBJECT, implementation = Customer.class)
            ))
    @APIResponse(
            responseCode = "404",
            description = "Customer does not exist for customerId",
            content = @Content(mediaType = MediaType.APPLICATION_JSON)
    )
    public Response getById(@Parameter(name = "customerId", required = true) @PathParam("customerId") Integer customerId) {
        return customerService.findById(customerId)
                .map(customer -> Response.ok(customer).build())
                .orElse(Response.status(Response.Status.NOT_FOUND).build());
    }
```

Addendum...

Dilema: Transacción, Concurrencia, Alta Demanda

Transacción

Capacidad de los sistemas computacionales de manejar operaciones de manera completa y correcta, de lo contrario, notificar el error sin que el sistema tenga afectación.

Tenemos la JTA (Java Transaction API) y Spring nos brinda la anotación @Transactional.

[15]

Concurrencia

Capacidad de los sistemas computacionales para manejar peticiones simultaneas (en terminos humanos*) de manera *coherente*. Se suelen usar pilas de prioridad y búfferes.

En pring se resuelve mediante @EnableAsync en unbean de configuración.

Alta Demanda

Capacidad de un sistema computaciónal para recibir grán cantidad de solicitudes sin saturación o bloqueo. Está intimamente ligado a las capacidades del hardware.

En spring se ***mitiga*** mediante @EnableAsync en un bean de configuración.

* Punto de fusión o *“integración sensorial”* [14]

Clientes Web...

Las tres partes del patrón de diseño de software MVC se pueden describir de la siguiente manera:

- **Modelo**: Maneja datos y lógica de negocios. (Capa de Persistencia)
- **Vista**: Se encarga del diseño y presentación acondicionamiento de datos. (Capa de lógica de negocio)
- **Controlador**: Enruta comandos a los modelos y vistas. (Capa de Routing. EntryPoint)

```
OkHttpClient client = new OkHttpClient().newBuilder()
  .build();
MediaType mediaType = MediaType.parse("application/json");
RequestBody body = RequestBody.create(mediaType, "{\n    \"nombre\": \"Logan\",\n
    \"apellidos\": \"Strange\",\n    \"pin\": \"12374\"\n}\n");
Request request = new Request.Builder()
  .url("127.0.0.1:8001/empleados")
  .method("POST", body)
  .addHeader("Content-Type", "applicatior
  .addHeader("Authorization", "Basic dXNl
  .build();
Response response = client.newCall(reques
```

```
Unirest.setTimeouts(0, 0);
HttpResponse<String> response = Unirest.post("127.0.0.1:8001/empleados")
  .header("Content-Type", "application/json")
  .header("Authorization", "Basic dXNlcm5hbWU6NWEyNWM5ZjI1YzMzNGY0MTk3ZGY0ZDJhYWZjYTVmZDk=")
  .body("{\n    \"nombre\": \"Logan\",\n    \"apellidos\": \"Strange\",\n    \"pin\":
    \"12374\"\n}\n")
  .asString();

```

Spring Batch...

- *Framework* trabajos por lotes completo y ligero.
- Spring Batch proporciona funciones reutilizables esenciales en el procesamiento de grandes volúmenes de registros.
- Gestión de transacciones.
- Procesamiento basado enfragmentos.
- Manejo de trabajos Start/Stop/Restart
- Proporciona optimizaciónes avanzadas de segmentación de grandes volúmenes de registros para garantizar el alto desempeño.
- Escalable.
- Procesamiento por lotes concurrente: procesamiento paralelo de un trabajo.
- Reinicio manual o programado después de una falla.
- Procesamiento secuencial de pasos dependientes (con extensiones a lotes controlados por flujo de trabajo).
- Procesamiento parcial/selectivo: puede omitir registros
- Transacción de lote completo, para casos con un tamaño de lote pequeño o procedimientos o scripts almacenados existentes.

Spring Batch...

```java
@Bean
public Job importUserJob(JobRepository jobRepository,
    JobCompletionNotificationListener listener, Step step1) {
  return new JobBuilder("importUserJob", jobRepository)
    .incrementer(new RunIdIncrementer())
    .listener(listener)
    .flow(step1)
    .end()
    .build();
}

@Bean
public Step step1(JobRepository jobRepository,
    PlatformTransactionManager transactionManager, JdbcBatcl
  return new StepBuilder("step1", jobRepository)
    .<Person, Person> chunk(10, transactionManager)
    .reader(reader())
    .processor(processor())
    .writer(writer)
    .build();
}
```

```java
@Component
public class JobCompletionNotificationListener implements JobExecutionListener {

  private static final Logger log = LoggerFactory.getLogger(JobCompletionNotificationListener.class);

  private final JdbcTemplate jdbcTemplate;

  @Autowired
  public JobCompletionNotificationListener(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
  }

  @Override
  public void afterJob(JobExecution jobExecution) {
    if(jobExecution.getStatus() == BatchStatus.COMPLETED) {
      log.info("!!! JOB FINISHED! Time to verify the results");

      jdbcTemplate.query("SELECT first_name, last_name FROM people",
        (rs, row) -> new Person(
          rs.getString(1),
          rs.getString(2))
      ).forEach(person -> log.info("Found <{{}}> in the database.", person));
    }
  }
}
```

Otros...

- **Cifrado**: ECIES, BouncyCastle
- ***Logs***: SLF4J, logback. Auditoria de aplicación.
- **Monitoreo/Telemetría**: Dynatrace, WatchDog, NewRelic...
- **Test**: JUnit, Mockito
- ...

Referencias

[1] https://developer.mozilla.org/en-US/docs/Glossary/MVC

[2] https://spring.io/guides/gs/serving-web-content/

[3] https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

[4] https://blogs.eclipse.org/post/tanja-obradovic/jakarta-ee-2021-review-and-community-update-january-2022

[5] https://www.baeldung.com/spring-component-repository-service

[6] https://jakarta.ee/specifications/mvc/

[7] https://jcp.org/en/jsr/detail?id=311

[8] https://jcp.org/en/jsr/detail?id=339

[9] https://www.ibm.com/docs/es/was-liberty/nd?topic=overview-java-persistence-api-jpa

[10] https://www.baeldung.com/spring-data-jpa-stored-procedures

[11] https://www.baeldung.com/spring-controller-vs-restcontroller

[12] https://spring.io/guides/tutorials/rest/

[13] https://www.geeksforgeeks.org/spring-boot-transaction-management-using-transactional-annotation/

[14] https://www.neurobidea.com/integracion-sensorial/

[15] https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/transaction.html

[16] https://spring.io/projects/spring-batch

[17] https://www.baeldung.com/mockito-series

[18] ***https://gitlab.com/bravotorres.alejandro/api-java-demo.git***

¡Gracias!

Alejandro Bravo Torres
alejandro.bravo@bancoazteca.com.mx