El propósito de este documento es especificar las funcionalidades de la aplicación "Canchas", así como las tecnologías y arquitecturas que se utilizarán para implementarlas. El objetivo es proporcionar una visión clara que permita seleccionar las tecnologías más adecuadas para el desarrollo del proyecto. Considerando las dimensiones del proyecto, se espera un alto volumen de peticiones y usuarios simultáneos. Por lo tanto, se busca asegurar una alta disponibilidad y optimización de los servicios.

**Listado de funcionalidades:**

1. Administración de usuarios.
   1. Registro y autenticación de usuarios.
   2. Administración de perfiles.
   3. Automación y roles.
2. Conexión entre jugadores.
   1. Conexión entre jugadores para organizar partidos.
   2. Búsqueda de jugadores y equipos.
   3. Gestión de invitaciones y aceptación de partidos.
3. Gestión de canchas.
   1. Reserva de canchas.
   2. Gestión de disponibilidad.
   3. Pagos y confirmación de reservas.
4. Funcionalidad de notificaciones.
   1. Envió de notificaciones push y correos electrónicos.
   2. Alerta de reservas, partidos y apuestas.
   3. Notificaciones de eventos.
5. Funcionalidad de apuestas.
   1. Gestión de apuestas entre partidos.
   2. Calculo y pago de premios.
   3. Historial de apuestas y transacciones.
6. Funcionalidad de chat.
   1. Búsqueda de usuarios.
   2. Envió de mensajes en tiempo real.

**Tecnologías sugeridas.**

Teniendo en cuenta que la finalidad es tener una plataforma administrativa dedicada a las actividades desempeñadas por los administradores de las canchas sintéticas y una plataforma móvil enfocada a los jugadores amateurs organizar y agendar partidos, se relacionan las tecnologías propuestas para llevar el desarrollo.

|  |  |  |
| --- | --- | --- |
| **Plataforma** | **Tecnología** | **Version** |
| Web | Angular |  |
| Móvil | Flutter |  |
| Backend | Spring boot |  |
| Motor BD | PostgresSQL |  |
| Mensajes | NodeJS, MongoDB |  |

**Descripción de arquitecturas.**

**Backend**

En la fase inicial del proyecto se utilizará una arquitectura que facilite la agilidad en el proceso de desarrollo y que no conlleve un gran esfuerzo para realizar procesos de despliegue, adicional es necesario mantener un bajo acoplamiento entre los compontes para aumentar la flexibilidad del aplicativo, para esto se plantea utilizar la arquitectura monolito modular con el objetivo de en algún punto futuro realizar la migración hacia microservicios.

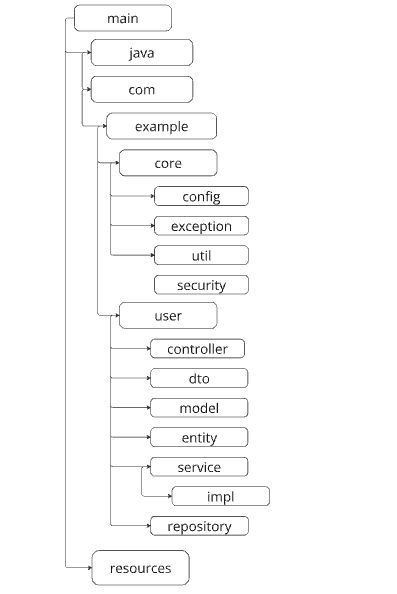
**Monolito modular:**

A continuación, se relacionan los módulos por los que estará conformado la aplicación, estos se definen en base a las funcionalidades anteriormente mencionadas.

1. User.
2. Player connection.
3. Field Management.
4. Notification.
5. Betting.
6. Chat.

**Distribución de carpetas.**

A continuación, se relaciona un ejemplo de la distribución de capetas para un servicio con la tecnología spring boot.



**Descripción**

* **config/**

Nombre archivo: AppConfig.java

Configuración general de la aplicación, como beans y propiedades específicas.

* **controller/**

Nombre archivo: UserController.java

Controladores REST que manejan las solicitudes HTTP relacionadas con usuarios.

* **dto/**

Nombre archivo: UserDTO.java

Transferencia de datos

* **exception/**

Nombre archivo: GlobalExceptionHandler.java

Maneja todas las excepciones globales de la aplicación

* **model/**

Nombre archivo: User.java

Clases que representan las entidades de la base de datos.

* **repository/**

Nombre archivo: UserRepository.java

Interfaces que extienden JpaRepository o CrudRepository para interactuar con la base de datos.

* **service/**

Nombre archivo: UserService.java

Clases abstractas que definen las funcionalidades.

* **service/impl/**

Nombre archivo: UserServiceImpl.java

Clases que extiende de UserService y contienen la lógica de negocio.

* **security/**

Nombre archivo: SecurityConfig.java

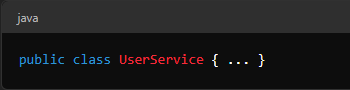
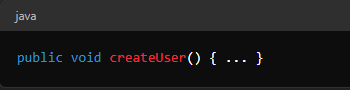
Configuración de seguridad, incluyendo reglas de acceso y protección de rutas.

