

Sistema integral de gestión de proyectos, GCBA

Especificación de Arquitectura







Historial de revisiones

|  |  |  |
| --- | --- | --- |
| VERSIÓN | FECHA | AUTOR – PRINCIPALES CAMBIOS |
| 0.1 | 05/08/2016 | Federico Bucich – Creación del contenido inicial. |

Índice

[1 Introducción 3](#_Toc458525468)

[1.1 Propósito 3](#_Toc458525469)

[1.2 Definición de términos y abreviaturas 3](#_Toc458525470)

[1.3 Referencias 3](#_Toc458525471)

[2 Organización de alto nivel 4](#_Toc458525472)

[2.1 Vista lógica 4](#_Toc458525473)

[2.2 Vista física 8](#_Toc458525474)

# Introducción

## Propósito

Este documento contiene todas las decisiones que determinan la arquitectura del Sistema de Gestión de Proyectos GCBA.

El objetivo es el desarrollo de un software que permita realizar seguimiento a los proyectos internos de las distintas Jurisdicciones de la Ciudad de Buenos Aires.

Se describen los aspectos técnicos del proyecto con los siguientes objetivos:

* Especificar la arquitectura del sistema, estableciendo el diseño de alto nivel.
* Indicar cómo las decisiones tomadas satisfacen los requerimientos no funcionales del sistema.
* Especificar las estrategias utilizadas para la integración con otros sistemas.
* Especificar los criterios de selección de componentes de terceros.

## Definición de términos y abreviaturas

Los siguientes términos o abreviaturas son usados a lo largo de este documento:

|  |  |
| --- | --- |
| término o abreviación | Descripción |
|  |  |
|  |  |

## Referencias

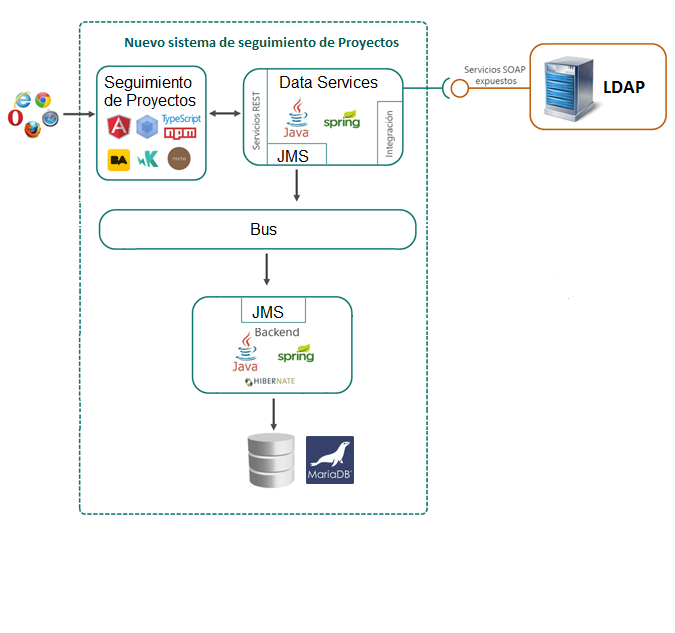
|  |  |
| --- | --- |
| documento | Descripción |
|  |  |
|  |  |

# Organización de alto nivel

A alto nivel, la plataforma de Proyectos está formada por tres aplicaciones bien definidas: un frontend y una capa de data services, y un backend.

A lo largo de esta sección se irán detallando cada una de ellas desde lo más general a lo más específico, con el objetivo de poder entender todas las decisiones de diseño tomadas.

## Vista lógica



Como se observa, la aplicación consiste básicamente en cuatro partes:

* El frontend utilizado por los usuarios para la carga, edición, visualización y seguimiento de proyectos.
* Una capa de data services que le brinda los servicios necesarios para la alta, baja y modificación de las entidades del sistemas.
* Un bus de servicios donde diversas aplicaciones podrán publicar los mensajes a ser procesados por el backend.
* Un backend que se encarga de persistir y recuperar información de la base de datos.

### Frontend para la carga de solicitudes

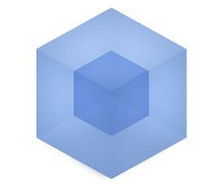
Es una aplicación web responsive desarrollada en AngularJs 1.5, la cual es utilizada por los ciudadanos en distintos browsers.

Los frameworks utilizados en dicha aplicación son:

**AngularJS:** framework javascript open-source escrito y mantenido por Google que permite crear SPA (Single Page Application) permitiendo extender el lenguaje HTML con el fin de incluir contenido dinámico. Cada componente en AngularJS puede tener su propio comportamiento y puede ser extendido y reemplazado si es necesario.

**https://upload.wikimedia.org/wikipedia/commons/a/a6/TypeScript_Logo.pngTypescript:** lenguaje que permite introducir el concepto de tipos en javascript, agregando además tipos adicionales, clases y módulos. Typescript compila a código javascript legible y basado en estándares.

**https://upload.wikimedia.org/wikipedia/commons/thumb/d/db/Npm-logo.svg/2000px-Npm-logo.svg.pngnpm:** gestor de paquetes para javascript. Permite buscar, compartir y reusar paquetes de código de miles de desarrolladores y utilizarlos de la forma que sea necesario.



**Webpack Module Bundler:** Bundler para módulos de archivos javascripts para usar en los browsers, con la capacidad de transformar, construir o empaquetar cualquier recurso.



**BAstrap:** herramienta que permite unificar la interfaz de los sitios web del Gobierno de la Ciudad de Buenos Aires. Usa HTML, CSS, y JS basado en Bootstrap.

