Ministério da Educação – MEC
Secretaria de Educação Superior – SESu
Programa de Educação Tutorial – PET

Universidade Federal Fluminense – UFF
Escola de Engenharia – TCE
Grupo PET do Curso de Eng. de Telecomunicações – PET-Tele

Projetos PEE PET-Tele

Estudos relativos ao uso do padrão MVC para o desenvolvimento *Web*

(Versão: A2021M07D14)

Autor: Lucca Sabbatini Reid Rodrigues
Gabriel Bueno dos Santos Doria Oliveira

Tutor: Alexandre Santos de la Vega

Niterói – RJ
Julho / 2021

Sumário

Parte I

Introdução

Capítulo 1

Introdução

O Programa de Educação Tutorial (PET) [Pro], do Ministério da Educação (MEC), exige que os grupos PET desenvolvam atividades que contemplem, de forma indissociável, itens de Pesquisa, de Ensino e de Extensão. Além disso, os grupos devem estimular uma evolução positiva dos seus integrantes, dos demais alunos do seu curso de graduação, do próprio curso e da sua instituição. Nesse sentido, o PET-Tele [PET] procura desenvolver atividades e/ou atender a demandas que cumpram tais exigências.

A seguir, são apresentadas as motivações e o objetivo para o trabalho em questão.

1.1 Motivações

A maior motivação para o desenvolvimento dos estudos aqui documentados surgiu da vontade do grupo de compartilhar, com qualquer um que seja interessado neste assunto, todo o conhecimento que foi adquirido sobre desenvolvimento *Web* por parte dos integrantes. O grupo também tem como motivação a criação de conteúdo educacional prático em vídeo sobre o desenvolvimento de aplicações *Web* para disponibilizar para a comunidade. Este documento deverá servir como base teórica para a realização de tal curso.

1.2 Objetivo

O objetivo deste documento é organizar e disponibilizar, para qualquer um que tenha interesse, os estudos do Grupo PET-Tele sobre o padrão de desenvolvimento MVC.

1.3 Resultados esperados

Ao final da leitura deste documento, é esperado que o leitor tenha adquirido conhecimentos básicos sobre arquitetura de *software* e desenvolvimento *Web* , um entendimento teórico sobre o *design pattern* MVC e algumas fontes de conteúdo para que seja possível maior aprofundamento nos assuntos aqui abordados.

Parte II

Contextualização

Capítulo 2

Arquitetura de *software*, *design patterns* e padrão MVC

O primeiro passo, para começar os estudos sobre o padrão MVC, encontra-se nos estudos sobre o planejamento que antecede a implementação de uma aplicação. Para entender esse processo de planejamento, vamos definir os seguintes conceitos: arquitetura de *software* e *design patterns*.

2.1 Arquitetura de *software*

Assim como no processo de construção de um prédio ou uma casa, o primeiro passo para o desenvolvimento de uma aplicação *Web* é o planejamento.

A arquitetura de *software* é o processo de planejamento, bem como o conjunto de decisões tomadas e de soluções escolhidas, que ditam o funcionamento de uma aplicação. Ela deve levar em consideração a escalabilidade, a facilidade de manutenção, a compatibilidade, o desempenho e as demandas que a aplicação deve ter para cumprir seu objetivo.

É uma das partes de maior importância no processo de implementação de uma aplicação, pois é nesta fase que ocorrem as definições das melhores maneiras de abordar os problemas que a aplicação em questão quer solucionar. Caso esta etapa seja executada de uma forma inadequada, a aplicação pode até ser finalizada e até funcionar em um primeiro momento, mas, quando ela necessitar de manutenção, ou quando ela precisar acoplar novas soluções, novos módulos ou até mesmo passar por um processo de refatoração, os desenvolvedores normalmente terão uma enorme quantidade de trabalho desnecessário.

A arquitetura de *software* da aplicação é que dita como os componentes internos da solução comunicam-se uns com os outros e a dificuldade ou a facilidade de manipulação da aplicação. Logo, ela é a pedra fundamental de qualquer projeto.

