

Evidencia de Aprendizaje – desarrollar software a partir de la integración de sus módulos componentes

GA8-220501096-AA1-EV01

Fase – 3 Ejecución – Área técnica

Por:

Santiago Arango Rodriguez

Centro de la Tecnología del Diseño y la Productividad Empresarial

Regional Cundinamarca – SENA - Girardot

Análisis y Desarrollo de Software

Ficha 2977481

Instructor: Milton Ivan Barbosa

30 de septiembre del 2025

Contenido

1. Introducción	6
2. Justificación	7
3. Objetivos.....	8
3.1 Objetivo general.....	8
3.2 Objetivos específicos	8
4.1 Requisitos funcionales	9
4.1.2 Requerimientos no funcionales.....	11
4.2 Diagrama de casos de uso	13
4.3 Historias de usuario.....	14
4.3.1 HU1 – Registro de usuario	14
4.3.2 HU2 – Inicio de sesión	14
4.3.3 HU3 – Cierre de sesión	14
4.3.4 HU4 – Visualización de estaciones	15
4.3.5 HU5 – Reserva de estación	15
4.3.6 HU6 – Iniciar uso de estación	15
4.3.7 HU7 – Finalizar uso y cobro	16
4.3.8 HU8 – Consulta de saldo	16
4.3.9 HU9 – Recargar saldo	16
4.3.10 HU10 – Gestión de productos y servicios	17
4.3.11 HU11 – Compra de productos y servicios	17
4.3.12 HU12 – Gestión de usuarios	17
4.3.13 HU13 – Reportes básicos	18
4.3.14 HU14 – Administración de estaciones.....	18
4.4 Herramienta de Desarrollo: IntelliJ IDEA.....	19
4.5 Diagrama de clases	21
4.6 Diagrama de paquetes.....	22
4.7 Diagrama de componentes	23
4.8 Mecanismos de seguridad	24

4.9 Descripción de la metodología Scrum	25
4.9.1 Grafico Fases de Scrum.....	25
4.9.2 Capa en la que estamos actualmente.....	26
4.10 Mapa de Navegación	27
4.11 Librerias y dependencias del proyecto	28
4.11.1 Backend (Spring Boot – Java).....	28
4.11.2 Frontend (integrado con backend)	28
4.11.3 Ejemplos de imports:.....	29
4.12 Frameworks del proyecto	30
4.13 Reutilización de código en el proyecto	31
4.13.1 Servicios (Service).....	31
4.13.2 Repositorios (Repository).....	32
4.13.3 Entidades (Entities)	33
4.13.4 Plantillas Thymeleaf	34
4.14 Buenas practicas	35
4.14.1 //GET mostrar perfil	35
4.14.2 // POST actualizar perfil	36
4.14.3 Repositorio	36
4.15.1 Paquete principal: com.GN.Gaming.Network	38
4.15.2 config	38
4.15.3 controller.....	38
4.15.4 model.....	38
4.15.5 repository	38
4.15.6 service	38
4.15.7 GamingNetworkApplication	39
4.15.8 resources	39
5. Conclusiones.....	49

Imagenes

Imagen 1 ej import 1	29
Imagen 2 ej import 2	29
Imagen 3 ej import 3	29
Imagen 4 testeo registro.....	42
Imagen 5 testeo inicio.....	43
Imagen 6 testeo dashboard.....	44
Imagen 7 testeo perfil	45
Imagen 8 testeo usuarios	46
Imagen 9 tabla estaciones	47
Imagen 10 tabla usuarios.....	47
Imagen 11 tabla transacciones.....	48

Ilustraciones

Ilustración 1 Diagrama casos de uso	13
Ilustración 2 Diagrama de clases	21
Ilustración 3 Diagrama de paquetes.....	22
Ilustración 4 Diagrama de componentes.....	23
Ilustración 5 Grafico fases de Scrum.....	25
Ilustración 6 Mapa de navegación.....	27

Tablas

Tabla 1 Requerimientos funcionales	9
Tabla 2 Hu Registro.....	14
Tabla 3 Hu inicio	14
Tabla 4 Hu cierre	14
Tabla 5 Hu visualizacion.....	15
Tabla 6 Hu reserva	15
Tabla 7 Hu inicio estacion	15
Tabla 8 Hu finalizar uso	16
Tabla 9 Hu consulta.....	16
Tabla 10 Hu recargar saldo	16
Tabla 11 Gestión productos y servicios	17
Tabla 12 Hu compra productos y servicios	17
Tabla 13 Hu gestión usuarios.....	17

Tabla 14 Hu reportes.....	18
Tabla 15 Hu administrar estaciones	18
Tabla 16 Mecanismos de seguridad	24
Tabla 17 Frameworks	30

1. Introducción

En el desarrollo de este proyecto hemos trabajado de manera organizada y progresiva en la construcción de una solución de software enfocada en la gestión integral de una sala de internet gaming. Desde el inicio, se definieron los requisitos principales y se fue consolidando una visión clara del sistema, siempre con la intención de que sea funcional, escalable y fácil de mantener.

Durante el proceso, hemos procurado aplicar buenas prácticas de programación, lo que incluye el uso de comentarios claros, organización del código y patrones de diseño que favorecen la claridad y el mantenimiento. También se ha fomentado la reutilización de código, evitando duplicaciones innecesarias y promoviendo componentes modulares que puedan ser usados en diferentes partes del sistema.

De igual manera, se han considerado aspectos técnicos fundamentales como la realización de pruebas unitarias para validar que cada módulo funcione de manera correcta y segura, así como la correcta configuración de bases de datos, asegurando la integridad de la información y la eficiencia en las operaciones que el sistema debe realizar.

En cuanto a las herramientas, hemos trabajado con IntelliJ IDEA Ultimate, aprovechando su potencia para el backend y dejando abierta la posibilidad de avanzar en el frontend de forma independiente conforme aumenten las habilidades técnicas. Todo esto se complementa con el uso de diagramas UML, la definición de requerimientos funcionales y no funcionales, y la aplicación de Scrum como metodología de desarrollo ágil que permite avanzar de manera iterativa y adaptativa.

En general, este documento recoge la base conceptual y técnica que sustenta el desarrollo del sistema, mostrando cómo cada decisión desde la elección de herramientas

hasta la aplicación de pruebas y prácticas de calidad, está orientada a construir un software confiable, eficiente y preparado para crecer con el paso del tiempo.

2. Justificación

El desarrollo de este sistema surge como respuesta a la necesidad de optimizar la gestión de una sala de internet gaming, donde se requiere un control claro de usuarios, estaciones, tiempos de uso, cobros, reservas y servicios adicionales. Actualmente, muchas de estas tareas se realizan de manera manual o con herramientas poco adaptadas al contexto, lo que genera ineficiencias, pérdida de información y dificultad en la administración.

Con la implementación de esta solución, se busca centralizar y automatizar los procesos clave del negocio, ofreciendo un sistema intuitivo tanto para el administrador como para los clientes. Esto no solo permitirá una mejor organización interna y reducción de errores, sino también una experiencia más ágil y transparente para los usuarios. Además, la propuesta está diseñada para ser escalable, de manera que pueda crecer junto con las necesidades futuras del establecimiento.

3. Objetivos

3.1 Objetivo general

- Desarrollar un sistema de software que permita la gestión integral de una sala de internet gaming, automatizando procesos de control de usuarios, estaciones, tiempos, cobros, reservas y servicios, con el fin de optimizar la administración y mejorar la experiencia de los clientes.

