

Universidad Autónoma del Estado de México

Centro Universitario UAEM Zumpango



Ingeniería en Computación

Ingeniería de software II

Periodo 2025A

Sistema de Gestión de Préstamos de Bicicletas Comunitarias

presentan:

Morales Jasso Jonathan

Reyes Cabrera Angel

Vargas Rangel José Erick

Docente:

Reyes Luna Rosa Eréndira

Entregado en Zumpango, Estado de México a 22 de mayo 2025

Acuerdo del equipo

Roles

- Lider -- Vargas Rangel José Erick
- Frontend – Morales Jasso Jonathan
- Backend – Reyes Cabrera Ángel

Tareas

- Redacción de una breve descripción del sistema: ¿qué va a hacer? (5 líneas mín.). -- Reyes Cabrera Ángel (7 de mayo de 2025)
- Diagramas preliminares de la arquitectura de capas y clases usada. -- Vargas Rangel José Erick y Morales Jasso Jonathan (7 de mayo de 2025)
- Código fuente en .zip o GitHub.
 - Capa de Presentación (UI) -- Morales Jasso Jonathan (18 de mayo de 2025)
 - Clase AppBicicletas.
 - Menú para registrar bicicletas/usuarios, prestar, devolver y consultar.
 - Capa de Lógica de Negocio (Service) -- Reyes Cabrera Ángel (19 de mayo de 2025)
 - Clases BicicletaService, UsuarioService, PrestamoService.
 - Validaciones (bicicleta disponible, usuario sin préstamo activo).
 - Capa de Acceso a Datos (DAO) -- Vargas Rangel José Erick (20 de mayo de 2025)
 - Clases BicicletaDAO, UsuarioDAO, PrestamoDAO.
 - Manejan colecciones internas (ArrayList) con la información.
 - Modelo – Vargas Rangel José Erick (20 de mayo de 2025)
 - Clases Bicicleta, Usuario, préstamo con atributos y métodos.
- Registro de actividades y tiempos en Trello. -- Reyes Cabrera Ángel (7 de mayo de 2025)
<https://trello.com/invite/b/681b60e204e876af237920b6/ATTI0b2e078609d91173354db7f4734fecaf3028A0A7/my-trello-board>
- Documento PDF con capturas de funcionamiento. -- Reyes Cabrera Ángel (21 de mayo de 2025)
- Breve reflexión escrita (1 cuartilla): ¿por qué separar en capas? -- Vargas Rangel José Erick y Morales Jasso Jonathan (21 de Mayo de 2025)

Objetivo general

Implementar una aplicación Java de arquitectura monolítica utilizando **arquitectura de capas** (Presentación, Lógica de Negocio y Acceso a Datos) que permita gestionar el préstamo y devolución de bicicletas en una comunidad.

Contexto

Una colonia ha implementado un programa de **bicicletas comunitarias** para que los vecinos puedan trasladarse dentro de la zona sin usar vehículos contaminantes. Para organizar este servicio, se necesita un sistema que permita:

- Registrar nuevas bicicletas (modelo, ID, estado).
- Registrar usuarios (nombre, ID de usuario).
- Realizar préstamos (una bicicleta por usuario).
- Registrar devoluciones.
- Consultar bicicletas disponibles.
- Consultar historial de préstamos por usuario.

Se almacenará todo en memoria con ArrayList.

Descripción del sistema

El sistema permitirá la gestión de préstamos y devoluciones de bicicletas dentro de una comunidad para que puedan trasladarse dentro de la zona sin usar vehículos contaminantes. Los usuarios registrados podrán solicitar bicicletas disponibles y devolverlas una vez utilizadas. No podrán solicitar otra si tienen un préstamo activo. El sistema llevará el control de usuarios, bicicletas y los préstamos activos o históricos. Se incluirá una interfaz básica (en terminal) para registrar acciones y consultar información. La arquitectura estará dividida por capas para mantener la separación de responsabilidades.

