

Aplicação demonstradora da Experience API (xAPI): Jogo Quiz

**Equipa:**

* Líder - Mário Ferreira – Func. nº: 2201122
* Verificador – Francisco Barata – Func. nº: 2000360
* Desenvolvedor – Roque Pina – Func. nº: 2101719

**2024**

**Endereço de repositório GIT:**

Este relatório apresenta o projeto da aplicação demonstradora da Experience API (xAPI): Jogo Quiz, desenvolvido pela equipa D3v-N3rds e composta pelos funcionários Francisco Barata, Mário Ferreira e Roque Pina, no âmbito da empresa SimProgramming da disciplina Laboratório de Desenvolvimento de Software. O projeto encontra-se disponível no repositório GitHub, no seguinte endereço:

[**https://github.com/D3v-N3rds/Exemplo.git**](https://github.com/D3v-N3rds/Exemplo.git)

**Descrição geral do projeto**

O projeto propôs-se demonstrar como desenvolver com a Experience API (xAPI) segundo o estilo arquitetônico Model-View-Controller (MVC) – Curry & Grace. Desta forma, utilizámos os conceitos deste estilo arquitetônico para coesão forte; usámos eventos e delegados para obter acoplamento fraco; recorremos a exceções no contexto deste estilo arquitetônico para reagir a problemas de forma prestável ao utilizador; e usámos os conceitos de interfaces para contribuir para separação de interesses e independência entre componentes.

O conceito-base da aplicação demonstradora é a criação de um jogo tipo Quiz que com a utilização da Experience API (xAPI), ficará equipada com uma ferramenta de investigação e análise, capaz de oferecer uma experiência de aprendizagem completa, personalizada e adaptável às necessidades individuais de cada utilizador. Para a funcionalidade do jogo utilizou-se a Tryvia API com a finalidade de obter uma base de dados de perguntas previamente existentes, dividas por categorias e dificuldade.

Para este projeto, o componente Model tem a responsabilidade de interligação com as API’s implementadas e armazenamento do perfil do jogador e da sua respectiva pontuação. O componente View tem como responsabilidade a disponiblização de informação para o ambiente gráfico ao utilizador. O componente Controller processa os input’s do utilizador aliando as componenetes View e Model de forma bidirecional, permitndo a modulação e escalabilidade do programa.

**Funcionamento esquemático**

Para o desenvolvimento da aplicação QuizGame, inicialmente foi optada pela variante Krasner & Pope, no entanto com o decorrer do projeto avaliou-se o modelo Curry & Grace que utilizava a componente View com os input’s e output’s para o utilizador, tornando este modelo o mais adequado ao nosso projeto. Abaixo representa-se o modelo com as responsabilidades de cada componente.

Uma imagem com texto, captura de ecrã, Tipo de letra, diagrama

Descrição gerada automaticamente

Diagrama 1: Interações entre componentes

O processo começa no Controller, mas como os inputs do utilizador são processados pela View, o controlo é transferido para este componente após a inicialização geral da aplicação (incluindo a atribuição de delegados a eventos, etc.). Para que o Controller possa tomar decisões críticas de controlo de fluxo, a interação do usuário passa por ele, conforme indicado pelo diagrama de componentes (Diagrama 1). O fluxo geral de execução é detalhado no diagrama de sequência a seguir (Diagrama 2).

Uma imagem com texto, captura de ecrã, diagrama, Paralelo

Descrição gerada automaticamente

Diagrama 2: Fluxo de execução para QuizGame

No diagrama 2 apresenta o fluxo do jogo pretendido, no qual descreve-se abaixo:

Inicialização do Jogo:

- O processo começa no Controller quando o método LigarJogo é chamado.  
- A View solicita ao utilizador que forneça o nome de Jogador, mostrando um prompt de entrada. Esse input é capturado pela View e enviado de volta ao Controller para processamento.

Processamento do Nome do Jogador:

- O Controller recebe o nome do jogador e faz uma chamada ao Model para pesquisar o perfil do jogador na base de dados através do método PesquisaJogador.

- O Model retorna os dados do perfil do jogador para o Controller, que então apresenta as opções iniciais do jogo na View.

Escolha de Opções:

- Enquanto o jogador não decide sair do jogo (Sair é False), ele interage com os botões da View. Cada clique de botão é capturado e animado pela View.

- A escolha do utilizador é processada pela View e comunicada ao Controller, que então inicia um novo jogo (NovoJogo) ou mostra o ranking (MostraRanking) dependendo da escolha feita.

