

Relatório de Projeto Equipa Sexta Feira 13

Laboratório de Desenvolvimento de Software

Licenciatura em Engenharia Informática

Aplicação demostradora de APIs em Windows Forms:

Autenticação por Facebook & Requisição de filmes por IMDb

Equipa:

* David Ferreira: 2102841
* Pedro Santos: 2000809
* Ruben Silva: 2000583
* Ricardo Pereira: 2100022

03 de junho de 2023

**Endereço do repositório GitHub:**

Neste relatório apresentamos o detalhe e opções do projeto da aplicação demonstradora em Windows Forms: Autenticação por Facebook & Requisição de filmes por IMDb. Este projeto foi desenvolvido pela equipa enunciada na capa de projeto: David Ferreira (Líder), Pedro Santos (Programador), Ruben Silva (Programador) e Ricardo Pereira (Tester). Este projeto foi desenvolvido no âmbito do estágio na empresa SimProgramming. Todo o projeto e documentação podem ser acedidos no GithHub no seguinte link:

<https://github.com/rmlp1983/Grupo13LabDS>

**Descrição Geral de Projeto**

Definindo a aplicação, a mesma prende-se com a criação de uma conta através do login pela API do Facebook, podendo depois disso, o utilizador listar filmes, ver os seus detalhes, requisitá-los e posteriormente devolvê-los e adicionar comentários. A lista de filmes e seus detalhes são provenientes da API do IMDb. Em resumo, esta aplicação demonstradora funciona como um videoclube online que permite ao utilizador posteriormente consultar a lista de filmes já requisitados.

O nosso projeto tem o intuito de demonstrar o potencial da utilização das APIs de Facebook para autenticação e do IMDb para o conceito de filmes e os seus detalhes. Para a sua implementação, de forma a garantirmos uma arquitetura sólida, robusta e escalável, aplicamos o modelo MVC (Model-View-Controller), variante Curry & Grace. Este modelo permite uma separação clara de responsabilidades entre as diferentes partes da aplicação, tal como uma elevada reutilização de código, uma grande capacidade de testabilidade e simplicidade de manutenção. Após análise de comparação com a variante Krasner & Pope, facilmente conseguimos perceber pelos diagramas de sequência que o modelo escolhido é o que apresenta maior simplicidade e facilidade de implementação.

Durante a implementação da aplicação e no decorrer do projeto adotamos técnicas de programação como delegação por eventos (obtenção de fraco acoplamento), implementação de interfaces (independência entre componentes e abstração de implementação) e exceções para tratamento de erros e acontecimentos imprevistos de forma a minimizar o impacto no utilizador final.

A distinção das componentes Model-View-Controller ocorre da seguinte forma no nosso projeto:

* Model: este componente tem a responsabilidade de gerar as respostas relativamente às listas de filmes, detalhes de filme, comentários e requisições, tal como processar os eventos pedidos pelo Controller;
* Controller: este componente tem a responsabilidade de gerir e decidir o que fazer face aos eventos e pedidos dos restantes componentes no modelo, tal como gerir exceções geradas durante os processos;
* View: este componente funciona como uma camada de obtenção/propagação de inputs e outputs, respetivamente. Deve receber os inputs necessários aos pedidos a efetuar e deve apresentar os outputs necessários de acordo com o resultado de processamentos geridos pelo Controller.

De reforçar que estas comunicações e responsabilidades estão assentes no modelo MVC Curry & Grace.

**Funcionamento Esquemático**

Tendo em conta o modelo arquitetónico escolhido e respetiva variante (MVC Curry & Grace), fica implícita a definição de que os inputs do utilizador são processados na View e não há comunicação do Model para o Controller. O seguinte diagrama (Fig. 1) demonstra de forma genérica as responsabilidades de cada um dos componentes do modelo com o detalhe das mensagens trocadas:

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

Descrição gerada automaticamente

*Fig. 1 – Troca de mensagens entre componentes e respetivo detalhe*

O Controller é o componente que inicia o programa, sendo responsável por despoletar o necessário para providenciar a autenticação do utilizador. O utilizador deve inserir nos respetivos inputs os dados de autenticação, sendo que o Controller irá gerir os casos de sucesso e insucesso. Podemos verificar com maior detalhe o fluxo apresentado no seguinte diagrama de sequência (Fig. 2):

Uma imagem com texto, diagrama, Paralelo, file

Descrição gerada automaticamente

*Fig. 2 – Fluxo de iniciação do programa e autenticação do utilizador*

No passo seguinte é gerida a criação de conta de utilizador (no caso de não existir uma), de forma a providenciar os dados necessários para o handling aplicacional de forma a garantir robustez e relação de dados. Podemos ver com maior detalhe o fluxo respetivo (Fig. 3):

