

# Projeto : CoChess

**Feito por :**

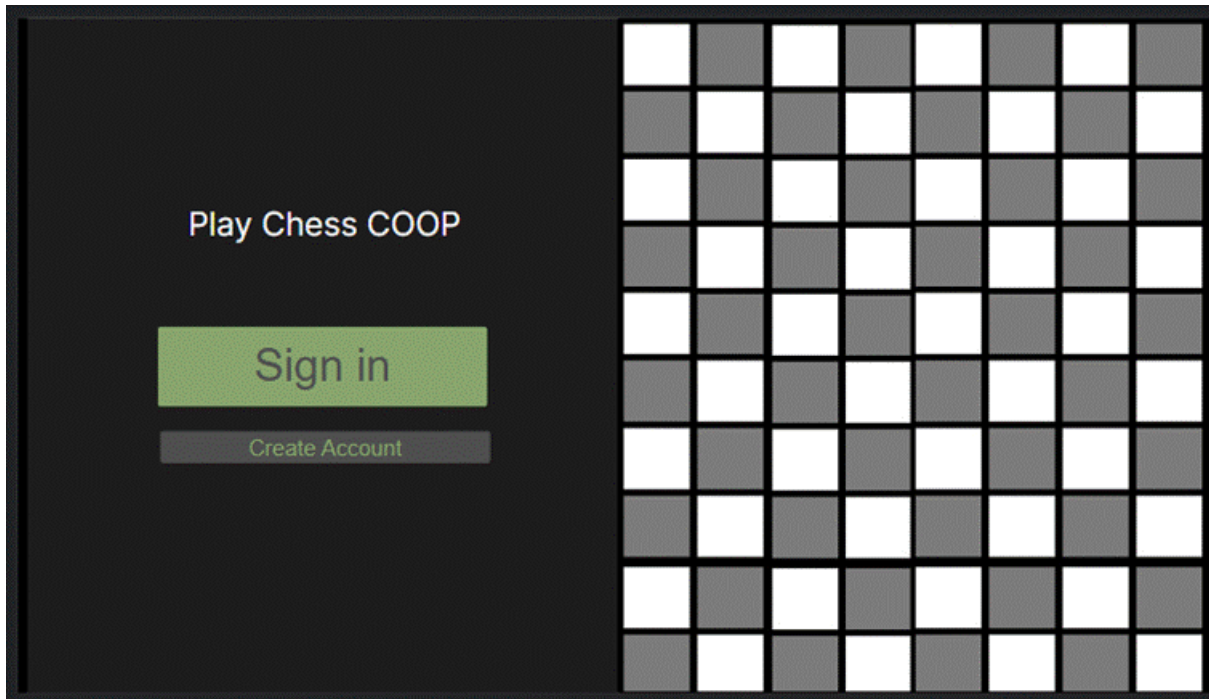
**Matheus Vidal Pereira, João Caio Oliveira Lins e João Henrique Rodrigues Lopes**

# Propósito Projeto 'CoChess'

O projeto CoChess foi desenvolvido com o objetivo de proporcionar uma experiência de xadrez cooperativo e interativo para os usuários. A aplicação permite partidas online, sugestões de jogadas, histórico de partidas e gerenciamento de perfil. Este documento apresenta os principais artefatos do sistema, incluindo diagramas de classes e telas da aplicação.