3.2 Objetivos específicos

- Implementar un módulo de autenticación y control de acceso seguro, que diferencie los roles de administrador y cliente para garantizar un uso adecuado del sistema.
- Desarrollar funcionalidades que gestionen estaciones, tiempos de uso, cobros y saldos, asegurando un control eficiente y transparente de los recursos.
- Integrar herramientas de reportes básicos y manejo de datos que faciliten la toma de decisiones y el seguimiento de las operaciones diarias.

4. Desarrollo de la solución de software

4.1 Requisitos funcionales

Tabla 1 Requerimientos funcionales

ID	Nombre	Descripción
RF1	Autenticación y Control de Acceso	- Pantalla de login con usuario/contraseña.- Roles: Administrador / Cliente.- Cierre de sesión manual.- Validación de formularios y mensajes de error claros.
RF2	Panel de Estaciones	- Vista en cuadrícula de todas las estaciones (PCs).- Cada estación es un cuadro coloreado según estado: En uso, Disponible, En mantenimiento.- Al hacer clic: ver detalles (usuario actual, tiempo restante).
RF3	Manejo de Tiempo y Cobros	- Iniciar/detener el tiempo de uso en una estación.- Cálculo automático del costo según tarifa y tiempo consumido.- Registro de cobro en historial de transacciones.
RF4	Manejo de Saldos	- Cada usuario posee un saldo interno en el sistema.- El administrador puede recargar saldo manualmente (ej. efectivo/datafono en local).- El sistema descuenta automáticamente saldo al usar estaciones o comprar productos.-

		Visualización del saldo disponible por parte del usuario.
RF5	Reservas de Estaciones	- Crear reservas indicando fecha, hora y duración.- Validación de disponibilidad antes de confirmar.- Notificación al administrador de reservas activas.
RF6	Tienda y Servicios	- Registro de productos (comestibles, bebidas) y servicios (impresiones, escaneos).- Compras de productos/servicios usando el saldo.- Historial de ventas con detalle de fecha, producto/servicio y costo.
RF7	Reportes Básicos	- Reporte diario de ingresos totales.- Reporte de uso de estaciones.- Exportación simple en tabla (no gráficas avanzadas).
RF8	Gestión de Usuarios	- CRUD básico de clientes y administradores (nombre, correo, cédula, fecha registro).- Estado del usuario (activo/inactivo).- Consulta de historial de sesiones y transacciones.

Nota: Tabla hecha por Santiago Arango Rodriguez

4.1.2 Requerimientos no funcionales

1. Rendimiento y eficiencia

- **RNF1.1:** El sistema debe permitir la autenticación de un usuario en un tiempo máximo de 3 segundos.
- **RNF1.2:** El sistema debe poder manejar simultáneamente al menos 50 usuarios conectados sin afectar el rendimiento.
- **RNF1.3:** Las consultas de reportes deben generarse en un tiempo no mayor a 5 segundos.

2. Usabilidad

- **RNF2.1:** La interfaz debe ser intuitiva y comprensible para usuarios con conocimientos básicos en informática.
- **RNF2.2:** El sistema debe contar con indicadores visuales de estado (colores en estaciones: rojo, verde, gris).
- **RNF2.3:** La curva de aprendizaje de los operadores no debe superar 2 horas de capacitación.

3. Seguridad

- **RNF3.1:** Las contraseñas de los usuarios deben almacenarse de forma encriptada.
- **RNF3.2:** El acceso a la sección administrativa debe estar protegido por autenticación de credenciales válidas.
- **RNF3.3:** Las operaciones críticas (como recargas de saldo y tarifas) deben requerir confirmación adicional del operador.

4. Fiabilidad

- **RNF4.1:** El sistema debe garantizar la integridad de los datos en caso de fallas inesperadas, mediante transacciones atómicas.
- **RNF4.2:** El sistema debe tener una disponibilidad mínima del 95 % durante el horario de funcionamiento.

5. Mantenibilidad y escalabilidad

- **RNF5.1:** El sistema debe estar diseñado con una arquitectura en capas para facilitar modificaciones y mantenimiento.
- **RNF5.2:** El sistema debe permitir la incorporación de nuevas funcionalidades (ej. pagos en línea) en versiones futuras sin afectar lo existente.
- **RNF5.3:** El código debe estar documentado con comentarios claros en las secciones principales.

6. Portabilidad

- **RNF6.1:** El sistema debe funcionar en navegadores web modernos (Google Chrome, Edge, Firefox).
- **RNF6.2:** El sistema debe ser compatible con el sistema operativo Windows 10 o superior en los equipos de las estaciones.

7. Trazabilidad y auditoría

- **RNF7.1:** El sistema debe permitir registrar en bitácoras (logs) los accesos y transacciones realizadas por cada usuario.
- **RNF7.2:** Los registros históricos deben mantenerse al menos por 6 meses para consulta administrativa.

4.2 Diagrama de casos de uso

Ilustración 1 Diagrama casos de uso



Nota: Diagrama hecho por Santiago Aarngó Rodríguez en draw.io

4.3 Historias de usuario

4.3.1 HU1 – Registro de usuario

Tabla 2 Hu Registro

Historia	Como usuario, quiero registrarme en la plataforma para crear mi cuenta y acceder a los servicios.
Criterios de aceptación	- Debo poder ingresar mis datos (nombre, correo, contraseña).- El sistema debe validar que el correo no esté registrado.- Al registrarme correctamente, debo ver un mensaje de confirmación.

4.3.2 HU2 – Inicio de sesión

Tabla 3 Hu inicio

Historia	Como usuario o administrador, quiero iniciar sesión en la plataforma para acceder a mis funciones según mi rol.
Criterios de aceptación	- Debo ingresar correo y contraseña válidos. - Si los datos son correctos, debo ser redirigido a mi dashboard. - Si son incorrectos, debo recibir un mensaje de error claro.

4.3.3 HU3 – Cierre de sesión

Tabla 4 Hu cierre

Historia	Como usuario o administrador, quiero cerrar sesión para finalizar mi uso del sistema de forma segura.
Criterios de aceptación	- Debo poder cerrar sesión manualmente.- La sesión debe cerrarse automáticamente después de un tiempo de inactividad configurado.

4.3.4 HU4 – Visualización de estaciones

Tabla 5 Hu visualizacion

Historia	Como usuario o administrador, quiero ver un panel con las estaciones y sus estados para saber cuáles están disponibles, en uso o en mantenimiento.
Criterios de aceptación	- Las estaciones deben mostrarse en una cuadrícula.- Cada estación debe tener un color según estado (verde, rojo, gris).- Al hacer clic en una estación, debo ver sus detalles.

4.3.5 HU5 – Reserva de estación

Tabla 6 Hu reserva

Historia	Como usuario, quiero reservar una estación en una fecha y hora específicas para asegurar mi lugar antes de llegar al local.
Criterios de aceptación	- Debo seleccionar fecha, hora y duración de la reserva.- El sistema debe verificar la disponibilidad antes de confirmar.- Debo recibir un mensaje de confirmación al completar la reserva.

4.3.6 HU6 – Iniciar uso de estación

Tabla 7 Hu inicio estacion

Historia	Como usuario, quiero iniciar sesión en una estación para empezar a usar el servicio y contar mi tiempo.
Criterios de aceptación	- Debo poder iniciar uso si la estación está disponible o tengo reserva.- El sistema debe iniciar un contador de tiempo.- Si no tengo saldo suficiente, debo recibir un aviso y no iniciar.

4.3.7 HU7 – Finalizar uso y cobro

Tabla 8 Hu finalizar uso

Historia	Como usuario, quiero finalizar el uso de una estación para que el sistema calcule el costo y descuento mi saldo.
Criterios de aceptación	- El sistema debe calcular el tiempo total y aplicar la tarifa correspondiente.- Debe descontar el saldo automáticamente o registrar pago manual por admin.- La estación debe volver al estado “disponible”.

