**SOLUCIONES LLM**

**VERSIÓN PERPLEXITY PRO**

**-Prompt 00:**

**Explicación:**

Este primer prompt establece una guía o roadmap detallado para desarrollar una aplicación básica de gestión de biblioteca usando arquitectura monolítica multicapa MVC y apoyándose en un LLM para asistir en el proceso. Define la planificación desde especificación hasta entrega, cubriendo gestión de libros, usuarios, préstamos, tecnologías (Java, Maven, MySQL) y buenas prácticas con GitHub, jUnit.

**Solución:**

Crear un documento o estructura de desarrollo que incluya los siguientes pasos:

Especificación detallada de requisitos.

Análisis mediante UML y modelado de actores y casos de uso.

Diseño arquitectónico con diagramas de clases organizados en paquetes MVC.

Configuración inicial de proyecto Maven con estructura correcta y empaquetado jar.

Integración progresiva de código con buenas prácticas, manejo de excepciones y documentación.

Definición de esquema y scripts SQL para base de datos.

Creación y ejecución de pruebas unitarias con jUnit.

Automatización de gestión de versiones con GitHub y documentación final.

Este Plan detalla fases y tecnologías clave, sirve como columna vertebral para todo el desarrollo y es punto de partida para los prompts siguientes.

**-Prompt 01:**

**Explicación:**

Se requiere una especificación formal de los requisitos software divididos en funcionales y no funcionales, con formato estructurado que incluya código identificativo, descripción, prioridad, tiempo estimado y dependencias. Funcionales cubren gestión libros, usuarios y préstamos incluyendo multas. No funcionales refieren a aspectos como rendimiento, seguridad y mantenibilidad.

**Solución:**

Ejemplo de tabla o listado con requisitos:

Código Tipo Descripción Prioridad Tiempo Est. Dependencias

RF.01 Funcional Alta libros en el sistema Alta 3 días Ninguna

RF.02 Funcional Actualizar datos de libros Alta 2 días RF.01

RF.03 Funcional Consultas variadas sobre libros Alta 3 días RF.01

RF.04 Funcional Baja de libros Media 2 días RF.01

RF.05 Funcional Alta de usuarios Alta 3 días Ninguna

RF.06 Funcional Actualizar usuarios Alta 2 días RF.05

RF.07 Funcional Baja de usuarios Media 2 días RF.05

RF.08 Funcional Consultas sobre usuarios Alta 3 días RF.05

RF.09 Funcional Alta préstamos Alta 3 días RF.01, RF.05

RF.10 Funcional Actualizar préstamos (renovaciones) Alta 2 días RF.09

RF.11 Funcional Baja préstamos Media 2 días RF.09

RF.12 Funcional Gestionar multas por retrasos Alta 3 días RF.11

RNF.01 No funcional Seguridad en acceso al sistema Alta 3 días Ninguna

RNF.02 No funcional Rendimiento adecuado para consultas simultáneas Alta 4 días Ninguna

RNF.03 No funcional Usabilidad correcta en interfaz Media 3 días Ninguna

RNF.04 No funcional Mantenibilidad y escalabilidad Alta 5 días Ninguna

Esta tabla debe incluirse en el documento con descripción clara y justificación.

**-Prompt 02:**

**Explicación:**

Se debe realizar un análisis del sistema con un diagrama general de casos de uso que muestre actores (usuario, bibliotecario, sistema), casos de uso (alta libro, consulta préstamo, etc.), además se identifican clases de análisis con estereotipos control, entidad, frontera y un ejemplo: diagrama de secuencia para un caso de uso.

**Solución:**

Crear un diagrama de casos de uso destacando:

Actores: Usuario, Bibliotecario, Sistema

Casos de uso: Alta libro, Baja libro, Consulta libro, Alta usuario, Renovación préstamo, etc.

Identificar clases de análisis:

Entidad: Libro, Usuario, Préstamo

Control: GestorPrestamos, GestorUsuarios

Frontera: InterfazAltaLibro, InterfazConsultaUsuarios

Diagrama de secuencia mostrando interacción para "Alta préstamo":

Usuario -> InterfazAltaPrestamo -> GestorPrestamos -> Base de Datos

Estos diagramas proveen una visión clara y sirven para transición al diseño.

**-Prompt 03:**

**Explicación:**

Diseño del sistema mediante un diagrama de clases que refleje la arquitectura MVC, con paquetes separados (modelo, vista, controlador) y detalle de atributos, métodos y relaciones.

**Solución:**

Diagramar clases principales:

Modelo: Libro, Usuario, Prestamo con atributos y getters/setters.

Vista: Clases para interfaces gráficas o consola.

Controlador: Clases que gestionan lógica de negocio.