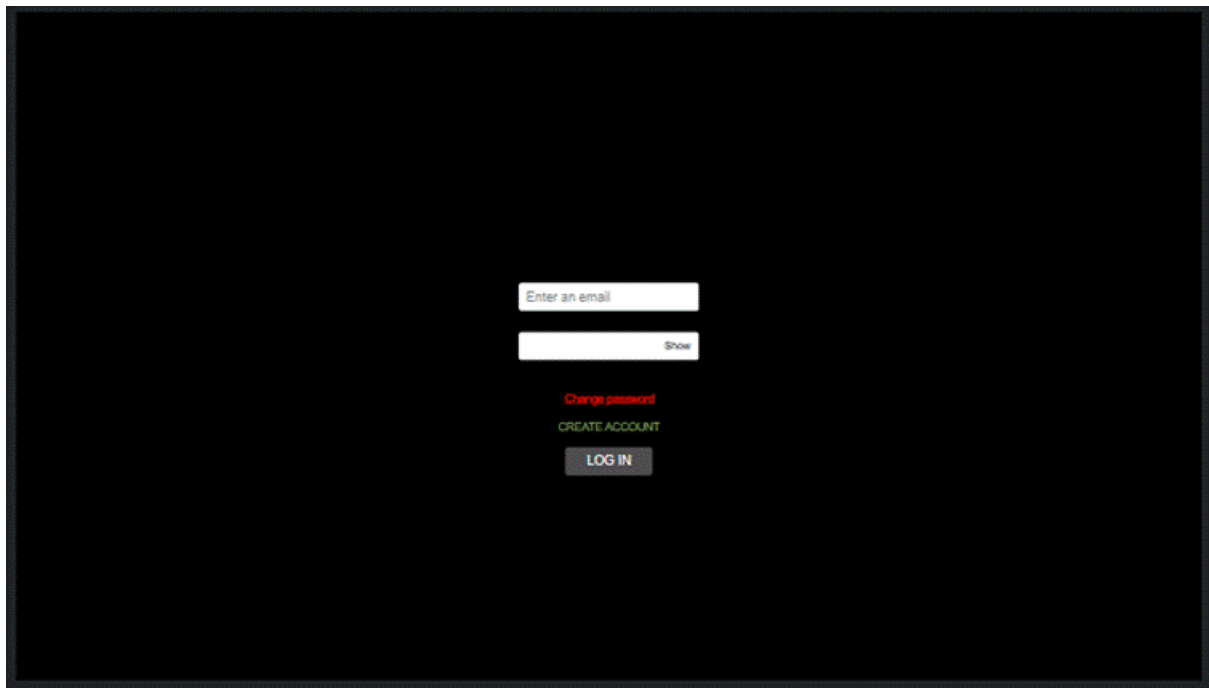
# Diagrama de Classes:

Tela (Início)



**Classe Usuario:** A tela inicial pode exibir opções de login ou cadastro, onde os usuários devem se autenticar.

## Tela (Log in)

A login screen with a dark background. In the center, there are two white input fields. The first field has the placeholder text "Enter an email". Below it is a second field with a "Show" button on its right side. Under the second field, there are three links: "Change password" in red, "CREATE ACCOUNT" in green, and a "LOG IN" button in a grey box.

**Classe Usuario:** Aqui, o usuário autentica seu acesso ao sistema

- **Atributos:** id, nome, email, senha
- **Métodos:** cadastrar(), autenticar(), editarPerfil()