*Uma imagem com texto, diagrama, Paralelo, número

Descrição gerada automaticamente*

*Fig. 3 – Fluxo de criação de conta do utilizador*

A partir deste passo podemos aceder às restantes funcionalidades da aplicação. Numa primeira iteração não temos requisições para visualizar, deixando deste modo esta funcionalidade para ser detalhada mais à frente. A primeira funcionalidade de base é a listagem de filmes. Essa listagem é providenciada por uma chamada à API do IMDb efetuada pelo Model, abstraindo assim a complexidade da implementação do seu lado. O tipo de dados a retornar é uma lista de listas de tuplos de forma a tornarmos a comunicação de informação o mais genérica possível e transportando apenas os dados necessários. O fluxo deste passo é representado pelo seguinte diagrama (Fig. 4):

Uma imagem com texto, diagrama, Paralelo, file

Descrição gerada automaticamente

*Fig. 4 – Fluxo de listagem de filmes*

Derivando da listagem de filmes, surge mais uma funcionalidade: obter o detalhe de um filme. Essa funcionalidade tem uma implementação semelhante à anterior, sendo que o Model recorre à chamada da API do IMDb para obter os dados, abstraindo em si a complexidade de tal implementação, não transpondo para qualquer outro componente do MVC tal conhecimento.

O fluxo deste passo é representado pelo seguinte diagrama (Fig. 5):

*Uma imagem com texto, diagrama, Paralelo, file

Descrição gerada automaticamente*

*Fig. 5 – Fluxo de detalhe de um dado filme*

Através das funcionalidades anteriores, o utilizador fica preparado para poder efetuar a sua requisição. Este conceito não tem ligação com APIs, ficando guardado no formato JSON e sendo tratado pelo Model. Desta forma, o fluxo de requisição utiliza dados obtidos durante os processos anteriores (como o identificador do utilizador e do filme) para poder proceder com sucesso. Esse mesmo fluxo fica representado da forma que apresentamos no seguinte diagrama (Fig. 6):

*Uma imagem com texto, diagrama, file, Paralelo

Descrição gerada automaticamente*

*Fig. 6 – Fluxo de requisição de um filme*

A partir do momento que temos pelo menos uma requisição, passa a fazer sentido o uso da funcionalidade de listar requisições. Este conceito está implementado de forma também genérica (recurso a Interface, como iremos detalhar mais à frente), sendo o seu fluxo bastante simples de visualizar e compreender conforme documentado no seguinte diagrama de sequência (Fig. 7):

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

Descrição gerada automaticamente*

*Fig. 7 – Fluxo de listagem de requisições de filmes*

Após a requisição de um filme o utilizador passa a estar habilitado a entregar o filme relativo à requisição. A requisição é atualizada, sendo que passa a existir uma data de entrega. O fluxo de entrega do filme é representado com maior detalhe no seguinte diagrama de sequência (Fig. 8), sendo que o final do fluxo é uma reutilização da funcionalidade de listagem de filmes (finalização do processo acaba com a disponibilização do ecrã de listagem de filmes ao utilizador):

*Uma imagem com texto, diagrama, Paralelo, file

Descrição gerada automaticamente*

*Fig. 8 – Fluxo de entrega de filme*

Tendo em conta o feedback do utilizador relativamente a um dado filme, existe a funcionalidade de adicionar comentários ao mesmo. Essa informação não é submetida por nenhuma API, sendo da responsabilidade do Model manter a mesma em formato JSON para futura disponibilização. Desta forma, o detalhe do fluxo de submeter comentário apresenta-se da seguinte forma (Fig. 9):

*Uma imagem com texto, diagrama, file, Esquema

Descrição gerada automaticamente*

*Fig. 9 – Fluxo de comentar um filme*

Finalizando, o utilizador quando não tiver mais a fazer na aplicação deve poder sair da aplicação, algo que está na responsabilidade do Controller gerir após interação do utilizador com a View a efetuar esse pedido. Desta forma, o fluxo de sair da aplicação é representado da seguinte forma (Fig. 10):