**TSLint:** linter para código typescript. Detecta errores y potenciales problemas con el objetivo de forzar las convenciones de código y las buenas prácticas de programación.

**Karma:** javascript test runner creado por los mismos creadores de AngularJS. Permite la ejecución de test tantos en ambientes de desarrollo como en ambiente de Integración Continua. Puede lanzar la ejecución en múltiples browsers (Chrome, Firefox, IE, etc.) y ejecutar los test contra ellos, permitiendo obtener la máxima confianza en el código.

****

**Mocha:** framework BDD (behavior-driven development) para testing de código javascript.

### Data Services

La REST API o capa de servicio es donde se publican todos los servicios que pueden ser usados por las aplicaciones clientes de presentación.

Dichos servicios son expuestos mediante interfaces bien definidas, las cuales se agrupan de acuerdo a las funcionalidades expuestas de cada entidad fuerte de negocio.

Es importante remarcar que en esta capa solo se publican los servicios de acuerdo a los criterios funcionales establecidos y no existe ninguna lógica de negocio en si desarrollada (esto es responsabilidad de la capa de modelo), evitando de este modo un acoplamiento indeseado entre las capas.

En resumen, esta capa es responsable de:

* Exponer una API REST simple que le permitan a otras aplicaciones realizar la carga, edición y seguimiento de proyectos.
* Delegar a la capa de modelo la resolución de las reglas de negocio.
* Proveer servicios que faciliten la seguridad (autenticación y autorización) mediante la integración con LDAP, utilizando el servicio SOAP provisto por el Gobierno de Buenos Aires.
* Manejar errores, ya sean excepciones de negocio conocidas como errores inesperados.
* Auditar el uso de los servicios.

Además, es importante destacar que todos los servicios expuestos como servicios REST usan json como tipo de datos en los parámetros de entrada y de salida. Toda la lógica de transformación de jsons en objetos de negocio y viceversa es encapsulada en esta capa. De este modo, la taxonomía típica de una implementación de un servicio es:

1. Autorizar el uso del servicio
2. Convertir los parámetros de entrada de json a objetos de negocio
3. Delegar el trabajo a la capa de modelo de negocio
4. Manejar excepciones si es necesario
5. Convertir el resultado en json
6. Auditar el uso del servicio
7. Retornar el json

Las principales tecnologías utilizadas en esta capa son:

Spring framework, Jackson y una arquitectura RESTful para definición de los mensajes.





### Bus de Servicios

Se utiliza [ActiveMQ](http://activemq.apache.org/) de Apache Software Foundation.

### BackEnd – Capa de Modelo

Esta capa es la responsable de implementar todas las reglas y procesos de negocio necesarios para la carga y seguimiento de proyectos y es accedida por la capa de servicios a través del Bus de servicios.

En esta capa, se modelan todas las entidades de negocio para que colaboren entre sí para implementar toda la lógica de negocio necesaria. Ejemplo de esas entidades son Proyecto, Objetivos Estrategicos, Ejes de Gobierno, Jurisdiccion, etc.

Es muy importante realizar un buen trabajo de análisis y modelado en esta capa, dado que se implementan muchas reglas de negocio y es clave mantener un buen modelo, con responsabilidades bien definidas de cada una de sus entidades para asegurar que la solución generada sea fácil de escalar y mantener.

Las principales tecnologías utilizadas en esta capa son:

Spring framework, JPA, Hibernate, Jackson y JMS





#### Capa de persistencia

La capa de persistencia es responsable de abstraer el modo en el cual la información es persistida en el repositorio (la base de datos relacional MariaDB en este caso).

Las principales tecnologías utilizadas en esta capa son:

Hibernate, JPA y MariaDB C:\Users\fbucich\Desktop\mariadb.png

#### Integración

Los servicios de integración se encargan de encapsular toda la lógica necesaria para realizar las interacciones con otras aplicaciones mediante los distintos protocolos (por ejemplo: webservices SOAP para validar usuarios).

## Vista física

### Software

Para el frontend se sugiere utilizar un servidor Apache HTTP. El frontend es básicamente una SPA (single page application) desarrollada en AngularJS, por lo que el servidor Apache HTTP satisface correctamente las necesidades que se solicitan para poder servir dicho contenido. En caso de que desde la ASI se proponga otra cosa (utilizar NGINX por ejemplo), se puede hacer sin ningún inconveniente.

Para los data services se sugiere utilizar un servidor Apache Tomcat. Los data services están hechos en Java 8 y spring, por lo que necesitamos tener un contenedor JEE para poder levantar el contexto y exponer los servicios REST, por eso es la recomendación de utilizar Apache Tomcat.

En cuanto a la base de datos, se utiliza MariaDB. Dada la experiencia, conocimientos y estándares que el área de Operaciones de la ASI tiene al respecto, se deja a su decisión el esquema de replicado de la base, backups y eliminación del SPOF.

Para todos los servidores recomendamos utilizar Linux como sistema operativo. La distribución a utilizar dependerá de los estándares que el área de operaciones

Finalmente, en la siguiente tabla se especifican el software que debe disponer el servidor:

|  |  |  |
| --- | --- | --- |
| Servidor | Sistema Operativo | Software instalado |
| Proyectos frontend - backend | Linux (distribución a definir por la ASI) | Apache2 HTTP Server – Versión 2.4.x  Java JRE – Versión 8.x  Apache Tomcat – Versión 8.0.x  MariaDB – Versión 10.x |

### Hardware

Como requerimiento necesitaremos que el servidor disponga con 100GB al inicio. Luego el almacenamiento ira creciendo por lo que se necesitara que se agreguen 20 GB por año. También necesitaremos que esto sea monitorizado en caso de que crezca más rápido de lo planificado.