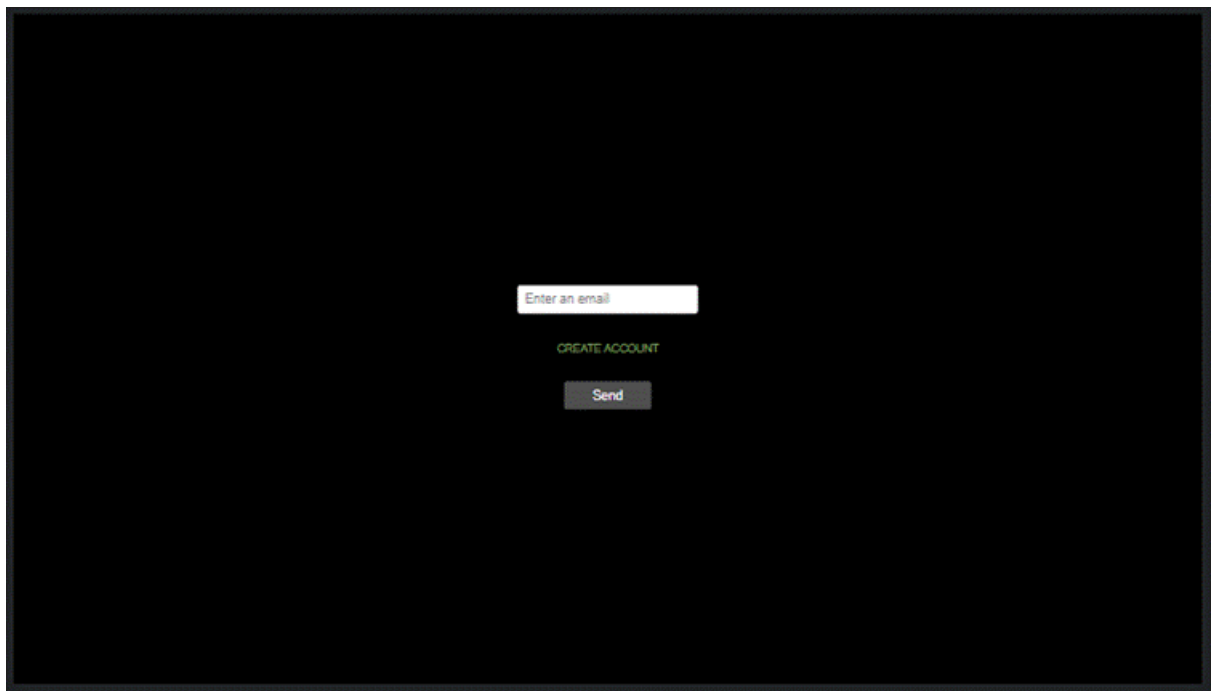
**Classe UserController:** Recebe o email e senha, e chama login(), compara o retorno e dá como válido caso verdadeiro, aparecerá uma mensagem de erro em caso falso.

- **Métodos:** create(), login(), recuperarSenha()

**Classe UserDAO:** Chama find(), e retorna o email e senha caso os dois sejam verdadeiros.

- **Métodos:** create(), find(), findByEmail()

## Tela (Recuperação de Senha)



Enter an email

[CREATE ACCOUNT](#)

Send

**Classe UserController:** Recebe o email e chama login(), compara o retorno com o input e válida, caso verdadeiro, então chama sendMail().

- **Métodos:** create(), login(), recuperarSenha()

**Classe UserDAO:** Chama find(), e retorna o email do banco de dados caso ache.

- **Métodos:** create(), find(), findByEmail()

### Tela (Criar Conta)

Nickname:  
Enter a value

Email:  
Enter a value

Confirm Email:  
Enter a value

Password:  
Enter a value

Confirm Password:  
Enter a value

- Password need to have a number  
and a symbol

[Sign In](#)

Create

**Classe Usuario:** Chama cadastrar() para permitir que novos usuários entrem no sistema.

- **Atributos:** id, nome, email, senha
- **Métodos:** cadastrar(), autenticar(), editarPerfil()

**Classe UserDAO:** Chama find() e verifica se email ou nome já existem, retorna erro se verdadeiro, caso falso chama create(), e adiciona no banco de dados um novo usuário caso todos os inputs sejam válidos.

- **Métodos:** create(), find(), findByEmail()

## Tela (Mudar Senha)



The screenshot shows a web form for changing a password. It is centered on a white background within a black rectangular frame. The form consists of the following elements from top to bottom:

- A label "Password:" in blue text.
- A text input field with the placeholder text "Enter a value".
- A label "Confirm Password:" in blue text.
- A second text input field with the placeholder text "Enter a value".
- A small note in grey text: "- Password need to have a number and a symbol".
- A "Sign In" link in green text.
- A black button with the word "Change" in white text.

**Classe Usuario:** Utiliza `editarPerfil()` para alterar a senha e garantir a segurança da conta.

- **Atributos:** id, nome, email, senha
- **Métodos:** `cadastrar()`, `autenticar()`, `editarPerfil()`

**Classe UserDAO:** Chama `find()`, e retorna o id caso seja válido, então chama `create()`, para dar update no banco de dados, caso a senha seja válida.