*Uma imagem com texto, file, diagrama, captura de ecrã

Descrição gerada automaticamente Fig. 10 – Fluxo de saída da aplicação*

**Divisão de Responsabilidades e Componentes**

A divisão de responsabilidades e componentes, tal como o previsto, foi feita tendo em conta as três componentes base do modelo arquitetónico escolhido: Model-View-Controller, e segue as ideias base da variante Curry & Grace, ficando então representadas da seguinte forma (Tab.1):

|  |  |  |
| --- | --- | --- |
| MODEL | VIEW | CONTROLLER |
| * *Gerir Dados Autenticação;* * *Gerir Dados de utilizador;* * *Gerir Dados de filmes;* * *Criar Requisições;* * *Registar erros;* * *Artefactos:*   + *MovieModel.cs*   + *RequestModel.cs*   + *UserModel.cs* | * *Mostrar Listas de filmes e detalhes;* * *Mostrar Lista de requisições e detalhes;* * *Tratar inputs/outputs das interações com o utilizador;* * *Artefactos:*   + *LoginView.cs*   + *MenuView.cs*   + *MoviesListView.cs* | * *Conectar todos os eventos e mensagens entre os componentes arquitetónicos Model e View;* * *Processar os eventos e chamadas provenientes da View;* * *Gerir Exceções;* * *Artefactos:*   + *LoginController.cs*   + *MenuController.cs*   + *MoviesListController.cs* |

*Tab. 1 – Responsabilidades de componentes segundo o modelo MVC*

A divisão de responsabilidades permitiu uma abstração bastante interessante ao ponto de não ser necessário o conhecimento por parte da View dos objetos originais lidados pelo Model, já que a informação passa entre componentes com tipos genéricos de forma a aumentar a capacidade de abstração e transparência.

Após análise de responsabilidade de cada componente e dada a informação e fluxos entre eles, obtivemos o seguinte relatório de dependências (Tab. 2):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Componente.Método (que passa dados ou retorna dados)** | **Informação passada** | **Componente.Método (recebedor)** | **Informação necessária** | **Comentários** | **Análise de dependências desnecessárias** |
| Controller.Login | nenhuma | View.Login | nenhuma | Passagem efetuada sem qualquer tipo de atributos (por evento), apenas para notificar a View que deve lançar a funcionalidade | nenhuma |
| Controller.Autenticacao | Username,password | Model.Autenticacao | Username,password | Os campos username e password são os únicos campos necessários para efetuar a autenticação, logo devem ser transmitidos entre componentes | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| View.EstadoLogin | nenhuma | Model.RetornoEstado | nenhuma | Neste caso apenas será necessário confirmar se a autenticação teve sucesso ou não, sendo que o erro é tratado de forma genérica | nenhuma |
| View.ListaFilmes | nenhuma | Controller.ListaFilmes | nenhuma | Passagem efetuada sem qualquer tipo de atributos (por evento), apenas para notificar o Controller que deve lançar a funcionalidade | nenhuma |
| Controller.ListaFilmes | nenhuma | Model.ListaFilmes | nenhuma | Passagem efetuada sem qualquer tipo de atributos (por evento), apenas para notificar o Model que deve lançar a funcionalidade | nenhuma |
| Model.ListaFilmes | Lista de listas de tuplos | View.ListarFilmes | Lista de listas de tuplos | O Model deve retornar uma lista com os dados dos filmes, sendo que, para manter uma implementação genérica, escolhe-se uma lista de listas de tuplos (par nome/valor). | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| View.EscolherFilme | Integer | Controller.EscolherFilme | Integer | A View tem que informar o Controller que foi escolhido um determinado filme, sendo apenas necessário informar-se o MovieId, passando como número inteiro | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| Controller.EscolherFilme | Integer | Model.EscolherFilme | Integer | O Controller tem que informar o Model que foi escolhido um determinado filme, sendo apenas necessário informar-se o MovieId, passando como número inteiro | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| Model.DetalheFilme | Lista de tuplos | View.DetalheFilme | Lista de tuplos | O Model deve retornar os dados do filme, sendo que, para manter uma implementação genérica, escolhe-se uma lista tuplos (par nome/valor). | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| View.RequisitarFilme | Integer | Controller.RequisitarFilme | Integer | A View tem que informar o Controller que se pretende requisitar um determinado filme, sendo apenas necessário informar-se o MovieId, passando como número inteiro | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| Controller.RequisitarFilme | Integer | Model.RequisitarFilme | Integer | O Controller tem que informar o Model que se pretende requisitar um determinado filme, sendo apenas necessário informar-se o MovieId, passando como número inteiro de forma a manter uma implementação genérica. | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| Model.ConfirmacaoRequisicao | Lista de tuplos | View.DetalheReserva | Lista de tuplos | O Model deve retornar à view os detalhes da reserva para que os mesmos sejam apresentados ao utilizador final. De forma a manter a implementação genérica, os dados serão transmitidos através de uma lista de tuplos com par nome/valor para cada um dos atributos. | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| View.ListaRequisicoes | nenhuma | Controller.ListaRequisicoes | nenhuma | Passagem efetuada sem qualquer tipo de atributos (por evento), apenas para notificar o Controller que deve lançar a funcionalidade | nenhuma |
| Controller.ListaRequisicoes | nenhuma | Model.ListaRequisicoes | nenhuma | Passagem efetuada sem qualquer tipo de atributos (por evento), apenas para notificar o Model que deve lançar a funcionalidade | nenhuma |
| Model.ListaRequisicoes | Lista de listas de tuplos | View.ListarRequisicoes | Lista de listas de tuplos | O Model deve retornar uma lista com os dados das requisições, sendo que, para manter uma implementação genérica, escolhe-se uma lista de listas de tuplos (par nome/valor). | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| View.EntregarFilme | Integer | Controller.EntregarFilme | Integer | Passagem efetuada apenas com o atributo do MovieId como número inteiro de forma a manter a generalidade (por evento), apenas para notificar o Controller que deve lançar a funcionalidade de entregar o filme | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| Controller.EntregarFilme | Integer | Model.EntregarFilme | Integer | Passagem efetuada apenas com o atributo MovieId como número inteiro de forma a manter a generalidade (por evento), apenas para notificar o Model que deve tratar a entrega do filme com o MovieId recebido | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| View.FazerComentario | Integer, String | Controller.FazerComentario | Integer, String | A View deve passar o comentário do utilizador ao Controller de forma que este possa gerir da melhor maneira. Os campos necessários são sempre o MovieId que passa como número inteiro e o comentário que passa como string, ambos de forma a manter a generalidade | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| Controller.FazerComentario | Integer, String | Model.InserirComentario | Integer, String | O Controller deve passar o comentário do utilizador ao Model de forma que este possa atualizar os dados. Os campos necessários são sempre o MovieId que passa como número inteiro e o comentário que passa como string, ambos de forma a manter a generalidade | tipos genéricos de forma a generalizar o pedido, transportando apenas os atributos necessários à medida do caso. |
| View.SairAplicacao | nenhuma | Controller.SairAplicacao | nenhuma | Passagem efetuada sem qualquer tipo de atributos (por evento), apenas para notificar a View que deve lançar a funcionalidade | nenhuma |