2.2 *Design patterns*

Na área de desenvolvimento *Web*, mesmo que existam infinitas possibilidades de soluções para infinitos problemas, é possível agrupar tais problemas em classes, levando em conta certas similaridades entre eles.

Vamos supor que um desenvolvedor precise criar uma aplicação para gerir um *website* de vendas de roupas. Ele passará por todo processo de análise e arquitetura, definição de ferramentas que serão utilizadas, modelagem de banco de dados e outros pontos que devem ser fixados antes da implementação da aplicação. Quando estiver passado dessa fase, ele terá um

plano eficiente e condizente com suas necessidades para finalmente implementar a aplicação de maneira correta.

Agora, vamos supor que, passado um tempo, este mesmo desenvolvedor tenha que criar uma aplicação para gerir um *website* de vendas de celulares. Claro que o desenvolvedor não vai copiar a exata solução que ele criou para o caso de venda de roupas, mas é evidente que existem inúmeras similaridades entre estes dois problemas e, consequentemente, entre as duas soluções.

É justamente neste cenário que surgem os *design patterns* (ou padrões de projeto), que são soluções genéricas, com arquiteturas de *software* que já foram pensadas e que se encaixam muito bem para solucionar problemas de uma determinada espécie.

Vale destacar que um padrão de projeto não é um pacote de código pronto para ser utilizado, mas, mais do que isso, é uma estipulação, em alto nível, de uma arquitetura de *software* que funciona bem para um determinado tipo de problema. O código será construído baseando-se nas regras e nas decisões fixadas no padrão de projeto.

2.3 Definição do padrão MVC

Após o estudo dos conceitos de arquitetura de *software* e *design patterns*, é possível entender melhor o que é, de fato, o MVC: um *design pattern* bastante relevante e renomado, capaz de servir de base para a construção de diversos tipos de aplicações, sejam elas com interfaces visuais para o usuário ou até mesmo com fins de servir de intermédio para a comunicação de outras aplicações, como APIs (*Application Programming Interfaces*).

Capítulo 3

A história do padrão MVC

O padrão MVC foi uma das primeiras abordagens pensadas para a implementação de *softwares* com GUIs (*Graphical User Interfaces* ou Interfaces Gráficas de Usuário) para computadores do tipo *desktop*.

A primeira versão de uma aplicação que se utiliza do padrão MVC foi desenvolvida na década de 70, com a linguagem Smalltalk-79 [Kay], desenvolvida pela Xerox. O cientista da computação Trygve Reenskaug deu início ao desenvolvimento do conceito enquanto estava no Centro de Pesquisas da Xerox em Palo Alto, na Califórnia.

Na década de 80, Jim Althoff e outros programadores implementaram uma versão do MVC com Smalltalk-80, uma versão mais nova da linguagem original.

Apenas em 1988 foi publicado um artigo no JOT (*Journal of Object Technology*) explicando o conceito por detrás da arquitetura MVC.

Ao longo do tempo, o *design pattern* evoluiu bastante e deu origem a diversas variantes. A utilização da arquitetura ficou extremamente famosa em 1996, devido à introdução do *framework* para desenvolvimento *Web* NeXT's WebObjects [App], que se utilizava da linguagem Objective-C. Essa popularidade continuou crescendo após a portabilidade do *framework* para o ambiente Java.

Atualmente, diversos *frameworks* para desenvolvimento *Web* utilizam-se de princípios do padrão MVC para a implementação de aplicações, fazendo com que a arquitetura seja extremamente relevante até os dias de hoje.

Capítulo 4

O padrão MVC no desenvolvimento Web

Mesmo que tenha sido pensado para o desenvolvimento de aplicações para *desktop*, o padrão MVC evoluiu bastante e, hoje, é usado majoritariamente no escopo de desenvolvimento de aplicações *Web*. Desde sua invenção, diversos *frameworks* foram desenvolvidos com base em seus princípios.

A partir do MVC, surgiram diversas variantes, que se utilizam de ideias e de conceitos do padrão MVC para a elaboração de novos *design patterns*. Alguns deles são os seguintes:

- *Model–view–adapter*;
- *Model–view–presenter*;
- *Model–view–viewmodel*;
- *Entity-Control-Boundary pattern*;
- *Presentation–abstraction–control*;
- *Action–domain–responder*;
- *Observer pattern*;
- *Strategy pattern*.

