Universidad Tecnológica de Torreón

Convenciones y Normativas en el Desarrollo del Casino en Línea

Equipo: “Alt + F4”

Docente: Luis Ernesto Ramírez Carrillo

Grupo: 4°C

Fecha de Entrega: 29/01/2024.

Convenciones y Normativas en el Desarrollo del Casino en Línea

El desarrollo de software en equipo requiere la adopción de **convenciones y normativas** que permitan mantener un código claro, reutilizable y fácil de mantener. En este documento, establecemos las reglas de nomenclatura, estructuración y codificación que seguiremos en el desarrollo del **Casino en Línea**.

Contemplaremos las siguientes metodologías / filosofías para llevar a cabo:

Filosofía de diseño:

- Menciona la importancia de adoptar una filosofía de diseño centrada en el usuario (User-Centered Design) para garantizar una experiencia de juego intuitiva, atractiva y accesible.
- Considera incluir principios de diseño de interacción (IxD) para optimizar la usabilidad y la satisfacción del jugador.

Metodologías ágiles:

- Introduce brevemente las metodologías ágiles (Scrum, Kanban) y cómo su enfoque iterativo e incremental puede beneficiar el desarrollo del casino en línea, permitiendo adaptarse rápidamente a los cambios y entregar valor de forma continua.

Consistencia y claridad:

- Enfatiza cómo una convención de nombramiento sólida facilita la comprensión del código, reduce la ambigüedad y mejora la colaboración entre desarrolladores.
- Explica cómo la consistencia en los nombres de variables, funciones y clases ayuda a mantener la coherencia y la calidad del código a largo plazo.

1. Convención de Nombramiento

Para mantener la consistencia en el código, utilizaremos **estándares de nomenclatura** en diferentes aspectos del proyecto.

Esto conlleva que ejemplifiquemos:

Consistencia y claridad:

- Enfatizaremos cómo una convención de nombramiento sólida facilita la comprensión del código, reduce la ambigüedad y mejora la colaboración entre desarrolladores.

- Explicaremos cómo la consistencia en los nombres de variables, funciones y clases ayuda a mantener la coherencia y la calidad del código a largo plazo.

1.1 Variables

- Usaremos **camelCase** para nombres de variables en JavaScript/TypeScript.
- Ejemplo: playerBalance, gameResult.
- Variables constantes en **SCREAMING_SNAKE_CASE**.
- Ejemplo: MAX_BET_AMOUNT.

1.2 Funciones / Métodos

- Nombres en **camelCase**, comenzando con un verbo que describa la acción.
- Ejemplo: calculateWinnings(), fetchPlayerStats().
- Métodos dentro de clases deben ser concisos y específicos.

1.3 Clases / Componentes

- Usaremos **PascalCase** para clases y componentes de React.
- Ejemplo: Player, GameTable, AdminDashboard.

1.4 Archivos y Directorios

- Archivos de componentes en **PascalCase**.
    - Ejemplo: RegisterForm.tsx, GameBoard.tsx.
- Archivos de utilidad en **camelCase**.
    - Ejemplo: gameLogic.ts, apiClient.ts.
- Directorios en **kebab-case**.
    - Ejemplo: game-logic/, user-profile/.

1.5 Base de Datos

- Tablas en **snake_case**, en plural.
    - Ejemplo: players, game_sessions.
- Columnas en **snake_case**.

- Ejemplo: player_id, bet_amount.

2. Estándares para Reducir el Tamaño del Código

Para mejorar la eficiencia y mantenibilidad del código, aplicaremos las siguientes prácticas:

2.1 Componentización en React

- Crearemos **componentes reutilizables** para evitar duplicación de código.
- Ejemplo:
  - ButtonPrimary.tsx para botones reutilizables.
  - RegisterForm.tsx para el formulario de registro.

2.2 Código Limpio

- Seguir principios **DRY (Don't Repeat Yourself)** y **KISS (Keep It Simple, Stupid)**.
- Evitar funciones con más de 50 líneas dividiéndolas en subfunciones cuando sea necesario.

2.3 Uso de Tipado Estricto

- Utilizaremos TypeScript para evitar errores de tipado.
- Definiremos **interfaces y tipos** para estructuras de datos.
- Ejemplo:
- interface Player {
- id: number;
- name: string;
- balance: number;
- }

3. Organización del Código y Estándares para Reducir el Tamaño del Código

Principios SOLID:

Introduce los principios SOLID de diseño de software y cómo su aplicación puede mejorar la calidad y la mantenibilidad del código.

Explica cómo SOLID fomenta la modularidad, la reutilización y la extensibilidad del código, lo cual es crucial en un proyecto de casino en línea que puede crecer y evolucionar con el tiempo.

Patrones de diseño:

Menciona algunos patrones de diseño relevantes para el desarrollo de casinos en línea, como el patrón Modelo-Vista-Controlador (MVC) para separar la lógica de negocio de la interfaz de usuario, o el patrón de Observador para gestionar eventos y notificaciones en tiempo real.

Para mantener la claridad y escalabilidad del proyecto, estructuraremos los archivos de la siguiente manera:

3.1 Estructura del Frontend (React + TypeScript)

/src

/components

/UI (Botones, modales, inputs reutilizables)

/Game (Componentes específicos de los juegos)

/pages

/Home

/Casino

/Admin

/hooks

/services

/utils

3.2 Estructura del Backend (Node.js + NestJS)

/src

/modules

/auth

/users

/games

/bets

/middlewares

/services

/controllers

/entities

4. Seguridad y Buenas Prácticas

- **Autenticación segura** con JWT y refresh tokens.
- **Encriptación de contraseñas** con bcrypt.
- **Validaciones estrictas** en formularios y API para evitar ataques de inyección.
- **Control de acceso** con middleware y roles en la base de datos.

5. Control de Versiones y CI/CD

- Uso de **Git y GitHub** con ramas organizadas (main, develop, feature/*).
- Implementación de **CI/CD** con GitHub Actions.
- Despliegue en **Vercel (Frontend)** y **AWS/Azure/Render (Backend)**.

6. Pruebas y Calidad del Código

- **Pruebas unitarias** con Jest en el backend y React Testing Library en el frontend.
- **Pruebas de integración** para verificar la comunicación entre componentes y servicios.
- **Pruebas de carga** para garantizar el rendimiento en eventos en tiempo real.

Conclusión

El uso de convenciones de nomenclatura, estándares de codificación y buenas prácticas asegurará que el desarrollo del **Casino en Línea** sea eficiente, escalable y mantenible. Al seguir estas reglas, facilitaremos el trabajo en equipo y mejoraremos la calidad del código, permitiendo la rápida implementación de nuevas funcionalidades y la estabilidad del sistema.