Desenvolvimento de Software utilizando Padrões de Projeto: estudo de caso de um sistema de gerência de contas em repúblicas estudantis

Carlos Henrique Lopes Zansavio, Jugurta Lisboa Filho

carloshlzansavio@gmail.com, jugurta@ufv.br

Universidade Federal de Viçosa, Departamento de Informática
Bacharelado em Ciência da Computação – Trabalho Final de Curso
Viçosa, dezembro de 2018

Resumo

Padrões de projeto possibilitam a experientes programadores registrarem técnicas já testadas e comprovadas de problemas frequentes. Com isso, projetistas conseguem utilizar essas soluções já colocadas à prova, em diferentes ambientes, não tendo a necessidade de partir do zero. O uso desses padrões reduz custo e erros ao processo de desenvolvimento, tornando o mesmo menos complexo com um produto final melhor. Esse trabalho apresenta um estudo de caso utilizando e testando alguns dos padrões bem aceitos pela comunidade visando a implementação de um gerenciador orçamentário para repúblicas estudantis.

1. Introdução

A evolução da informática desafia constantemente os programadores e analistas a desenvolverem novos programas e sistemas de informações. Mas o emprego efetivo das técnicas da engenharia de software nem sempre é alcançado. Há programadores que relutam em aplicar técnicas já comprovadamente validadas. Um dos exemplos dessa falta de uso efetivo de técnicas de engenharia de software é a baixa adoção do uso de padrões de projeto. Normalmente, é possível encontrar sistemas feitos com grande grau de improvisação, gerando produtos de baixa qualidade e com alto custo de manutenção, gerando prejuízo para as empresas e insatisfação aos usuários.

Este projeto final de curso faz um estudo sobre o desenvolvimento de software reutilizando diversos padrões de projeto encontrados na literatura. Como estudo de caso foi projetado e implementado um sistema de gerenciamento de contas em repúblicas estudantis, um problema simples, mas de grande relevância para a comunidade estudantil brasileira.

O termo “república” tem origem do latim *res publica*, ou seja, coisa pública, e expressa bem público, de todos (MORAES & MIRANDA, 2011). Já as primeiras repúblicas remetem ao século XIV, em Coimbra, Portugal. Dom Dinis, por diploma régio em 1309, promoveu a construção de casas que deveriam ser habitadas por estudantes se pagassem um aluguel, cujo valor seria estabelecido por uma comissão nomeada pelo Rei. Com isso, as repúblicas evoluíram e chegaram ao que temos hoje.

Um sistema de gerenciamento de contas em repúblicas estudantis foi escolhido para ser implementado como estudo de caso, visando auxiliar os jovens que saem de casa e precisam apender a controlar as despesas de uma residência compartilhada. República estudantil é uma alternativa viável e barata de morar longe da cidade natal. No entanto, uma dificuldade que gera muitos conflitos quando se escolhe morar em repúblicas estudantis é o controle da divisão das contas. Entre a liberdade de estarem longe dos pais e responsabilidade de cuidar de si, há a grande tarefa de aprender muitas coisas novas. São várias questões e ensinamentos.

No desenvolvimento do estudo de caso foram implementados vários padrões de projeto para posterior avaliação. Há padrões controversos, como ActiveRecord, e outros bem aceitos pela comunidade, como Query Object e Service Layer (FOWLER, 2002). A linguagem de programação escolhida foi Ruby, pela sua simplicidade e por ela ser orientada a objetos com algumas pitadas de programação funcional. Todas as camadas foram divididas em pastas para melhor visualização e controle de código, o qual está disponibilizado no repositório hospedado no Github (ZANSAVIO, 2018).

O restante desse artigo está estruturado como segue: caracterização do problema, banco de dados, padrões utilizados, requisitos implementados e conclusão.

2. Revisão Bibliográfica

2.1 Método de desenvolvimento de software

O método Scrum foi utilizado para projetar e desenvolver o estudo de caso, por ser um método de desenvolvimento ágil para gestão e planejamento de projetos de software (HE:labs, 2013).

Os projetos que seguem essa metodologia são divididos em ciclos que são chamados de Sprints. Um sprint é um pedaço do software que deve ser entregue em no máximo algumas semanas e são revisados diariamente.