Alguns dos *frameworks* mais utilizados na área de desenvolvimento *Web* empregam, de forma completa ou parcial, os conceitos do padrão MVC para a elaboração de aplicações, assim como as linguagens que cada um deles utiliza. Entre eles, podem ser citados os seguintes:

- Laravel (PHP);
- AngularJS (JavaScript);
- ExpressJS (JavaScript);
- Django (Python);
- Ruby on Rails (Ruby).

Parte III

Funcionamento do padrão MVC

Capítulo 5

Introdução técnica sobre o padrão MVC

Como foi definido na parte de contextualização deste documento, a sigla MVC significa *Model-View-Controller* e é um *design pattern* de arquitetura de *software*.

O MVC é um *design pattern* ampla e frequentemente usado por desenvolvedores, uma vez que sua arquitetura promove diversas vantagens e qualidades para o código, caso os seus princípios sejam seguidos da maneira correta.

O MVC apresenta uma lista de vantagens para o desenvolvimento *Web*, tais como:

- Promove escalabilidade;
- Promove encapsulamento: os diversos conjuntos *Model-View-Controller* são independentes entre si;
- Favorece a manutenção do código: é possível alterar cada camada de forma independente, sem afetar o resto da aplicação;
- Fácil modificação da interface: não é necessário reconstruir nem a lógica nem as funcionalidades;
- Bom desempenho: por conta da sua forma modular de construção;
- Boa experiência de desenvolvimento: uma vez que promove um código organizado.

Nos próximos capítulos, os elementos *Model*, *View* e *Controller*, são brevemente discutidos isoladamente e, posteriormente, são abordadas as relações entre eles, para que seja possível compreender o padrão MVC por completo.

Capítulo 6

A camada *Model*

A camada *Model* tem esse nome pois é responsável pela modelagem da solução do problema que a aplicação quer solucionar. Ela é responsável pelas operações que são realizadas com as informações do servidor e do cliente, por trás de toda a aplicação. Toda e qualquer regra de negócio, função e lógica, que pertencem à aplicação são implementadas nessa camada. Ela é essencialmente o núcleo do *software* em questão.

Dentre as funções da camada *Model*, podem-se destacar as seguintes:

- Geração de dados;
- Armazenamento de dados em um banco;
- Recuperação de dados de um banco;
- Manipulação de dados;
- Aplicação de regras de negócios da aplicação.

Assim como as outras camadas do MVC, os *Models* são separados por entidades. Isso significa que existirá um *Model* diferente para cada entidade que faz parte do escopo da aplicação. Por exemplo, em uma aplicação para gerenciar os alunos de uma escola, as funções de cadastrar um aluno e de cadastrar um professor não devem estar no mesmo lugar. Funções relacionadas à entidade aluno devem estar no *Model* dos alunos. O mesmo vale para os professores.

Vale ressaltar que a conexão com o banco de dados da aplicação deve acontecer somente nesta camada. O módulo responsável pela conexão com o banco de dados não deve ser utilizado em nenhum outro tipo de camada de uma aplicação que segue os princípios do MVC.

Para fins ilustrativos, são apresentados, a seguir, dois exemplos de *Model*, condizentes com os princípios MVC e implementados em pseudocódigo.

Exemplo 1 de pseudocódigo para um *Model*

```
/////// Arquivo AlunoModel ///////
AlunoModel() {
    CriarAluno(<informacoes_do_aluno>) {
        retornar database.inserir(
            tabela: "aluno",
            info: <informacoes_do_aluno>
        );
    },

    ExcluirAluno(<id_do_aluno>) {
        retornar database.excluir(tabela: "aluno", id: <id_do_aluno>);
    }
 }
///////// Fim do Arquivo /////////
```

Exemplo 2 de pseudocódigo para um *Model*

```
///// Arquivo ProfessorModel /////
ProfessorModel() {
    CriarProfessor(<informacoes_do_professor>) {
        retornar database.inserir(
            tabela: "professor",
            info: <informacoes_do_professor>
        );
    },

    ExcluirProfessor(<id_do_professor>) {
        retornar database.excluir(tabela: "professor", id: <id_do_professor>);
    }
 }
///////// Fim do Arquivo /////////
```

Capítulo 7

A camada *View*

A camada *View* tem esse nome pois é responsável pela parte visível ao usuário da aplicação. É através desta camada que o usuário consegue requisitar informações à aplicação. Também é através desta camada que o usuário é capaz de visualizar as informações requisitadas. Ela é, essencialmente, o *front-end* da aplicação. Logo, na sua construção. comumente são utilizadas as linguagens de programação HTML, XML e CSS.