4.3.8 HU8 – Consulta de saldo

Tabla 9 Hu consulta

Historia	Como usuario, quiero consultar mi saldo disponible para saber cuánto dinero tengo en mi cuenta.
Criterios de aceptación	- Debo ver mi saldo actual en la plataforma.- Debo ver un historial de recargas y transacciones.

4.3.9 HU9 – Recargar saldo

Tabla 10 Hu recargar saldo

Historia	Como administrador, quiero registrar recargas de saldo de los clientes para que puedan pagar sus sesiones y compras.
Criterios de aceptación	- Debo poder seleccionar usuario y monto a recargar.- El sistema debe actualizar el saldo inmediatamente.- Debe registrarse la transacción en el historial.

4.3.10 HU10 – Gestión de productos y servicios

Tabla 11 Gestión de productos y servicios

Historia	Como administrador, quiero gestionar productos y servicios (CRUD) para mantener actualizado el catálogo de la tienda.
Criterios de aceptación	- Debo poder agregar, editar y eliminar productos/servicios.- Los cambios deben reflejarse en la tienda de los usuarios.- Debo ver un listado con todos los ítems disponibles.

4.3.11 HU11 – Compra de productos y servicios

Tabla 12 Hu compra de productos y servicios

Historia	Como usuario, quiero comprar productos y servicios usando mi saldo para consumirlos en el local.
Criterios de aceptación	- Debo poder seleccionar productos y confirmar compra.- El sistema debe validar stock y saldo suficiente.- El sistema debe registrar la compra en el historial.

4.3.12 HU12 – Gestión de usuarios

Tabla 13 Hu gestión de usuarios

Historia	Como administrador, quiero gestionar las cuentas de los usuarios para mantener actualizada la información y controlar el acceso.
Criterios de aceptación	- Debo poder crear, editar y desactivar usuarios.- El sistema debe guardar cambios y registrar acciones del admin.- No debe permitirse duplicar correos.

4.3.13 HU13 – Reportes básicos

Tabla 14 Hu reportes

Historia	Como administrador, quiero generar reportes de ingresos y uso para analizar el rendimiento del negocio.
Criterios de aceptación	- Debo poder elegir rango de fechas.- El sistema debe mostrar los datos en tablas con totales.- Debo poder exportar los reportes en CSV.

4.3.14 HU14 – Administración de estaciones

Tabla 15 Hu administrar estaciones

Historia	Como administrador, quiero marcar estaciones en mantenimiento o forzar cierre de sesiones para gestionar incidencias y disponibilidad.
Criterios de aceptación	- Debo poder poner una estación en mantenimiento.- Debo poder forzar el cierre de una estación en uso.- El sistema debe actualizar el estado y registrar la acción.

4.4 Herramienta de Desarrollo: IntelliJ IDEA

En el desarrollo de *Gaming Network* hemos elegido IntelliJ IDEA Ultimate como nuestro entorno de trabajo principal. La decisión se tomó porque este IDE ofrece un soporte muy completo para Java y Spring Boot, que son la base del backend de nuestro sistema.

Gracias al correo institucional, contamos con la licencia de la versión Ultimate, lo que nos permite acceder a todas las características avanzadas sin costo adicional. Esto ha sido una ventaja importante, ya que facilita no solo el desarrollo del backend, sino también la posibilidad de trabajar con tecnologías de frontend, bases de datos y herramientas web en el futuro.

Lo que más valoramos de IntelliJ es la productividad que nos brinda. Su autocompletado inteligente, la refactorización automática y las sugerencias en tiempo real hacen que el trabajo sea más rápido y con menos errores. Además, la integración con Spring Boot permite que podamos ejecutar, probar y depurar el proyecto directamente desde el IDE, sin pasos complicados.

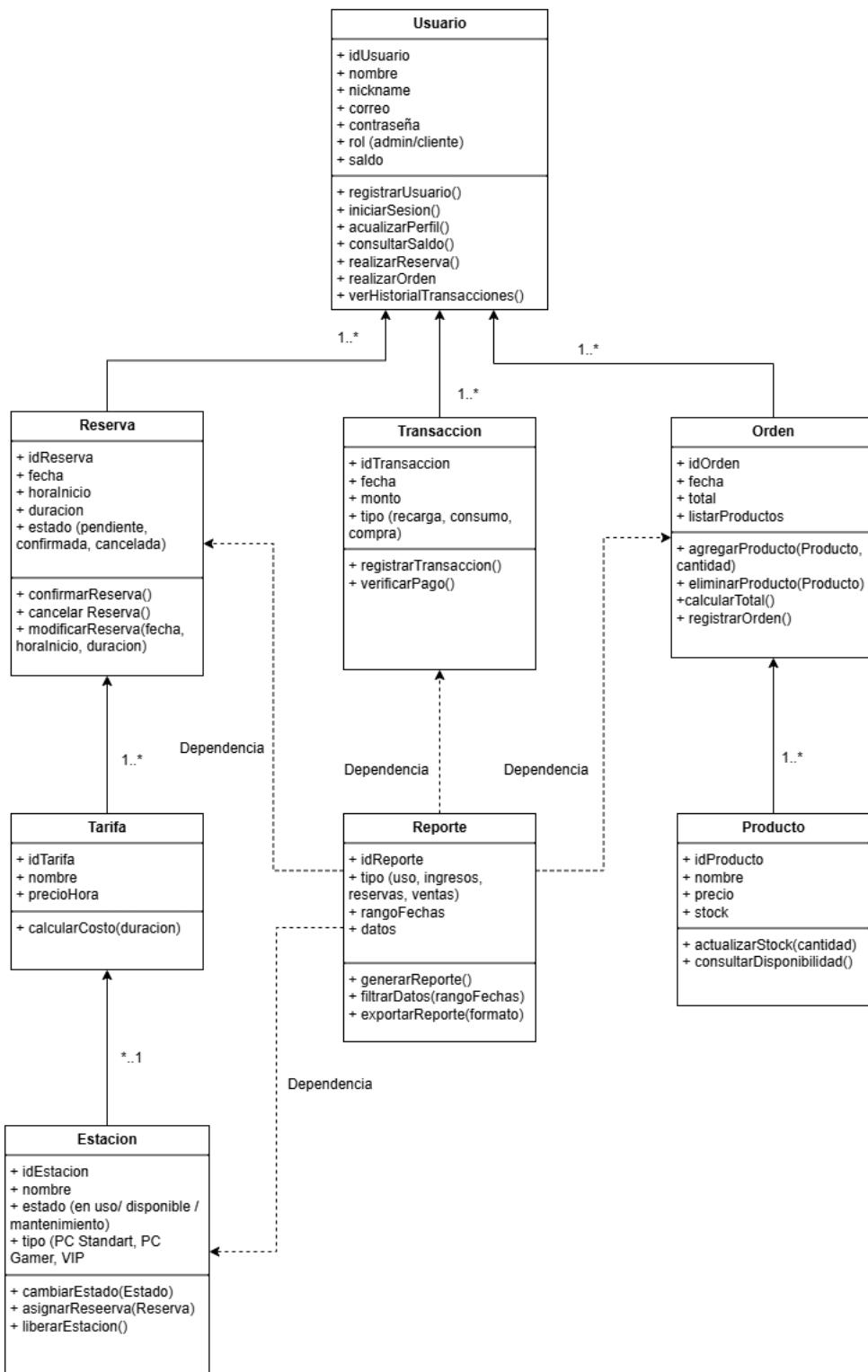
Otra gran ventaja es que incluye herramientas integradas para gestionar bases de datos, controlar versiones con Git y hasta crear consultas SQL sin salir del entorno. Esto reduce la necesidad de estar saltando entre programas, lo que nos ahorra tiempo y mantiene todo centralizado.