- **Métodos:** `create()`, `find()`, `findByEmail()`

### Tela (Editar Perfil)

Nickname:  
Enter a value

Password:  
Enter a value

Confirm Password:  
Enter a value

- Password need to have a number and a symbol

Edit

**Classe Usuario :** Permite modificar nome e senha através de editarPerfil()

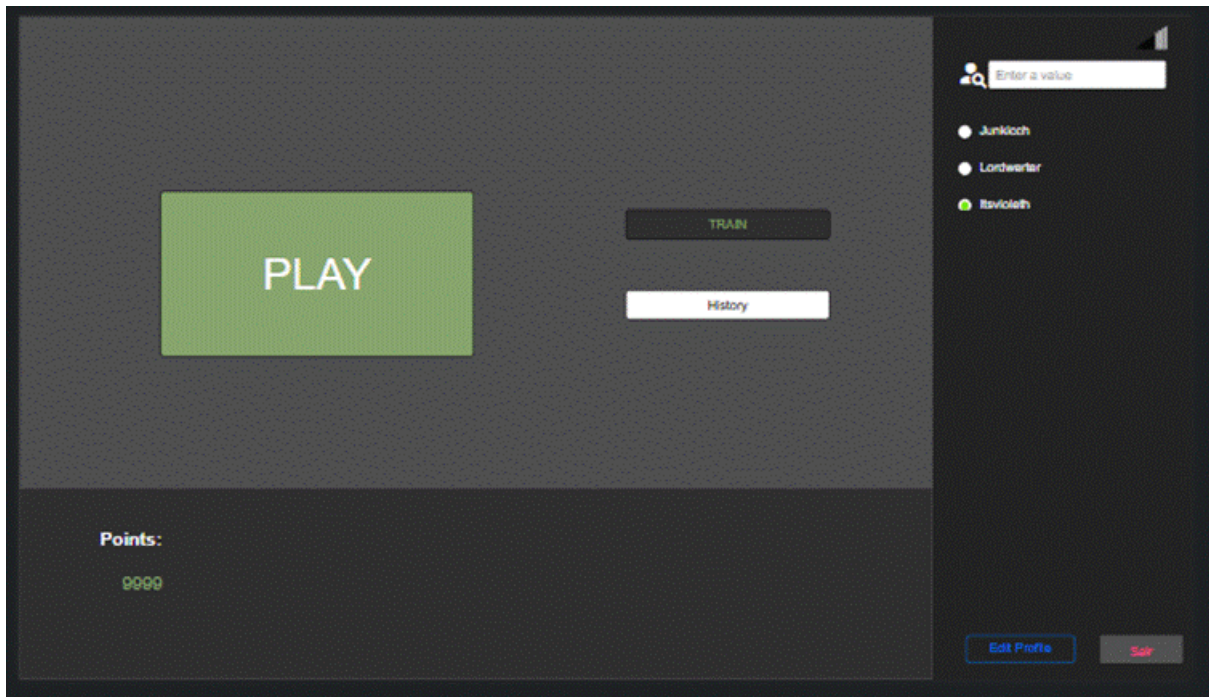
- **Atributos:** id, nome, senha
- **Métodos:** cadastrar(), autenticar(), editarPerfil()

**Classe UserDAO:** Chama find(), e retorna o nome caso seja válido, então chama create(), para dar update no banco de dados, caso todos os inputs sejam válidos.

- **Métodos:** create(), find(), findByEmail()



### Tela (Lobby)



**Classe Jogador:** Exibe informações sobre ranking, jogadores disponíveis e opções de jogo.

- **Atributos:** ranking, time
- **Método:** visualizarRanking(), getRanking()

## Tela (Histórico de Jogos)

PLAY	TRAIN	HOME
RESULT	ANALYSIS	SCORE
victory	-	-
defeat	-	-
defeat	-	-
defeat	-	-
defeat	-	-
defeat	-	-
defeat	-	-