Conexão Estabelecida:

- O Controller tenta estabelecer uma conexão com a base de dados (representada por BuscaBD no diagrama). Caso a conexão falhe, uma mensagem de erro é mostrada.

- Se a conexão for bem-sucedida, o Controller envia uma solicitação ao Model para buscar perguntas e rankings da base de dados.

Escolha e Envio de Respostas:

- O jogador escolhe uma resposta através da View, que é então enviada para o Controller.

- O Controller envia essa resposta ao Model através do método EnviaResposta, que verifica a resposta (VerificaResposta).

- O Model devolve o resultado da verificação ao Controller, que acumula os resultados e atualiza o ranking (AtualizaRanking).

Animação de Resultados:

- A View anima o resultado da resposta do jogador e mostra as próximas instruções ou resultados acumulados.

Finalização do Jogo:

- Quando o jogador decide sair (Botao Sair), a View comunica ao Controller para encerrar o programa.

- O Controller realiza as operações de limpeza necessárias e encerra o programa.

Funções Principais da API

PesquisaJogador (NomeJogador):

- Função que pesquisa e retorna o perfil do jogador na base de dados, utilizando o nome fornecido pelo jogador.

BuscaBD:

- Método que tenta estabelecer uma conexão com a base de dados para buscar perguntas e rankings.

EnviaResposta (Resposta):

- Função que envia a resposta do jogador para verificação na base de dados.

VerificaResposta (Resposta):

- Função que verifica a resposta do jogador e retorna o resultado da verificação.

AtualizaRanking (Resultado):

- Função que atualiza o ranking do jogador com base nos resultados acumulados.

Entrada de Dados:

- A API recebe dados de entrada através de métodos chamados pelo Controller. Esses dados podem incluir o nome do jogador, respostas selecionadas e outras interações do utilizador.

Processamento e Verificação:

- A API processa as entradas, interage com a base de dados para verificar a integridade e validade das respostas, e realiza cálculos para atualizar os rankings.

Saída de Dados:

- Os resultados do processamento são retornados ao Controller, que os utiliza para atualizar a View e fornecer feedback ao utilizador.

**Divisão de Responsabilidades pelos Componentes**

A implementação foi pensada segundo o método de arquitectura MVC, neste caso, no modelo Curry & Grace na qual abaixo são representadas as responsabilidades:

* Apresenta Opções ao Jogador
* Desenha a Interface Utilizador
* Processa Input’s e Output’s
* Artefatos:
* View.cs
* View.Designer.cs
* Program.cs
* Responses.cs

View

* Liga eventos entre os components Model e View
* Transforma os I/O do Utilizador em ações/eventos
* Gere os erros
* Artefatos:
* Controller.cs
* IController.cs

Controller

* Armazenar perfil jogador
* Verificar respostas
* Atualizar Ranking
* Artefatos:
* Model.cs
* XapiHelper.cs

Model

Figura 1: Responsabilidades entre componentes

**Recurso a eventos para acomplamento fraco**

Após a análise do programa implementado, verifica-se que não existe eventos ou delegados declarados originalmente no código. Este processo deveria ter sido criado, no entanto, priorizou-se a funcionalidade do programa ao invés desta caracteristica.

O código utiliza métodos de interface (IController) e chama diretamente esses métodos a partir da View.

Nesta perspectiva consegue-se afirmar que o programa baseia-se num acomplamento fraco com forte coesão nos componentes.

**Emprego de exceções respeitando o estilo MVC**

A situação de exceção considerada na aplicação foi a de falha de ligação à API.

Este erro era controlado através de uma verificação de conexão pelo Controller, bem como o seu tratamento, dando a opção de um timer durante 30 segundos para reestabelecer a conexão com a API; Em simultâneo é enviado para a View e posteriormente para o utilizador uma mensagem onde é apresentada esta exceção. Caso se verifique a conexão, o Controller passa a informação à View que está funcional dando origem às opções do jogo para interagir com o utilizador.