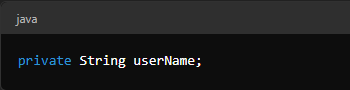
* **util/J**wtUtil.java

**Convenciones de codificación.**

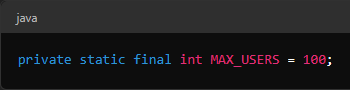
Definir un estándar de codificación es importante para mantener una coherencia y una alta calidad de código.

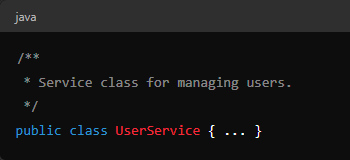
Teniendo en cuenta Google Java Style Guide se listarán los puntos más importantes que se deben tener en cuenta al momento de la escritura de código en el proyecto java.

1. **Idioma.**
   1. Se define que la codificación será exclusivamente en inglés con algunas excepciones que se mencionaran en este archivo.
2. **Nombres de clases.**
   1. Deben ser sustantivos y en CamelCase
   2. Los nombres de clases no pueden ser verbos.
   3. Siempre se deben usar palabras completas evitando siempre las abreviaturas a menos que estas sean mas comunes que la palabra completa.
   4. No es posible que los nombres tengan caracteres especiales, solo se puede usan usar letras de la “a” a la “z”.
   5. Los nombres de clases de tipo DAO, DTO o similares deben tener el sufijo correspondiente como UserDAO o UserDTO.
3. **Nombres de métodos.**
   1. Deben ser verbos y usar lowerCamelCase
   2. No es posible asignar espacios ni caracteres especiales.
4. **Nombres de variables.**
   1. Deben ser descriptivos y en lowerCamelCase

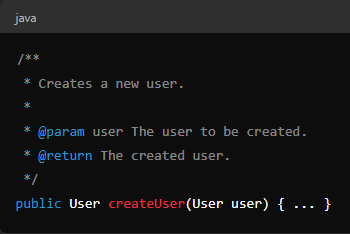


* 1. Se deben evitar los nombres de variables de un solo carácter, solo se admiten para variables temporales como j, i o k al interior de ciclos.

1. **Nombres de constantes.**
   1. Se define usar mayúsculas y snake\_case
2. **Comentarios de clases.**
   1. Cada clase debe tener un breve comentario **EN ESPAÑOL** que defina cuál es su propósito



* 1. **Comentarios de métodos.**Todos los métodos deben tener un comentario **EN ESPAÑOL** que describa su funcionalidad, parámetros que recibe y cual es el valor de retorno.



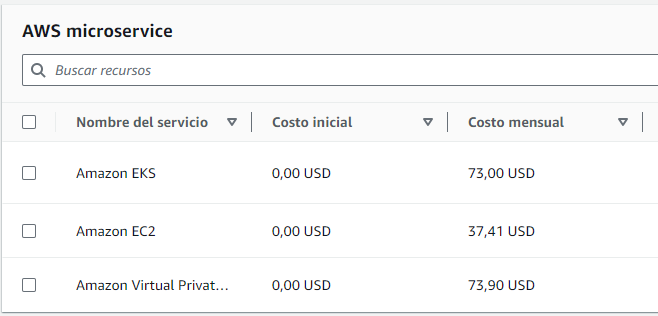
**Costos**

2024-05-08

A continuación, se muestra cotización realizada en los principales prestadores de servicio nube, teniendo en cuenta la arquitectura, tecnologías y servicios anteriormente planteados.

**AWS**

|  |  |  |
| --- | --- | --- |
| **Descripción** | **Valor USD** | **Datos adicionales** |
| Amazon EKS | 73 |  |
| Amazon Virtual Private Cloud (VPC) | 73.90 | 10 GB mes |
| Amazon EC2 | 38.61 | 2 CPU, 8GB RAM, 32GB SSD |
| Total | 185.51 |  |



[**https://calculator.aws/#/estimate?id=793f239a178f55ece007474fb0e61b8a5987a866**](https://calculator.aws/#/estimate?id=793f239a178f55ece007474fb0e61b8a5987a866)

**Amazon EKS:** Servicio administrado que permite realizar la ejecución de kubernetes en la nube de AWS. Facilita la instalación mantenimiento y operación de kubernestes.

**Kubernetes:** Es una plataforma de código abierto que facilita la implementación, escalado y gestión de aplicaciones en contenedores. Permite administrar automáticamente las cargas de trabajo y servicios, lo que optimiza recursos y mejora la portabilidad de las aplicaciones.

**Amazon Virtual Private Cloud (VPC):** Es una red virtual dedicada a la cuenta de aws contratada, ofrece un control total sobre el entorno real de redes virtuales. Esta infraestructura está aislada lógicamente de otras redes permitiendo gestionar la ubicación de los recursos y configuraciones como habitualmente lo hacen los centros de datos.

**Amazon EC2:** Es un servicio web de Amazon que proporciona capacidad de computación en la nube, segura y de tamaño modificable. Permite lanzar instancias de servidores virtuales en la nube, con la flexibilidad de elegir el hardware y la configuración que mejor se adapte a los requerimientos.