Dentre as funções da camada *View*, podem-se destacar as seguintes:

- Servir de interface para que o usuário consiga interagir com a aplicação e requisitar as informações que deseja.
- Expor as informações recuperadas pelo *Model* acionado pelo usuário;

As *Views* também devem ser separadas por entidades, assim como os *Models*. Com isso, cada *Model* será associado a uma *View*, o que contribuirá para a característica de encapsulamento proposta pelo *design pattern*.

Capítulo 8

A camada *Controller*

A camada *Controller* tem esse nome pois é responsável pelo controle da aplicação. Ela é responsável pela entrada e saída de dados na aplicação. Sua função é receber o comando do usuário, acionar o método do *Model* condizente com o pedido do cliente e atualizar a *View* associada, de acordo com a resposta do *Model*. Também é função do *Controller* formatar as informações fornecidas pelo usuário antes de repassá-las para o *Model*, promovendo assim uma padronização de entrada e saída de dados.

Dentre as funções da camada *Controller*, podem-se destacar as seguintes:

- Receber os comandos e os dados de entrada do usuário, por meio da *View*.
- Acionar os métodos do *Model* associado, de acordo com as demandas do usuário.
- Repassar ao *Model* as informações fornecidas pelo usuário, de maneira formatada e organizada.
- Atualizar a *View* associada àquela entidade manipulada pelo usuário.

Os *Controllers* também devem ser separados por entidades, assim como os *Models* e as *Views*. Com isso, cada *Controller* intermediará um *Model* e uma *View*, fechando o ciclo de funcionamento do padrão MVC.

Para fins ilustrativos, são apresentados, a seguir, dois exemplos de *Controllers*, condizentes com os princípios MVC e implementados em pseudocódigo.

Exemplo 1 de pseudocódigo para um *Controller*

```
/////// Arquivo AlunoController ///////
AlunoController() {
    CriarAluno(<informacoes_do_aluno>) {
      AlunoModel.CriarAluno(<informacoes_do_aluno>);
      AtualizarAlunoView();
    },

    ExcluirAluno(<id_do_aluno>) {
      AlunoModel.ExcluirAluno(<id_do_aluno>);
      AtualizarAlunoView();
    }
 }
//////////// Fim do Arquivo ///////////
```

Exemplo 2 de pseudocódigo para um *Controller*

```
///// Arquivo ProfessorController /////
ProfessorController() {
    CriarProfessor(<informacoes_do_professor>) {
      ProfessorModel.CriarProfessor(<informacoes_do_professor>);
      AtualizarProfessorView();
    },

    ExcluirProfessor(<id_do_professor>) {
      ProfessorModel.ExcluirProfessor(<id_do_professor>);
      AtualizarProfessorView();
    }
 }
//////////// Fim do Arquivo ///////////
```

Capítulo 9

O conjunto MVC

Uma vez conhecidas as características de cada camada isoladamente, deve-se entender as relações que podem ser estabelecidas entre elas, compreendendo-se o padrão MVC por completo.