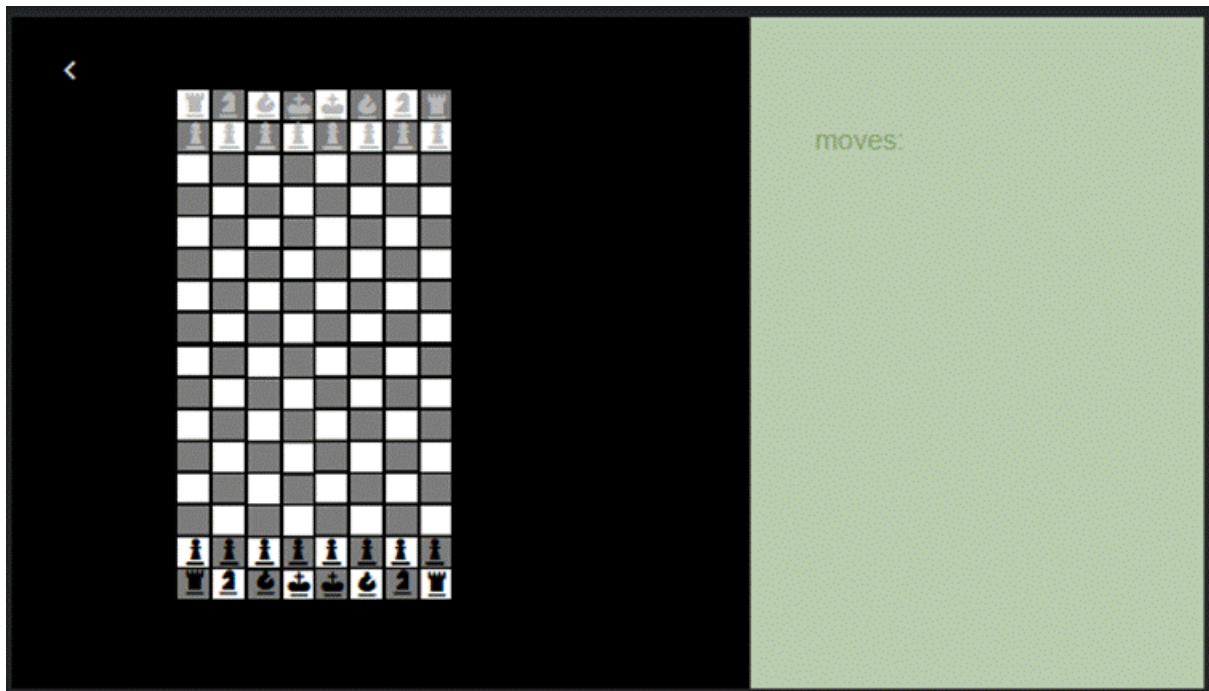
**Classe Ranking:** Permite análise de desempenho por meio de `visualizarRanking()`.

- **Atributos:** id, jogador, pontuacao, vitorias, derrotas
- **Métodos:** `atualizarRanking()`, `visualizarRanking()`, `calcularPontuacao()`

**Classe Partida :** Apresenta os detalhes das partidas jogadas anteriormente através de `verHistoricoPartida()`.

- **Atributos:** id, jogadores, tabuleiroEstado, status, historicoMovimentos
- **Métodos:** `iniciarPartida()`, `finalizarPartida()`, `registrarMovimento()`, `verHistoricoPartida()`, `atualizarTabuleiroEstado()`, `validarMovimento`

### Tela (Treino)



**Classe SugestaoJogada :** Sugere jogadas para aprimorar as habilidades dos jogadores através de gerarSugestao().

- **Atributos:** id, jogador, movimentoSugerido
- **Métodos:** gerarSugestao(), aceitarSugestao()

**Classe UserRoutes:**

- **Métodos:** createRoutes()