*Tab. 2 – Relatório de dependências*

**Recurso a eventos para acoplamento fraco**

O recurso a eventos prendeu-se com a necessidade e intenção de obter um acoplamento fraco (potenciando a agilidade para reutilização e mudança). Os componentes emissores e recetores de eventos tiveram na sua base o conhecimento dos seguintes inputs/outputs (Fig. 11):

*Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamenteFig. 11 – Lista de inputs e outputs identificados*

Um exemplo de como os eventos foram implementados está representado nas seguintes figuras (Fig. 12 – View, Fig. 13 – Model, Fig. 14 - Controller):

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente

*Fig. 12 – Evento processado ao nível do componente View*

*Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente*

*Fig. 13 – Evento processado ao nível do componente Model*

*Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente*

*Fig. 14 – Evento processado ao nível do componente Model*

O uso de eventos proporciona-nos vantagens como o desacoplamento, a reutilização de código, extensibilidade e separação de responsabilidades, tudo características que tentamos obter durante todo o desenvolvimento do projeto.

**Exceções no estilo MVC**

O recurso às exceções para o tratamento de erros e acontecimentos inesperados prende-se com a necessidade de centralizar o tratamento dos mesmos num local específico (Controller). Desta forma conseguimos atenuar o impacto desses eventos e tratar da forma mais conveniente para cada um dos cenários que surja. Através desta abordagem conseguimos garantir uma maior robustez e qualidade de código, tal como facilitar o rastreamento de erros.

De forma a demonstrar como esta técnica foi discutida no nosso projeto, apresenta-se o seguinte diagrama de sequência com um exemplo de tratamento de erro por exceção (Fig. 15):

*Uma imagem com texto, diagrama, número, captura de ecrã

Descrição gerada automaticamente*

*Fig. 15 – Fluxo com tratamento de erro por exceção*

Podemos verificar que o Model lança a Exceção, sendo ela capturada pelo Controller, lidando este com a resolução do problema de forma mais eficiente possível, notificando o utilizador com uma mensagem customizada invés de permitir que o programa falhe na generalidade, provocando uma má experiência de utilização.