Finalmente, el hecho de que IntelliJ IDEA Ultimate tenga soporte para múltiples lenguajes y frameworks nos da la seguridad de que, cuando avancemos en la parte de frontend, no tendremos que cambiar de entorno: el mismo IDE nos servirá para manejar tanto el backend como el frontend.

En resumen, IntelliJ IDEA Ultimate es más que un editor de código; es una plataforma integral de desarrollo que se adapta perfectamente a las necesidades actuales de *Gaming Network* y también a su crecimiento futuro.

4.5 Diagrama de clases

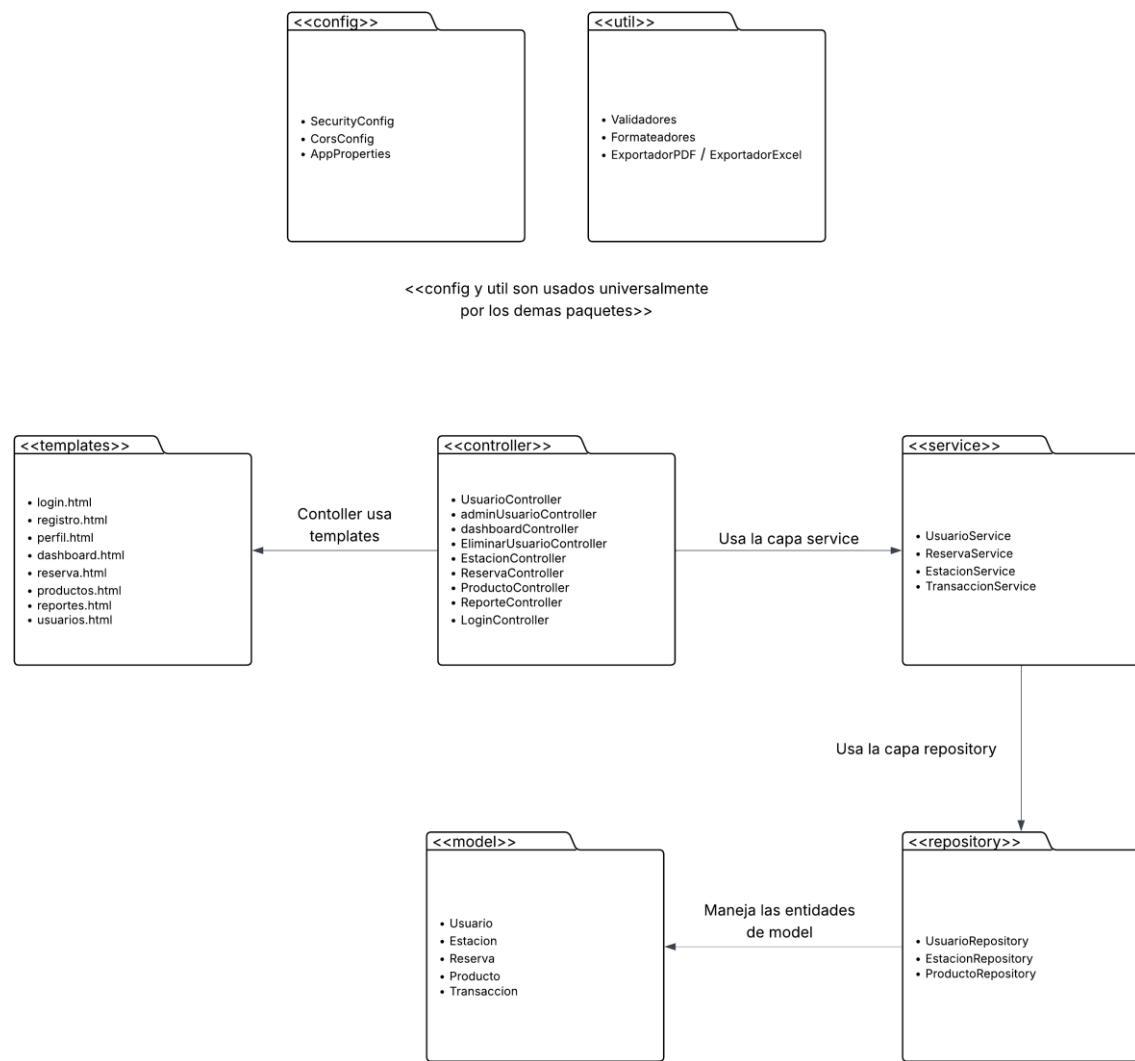
Ilustración 2 Diagrama de clases



Nota: Diagrama hecho por Santiago Arango Rodriguez en Draw.io

4.6 Diagrama de paquetes

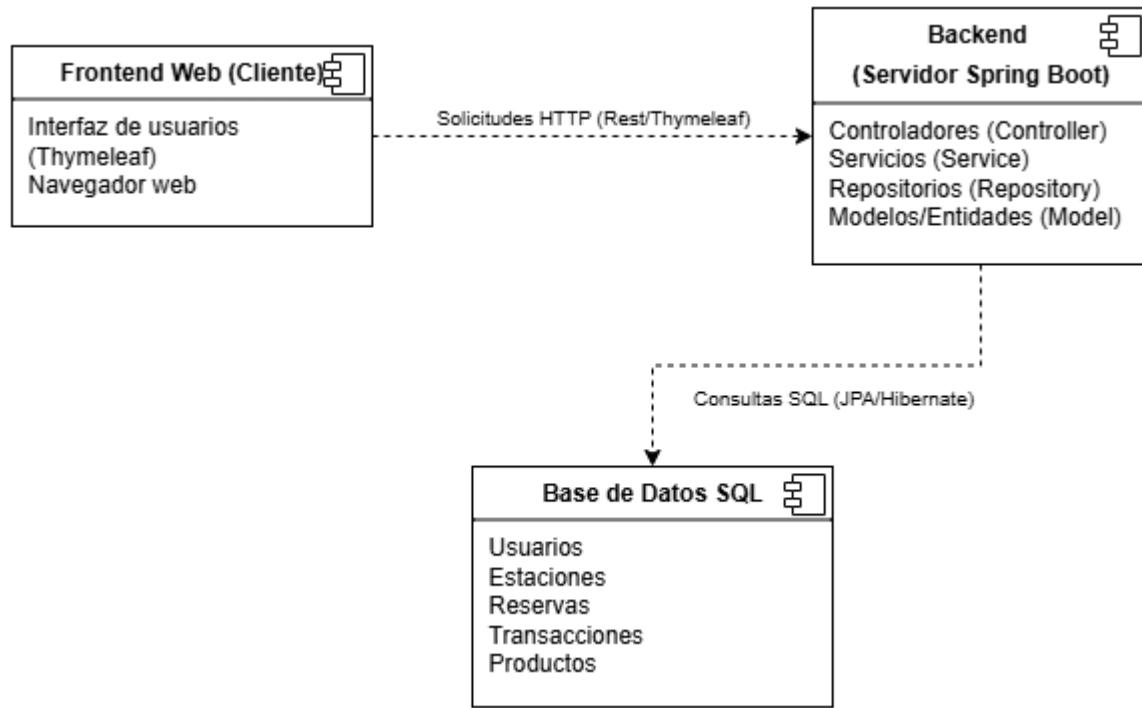
Ilustración 3 Diagrama de paquetes



Nota: Diagrama hecho por Santiago Arango Rodriguez en Draw.io

4.7 Diagrama de componentes

Ilustración 4 Diagrama de componentes



Nota: Diagrama hecho por Santiago Arango Rodriguez en Draw.io

4.8 Mecanismos de seguridad

Tabla 16 Mecanismos de seguridad

Mecanismo	Descripción	Propósito
Autenticación de usuarios	Los usuarios deben iniciar sesión con su correo y contraseña antes de acceder al sistema.	Evitar que personas no autorizadas entren a la plataforma.
Gestión de roles y permisos	No todos ven lo mismo: el administrador tiene más funciones que un cliente común.	Asegurar que cada usuario solo pueda hacer lo que le corresponde.
Cifrado de contraseñas	Las contraseñas no se guardan en texto plano, se almacenan con un algoritmo seguro (por ejemplo, BCrypt).	Proteger las credenciales en caso de fuga de datos.
Sesiones seguras	Cuando un usuario inicia sesión, se crea una sesión controlada, con tiempo de expiración automático.	Evitar accesos indebidos si alguien deja la cuenta abierta.