Junto com Scrum, também foi utilizado Test driven Development (desenvolvimento guiado a teste). Essa técnica descreve que os testes unitários devem ser construídos antes do código. Além de haver menos erros nos códigos, há também a documentação e exemplos de como utilizar as classes. Desta forma, o código tende a ficar menor, mais bem implementado e mais organizado.

2.2 Padrões de projeto

Quando os tipos de dados começaram a ser estudados nos anos 1970, os projetistas viram que padrões eram sempre repetidos e deixavam a programação mais fácil (Garlan & Shaw, 1994). E desde então, os programadores começaram a catalogar e testar diversos padrões.

“Os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem-sucedidas. Expressar técnicas testadas e aprovadas as torna mais acessíveis para os desenvolvedores de novos sistemas” (GAMMA, HELM, JOHSON, VLISSIDES, 2000).

Portando, padrões de projeto são indispensáveis na construção de software. Eles não dependem de nenhum recurso incomum de linguagem de programação e nem de truques. Eles devem ser simples e fáceis de entender.

As seções seguintes descrevem os padrões utilizados neste projeto.

2.2.1 Model View Controller

Model-View-Controller é um padrão arquitetural que separa o projeto em três camadas: Model, View e Controller.

Model: são classes que representa os dados do projeto. A lógica de validação e regras de negócio também são feitas nestas classes. Seguindo o padrão Active Record, essas classes também fazem as consultas com o banco de dados.

View: são classes que representam a parte visual do projeto.

Controller: são classes que orquestram a camada Model e a camada View. As respostas da Model são tratadas e são passadas para View, onde é feito um tratamento desses dados e apresentados para o usuário final.

O padrão MVC ajuda os desenvolvedores a separarem os aspectos mais importantes de um projeto (logica de entrada, regras de negócio e visualização) e desacopla-los. Esse padrão ainda define onde cada lógica deve ficar localizada. As regras de negócio devem ficar no Model. As regras de visualização na View. E as regras de entrada de dados e afins devem ser tratadas no Controller. Toda essa separação ajuda os desenvolvedores a se planejarem antes mesmo da fase de concepção.

A Figura 1 é ilustra um diagrama de sequência de como um MVC convencional é implementado em aplicações web.

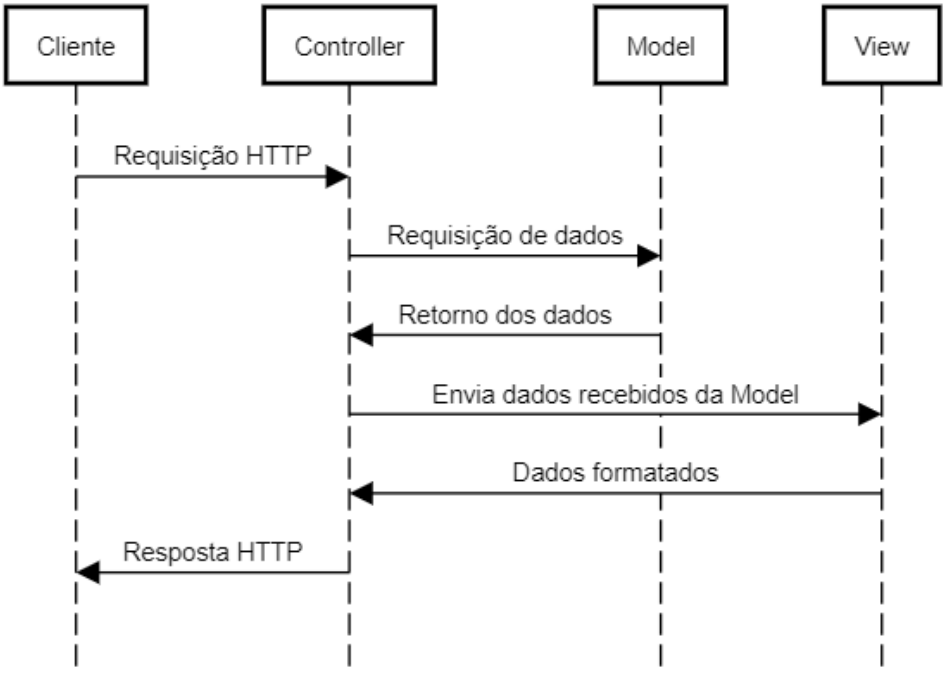

Figura 1 – MVC convencional. Fonte: próprio autor

2.2.2 Active Record

Active Record é um padrão em que toda linha de uma tabela de banco de dados representa um objeto. Nesse objeto há também regras de negócio e regras de como obter esses dados.

