

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

“Universidad del Perú, DECANA DE AMÉRICA”

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMATICA

Escuela de Ingeniería de Software

**TÍTULO**

Diseño WHALE Software

**INTEGRANTES G11**

Aguilar Burga, Piero André 16200203

Ávila Chunga, Emanuel Jeremy 16200206

Escobar Bendezú, Álvaro José 16200216

Cruz Huaman, Carlos Cruz 16200211

Rolando David Tueros Montes 16200243

Lima, Perú

**Índice**

[**1. Introducción**](#_loxdu5bydaj5) **4**

[1.1. Propósito.](#_4p55usf5brpr) 4

[1.2. Objetivos del diseño.](#_z731jn8ohjg1) 4

[1.3. Alcance](#_8ssbgus18af2) 4

[1.4. Definiciones, acrónimos y abreviaturas](#_433dtklz3acg) 4

[**2. Referencias**](#_dpvac8q03rol) **5**

[**3. Arquitectura del sistema**](#_mngiv65kymcm) **5**

[3.1. Arquitectura actual](#_m11a4yxvqczs) 5

[3.2. Arquitectura propuesta](#_tu3kd9wawssf) 5

[**4. Descripción de descomposición**](#_qphedaw390rq) **6**

[4.1. Descomposición en módulos](#_hstkiyd0ia29) 6

**Historial de Versiones**

| **Fecha** | **Versión** | **Descripción** | **Autor** |
| --- | --- | --- | --- |
| 30/05/2022 | 1.0 | Establecimiento del diseño | Escobar, Alvaro |
|  |  |  |  |

# 

# **1. Introducción**

## 1.1. Propósito.

Se implementará un diseño modular con una orientación a programación funcional que nos permitirá poder implementar los requerimientos del cliente de manera más directa y concisa. A su vez la arquitectura que usaremos será la arquitectura Onion o Cebolla que nos facilita la modularización del código.

## 1.2. Objetivos del diseño.

* Poder realizar un código escalable
* Permitir la modularización del código
* Agilizar el proceso de desarrollo
* Facilitar el uso de estructuras de código reutilizables
* Mantener la legibilidad del código
* Permitir cambios durante el desarrollo de los flujos de negocio

## 1.3. Alcance

El alcance que estamos esperando en este proyecto con respecto al diseño abarca desde el inicio del desarrollo del proyecto, hasta su fin, dicho de otra manera abarca la totalidad del desarrollo y a su vez también permite que el software sea mantenible y entendible en el tiempo.

## 1.4. Definiciones, acrónimos y abreviaturas

* Onion, hace referencia al nombre de la arquitectura que estaremos implementando
* WHALE, hace referencia al nombre del proyecto

# **2. Referencias**

Algunos documentos que estamos implementando para el orden del proyecto son:

* CRONOGRAMA DEL PROYECTO
* PRODUCT ROAD MAP
* PROJECT CHARTER

# **3. Arquitectura del sistema**

## 3.1. Arquitectura actual

Como este proyecto está iniciando, estaremos describiendo las arquitecturas que hemos analizado en el proceso de análisis del proyecto, algunas definiciones y características generales

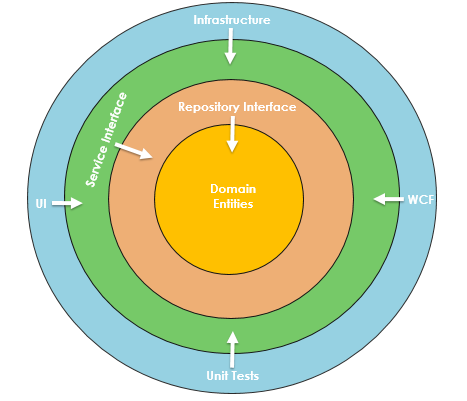
MVC: Significa modelo (model) vista (view) controlador (controller). Esto es lo que significan cada uno de esos componentes. Modelo: El backend que contiene toda la lógica de datos. Vista: El front end o interfaz gráfica de usuario (GUI). Controlador: El cerebro de la aplicación que controla como se muestran los datos.

Hexagonal: La arquitectura hexagonal es un estilo de [arquitectura de software](https://apiumhub.com/es/arquitectura-software-barcelona/) que mueve el foco de un programador desde un plano más conceptual hacia la distinción entre el interior y el exterior del software. La parte interior son los casos prácticos y el modelo domain está construido sobre ello. La parte exterior es UI, base de datos, mensajería, etc. La conexión entre el interior y el exterior es mediante puertos, y su implementación equivalente se conoce como adaptadores. Por esta razón, este estilo de arquitectura hexagonal se conoce habitualmente como Puertos y Adaptadores.

4 + 1: Las vistas suelen describir el sistema desde el punto de vista de diferentes interesados, tales como usuarios finales, desarrolladores o directores de proyecto. Las cuatro vistas del modelo son: vista lógica, vista de desarrollo, vista de proceso y vista física. Además, una selección de casos de uso o escenarios suele utilizarse para ilustrar la arquitectura sirviendo como una vista más. Por ello el modelo contiene 4+1 vistas

## 3.2. Arquitectura propuesta

Onion La Arquitectura de la Cebolla utiliza el concepto de capas, pero son diferentes de las capas de la arquitectura de 3 niveles y de n niveles. Veamos lo que representa y debe contener cada una de estas capas.



# **4. Descripción de descomposición**

## 4.1. Descomposición en módulos

Capa de dominio

En la parte central de la Arquitectura de la Cebolla, existe la capa de dominio; esta capa representa los objetos de negocio y de comportamiento. La idea es tener todos sus objetos de dominio en este núcleo. Contiene todos los objetos de dominio de la aplicación. Además de los objetos de dominio, también podrías tener interfaces de dominio. Estas entidades de dominio no tienen ninguna dependencia. Los objetos de dominio también son planos como deberían ser, sin ningún código pesado o dependencias.

Capa de repositorio

Esta capa crea una abstracción entre las entidades de dominio y la lógica de negocio de una aplicación. En esta capa, normalmente añadimos interfaces que proporcionan un comportamiento de guardado y recuperación de objetos, normalmente involucrando una base de datos. Esta capa consiste en el patrón de acceso a los datos, que es un enfoque más libremente acoplado al acceso a los datos. También creamos un repositorio genérico, y añadimos consultas para recuperar datos de la fuente, mapear los datos de la fuente de datos a una entidad de negocio, y persistir los cambios en la entidad de negocio a la fuente de datos.

Capa de servicios

La capa de servicios contiene interfaces con operaciones comunes, como añadir, guardar, editar y eliminar. Además, esta capa se utiliza para comunicarse entre la capa de interfaz de usuario y la capa de repositorio. La capa de servicios también puede contener la lógica de negocio de una entidad. En esta capa, las interfaces de servicio se mantienen separadas de su implementación, teniendo en cuenta el acoplamiento débil y la separación de preocupaciones.

Capa de interfaz de usuario

Es la capa más externa, y mantiene preocupaciones periféricas como la UI y las pruebas. Para una aplicación web, representa la API web o el proyecto de pruebas unitarias. Esta capa tiene una implementación del principio de inyección de dependencias para que la aplicación construya una estructura débilmente acoplada y pueda comunicarse con la capa interna a través de interfaces.