O fluxo do funcionamento de uma aplicação que segue o padrão MVC é o seguinte:

1. O usuário acessa a aplicação através da camada *View* e, por meio dela, é capaz de visualizar suas funções e seus dados. A partir dela, o usuário seleciona uma funcionalidade da aplicação, que será requisitada pela camada *Controller*. O *Controller*, por sua vez, recebe os comandos e os dados de entrada do usuário;

2. Com as informações pertinentes em mãos, o *Controller* aciona os métodos da camada *Model* condizentes com a demanda do usuário, repassando as informações de entrada;

3. O método selecionado do *Model* processa as informações, aplica a lógica da aplicação e as regras de negócio, executa funções e manipula os dados, gerando com isso uma resposta, que, por sua vez, é passada para o *Controller*;

4. De posse da resposta do *Model*, e baseando-se nela, o *Controller* atualiza a *View*, para que o usuário seja capaz de visualizar o resultado da sua requisição;

5. Finalmente, o usuário analisa a resposta da aplicação e, caso seja necessário, reinicia o ciclo com novos comandos e informações, até que o propósito da utilização da aplicação seja cumprido.

A Figura 9.1 ilustra o fluxo do funcionamento de uma aplicação que segue o padrão MVC.

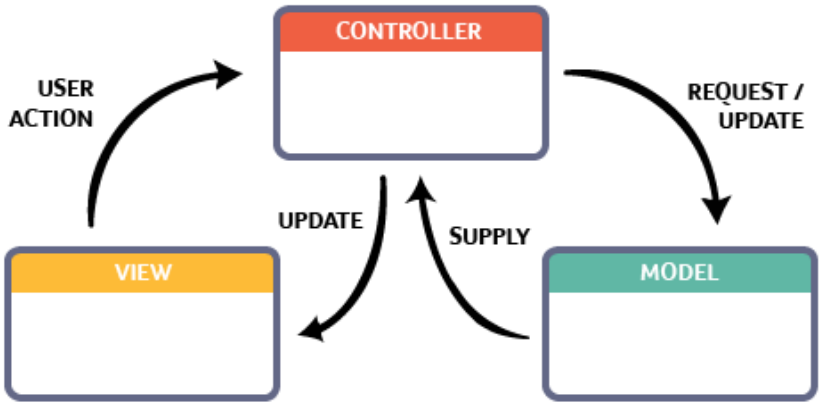

Figura 9.1: Fluxo do funcionamento de uma aplicação que segue o padrão MVC.

Referências Bibliográficas

[App] Apple. *WebObjects Overview*. Disponível em: "https://developer.apple.com/library/archive/documentation/LegacyTechnologies/WebObjects/WebObjects_5/WebObjectsOverview/WebObjectsOverview.pdf". Acesso em: 14/07/2021.

[Kay] Alan C. Kay. *The Early History Of Smalltalk*. Disponível em: "http://worrydream.com/EarlyHistoryOfSmalltalk/". Acesso em: 14/07/2021.

[Lut] Lutti. *Design Patterns – O que são e quais os benefícios?* Disponível em: "https://www.opus-software.com.br/design-patterns/". Acesso em: 14/07/2021.

[NR7] NR7. *Arquitetura de software: sua definição e aplicação nos negócios*. Disponível em: "https://www.igti.com.br/blog/arquitetura-de-software-definicao-negocios". Acesso em: 14/07/2021.

[PET] Grupo PET-Tele. Disponível em: "http://www.telecom.uff.br/pet". Acesso em: 14/07/2021.

[Poi] Tutorials Point. *MVC Framework - Introduction*. Disponível em: "https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm". Acesso em: 14/07/2021.

[Pro] Programa de Educação Tutorial - PET. Disponível em: "http://portal.mec.gov.br/index.php?option=com_conent&view=article&id=12223&ativo=481&Itemid=480". Acesso em: 14/07/2021.

[Wika] Wikibooks. *WebObjects/Overview/History*. Disponível em: "https://en.wikibooks.org/wiki/WebObjects/Overview/History". Acesso em: 14/07/2021.

[Wikb] Wikipedia. *Model–View–Controller*. Disponível em: "https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller". Acesso em: 14/07/2021.

[Wikc] Wikipédia. MVC. Disponível em: "https://pt.wikipedia.org/wiki/MVC". Acesso em: 14/07/2021.

[Zee] Afzaal Ahmad Zeeshan. *Understanding ASP.NET MVC using Real World Example, for Beginners and Intermediate Programmers*. Disponível em: "https://www.codeproject.com/Articles/871375/Understanding-ASP-NET-MVC-using-Real-World-Example". Acesso em: 14/07/2021.