Esse padrão é ideal para pequenos projetos e projetos em que não há muita associação de tabelas.

2.2.3 Query Object

Enquanto Active Record é um padrão que concentra todas as operações do banco de dados e regras de negócio em um objeto, o Query Object separa as regras de obtenção de dados em objetos.

Esse padrão é excelente para casos em que é desejado a separação e concentração das consultas do banco de dados em apenas um objeto.

2.2.4 Service Layer

Esse padrão encapsula as regras de negócio em um objeto. Com isso, o desenvolvedor sabe exatamente qual objeto está por trás de um serviço, como a sequência de passos após a criação de algum item.

Além de encapsular muito bem as regras de negócio, há também um ganho em testes unitários.

Esses objetos coordenam toda a aplicação.

O padrão de serviço trabalha muito bem com o MVC, como pode ser visto na Figura 2. Esse padrão separa tarefas em vários objetos deixando a aplicação mais robusta. Sem esse padrão, tudo estaria apenas na camada Model, o que a sobrecarregaria e a tornaria difícil de ser testada.

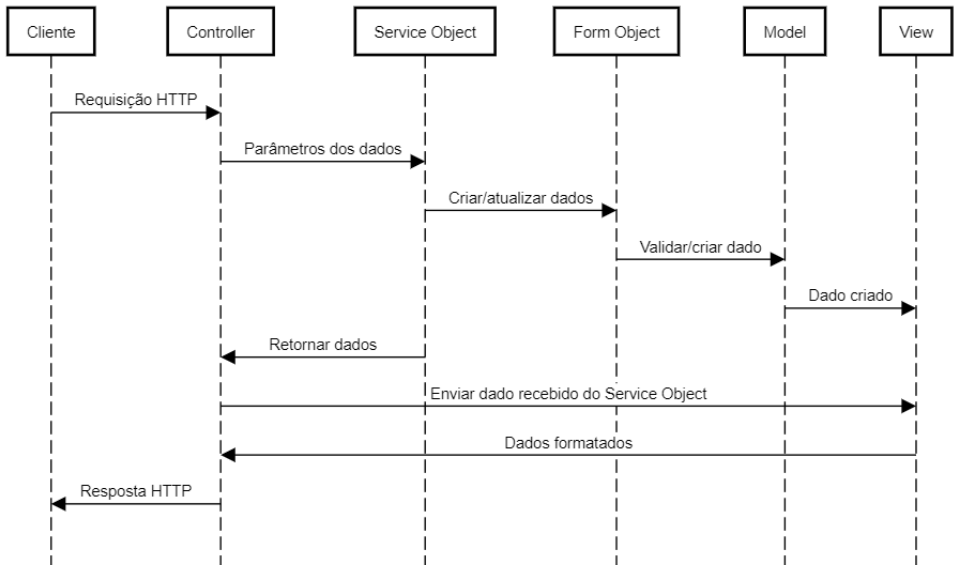

Figura 2 – MVC com camada de serviço e validação. Fonte: próprio autor

Além disso, quando se deseja a obtenção de algum dado devido a alguma consulta (um relatório por exemplo), a Figura 2 muda um pouco. Nele, é introduzido um objeto de consulta para que haja essa ligação entre a camada de serviço e a camada de Model e banco de dados. A Figura 3 ilustra esse tipo de comportamento.