Validación de datos de entrada	El sistema revisa que los datos que ingresa el usuario sean correctos (ejemplo: formatos de correo, evitar inyecciones SQL).	Prevenir errores y ataques malintencionados.
Control de intentos fallidos	Si alguien intenta muchas veces entrar con una clave errónea, se bloquea temporalmente el acceso.	Proteger contra ataques de fuerza bruta.
Respaldo de información básica	Copias de seguridad periódicas de datos importantes, como usuarios y registros de uso.	Garantizar recuperación en caso de fallos o pérdidas.

Nota: Tabla hecha por Santiago Arango Rodriguez

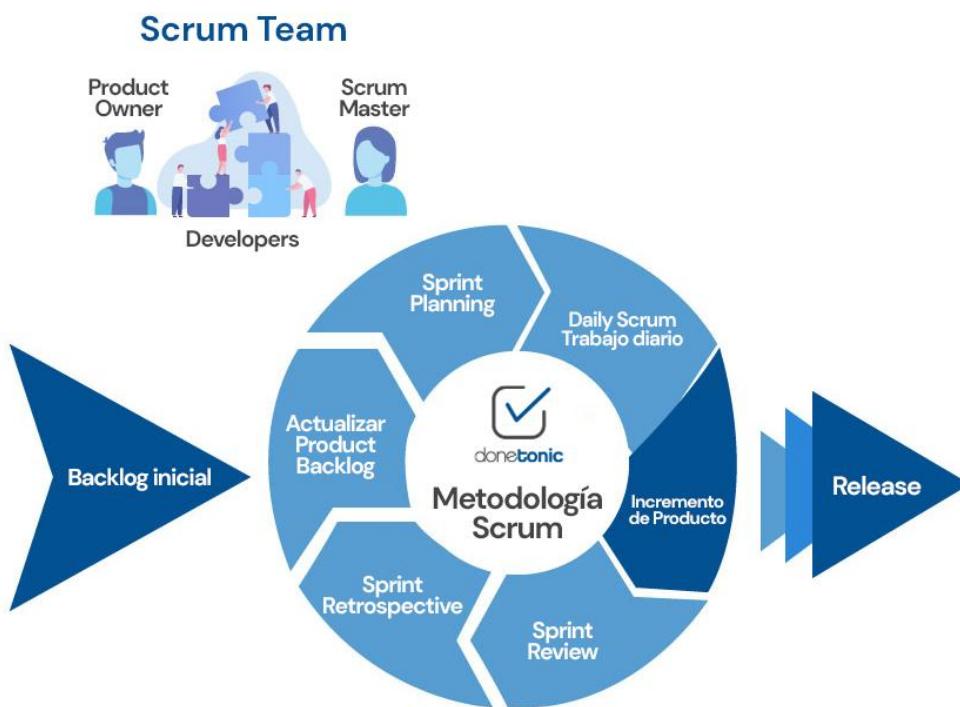
4.9 Descripción de la metodología Scrum

Scrum es una metodología ágil que se centra en trabajar en equipo de manera organizada para entregar software de calidad en tiempos cortos. No se trata de hacer todo de una sola vez, sino de dividir el proyecto en etapas pequeñas llamadas sprints, donde en cada una se entrega una parte funcional del sistema.

Su filosofía es la colaboración, la adaptabilidad y la entrega continua. Esto permite que, si hay cambios en los requisitos, podamos ajustarnos sin perder el rumbo.

4.9.1 Gráfico Fases de Scrum

Ilustración 5 Gráfico fases de Scrum



Nota: Imagen obtenida de <https://donetonic.com/wp-content/uploads/2021/05/la-metodologia-scrum.png>

4.9.2 Capa en la que estamos actualmente

En Scrum no se llama lo llamamos capas sino fase o evento en el ciclo.

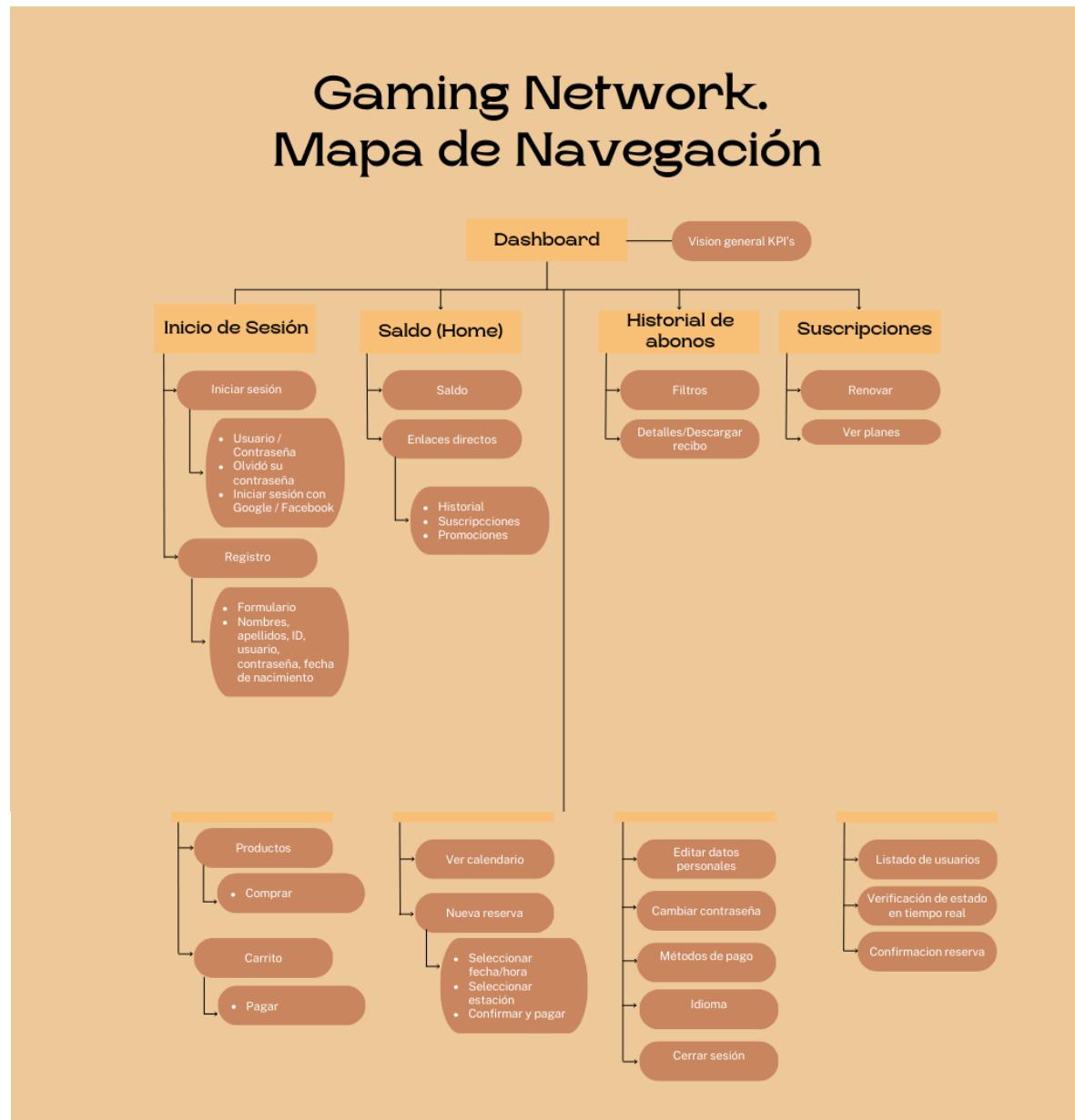
Por lo que me cuentas, ya hicimos levantamiento de requisitos, casos de uso, historias de usuario, requerimientos no funcionales y diagramas → eso corresponde al inicio de un Sprint de Desarrollo.