Diagramas

Diagrama de capas

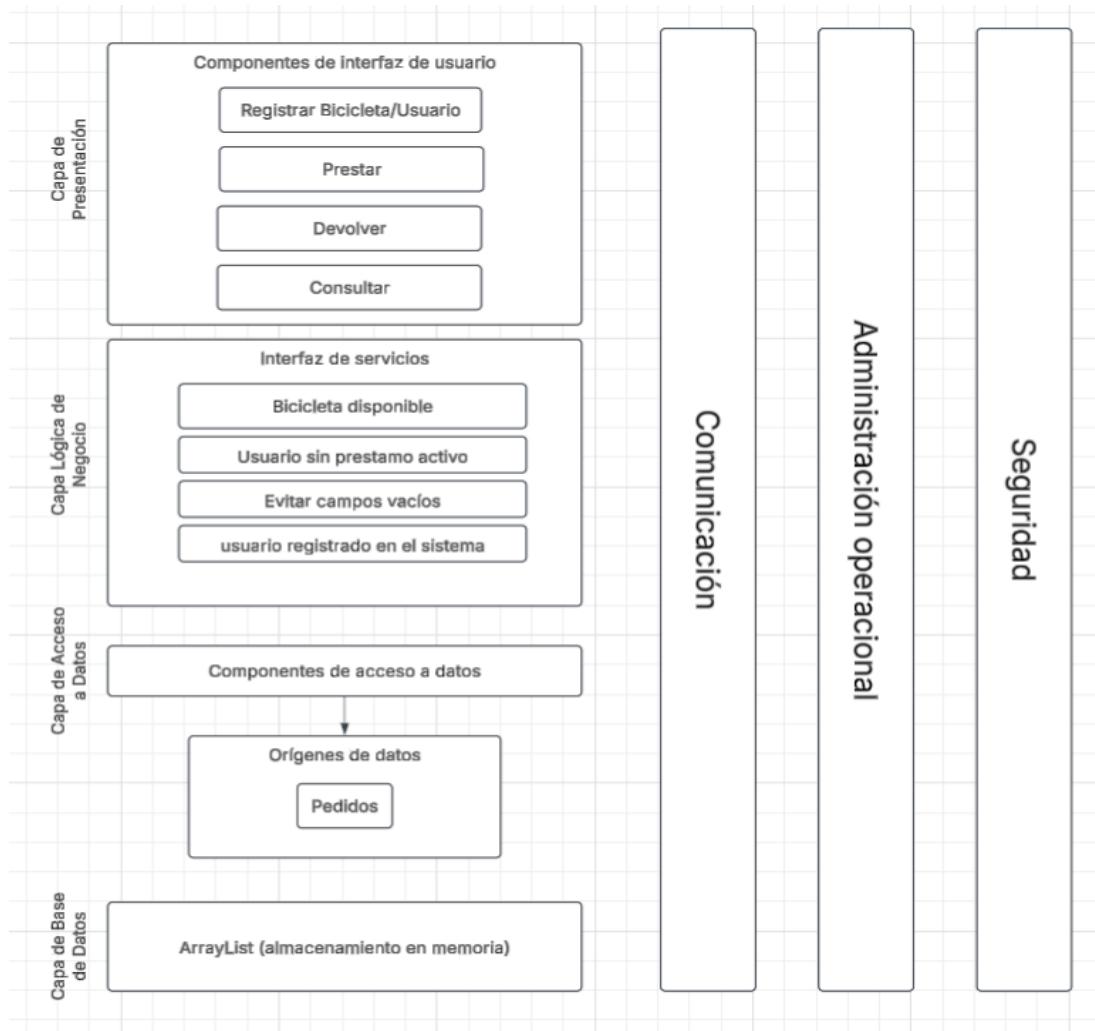
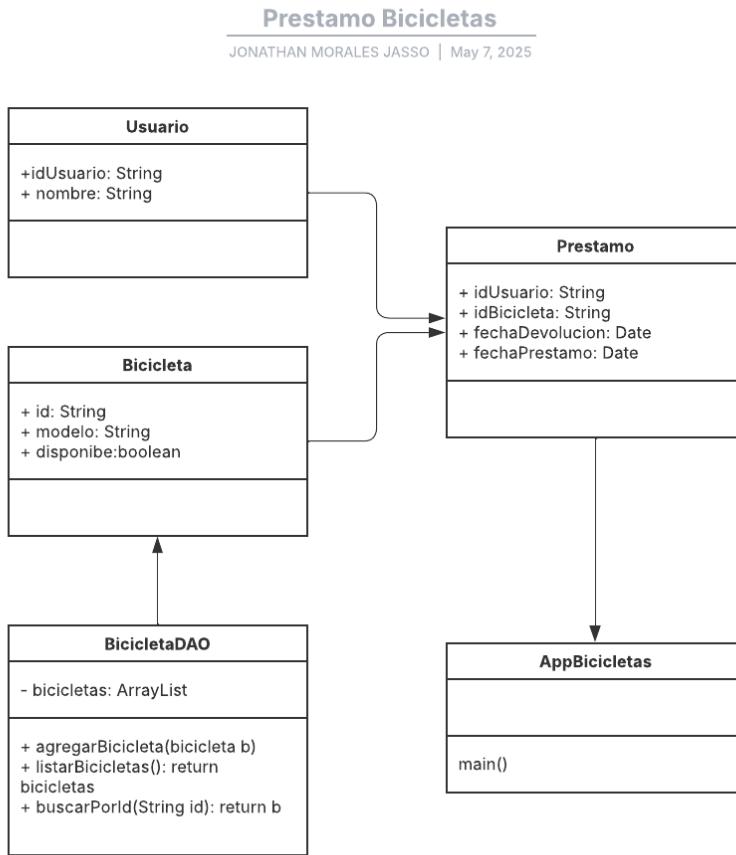