Abaixo representa-se o diagrama de sequência com a exceção desenhada e respectivamente assinalada em cor verde:

Uma imagem com texto, captura de ecrã, diagrama, Paralelo

Descrição gerada automaticamente

Implementação do tratamento da exceção da conexão da xAPI

Diagrama 3: Implementação de exceção no QuizGame

**Uso de interfaces para indepência entre componenetes**

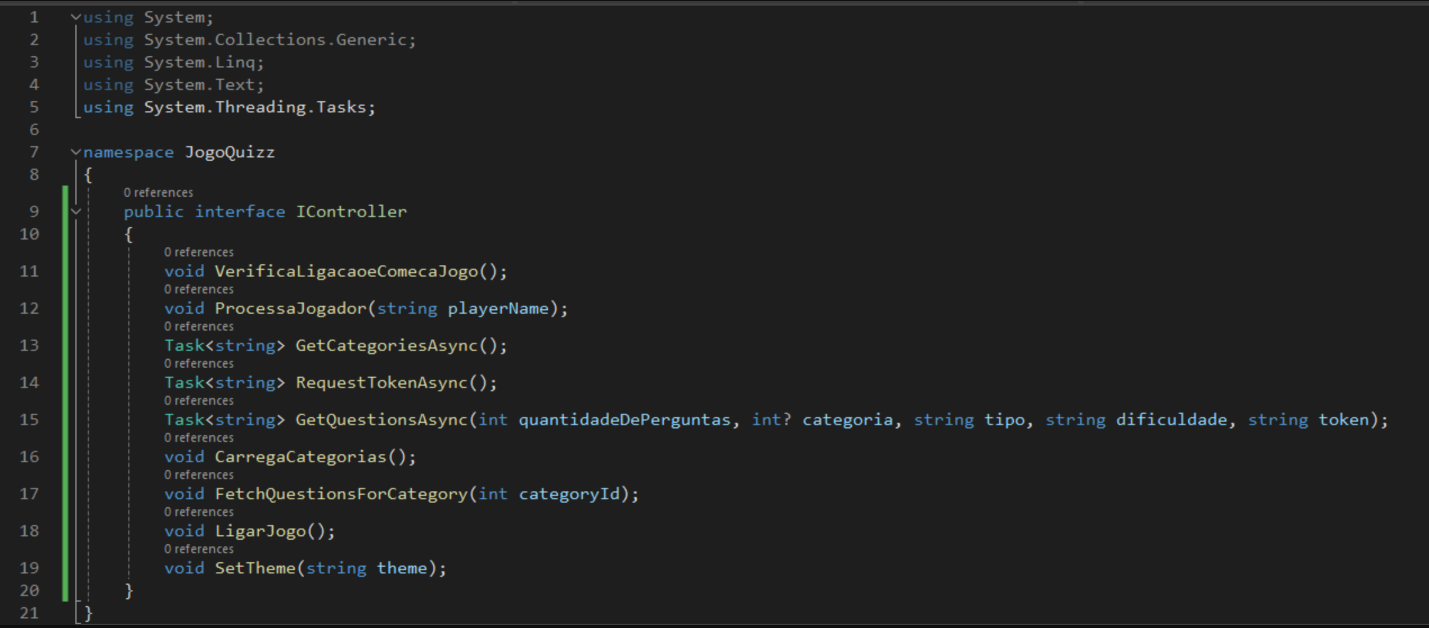
A interface IController define os métodos que qualquer controlador deve implementar. Isso inclui métodos para verificar a conexão, processar dados do jogador, obter categorias e perguntas, e atualizar o tema.

Exemplos de Métodos Implementados:

* VerificaLigacaoeComecaJogo(): Verifica se há conexão com a API e inicia o jogo.
* ProcessaJogador(string playerName): Processa os dados do jogador e inicializa o jogo.
* GetCategoriesAsync(): Obtém categorias de perguntas da API.
* RequestTokenAsync(): Solicita um token da API para autenticação.
* GetQuestionsAsync(int quantidadeDePerguntas, int? categoria, string tipo, string dificuldade, string token): Obtém perguntas da API com base nos parâmetros fornecidos.
* CarregaCategorias(): Carrega categorias de perguntas.
* FetchQuestionsForCategory(int categoryId): Busca perguntas para uma categoria específica.
* LigarJogo(): Inicia o jogo.
* SetTheme(string theme): Define o tema da aplicação.

A utilização de interfaces na aplicação foi uma escolha estratégica para garantir adaptabilidade, manutenção e inteligibilidade. Ao adicionar eventos e delegados, poderíamos ainda melhorar a comunicação entre componentes, promovendo um maior desacoplamento e modularidade.

Abaixo segue o exemplo de interface de indepedência entre componentes:

****

Código 1: Interface Icontroller