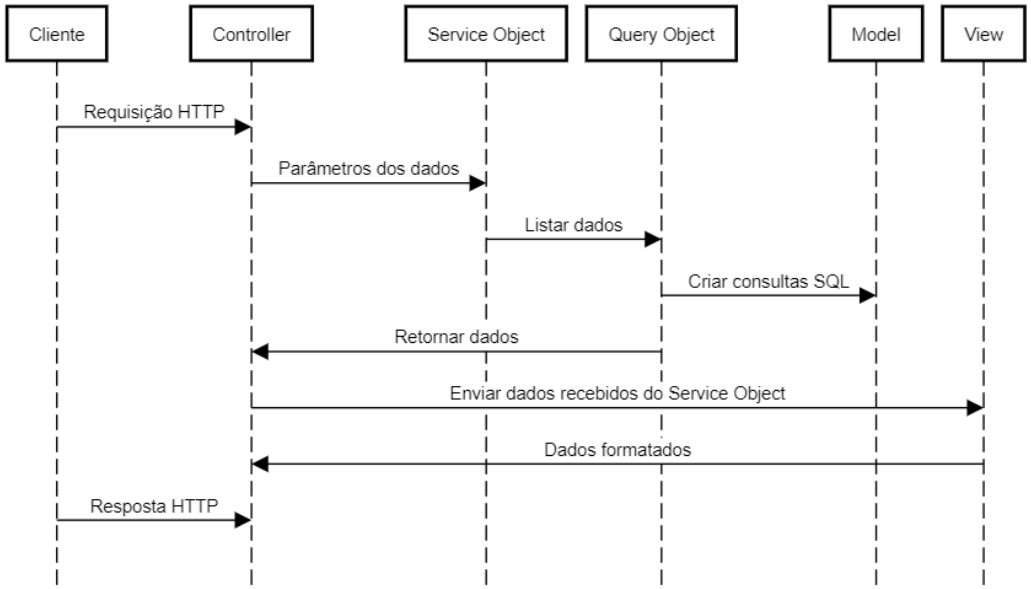

Figura 3 – MVC com camada de serviço e consulta. Fonte: próprio autor

2.2.5 Form Object

A validação de dados em um sistema web é um processo importante. Esse padrão foi criado para encapsular as regras de validação.

Todo objeto que corresponde a uma entidade deve ser validado por um objeto desse padrão. Assim sendo, se ele não for válido ou não passar nos testes, esse objeto entidade não deve ser criado.

2.3. Linguagem de Programação

Para um programador evoluir suas habilidades técnicas, uma das primeiras coisas que ele deve fazer é estudar sobre padrões de projeto. Além dessas técnicas serem universais, elas já foram amplamente testadas e aprovadas. Portanto, mais pessoas podem se envolver com o projeto sem grandes dificuldades e o código ser menos propenso a erros.

Nesse artigo foi preciso escolher em um estudo de caso para assim testar o comportamento desses padrões. Então foi projetado um gerenciador de repúblicas estudantis. A razão dessa escolha é o domínio dessa área que o autor tem. Esse problema é simples, mas de grande relevância já que problemas de divisão financeiras são frequências em ambientes como de repúblicas estudantis, onde existe a falta de experiência de moradores recém-saídos de casa. Além disso, esse sistema pode facilmente ser adaptado para a gestão financeira de uma família.

Outro ponto importante é que o estudo de caso foi desenvolvido utilizando a linguagem de programação Ruby. Criada em 1995 por Yukihiro Matsumoto, é uma linguagem multiparadigma baseada em Smalltalk e focada em metaprogramação.

Para auxiliar foi escolhido o framework Ruby on Rails pela fácil implementação de alguns padrões.

4. Estudo de Caso: Gestor de República Estudantil

Repúblicas sofrem com problemas constantes de gerenciamento orçamentário. Por isso um sistema que faça toda a gerência e controle, gerando relatórios se faz necessário nesse domínio.

Para utilizar o sistema, o usuário deve criar um Tipo de conta em que ele passa o valor da conta e a divisão de contas sugerida entre os moradores.

Posteriormente, todo mês, ele cria uma Conta e a associa com o Tipo de conta correspondente. Com isso, é possível pegar todas as contas de um determinado tipo, como todas as contas de luz.

Há também uma área para a consulta de relatórios que são gerados a partir dos dados coletados pelo sistema.

Existe o mural em que os moradores podem postar recados e também é possível visualizar dados dos moradores, como telefone dos pais ou número de conta bancária.