Diagrama de clases



Problemas detectados y dudas

Una duda detectada fue si hacer o no un login para tener control sobre los usuarios

Una duda fue si la relación de clases es correcta en el diagrama de clases

Un pequeño problema fue la asignación de roles y la distribución de trabajos a lo largo del tiempo disponible hasta el trabajo final.

Capturas del funcionamiento

Primera pantalla



Imagen 2. Pantalla inicial.

Ver bicicletas disponibles



Imagen 3. Visualización de las bicicletas disponibles.

Consultar bicicleta



Imagen 4. Consultar bicicleta con ID B001.

Consultar usuario



Imagen 5. Consultar usuario con ID U003.

Ver historial



Imagen 6. Ver historial de préstamos del usuario.

Registrar bicicleta



Imagen 7. Registrar bicicleta. Pide un ID bicicleta.



Imagen 7.1 Registrar bicicleta. Despues pide el modelo de la bicicleta.



Imagen 7.2 Registrar bicicleta. Muestra un mensaje de registro exitoso.



Imagen 7.3 Comprobación del registro al ver bicicletas disponibles.

Registrar usuario



Imagen 8. Registrar usuario. Pide ID del usuario.



Imagen 8.I Registrar usuario. Pide nombre del usuario.



Imagen 8.2 Registrar usuario. Muestra mensaje de registro exitoso.

Prestar



Imagen 9. Prestar bicicleta. Pide ID del usuario.