Específicamente, podemos ubicarlo en la etapa de Sprint Planning y Desarrollo:

- Ya definimos qué se va a construir (backlog refinado).
- Estamos entrando en cómo se va a implementar (diagramas, arquitectura, prototipos).
- Próximamente sería el incremento: entregar mas módulos funcionales además de login y registro.

4.10 Mapa de Navegación

Ilustración 6 Mapa de navegación



Nota: Mapa hecho por Santiago Arango Rodriguez en Canva

4.11 Librerías y dependencias del proyecto

4.11.1 Backend (*Spring Boot – Java*)

- **Spring Boot Starter Web**: para crear controladores y exponer endpoints REST.
- **Spring Boot Starter Thymeleaf** : para integrar vistas HTML dinámicas (si usas Thymeleaf).
- **Spring Boot Starter Data JPA**: para trabajar con base de datos relacional mediante repositorios.
- **Spring Boot Starter Validation**: validación de formularios y mensajes de error claros (RF1).
- **Spring Boot Starter Security**: autenticación básica y manejo de sesiones.
- **H2 Database** (para desarrollo): base de datos en memoria para pruebas rápidas.
- **MySQL / PostgreSQL Driver** (para producción): conexión con la BD real.
- **Lombok**: reduce código repetitivo (getters, setters, constructores).

4.11.2 Frontend (*integrado con backend*)

- Tailwind CSS → para dar estilos modernos y responsivos a las vistas.
- Font Awesome → iconos para la interfaz (estaciones, saldo, etc.).

si más adelante decides separar el frontend con React o Angular, se evalúan otras librerías)

4.11.3 Ejemplos de imports:

Imagen 1 ej import 1

```
import com.GN.Gaming.Network.model.Usuario;
import com.GN.Gaming.Network.service.UsuarioService;
import jakarta.servlet.http.HttpSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
```

Nota: Captura de pantalla tomada por Santiago Arango Rodriguez desde IntelliJ

Imagen 2 ej import 2

```
import com.GN.Gaming.Network.model.Usuario;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
```

Nota: Captura de pantalla tomada por Santiago Arango Rodriguez desde IntelliJ

Imagen 3 ej import 3

```
import com.GN.Gaming.Network.model.Usuario;
import com.GN.Gaming.Network.repository.UsuarioRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
```

Nota: Captura de pantalla tomada por Santiago Arango Rodriguez desde IntelliJ

4.12 Frameworks del proyecto

Tabla 17 Frameworks

Framework	Descripción	Uso en el proyecto
Spring Boot	Framework de Java que simplifica la creación de aplicaciones backend con configuración mínima.	Base del proyecto: controladores, servicios, repositorios y lógica de negocio.
Spring Data JPA	Extensión de Spring que facilita el acceso a bases de datos usando repositorios sin escribir queries SQL manuales.	Persistencia de datos (usuarios, estaciones, reservas, transacciones, productos, reportes).
Thymeleaf	Motor de plantillas para Java que permite crear páginas HTML dinámicas y conectarlas al backend.	Vistas para clientes y administradores (login, panel de estaciones, reservas, reportes).
TailwindCSS	Frameworks de diseño frontend (CSS). Tailwind es moderno y flexible.	Interfaz visual amigable y responsive sin necesidad de diseñar todo desde cero.

Nota: Tabla hecha por Santiago Arango Rodriguez

4.13 Reutilización de código en el proyecto

En el desarrollo de Gaming Network con Spring Boot se aplicó el principio de reutilización de código en varias capas del sistema. Esto permite mantener una arquitectura limpia, reducir la duplicación y facilitar el mantenimiento del software. A continuación, se muestran algunos ejemplos claros:

4.13.1 Servicios (Service)

La lógica de negocio se centraliza en las clases de servicio. Por ejemplo, en la clase UsuarioService se implementan unos métodos como findAll() o save(), los cuales son reutilizados por distintos controladores cuando necesitan obtener o registrar usuarios, evitando así repetir la misma lógica en cada controlador.

```
// Guardar o actualizar usuario
public Usuario save(Usuario usuario) { 1 usage
    return usuarioRepository.save(usuario);
}

// Metodos para CRUD listado y búsqueda

// Listar todos los usuarios
public List<Usuario> findAll() { 1 usage
    return usuarioRepository.findAll();
}
```

4.13.2 Repositorios (Repository)

Las interfaces de repositorio, como UsuarioRepository, también son reutilizadas en diferentes servicios. Por ejemplo, el método findByNicknameAndContraseña se usa para permitir el inicio de sesión de los usuarios. Esto para que consulte correctamente las credenciales.

```
//Validar usuario para login
Usuario findByNicknameAndContraseña(String nickname, String contraseña); 1 usage
```

4.13.3 Entidades (Entities)

Las clases de entidad (Usuario, Estacion, Reserva) se definen una sola vez y son utilizadas en toda la aplicación: en los repositorios para consultas, en los servicios para la lógica y en los controladores para enviar la información a las vistas. Esta reutilización permite que los cambios en una entidad (por ejemplo, añadir un atributo a Usuario) se reflejen en toda la aplicación sin necesidad de modificar varias clases diferentes.

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private int idUsuario;  
  
@Column(nullable = false)  
private String nombre;  
  
@Column(nullable = false, unique = true)  
private String nickname;  
  
@Column(nullable = false, unique = true)  
private String correo;  
  
@Column(nullable = false)  
private String contraseña;  
  
@Column(nullable = false)  
private String rol; // "admin o cliente"
```

4.13.4 Plantillas Thymeleaf

A nivel de la interfaz de usuario también existe esa reutilización. Por ejemplo, fragmentos como el **header** o el **footer** se definen en un archivo independiente y se incluyen en diferentes vistas mediante la instrucción th:replace. Esto en el futuro evita tener que repetir el mismo código HTML en cada plantilla, facilitando además las modificaciones globales en el diseño.

4.14 Buenas prácticas

Además de la reutilización de código, se aplicaron buenas prácticas de desarrollo como la documentación interna mediante comentarios. Estos comentarios permiten que cualquier desarrollador que trabaje en el proyecto pueda comprender de forma rápida qué hace cada método o clase, facilitando la colaboración y el mantenimiento.

