***Constructor de Aplicaciones para Java***

Presentación

**AppsBuilder*4J***

**2014**

**Presentación de la arquitectura: 1ª Edición**

AppsBuilder4J

Autor:

Versión: 1.0

Copyright © 2014

**Tabla de Contenidos**

[Introducción 5](#_Toc373082999)

[Principios fundamentales 6](#_Toc373083000)

[Cuestiones a las que tenemos que tener respuestas para ofrecer el producto. 6](#_Toc373083001)

[¿Cuáles son sus características? 6](#_Toc373083002)

[¿Qué beneficios aportaría a mi empresa? 6](#_Toc373083003)

[Arquitectura tecnológica 8](#_Toc373083004)

[CAPA DE MODELADO 9](#_Toc373083005)

[SEGURIDAD 9](#_Toc373083006)

[CONFIGURACIÓN 9](#_Toc373083007)

[MODELO DE NEGOCIO 9](#_Toc373083008)

[CAPA DE PRESENTACIÓN 11](#_Toc373083009)

[MÓDULOS 11](#_Toc373083010)

[FORMULARIOS 11](#_Toc373083011)

[COMPONENTES 11](#_Toc373083012)

[RECURSOS 11](#_Toc373083013)

[CAPA DE SERVICIOS 12](#_Toc373083014)

[FEATURES *f++* 14](#_Toc373083015)

[HERRAMIENTAS 14](#_Toc373083016)

[Arquitectura de negocio 15](#_Toc373083017)

[Entorno de desarrollo 16](#_Toc373083018)

[HERRAMIENTAS DE DESARROLLO 16](#_Toc373083019)

[CONTROL DE VERSIONES 16](#_Toc373083020)

[TDD 16](#_Toc373083021)

[LIBRERÍAS DE TERCEROS (dependencias por defecto): 16](#_Toc373083022)

[METODOLOGÍA DE TRABAJO 16](#_Toc373083023)

[Caso de uso *(borrador)* 17](#_Toc373083024)

[Publicidad 19](#_Toc373083025)

[Epílogo 20](#_Toc373083026)

Capítulo 1

Introducción

Esta guía está dedicada a proporcionar una visión general de la arquitectura **AppsBuilder4J**.

Capítulo 2

Principios fundamentales

# Cuestiones a las que tenemos que tener respuestas para ofrecer el producto.

# ¿Cuáles son sus características?

1. Arquitectura fácil de entender, usar y extender.
2. Arquitectura ágil para aumentar nuestra productividad desde soluciones simples hasta las más complejas.
3. Arquitectura abierta que permita integrar nuevas características mediante la creación de Features. Configuración parecida a un plugin donde podremos definir nuevos proveedores de servicios o definir nuevos servicios. Además, podrá contar con su propio modelo de configuración.
4. Arquitectura basada en modelos de configuración programados (recomendamos Groovy por su simplicidad) que evite el uso de archivos de texto tradicionales como .xml o .properties.
5. Arquitectura desacoplada (uso de interfaces) que permita la selección de los tipos de modelos y servicios que necesitemos.
6. Arquitectura que nos permita integrar desde aplicaciones Legacy (heredadas) hasta productos comerciales utilizando tecnologías como REST o SOA.
7. Arquitectura basada en la instanciación de objetos usando Spring IoC o JEE para evitar el uso masivo de factorías.
8. …

# ¿Qué beneficios aportaría a mi empresa?

1. Reducción drástica de tiempos.
2. Reducción de la inversión.
3. Escalabilidad segura
4. Integración con otras aplicaciones y productos de su empresa.
5. Garantía de actualización a tecnologías futuras.
6. Mejora el trabajo en equipo
7. Reduce los riesgos ante cambios evolutivos o correctivos.
8. Podría informatizar procesos complejos con una inversión reducida.
9. Mejoraría la productividad de su empresa al integrar todos sus procesos.
10. …

Arquitectura tecnológica

**Modelo de arquitectura basado en capas (MPS):**

**CAPA DE MODELADO**

**CAPA DE SERVICIOS**

**CAPA DE PRESENTACIÓN**

# CAPA DE MODELADO

Capa donde incluiremos toda la información relativa a la configuración de nuestra solución. Sólo contiene información que será utilizada por los servicios.

## SEGURIDAD

* **Modelo**: Roles, Grupos de usuarios.
* **Permisos**: Asociación de elementos del modelo a los roles y grupos.
* **Servicios**: En la capa de servicios hay que incluir esta misma asociación.