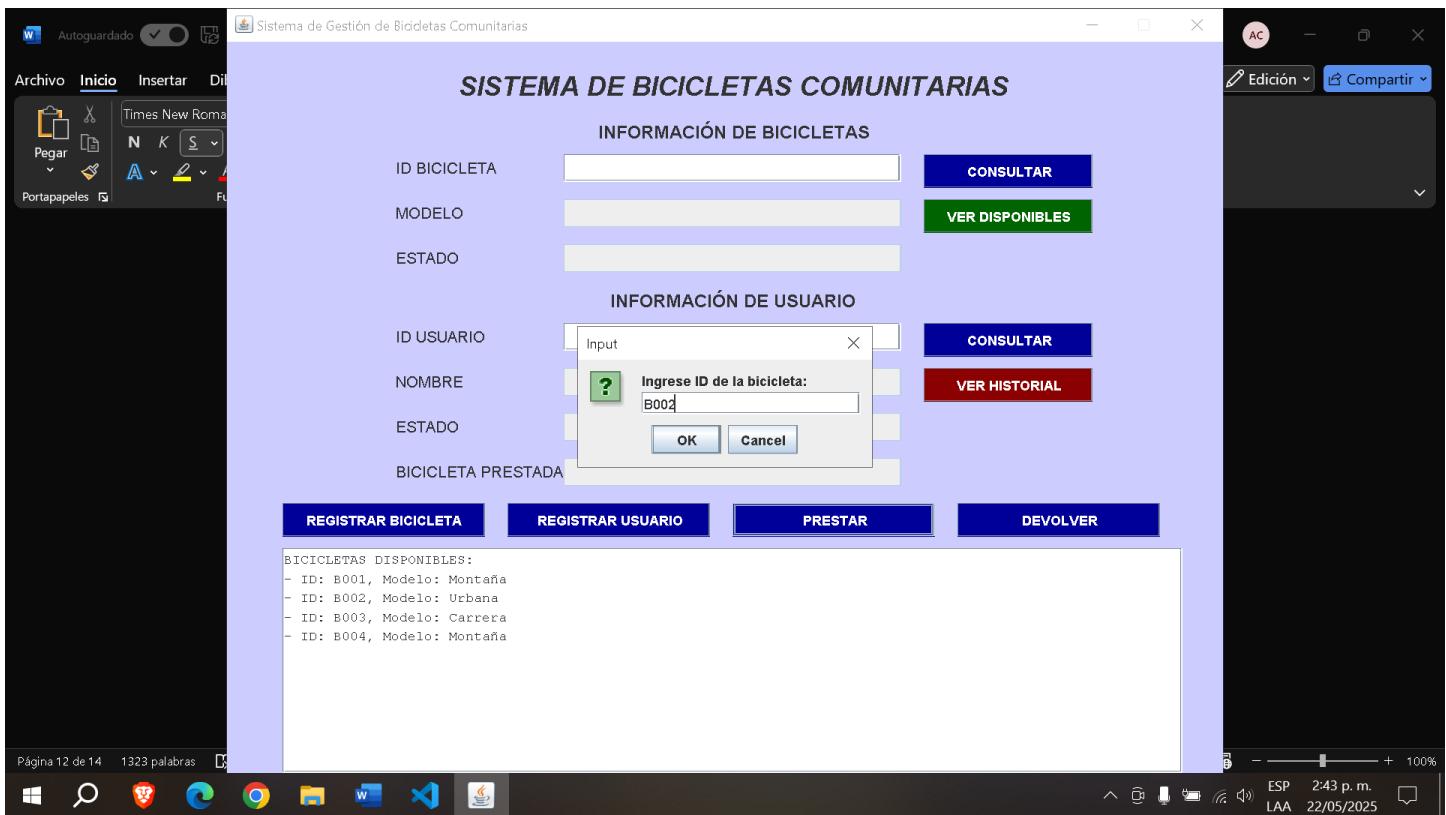


Imagen 9.1 Prestar bicicleta. Pide ID de la bicicleta.



Imagen 9.2 Prestar bicicleta. Mensaje de préstamo realizado. Da un ID de préstamo.



Imagen 9.3 Presta bicicleta. Comprobación de historial de préstamos con la acción “ver historial”.