**Uso de Interfaces para independência entre componentes**

De modo a criar independência e abstração de implementação de classes e tipos, recorremos à implementação de 3 interfaces principais relativamente aos conceitos que são foco da aplicação: IComment, IMovie e IRequest. Estas interfaces estão implementadas no recurso ‘Datatypes.cs’, sendo as respetivas classes implementadas nos seguintes recursos (Tab. 3):

|  |  |  |
| --- | --- | --- |
| **Interface** | **Classe implementadora** | **Recurso de implementação** |
| IComment | public class CommentModel | MovieModel.cs |
| IMovie | public class MovieModel | MovieModel.cs |
| IRequest | public class RequestModel | RequestModel.cs |

*Tab. 3 – Responsabilidades de componentes segundo o modelo MVC*

De forma atirar o máximo partido das interfaces, os tipos de dados retornados e de argumentos entre as respetivas e as classes implementadoras são de tipos genéricos (listas de tuplas para transportar o par nome do atributo/valor, inteiros ou strings). Com isto conseguimos obter um elevado grau de transparência e reutilização do código. Durante o desenvolvimento, conseguiu-se perceber que a flexibilidade obtida com esta abordagem foi superior ao caso de implementação sem Interfaces.

**Reflexão Final**

Durante todo processo de análise e desenvolvimento, aprendemos várias abordagens e ferramentas vitais para o nosso correto crescimento enquanto programadores. De formar mais genérica o que resultou desta fase foi:

* Conseguimos compreender que é sempre necessário estudar bem as vantagens desvantagens dos diversos modelos de arquitetura de forma a compreender qual dos modelos melhor se ajusta às nossas necessidades. Os diagramas de sequência são uma ferramenta muito útil para este exercício.
* Os eventos são uma ferramenta extremamente poderosa de forma a tratar as responsabilidades nos respetivos componentes, garantindo também uma adaptabilidade e flexibilidade tremenda. Consegue-se reutilizar componentes de forma bastante atrativa através desta abordagem, já que passa para o delegado a responsabilidade total de como tratar o caso despoletado.
* As Exceções de erro são uma forma bastante inteligente de gerir e centralizar o tratamento de erros e cenários não esperados. O facto de permitir a capacidade centralizada de decisão com o intuito de reduzir o impacto no utilizador final e melhor a sua experiência de utilização é um dos pontos bastantes favoráveis desta técnica.
* As interfaces garantem-nos um elevado nível de abstração, encapsulamento e uma grande flexibilidade e extensibilidade. Desta forma o processo de implementação torna-se mais simples e fácil de compreender, já que a complexidade fica abstraída na classe, sem interferir com quem utiliza a Interface.

Tendo com base este conhecimento aprendido, de forma mais específica, obtemos as seguintes dificuldades:

* Inicialmente encontramos dificuldades em compreender que melhor modelo se adaptava ao nosso projeto. Foi necessária uma forte análise dos fluxos comparativos e respetivas correções de forma a compreende de que forma realmente cada variante funcionava.
* Relativamente aos eventos, tivemos algumas dificuldades em compreende os cenários corretos em que usar os eventos e não fazer chamadas convencionais. Após uma análise do que realmente poderia ser reutilizável e altamente escalável começamos a compreender melhoro caminho a seguir.
* Numa fase inicial existiram algumas dificuldades nos elementos da equipa com menor conhecimento em C# no que respeita à assinatura das Interfaces. Após rápidos esclarecimentos o problema ficou sanado.
* A complexidade técnica e o facto de integrarmos duas APIs diferentes trouxe alguma dificuldade quanto à obtenção de sucesso na invocação dos seus métodos. Após compreensão das respetivas implementações conseguiu-se obter o esperado.

Em resumo, a nível de trabalho em equipa, conseguimos compreender que o trabalho assíncrono tem de ser bem gerido e planeado, sendo que é preciso ter muito cuidado para não seguirmos por pitfalls. Este é um dos temas que sabemos que devemos melhorar. A comunicação ente todos foi melhorando ao longo do tempo, sendo algo que é um trabalho contínuo e também onde sabemos que podemos melhorar.

Numa vista mais técnica, compreendemos que devemos continuar a praticar todas as abordagens que aprendemos e colocar em prática a aprendizagem obtida de forma a podermos refinar e obter novas aprendizagens a partir daí. Reconhecemos que foi um estágio bastante enriquecedor, em que, além de aprendizagem de conteúdo e experiência no campo, conseguimos observar e compreender pontos mais fracos e de melhoria existentes em cada um de nós.