Estándar de Desarrollo del Proyecto IoT

1. Requisitos del Software
   1. Los requisitos del software deben ser organizados en:

* Requisitos funcionales. Aquellos relacionados con la gestión de datos
* Subsistemas. Los subsistemas son los aplicativos que pertenecen a un mismo sistema que pueden ser desarrollado de manera separada pero que se integren entre si.
* Funciones. Módulo o componente principal del software o subsistema.
* Subfunciones. Módulos más pequeños que pertenecen a las funciones.
* Requisitos específicos. Son operaciones directas, es decir, los algoritmos a codificar.
* Requisitos no funcionales. Son aquellos relacionados con:
  + Seguridad de la aplicación. Definir el tipo de seguridad de la aplicación. Por ejemplo: encriptación, sistema de usuario y contraseña, roles, captchas, entre otros.
  + Interfaces. Tipos de interfaces:
    - Usuarios: escritorio, web, móvil
    - Hardware: red lan cableada o wifi, bluetooth, zeegbe y cualquier otra interfaz que se vaya a utilizar
    - otras aplicaciones: Middleware que ayudan a coordinar la comunicación entre aplicaciones. Ejem Web Service(WSDL, REST), Protocolos o middleware personalizado.
  + Rendimiento. Tiempo máximo para la carga de una página web, tiempo para consultas, usuarios permitidos en líneas, entre otras políticas.
  + Tecnologías. Aspectos Hardware, Software y elementos de comunicación tanto para desarrollar como para implementar.
  + Otros. Los que se establezcan en el equipo de desarrollo y el cliente.

1. Análisis del Software:
   1. Procesos. Se aplicaran diagramas de casos de uso y de actividades de UML
   2. Datos. Modelo entidad relación y diagrama de clases a nivel conceptual.
2. Diseño del Software
   1. Arquitectura. Establecer un modelo de arquitectura del software:
      * MVC , MVT, basado en capas o componentes
   2. Modelo de Datos.
      * Modelo de Archivos
        + Definir el tipo y la estructura de archivos (Textos, binario, Json, XML, u otros)
      * Modelado de la Base de Datos
        + La base de datos debe tener un diseño modular:
          1. Se deben definir los módulos y sus prefijos: Ejm:

|  |  |
| --- | --- |
| Módulo | Prefijo |
| Seguridad | seg\_ |
| Gestión del Proyecto Iot | proy\_ |
| Red de sensores | red\_ |
| Monitoreo | mon\_ |
| Gestión de Cultivo | cul\_ |
| Gestión de Riego | rieg\_ |

* + - * Cada tabla de la base de datos debe pertenecer a un módulo. Se deben nombrar las tablas anteponiendo el prefijo del módulo. Ejemplo: seg\_usuario, seg\_grupos.
      * Los campos deben ser nombrados anteponiendo el prefijo de la tabla a la que pertenecen directamente.
      * Los campos foráneos (FK) deben mantener el nombre que obtuvieron en la tabla padre a excepción de los campos recursivos, los cuales deben agregarle a su nombre original un posfijo que lo diferencie.
      * Los campos foráneos primarios (PFK) en el caso de que sean derivados, deben modificarse su nombre para evitar conflictos, agregándoles un posfijo a su nombre original.
    - Modelado de Clases a nivel del aplicativo
      * Debe crearse un diagrama de clases para cada subsistema, puede ser también por componentes-paquete.

1. Desarrollo del Software

4.1. Estructura de repositorios del Proyecto para llevar el control de cambios realizarlo mediante la herramienta **Github**

* 1. Crear un Repositorio Master (con el nombre del proyecto) que agrupará todos los repositorios del proyecto
  2. Crear un Repositorio Final, donde se coloque los entregables verificados y validados del proyecto listos para poner en producción (implementar)
  3. Crear un repositorio de pruebas (de nombre pruebas) basado en el Repositorio final que contendrá la organización de carpetas según la arquitectura de desarrollo elegida. Los cambios de cada desarrollador que sean aprobados, se integrarán en este repositorio con la finalidad de probar, depurar y validarlos.
  4. Crear un Repositorio de trabajo por cada desarrollador basado en el repositorio de pruebas.

dingo\_Iot

dingo\_remoto\_test

dingo\_remot\_des\_A

dingo\_remot\_des\_B

dingo\_iot\_remoto

dingo\_local\_test

dingo\_local\_des\_C

dingo\_local\_des\_D

dingo\_iot\_local

4.2. Estructura de carpetas. Dependerá del entorno de desarrollo, del lenguaje de programación y del tipo de arquitectura elegidos.

4.3. Código fuente

4.3.1 Aplicativo para Cloud Comouting (Dingo\_iot\_remoto).

La tecnología a utilizar es: Lenguaje Python con framework Django, base de datos relacional Postgres.

**Estandarización en HTML**

|  |  |  |  |
| --- | --- | --- | --- |
| **Imagen** | **Elementos** | **prefijo** | **Ejemplo** |
|  | Input text | txt | txtNombre |
|  | Input btn o  Button | btn | btnGuardar |
|  | Input date | tdt | tdtFechaNacimiento |
|  | Combo box | cmb | cmbTipo |
|  | Radio button | rbt | rbtGenero |
|  | Check box | cbx | cbxEmpleo |
|  | Text area | txa | txaComentario |
|  | Input file | fic | ficFoto |
|  | Div | div | divFondo |
|  | Images | img | imgLogo |
|  | Menú | mnu | mnuPrincipal |
|  | Menu ítem | mni | mniEstado |
|  |  |  |  |

Estandarización en javascript

Estandarización en Python