Devolver

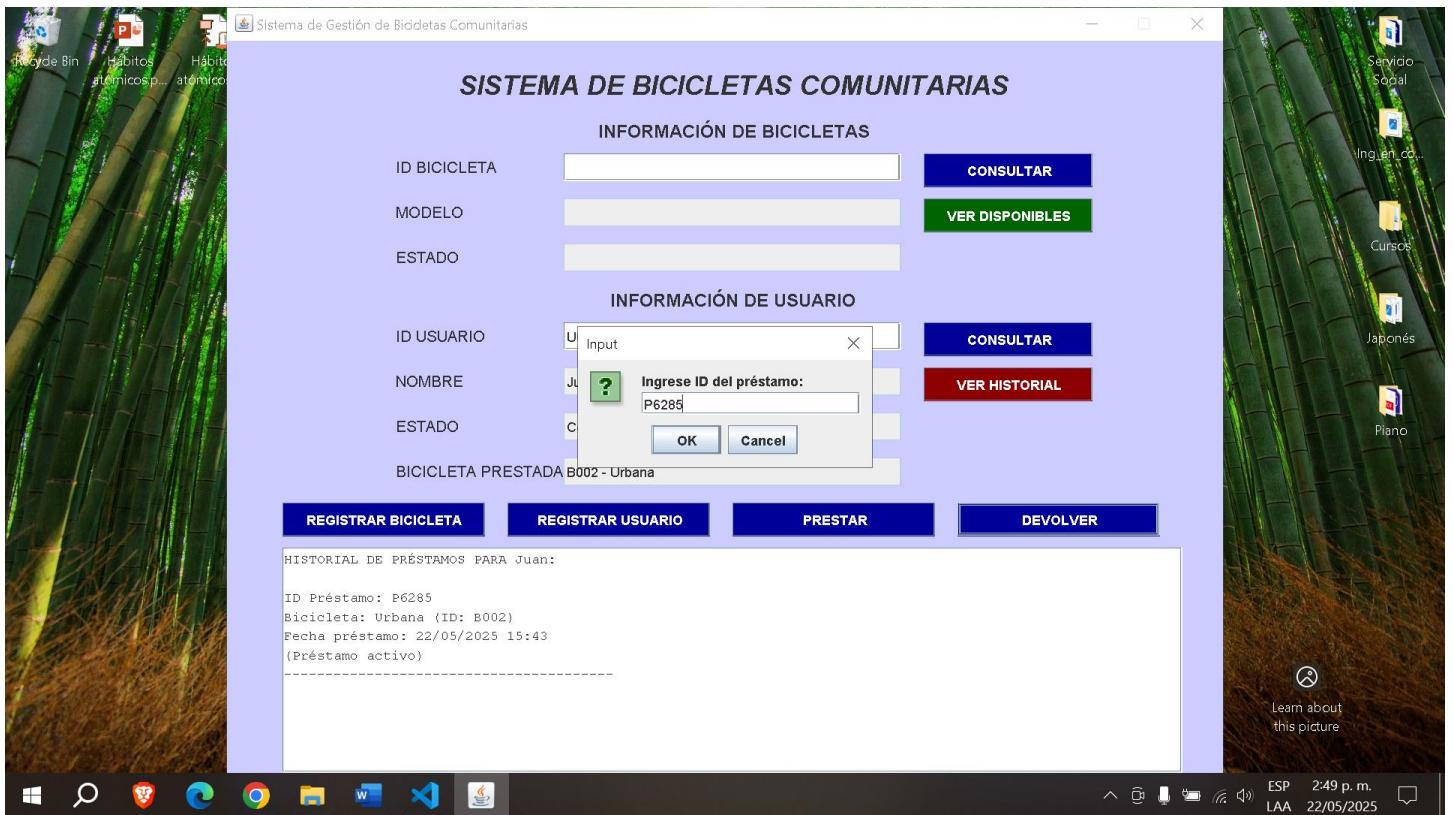


Imagen 10. Devolver bicicleta. Pide el ID de préstamo.



Imagen 10.1 Devolver bicicleta. Muestra mensaje de devolución exitoso.



Imagen 10.2 Comprobación de devolución al ver el historial de préstamos.



Imagen 10.3 Comprobación de bicicleta nuevamente disponible al ver bicicletas disponibles.

Reflexión

La separación en capas dentro del desarrollo de software es un principio fundamental que busca mejorar la organización del código, la claridad de las responsabilidades y la capacidad de mantener, escalar y reutilizar el sistema a lo largo del tiempo. En nuestro proyecto de préstamo de bicicletas comunitarias, esta estructura resulta clave para lograr una implementación ordenada, donde cada parte del sistema cumple una función clara y delimitada.