## CONFIGURACIÓN

* + - * **Entornos**: Bases de datos, seguridad, logs,...
      * **Localización**: Gestión de recursos de idiomas para la traducción de texto del sistema y del modelo de la solución. Formatos, moneda,...

## MODELO DE NEGOCIO

Modelo de Dominio

* + - **Entidades**
    - **Repositorios** (acceso a datos mediante funciones o querys Spring Data NoSQL, JPA,...).
    - **Modelo de Entidad** (Configuración de elementos de una entidad)
      * **QueryView** (Vista de entidad asociadas a métodos del repositorio @Query).
      * **RelationView** (Relaciones entre entidades @ManyToOne, @OneToMany, ... para la edición de campos relacionados con otras entidades).
      * **Filter** (Filtros de entidad basado en expresiones y entrada de datos mediante parámetros (Ej.: where firstname = :firstname).
      * **Rule** (Reglas de negocio: Arquitectura, Drools, ?) Contexto servidor (ver Edition Rules en contexto de presentación).
      * **Layout** (tipologías para diferentes dispositivos y estilos).

...

CMS (Sistema de gestión de contenidos)

Alertas

Tareas de usuario.

Procesos batch

Interfaces

Informes

# CAPA DE PRESENTACIÓN

Capa donde incluiremos toda la información relativa a la configuración de nuestra interface gráfica. Sólo contiene información que será utilizada por los servicios de presentación.

## MÓDULOS

Items y grupos de ítems (Formularios, informes, procesos, …)

## FORMULARIOS

* **Editores**: tipologías CRUD, Grid, edición de objeto, MasterDetail, Jerárquicos, custom,...
* **Acciones** vinculadas a los editores y formularios
* CRUD (+ - update, delete)
* **Enlace** otros formularios,...
* **Validaciones** y **reglas** de edición.

## COMPONENTES

* **Tipología de campos**

(FieldUI) Ej.: moneda, email, url, dirección, nif, relación,... (se podrán crear Features para incluir más tipologías).

* **Acciones** (categorías).
* **Home**

(Pantalla principal de la aplicación que gestionará los módulos, Dashboard, ...)

* **Custom Window**.

## RECURSOS

Imágenes, iconos, idiomas,...

# CAPA DE SERVICIOS

Los servicios serán los encargados de ejecutar el modelo definido en el resto de capas.

* **DATA** (Implementación de IDao, Respository, ...) : Provider: Spring Data, Hibernate, JPA, Custom, ... ?
* **ENVIRONMENT** Provider: Arquitectura (configura los servicios y el modelo Usa: Model Environment)
* **DOCUMENT** Provider: Arquitectura (Generador de documentación de los componentes de la solución)
* **REPORTS** (Información de gestión) Provider: Arquitectura (Excel y Doc), BI Birt, BI Jaspert Report, ?. Usa: Model Reports (categorizados y obtenidos de otras fuentes de negocio legacy). Revisar el tema para una solución abierta e integral.
* **BUSINESS** (Servicios de Negocio) Provider: Spring REST, Apache CXF SOA, Arquitectura, ... (Analizar como en BdE) Usa: Model Business definición de servicios por categorías.
* **SECURITY** (Control de accesos, usuarios, ...) Provider: Spring Security, Shiro, Custom, ? Usa: Model Security
* **CMS** Provider: Google Drive, Database, Archivos, Custom, ?
* **PRESENTATION UI** (Pattern MVP reutilizar de G.Mobile) Provider: Vaadin Desktop, Vaadin Mobile, Spring MVC, ?
* **LOCALIZACIÓN** (conversor de texto solución y sistema, excepciones multilenguaje, formato, horario, ...) Usa: MODEL LOCALE (recursos multilenguaje). Ver con enums.
* **INTERFACES**: Provider: Arquitectura (procesos de importación y exportación de datos en diferentes fuentes (.xls, .txt, .csv, .xml, .json, ...), ?
* **REGLAS DE NEGOCIO**: Provider: Drools, ?
* **ALERTAS**: Provider: Arquitectura, ?
* **TASKS**: Provider: Arquitectura (reutilizar de la anterior arquitectura creando tipologías. analizar), ?
* **BATCH**: Provider: Spring Batch, ?
* **WORKFLOW**: Provider: Activity, jBPM, ?
* **PORTALES**: Provider: Liferay, ?
* **Custom service** (***f++*** Feature)

# FEATURES *f++*

Extensiones de la arquitectura mediante la integración de nuevas funcionalidades o características (similar al concepto Plugin de Eclipse).

Cada *feature* puede incluir cualquier de las capas de la arquitectura: Modelo, Presentación y Servicios.