```
// GET -> mostrar perfil
@GetMapping(@RequestMapping("/perfil"))
public String mostrarPerfil(@RequestParam("idUsuario") int idUsuario, Model model) {
    Optional<Usuario> usuarioOpt = userService.findById(idUsuario);

    if (usuarioOpt.isPresent()) {
        model.addAttribute(attributeName: "usuario", usuarioOpt.get());
        return "perfil"; // <- Thymeleaf buscará perfil.html en templates/
    } else {
        return "redirect:/inicioUsuario"; // redirige si no existe
    }
}

// POST -> actualizar perfil
@PostMapping(@RequestMapping("/perfil"))
public String actualizarPerfil(
    @RequestParam("idUsuario") int idUsuario,
    @RequestParam("nombre") String nombre,
    @RequestParam("nickname") String nickname,
    @RequestParam("correo") String correo,
    @RequestParam(value = "contraseña", required = false) String contraseña,
    Model model) {
```

4.14.1 //GET mostrar perfil

Este comentario describe que el método corresponde a una petición **GET** y su función principal es **mostrar el perfil del usuario**.

En este caso, el comentario ayuda a identificar que el método se conecta con la vista perfil.html para mostrar la información del usuario.

4.14.2 // POST actualizar perfil

Este comentario indica que el método está asociado a una petición POST y que su objetivo es actualizar el perfil del usuario.

Permite diferenciarlo del anterior (que sirve solo para mostrar) y facilita la comprensión del flujo de la aplicación.

Aquí el comentario es útil tambien porque muestra claramente que este método procesa datos enviados por un formulario (nombre, nickname, correo, contraseña) para actualizar la información en la base de datos.

```
// búsqueda combinada por filtro de texto (nombre, correo, nickname)
// si el filtro es null, devuelve todos
@Query("SELECT u FROM Usuario u " + 1 usage
        "WHERE (:filtro IS NULL OR " +
        "LOWER(u.nombre) LIKE CONCAT('%', LOWER(:filtro), '%') OR " +
        "LOWER(u.correo) LIKE CONCAT('%', LOWER(:filtro), '%') OR " +
        "LOWER(u.nickname) LIKE CONCAT('%', LOWER(:filtro), '%'))")
List<Usuario> searchByFiltro(@Param("filtro") String filtro);
```

4.14.3 Repositorio

En este método del repositorio se agregaron comentarios que explican claramente su propósito: realizar una búsqueda combinada por nombre, correo o nickname, y además indicar que si no se envía ningun filtro se devuelven todos los registros. Estos comentarios también permiten entender la lógica de la consulta sin necesidad de analizar toda la sintaxis, lo que mejora la **claridad y buenas prácticas** en el proyecto.

4.15 Paquetes de los componentes