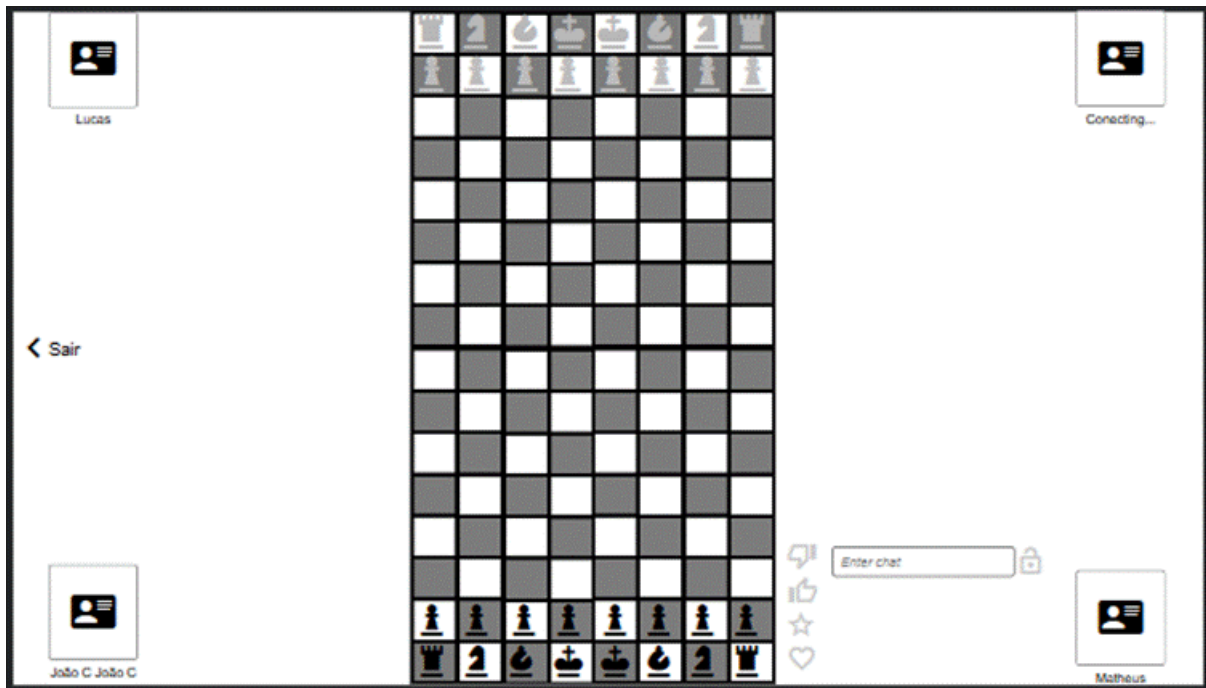
**Classe Movimento:** Armazena os movimentos realizados no treino com validarMovimento().

- **Atributos:** id, peca, posicaoInicial, posicaoFinal, valido
- **Métodos:** validarMovimento(), executarMovimento()

**Classe Partida:** Gerencia o andamento do jogo com registrarMovimento() e finalizarPartida().

- **Atributos:** id, jogadores, tabuleiroEstado, status, historicoMovimentos
- **Métodos:** Métodos: iniciarPartida(), finalizarPartida(), registrarMovimento(), verHistoricoPartida(), atualizarTabuleiroEstado(), validarMovimento()

### Tela (Partida)



**Classe SugestaoJogada :** Caso ativado, pode fornecer sugestões de jogadas durante a partida..

- **Atributos:** id, jogador, movimentoSugerido
- **Métodos:** gerarSugestao(), aceitarSugestao()

**Classe UserRoutes:**

- **Métodos:** createRoutes()

**Classe Movimento:** Armazena os movimentos realizados com validarMovimento().

- **Atributos:** id, peca, posicaoInicial, posicaoFinal, valido
- **Métodos:** validarMovimento(), executarMovimento()

**Classe Partida:** Gerencia o andamento do jogo com registrarMovimento() e finalizarPartida().

- **Atributos:** id, jogadores, tabuleiroEstado, status, historicoMovimentos
- **Métodos:** iniciarPartida(), finalizarPartida(), registrarMovimento(), verHistoricoPartida(), atualizarTabuleiroEstado(), validarMovimento()

**Classe Ranking:** quando finalizarPartida() é chamado, é chamado calcularPontuacao().

- **Atributos:** id, jogador, pontuacao, vitorias, derrotas
- **Métodos:** atualizarRanking(), visualizarRanking(), calcularPontuacao()

### **Classe Jogador** (herda de Usuario)

- **Atributos:** ranking, time
- **Métodos:** visualizarRanking(), getRanking()