(Ej.: CMS Google Drive debe implementar la interface ICMS model y servicios ICMSServices del API Google).

(Ej.: componente NIF con model IField para incluir nuevos tipos de datos. Incluirá una capa de presentación: IProviderView (Web y Mobile) para incluir en UI y Presenter IPresenterProperty)

# HERRAMIENTAS

1. **Monitorización** de datos del entorno: usuarios conectados, logs,...
2. **Reports Builder** (reutilizar anterior arquitectura pero adaptada al nuevo modelo y servicios).
3. **Dashboard**: Centro de trabajo donde el usuario puede acceder a todas sus tareas asignadas (tareas pendientes, tareas realizadas, planificación de alarmas o avisos, procesos, informes o interfaces con otros sistemas).
4. Explorer: Herramienta de monitorización de todos los elementos de su solución y generación de documentación.
5. Creación de nuevas herramientas o integración de otras existentes en la empresa mediante features.

Arquitectura de negocio

Capacidades que tiene la arquitectura para extender nuestras soluciones dentro de la empresa.

1. Integración entre soluciones conectando modelos y servicios
2. Soluciones extensibles para adaptarlas a los diferentes requerimientos de implantación por empresas.
3. Mecanismos de extensión:

* Extender modelos
* Extender servicios
* Crear features conectadas a la arquitectura que contengan modelos y servicios de negocio para su reutilización en la creación de nuevas soluciones o soluciones extendidas.

Entorno de desarrollo

# HERRAMIENTAS DE DESARROLLO

**Eclipse** con plugins: **Maven** o **Gradle**, Plugin **Vaadin**, **Groovy** plugins, **Sonar**, **Jenkins**,...

# CONTROL DE VERSIONES

**GitHub** para proyectos y dependencias de la arquitectura, ++features y otros proyectos que necesitemos incluir en nuestros proyectos como dependencias (jars)

# TDD

Desarrollo guiado por pruebas de software, o Test-driven development

* + - Pruebas unitarias: **JUnit**
    - Pruebas de integración: **Jenkins**
    - Cobertura de código: **Sonar**

# LIBRERÍAS DE TERCEROS (dependencias por defecto):

**Log4j** o **SLF4J** (consensuar)

# METODOLOGÍA DE TRABAJO

**SCRUM**

* + - Buscar herramienta para trabajar online (Ver **Mylyn** con repositorio en Github)

Caso de uso ***(borrador)***

Primeros pasos para consolidar los conceptos de la arquitectura expuestos en esta presentación.

1. Crear una solución que contenga:
2. Una entidad en el modelo de datos.
3. Un formulario de tipo grid de entidad que nos permita editar, añadir objetos y eliminarlos. CRUD básico.
4. Definir un entorno de ejecución básico donde incluiremos los datos de conexión a nuestra base de datos (MySQL ej.:)
5. Realizar la persistencia utilizando el servicio de datos del proveedor Spring Data.

1. Pasos a seguir:
2. Crear proyecto Java Maven.
3. Instanciar un objeto Solution
4. Crear el MODELO con:
5. Datos: Configurar una entidad (JPA) mediante Goovy en /model/business/entity/Person (utilizar inglés para poder subir este ejemplo a la web www.appbuilder4j.com).
6. Environment: Crear un repositorio para la entidad de tipo CRUD (Spring Data).
7. UI: Configurar un módulo y un formulario de tipo CRUD asociado a la entidad de datos creada.
8. Crear los SERVICIOS con:
   * Spring Data
   * Vaadin Web y Mobile para ejecutar la solución.

Propuesta de Estructura de packages:

/solution

solution.groovy

/business

/domainmodel

/entities

/entityViews

/security

/roles

/...

/ui

/applications RRHH

rrhhapp.groovy

/modules PLANTILLA, FACTURACION, NOMINAS

/views Plantilla, Organigrama, ... (vistas de mantenimiento, navegadores, informes, ...)

....

Crear un documento de guía de usuario con imágenes o video.

Definir la solución, el modelo y los servicios que necesitamos para empezar a constuir nuestra aplicación.

Ver codificación Java y clases actuales (SolutionModel, SolutionManager, ...

Al final incluir "too easy" como slogan de la arquitectura.

Publicidad

1. Crear web e incluir publicidad en redes sociales (blog, twiter, facebook, Linkedin..) revisar
2. Contactar con Autentia como posible colaborador.
3. .... (pendiente de análisis)

Epílogo

***"Lo más hermoso del trabajo en equipo es que siempre tienes a otros de tu lado."***

***"El hombre nunca sabe de lo que es capaz hasta que lo intenta."***