La arquitectura por capas divide la aplicación en distintos niveles o módulos, generalmente definidos como: Capa de Presentación (UI), Capa de Lógica de Negocio (Servicio o Service), Capa de Acceso a Datos (DAO) y Modelo de Datos (Modelo). Cada una de estas capas tiene un propósito específico y no debería depender directamente de detalles internos de las otras, salvo mediante interfaces o servicios bien definidos. Por ejemplo, la Capa de Presentación en nuestro proyecto está representada por la clase AppBicicletas, la cual ofrece un menú interactivo en consola para que el usuario pueda registrar bicicletas, registrar usuarios, solicitar préstamos o devoluciones, y consultar el estado del sistema. Esta capa se comunica con la capa de servicio para ejecutar acciones, pero no sabe cómo se almacenan los datos ni cuál es la lógica interna exacta para validar operaciones.

Esto permite que cualquier cambio en la lógica del negocio o en la forma de almacenar los datos no afecte directamente la interfaz del usuario. La Capa de Lógica de Negocio, compuesta por clases como BicicletaService, UsuarioService y PrestamoService, es la encargada de implementar todas las reglas que rigen el funcionamiento del sistema. Aquí es donde se determina si un usuario puede tomar prestada una bicicleta, si ya tiene un préstamo activo o si la bicicleta que quiere devolver está efectivamente prestada. Estas clases actúan como intermediarias entre la UI y la capa de datos, asegurando que todas las operaciones cumplan con las reglas definidas. Gracias a esta capa, podemos tener control total del comportamiento del sistema y asegurar que no se hagan operaciones inválidas. Por su parte, la Capa de Acceso a Datos (DAO) se encarga de simular la persistencia de los datos.

En este caso, usamos estructuras de datos como ArrayList para guardar objetos de tipo Usuario, Bicicleta o Prestamo. Las clases DAO como BicicletaDAO, UsuarioDAO y PrestamoDAO se encargan de agregar, buscar, listar y modificar los datos sin que el resto del sistema necesite saber cómo están organizados. Este aislamiento facilita que en el futuro podamos reemplazar la lógica de almacenamiento en memoria por una base de datos real, sin necesidad de modificar la lógica del negocio ni la interfaz del usuario.

El Modelo de Datos representa las entidades principales del sistema. En nuestro caso, son Usuario, Bicicleta y Prestamo. Cada una tiene atributos y métodos get y set que permiten acceder y modificar su información. Estas clases son compartidas entre las distintas capas, pero no contienen lógica compleja: solo representan datos. Esto hace que el modelo sea reutilizable y coherente a lo largo de toda la aplicación.

Una de las ventajas más evidentes de la separación en capas es que permite trabajar de forma colaborativa sin interferencias. Por ejemplo, mientras un integrante del equipo diseña la interfaz, otro puede estar desarrollando los servicios o escribiendo el código para acceder a los datos. Esto mejora la productividad y reduce conflictos, ya que cada integrante puede enfocarse en una responsabilidad distinta sin tener que entender todo el sistema de una vez.

Otra ventaja importante es la modularidad. Si más adelante se desea modificar una parte específica del sistema (como añadir un nuevo criterio de búsqueda en los DAO o implementar nuevas validaciones en los servicios), es posible hacerlo sin reescribir todo el proyecto. Esta capacidad de aislar los cambios facilita el mantenimiento del software y reduce el riesgo de introducir errores.

También es importante considerar que la arquitectura en capas promueve el uso de buenas prácticas de programación orientada a objetos, como la encapsulación y la abstracción. Cada clase tiene una función clara, los datos están protegidos mediante atributos privados y accesores, y las responsabilidades están distribuidas de manera lógica. Esto no solo mejora la calidad del código, sino que también lo hace más fácil de leer, probar y depurar.

En el ámbito educativo y de desarrollo académico, este enfoque también permite que los estudiantes y desarrolladores en formación entiendan mejor cómo funciona una aplicación completa, diferenciando entre los datos, la lógica y la interacción con el usuario. Facilita además la realización de pruebas unitarias en cada capa por separado, lo cual es esencial para garantizar la calidad del software.