4.1 Esquema do Banco de dados

A partir do levantamento de requisitos foi elaborado um diagrama ER (Figura 4) para representar a necessidade de dados do sistema.

O diagrama ER contém uma entidade t*ipo de conta*, que constitui uma conta abstrata. Ela será registrada apenas uma vez pelos moradores, informando quanto geralmente é o custo dessa conta no atributo valor. Além disso, na entidade divisão de contas é feita a divisão, que na verdade é uma sugestão de como será feito o pagamento de todas as contas.

Finalmente, todo mês o morador cria um registro na entidade *conta,* associando-a ao seu *tipo de conta*, informando o valor real que foi pago no atributo valor_pago e o valor do mês no valor_mês. Além da data que foi paga no atributo data_paga.

Também há a entidade *ContaMes* que são as tags das contas. O atributo status diz se a conta está aberta ou fechada.

Os recados que os moradores podem criar estão na entidade *recado*. Esses recados desaparecem após um certo período de tempo e ficam disponíveis para todos visualizarem.

A entidade *usuário* serve para fazer o login. Ela está associada 1:1 com a entidade *morador*. Com isso, podem ser feitas alterações no comportamento do login sem afetar diretamente a entidade *morador*. Um exemplo disso, seria um atributo para saber qual o último IP ou a data do último login.