```
└ com.GN.Gaming.Network
  └ config
  └ controller
    └ AdminUsuarioController
    └ DashboardController
    └ EliminarUsuarioController
    └ HolaController
    └ LoginController
    └ PerfilController
    └ UsuarioController
  └ model
    └ Usuario
  └ repository
    └ UsuarioRepository
  └ service
    └ UsuarioService
  └ GamingNetworkApplication
└ resources
  └ static
    └ logo.jpg
  └ templates
    └ dashboard.html
    └ inicioUsuario.html
    └ perfil.html
    └ registroUsuario.html
    └ usuarios.html
  └ application.properties
```

4.15.1 Paquete principal: com.GN.Gaming.Network

Este es el paquete raíz del proyecto. Aquí se agrupan todos los subpaquetes y también está la clase principal que se encarga de arrancar la aplicación.

4.15.2 config

En esta carpeta van las configuraciones generales del proyecto (aunque por ahora no hay clases en la captura).

Ejemplos: configuración de seguridad, etc.

4.15.3 controller

Aquí se encuentran los controladores (REST o MVC) que manejan las peticiones HTTP, se conectan con los servicios y devuelven respuestas o vistas. En este caso tengo varios:

- **AdminUsuarioController** → Maneja funcionalidades para administradores.
- **DashboardController** → Controla la vista principal o dashboard.
- **EliminarUsuarioController** → Encargado de la eliminación de usuarios.
- **HolaController** → Posiblemente usado para pruebas o mensajes de ejemplo.
- **LoginController** → Maneja el inicio de sesión.
- **PerfilController** → Se encarga del perfil del usuario.
- **UsuarioController** → Control general de los usuarios.

4.15.4 model

Aquí defino las entidades o modelos de datos que representan las tablas en la base de datos o los objetos principales del sistema.

Por ejemplo:

- **Usuario** : Clase que representa a un usuario con sus atributos (id, nombre, correo, etc.).

4.15.5 repository

Esta carpeta contiene las interfaces de acceso a datos, que normalmente extienden de JpaRepository o CrudRepository.

- **UsuarioRepository**: Es el que se encarga de interactuar con la base de datos para la entidad Usuario.

4.15.6 service

Aquí va la lógica de negocio. Los servicios hablan con los repositorios y luego son usados por los controladores.

- **UsuarioService**: Implementa las operaciones principales de los usuarios (crear, actualizar, eliminar, buscar, etc.).

4.15.7 GamingNetworkApplication

Esta es la clase principal del proyecto, marcada con `@SpringBootApplication`, que es la que arranca toda la aplicación.

4.15.8 resources

Carpeta estándar en proyectos Spring Boot donde van los recursos de la aplicación.

- **static:** Aquí se guardan archivos estáticos como imágenes, CSS o JS.
 - logo.jpg: Un recurso de imagen para el login.
- **Templates:** Vistas hechas con Thymeleaf (HTML que se puede renderizar en el servidor).
 - dashboard.html, inicioUsuario.html, perfil.html, registroUsuario.html, usuarios.html.
- **application.properties:** Un archivo de configuración de la aplicación (puerto, conexión a la base de datos, etc.).

4.16 Patrón de diseño MVC (Modelo – Vista – Controlador)

El patrón MVC es el que vamos a usar porque organiza la aplicación separando responsabilidades:

- **Modelo (M):**

Representa los datos y la lógica del negocio.

Ejemplo en tu proyecto: entidades como Usuario, Estación, Reserva, Transacción, Producto, Factura.

- **Vista (V):**

Es lo que ve el usuario (interfaz gráfica).

Ejemplo: el dashboard del administrador y la interfaz del cliente (reservas, pagos, tienda).

- **Controlador (C):**

Actúa como intermediario entre la Vista y el Modelo.

Ejemplo: los endpoints REST de Spring Boot (/login, /reservas, /estaciones) que reciben las solicitudes y llaman a los servicios del negocio.

Una ventaja es que facilita el mantenimiento, evita mezclar lógica de negocio con la interfaz, y permite escalar mejor.

4.16.1 Arquitectura en capas

La arquitectura estará organizada en tres capas principales dentro de un esquema cliente-servidor:

Capa de Presentación (Frontend)

- Puede ser Thymeleaf (integrado en Tailwind) o React si se requiere interactividad.
- Es lo que usan los usuarios (administrador o cliente).
- Se conecta al backend mediante llamadas HTTP (REST API).

Capa de Lógica de Negocio (Backend – Spring Boot)

Aquí trabaja por así decirlo la inteligencia del sistema y contiene:

- **Controladores:** reciben solicitudes.
- **Servicios:** procesan reglas de negocio (ej. calcular tarifas, validar disponibilidad de estaciones, aplicar descuentos de fidelización).

Capa de Datos (Persistencia – Base de Datos)

- Con Spring Data JPA se conecta a la base de datos (MySQL/PostgreSQL).
- Gestiona entidades y repositorios para almacenar usuarios, estaciones, reservas, transacciones, etc.

4.17 Evidencia testeo en Postman

Para verificar el correcto funcionamiento de las rutas y la comunicación con el backend del sistema, se realizaron pruebas de tipo GET utilizando Postman y el navegador, evaluando la respuesta de cada endpoint. Todos los testeos arrojaron un código de estado 200 OK, lo que indica que las solicitudes fueron procesadas correctamente por el servidor y que los recursos solicitados están disponibles.

4.17.1 Registro de usuario:

<http://localhost:8080/registroUsuario>

- **Objetivo:** Verificar que la página de registro de usuarios se cargue correctamente.
- **Resultado:** 200 OK, mostrando el formulario de registro.

Imagen 4 testeo registro

The screenshot shows the Postman interface with a successful HTTP request to `http://localhost:8080/registroUsuario`. The response status is `200 OK` with a response time of `9 ms` and a size of `6.13 KB`. The response body displays a registration form with fields for Nombre de usuario, Contraseña, and Recuérdame, along with a large red 'Iniciar sesión' button.

Nota: Captura de pantalla tomada por Santiago Arango Rodriguez desde Postman

4.17.2 Inicio de sesión:

http://localhost:8080/inicio

- **Objetivo:** Comprobar que la página de login sea accesible y que la sesión pueda inicializarse correctamente.
- **Resultado:** 200 OK, con carga correcta de la interfaz de inicio de sesión.

Imagen 5 testeo inicio

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:8080/registroUsuario
- Method:** GET
- Headers:** Authorization, Headers (6), Body, Scripts, Settings
- Query Params:** A table with one row:

Key	Value	Description
Key	Value	Description
- Response Status:** 200 OK
- Response Time:** 6 ms
- Response Size:** 6.95 KB
- Preview:** Shows a registration form titled "Crear una cuenta" with fields for Nombre (placeholder: Tu nombre completo) and Nombre de usuario (placeholder: Nombre de usuario).

Nota: Captura de pantalla tomada por Santiago Arango Rodriguez desde Postman

4.17.3 Dashboard del sistema:

http://localhost:8080/dashboard;jsessionid=AFABF8EF639382271B1D1D20A7B90948

- **Objetivo:** Validar que la página principal del dashboard se muestre para un usuario autenticado.
- **Resultado:** 200 OK, confirmando que la sesión se mantiene activa y que el dashboard carga con todos los elementos correspondientes.

Imagen 6 testeo dashboard

The screenshot shows a Postman interface. At the top, it displays the URL `http://localhost:8080/dashboard;jsessionid=AFABF8EF639382271B1D1D20A7B90948`. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Scripts', and 'Settings'. Under 'Headers (6)', there is a table with one row containing 'Key' and 'Value'. The 'Body' tab is selected, showing the response from the API. The response status is '200 OK' with a response time of '4 ms' and a size of '5.27 KB'. The response content is titled 'Panel de Control' and contains three cards: 'Estaciones' (with a monitor icon), 'Reservas' (with a calendar icon), and 'Usuarios' (with a people icon). Each card has a 'Gestionar' button.

Nota: Captura de pantalla tomada por Santiago Arango Rodriguez desde Postman

4.17.4 Perfil de usuario específico:

http://localhost:8080/perfil?idUsuario=2

- **Objetivo:** Comprobar que la información de un usuario específico pueda ser consultada correctamente.
- **Resultado:** 200 OK, cargando la información del usuario con ID 2.

Imagen 7 testeo perfil

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:8080/perfil?idUsuario=2
- Method:** GET
- Params:** idUsuario = 2
- Response Status:** 200 OK
- Response Body:** (HTML view) **Perfil de Usuario**

Nombre:	William Rosenberg
Nickname:	wew
Correo:	wew@example.com
Contraseña:	*****
Rol:	admin

Nota: Captura de pantalla tomada por Santiago Arango Rodriguez desde Postman

4.17.5 Listado de usuarios:

<http://localhost:8080/usuarios>

- **Objetivo:** Asegurar que la tabla de usuarios sea accesible y se muestren correctamente los datos registrados.
- **Resultado:** 200 OK, mostrando la lista completa de usuarios en el sistema.

Imagen 8 testeo usuarios

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:8080/usuarios
- Method:** GET
- Headers:** Authorization, Headers (6), Body, Scripts, Settings
- Query Params:** Key, Value, Description
- Body:** Cookies, Headers (6), Test Results, 200 OK, 9 ms, 5.18 KB
- Preview:** HTML, Visualize
- Table Data:** Shows a list of users with columns: Usuario (nickname), Nombre, Correo electrónico, Rol, and Estado. The data is as follows:

Usuario (nickname)	Nombre	Correo electrónico	Rol	Estado
alejo12	Alejandro Zapata Rodriguez	alejo12@example.com	admin	Activo
wew	William Rosemberg	wew@example.com	admin	Activo
santiX	Santiago Arango	santi@example.com	cliente	Activo
CamiGamer	Camila Torres	cami@example.com	cliente	Activo
JDark	Juan Perez	juanp@example.com	cliente	Activo

Nota: Captura de pantalla tomada por Santiago Arango Rodriguez desde Postman

4.18 Configuracion de base de datos

La base de datos en esta fase compuesta por tres tablas principales:

- **Usuarios:** almacena la información de las personas que interactúan con el sistema, asegurando que cada una tenga un identificador único y datos que no se repitan como el correo o el nombre de usuario.
- **Estaciones:** representa los equipos o consolas disponibles en el local, registrando su tipo, características y estado (si están libres, ocupadas o en mantenimiento).
- **Transacciones:** conecta a los usuarios con las estaciones, registrando cada uso junto con el monto, la fecha y el estado de la operación, lo que permite llevar un control de pagos y reservas.

Imagen 9 tabla estaciones

	Field	Type	Null	Key	Default	Extra
▶	id_estacion	int	NO	PRI	NULL	auto_increment
	tipo	varchar(50)	NO		NULL	
	descripcion	text	YES		NULL	
	estado	varchar(20)	NO		NULL	

Nota: Imagen tomada por Santiago Arango Rodriguez desde MYSQL Workbench

Imagen 10 tabla usuarios

	Field	Type	Null	Key	Default	Extra
▶	id_usuario	int	NO	PRI	NULL	auto_increment
	nombre	varchar(255)	NO		NULL	
	nickname	varchar(255)	NO	UNI	NULL	
	correo	varchar(255)	NO	UNI	NULL	
	contraseña	varchar(255)	NO		NULL	
	rol	varchar(255)	NO		NULL	

Nota: Imagen tomada por Santiago Arango Rodriguez desde MYSQL Workbench

Imagen 11 tabla transacciones

	Field	Type	Null	Key	Default	Extra
▶	id_transaccion	int	NO	PRI	NULL	auto_increment
	id_usuario	int	NO	MUL	NULL	
	id_estacion	int	NO	MUL	NULL	
	monto	decimal(10,2)	NO		NULL	
	fecha	datetime	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
	estado	varchar(20)	YES		EN_CURSO	

Nota: Imagen tomada por Santiago Arango Rodriguez desde MYSQL Workbench

4.19 Ambientes de desarrollo y pruebas

Enlace video de YT explicativo:

<https://youtu.be/K8zOzKXwRQE>

5. Conclusiones

El desarrollo de este documento permitió sentar las bases de un sistema de software robusto y bien estructurado para la función del proyecto. A lo largo del proceso, se establecieron los requerimientos funcionales y no funcionales más relevantes, haciendo unos ajustes priorizando aquellos que podían implementarse de forma realista en el tiempo disponible, sin perder de vista la posibilidad de ampliar funcionalidades en el futuro con más disponibilidades.

El uso de la metodología Scrum facilita mucho avanzar de manera iterativa y ordenada, permitiendo ajustar los objetivos conforme se iba adquiriendo una mayor claridad técnica. Asimismo, se aplicaron buenas prácticas de programación, reutilización, pruebas configuraciones adecuadas de bases de datos, lo que hace una base sólida y confiable para el sistema.

Este documento refleja en su totalidad la planeación técnica del proyecto, y también el compromiso por entregar una solución funcional, escalable y centrada en las necesidades del usuario, que servirá como apoyo directo para la administración y el crecimiento del negocio.