# Código dos diagramas no plantUML:

@startuml

package "sysfit" {

' \*\*\* Camada de Persistência (Persistence) \*\*\*

package "persistence" {

class Usuario {

+id: int

+nome: String

+email: String

+senha: String

+cadastrar()

+autenticar()

+editarPerfil(nome: String, senha: String)

}

}

' \*\*\* Camada de Persistência (PersistenceLayer) \*\*\*

package "persistencelayer" {

' --- DAO (Data Access Object) ---

package "DAO" {

interface IUserDAO {

+boolean create(Usuario user)

+List<Usuario> find()

+Usuario findByEmail(email: String) ' Novo método para busca por e-mail

}

class UserDAO {

+boolean create(Usuario user)

+List<Usuario> find()

+Usuario findByEmail(email: String) ' Novo método para busca por e-mail

}

IUserDAO <|.. UserDAO

}

' --- Model ---

package "model" {

class Jogador {

+ranking: int

+time: String

+visualizarRanking()

+getRanking(): int

```

    }

    class Partida {
        +id: int
        +jogadores: Jogador[4]
        +tabuleiroEstado: String ' Tabuleiro agora é um atributo interno
        +status: String
        +historicoMovimentos: Movimento[]
        +iniciarPartida()
        +finalizarPartida()
        +registrarMovimento()
        +verHistoricoPartida()
        +atualizarTabuleiroEstado()
        +validarMovimento()
    }

    class Movimento {
        +id: int
        +peca: String
        +posicaoInicial: String
        +posicaoFinal: String
        +valido: boolean
        +dataHora: DateTime
        +validarMovimento()
        +executarMovimento()
    }

    class Ranking {
        +id: int
        +jogador: Jogador
        +pontuacao: int
        +vitorias: int
        +derrotas: int
        +atualizarRanking()
        +visualizarRanking()
        +calcularPontuacao()
    }
}

' --- Controller ---
package "controller" {
    interface IUserController {
        +Usuario create(Usuario user)
        +boolean login(Usuario user)
        +boolean recuperarSenha(email: String) ' email reset!
    }

    class UserController {

```

```

        +Usuario create(Usuario user)
        +boolean login(Usuario user)
        +boolean recuperarSenha(email: String) ' email reset!
    }

    IUserController <|.. UserController
}

```

```

' *** Camada de Serviço (ServiceLayer) ***
package "servicelayer" {

```

```

' --- DTO (Data Transfer Object) ---

```

```

package "DTO" {
    class UserLoginReq {
        "email": String
        "password": String
    }

```

```

    class UserLoginRes {
        "msgcode": "001"
        "userobject": "{}"
    }

```

```

    class UserCreateReq {
        "email": String
        "password": String
        "confpassword": String
    }

```

```

    class UserCreateRes {
        "msgcode": "002"
        "userobject": "{}"
    }
}

```

```

' --- Serviço de Sugestão de Jogadas ---

```

```

package "SuggestionService" {
    class SugestaoJogada {
        +id: int
        +jogador: Jogador
        +movimentoSugerido: Movimento
        +gerarSugestao()
        +aceitarSugestao()
    }
}

```

```

' --- Serviço de Email (Recuperação de Senha) ---

```



```

package "EmailService" {
    class EmailService {
        +boolean sendPasswordResetEmail(email: String, token: String) ' Método
para envio de e-mail de reset
    }
}

' --- View (Interface com o usuário) ---
package "View" {
    interface IRoutes {
        +void createroutes()
    }

    class UserRoutes {
        +void createroutes()
    }

    IRoutes <|.. UserRoutes
}

' *** Relacionamentos ***
Usuario <|-- Jogador
Partida "1" -- "4" Jogador : Participa
Partida "1" -- "*" Movimento : Registra jogadas
Jogador "1" -- "1" Ranking : Está associado
Jogador "1" -- "*" SugestaoJogada : Pode sugerir jogadas
Movimento "1" -- "1" SugestaoJogada : Ligado às sugestões

' Representação da dependência: o UserController utiliza o EmailService para
recuperar a senha
    persistencelayer.controller.UserController ..> servicelayer.EmailService : calls
sendPasswordResetEmail()
}
@enduml

```