Organizar en paquetes:

es.uclm.esi.iso2.bibliotecamonolitica.modelo

es.uclm.esi.iso2.bibliotecamonolitica.vista

es.uclm.esi.iso2.bibliotecamonolitica.controlador

Representar asociaciones, herencias (si existen) y multiplicidades.

Añadir interfaces y clases abstractas si aplican.

Este diseño sirve de blueprint para codificación.

**-Prompt 04:**

**Explicación:**

Creación inicial del proyecto con Maven usando archetype estándar, configurando groupID y artifactID, ajustando estructura y empaquetado.

**Solución:**

Ejecutar:

bash

mvn archetype:generate -DgroupId=es.uclm.esi.iso2 -DartifactId=bibliotecamonolitica -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

Organizar las clases generadas en paquetes correctos bajo src/main/java.

Configurar pom.xml para empaquetado jar:

xml

<packaging>jar</packaging>

Añadir manifiesto para ejecución en pom.xml o archivo MANIFEST.MF.

Esto crea la base técnica para empezar a programar.

**-Prompt 4.1 a 4.5:**

**Explicación:**

Mejoras relacionadas con:

Manifiesto para que el jar sea ejecutable.

Uso consistente de getters y setters para encapsulación.

Implementar excepción NotNullValueAllowedException en setters que no admitan cadena vacía.

Documentar con Javadoc todo el código.

Verificar que las rutas en pom.xml corresponden con los paquetes.

**Solución:**

Manifiesto: añadir a pom.xml plugin maven-jar-plugin con configuración para Main-Class.

Modificar todas las clases para usar getters/setters y validar atributos String no vacíos lanzando NotNullValueAllowedException.

Generar Javadocs con plugin maven-javadoc-plugin.

Revisar pom.xml para correcta referencia a paquetes en configuración de plugins o dependencias.

**-Prompt 05 y subprompts:**

**Explicación:**

Configurar dependencias en pom.xml para:

Conectar con MySQL (conector JDBC).

Gestionar pruebas con jUnit.

Generar Javadoc.

Además, implementar patrón Singleton para clase abstracta Agent y concreta AgentMySQL para conexión base de datos, clases DAO para cada entidad que usen AgentMySQL. Crear esquema SQL y script de inserción masiva.

**Solución:**

Añadir a pom.xml:

xml

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.x</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

Más el plugin de javadoc.

Clase abstracta Agent con patrón Singleton para manejar conexión.

AgentMySQL extiende Agent y gestiona operaciones SQL.

DAOs por entidad usan AgentMySQL para CRUD.

Crear script SQL creando tablas Libro, Usuario, Prestamo con sus campos y llaves, y script para insertar 100 usuarios y 100 libros.

**-Prompt 06:**

**Explicación:**

Diseñar y desarrollar pruebas unitarias con jUnit para cubrir al menos 90% del código fuente, asegurando confiabilidad.

**Solución:**

Para cada clase, crear test classes en src/test/java.

Implementar pruebas para métodos críticos: altas, bajas, actualizaciones y consultas.

Usar aserciones para validar resultados.

Ejecutar cobertura con plugins como JaCoCo para verificar meta de 90%.

**-Prompt 07 y 08:**

**Explicación**:

Generar código fuente completo, garantizar setters/getters, empaquetar proyecto Maven, implementar pruebas y configurarlas en pom.xml.

**Solución:**

Codificar todas las clases según diseño asegurando encapsulación.

Organizar estructura proyecto conforme estándares Maven.

Añadir dependencia jUnit y configurar pruebas para ejecución automática.

Empaquetar con mvn package.

**-Prompt 09:**

**Explicación:**

Instrucciones para crear proyecto en Visual Studio Code, conectarlo a repositorio GitHub y realizar commits.

**Solución:**

Crear proyecto importando estructura Maven.

Configurar extensión Git en VS Code.

Usar comandos Git para git init, git remote add origin, git add ., git commit -m "mensaje", git push.

Incluir pasos detallados en documento.

**-Prompt 10:**

**Explicación:**

Crear matriz de trazabilidad relacionando cada requisito funcional con implementaciones y calcular métrica de completitud según ISO/IEC 25010.

**Solución:**

Tabla cruzada requisitos vs casos/clases implementadas.

Marcar cumplimiento o falta.

Calcular porcentaje implementado: (requisitos cumplidos / totales) \* 100.

Añadir análisis en documento.

**-Prompt 11:**

**Explicación:**

Generar versión final del documento, empaquetar código fuente completo con pom.xml, README.md, manifiesto y documentación Javadoc en carpeta source y comprimir para entrega.

**Solución:**

Exportar documento consolidado.

Preparar estructura con carpetas src, documentación, scripts.

Generar Javadoc en carpeta source.

Comprimir todo en zip estándar.

Este es el contenido con la explicación detallada y la solución para cada prompt, siguiendo el formato pedido.