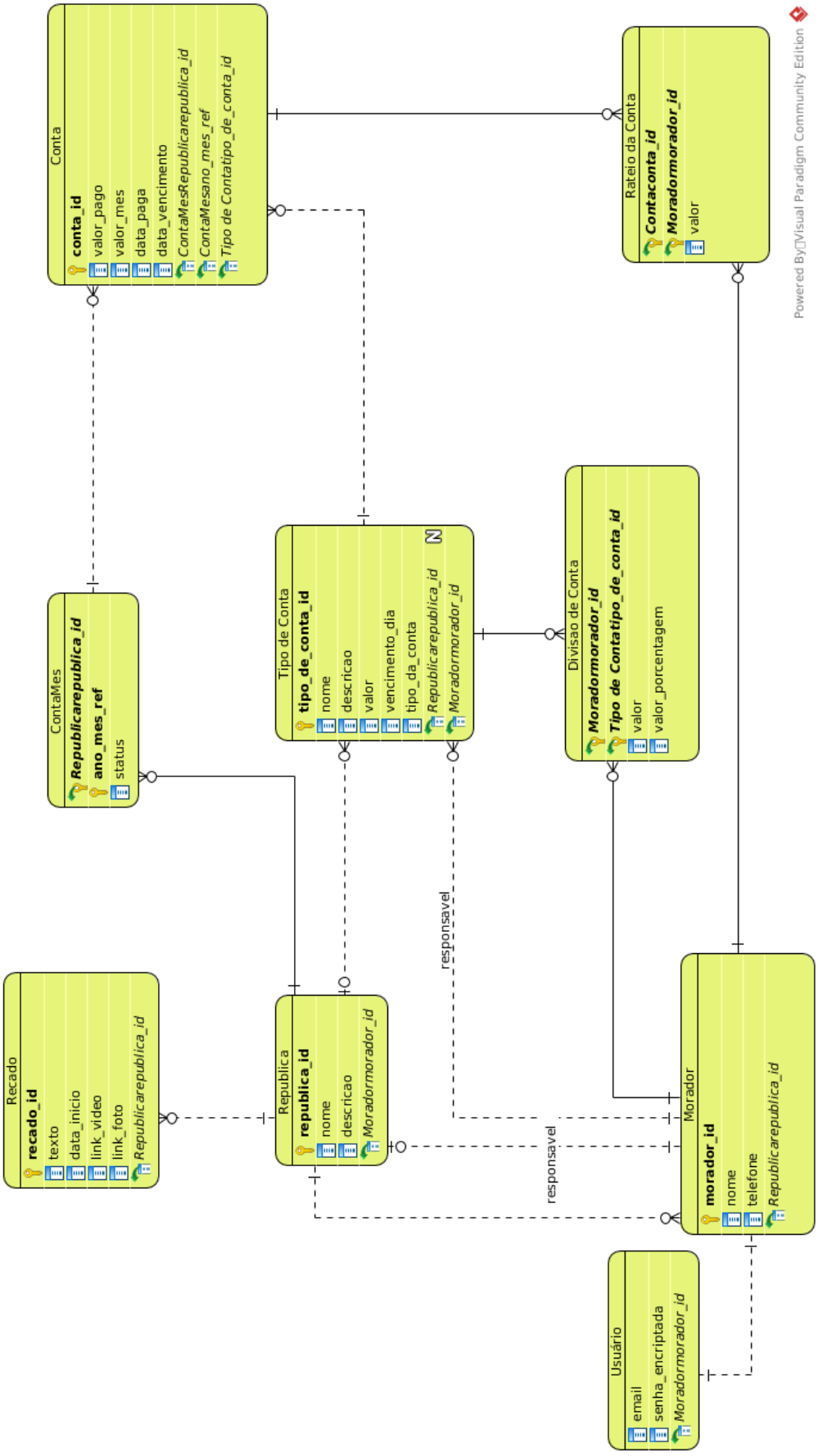

Figura 4 – Esquema conceitual de dados. Fonte: próprio autor

4.2 Requisitos implementados

O sistema desenvolvido neste estudo de caso é um bom exemplo de sistema CRUD. CRUD vem da língua inglesa e significa Create, Retrieve, Update e Delete (KILOV, 1998), e é um termo frequentemente usado para definir as quatro operações básicas de um banco de dados relacionais.

A seguir, estão listados os principais requisitos implementados:

- O sistema deve prover um administrador (ou dono) da república. Ele deve ser responsável pela mesma, tratando e editando informações da república.
- O sistema deve prover o CRUD dos Tipos de Contas bem como um valor sugerido por mês dessa conta e como ela normalmente é dividido entre os moradores.
- O sistema deve prover o CRUD da Conta, bem como o valor real, o rateio que vai haver ou já houve e se já foi paga ou não.
- O sistema deve prover notificações para todos os moradores quando um usuário lança uma conta.
- O sistema deve prover CRUD de um mural de mensagens em que seja possível deixar um recado durante um certo período de tempo.
- O sistema deve prover autenticação de usuários de várias formas.
- O sistema deve prover relatórios em forma textual e gráfica em diversos formatos de arquivo.
- O sistema deve ser multi-línguas.

4.3 Implementação com Padrões de Projeto

O framework utilizado já é MVC. O ORM implementado nele também utiliza o padrão Active Record. Já os outros padrões foram implementados.

A validação de dados é feita a partir do momento que usuário submeter as informações. Para que isso ocorra foram criados quatro forms objects – conta, tipo de conta, divisão de conta e rateio de conta. Cada um desses objetos é responsável por uma validação diferente.

O padrão Service Layer é o que coordena toda a aplicação e por isso foi implementado um objeto para cada entidade importante no sistema. São elas: Conta, Tipo de Conta, Divisão de Conta, Rateio de Conta e criação de usuário.

O serviço de tipo de conta primeiro válida os dados e após isso atrela a responsabilidade da conta para o usuário que a criou.

O serviço conta válida as informações e depois notifica todos os moradores daquela república. Essa notificação pode ser via email, sms ou no celular. A mudança requer apenas uma linha de código.

O serviço de divisão de contas e o serviço de rateio apenas fazem a validação das informações. Ele foi implementado para haver mudanças fáceis no futuro.

Quando há um novo registro, esse registro, a princípio, não cria um novo morador. O serviço de usuário é responsável por isso. Ele orquestra toda essa regra de negócio, passando as informações recebidas para a entidade morador.

5. Discussão do Método Utilizado

O uso de padrões de projeto se torna essencial em projetos de médio a grande porte. Nele são descritas boas práticas já testadas e comprovadas. Foram poucos padrões testados perto dos padrões já catalogados, mas isso não tira a relevância desse trabalho.

Foram estudados cinco padrões de projeto que são realmente utilizados na academia e no mercado de trabalho. A ideia e a implementação de cada um deles é simples, não havendo grandes curvas de aprendizado.

É de fácil percepção que o projeto ficou muito mais simples de ser entendido. Sabendo um pouco da ideia dos padrões, não há grandes dificuldades para que uma nova pessoa possa colaborar com o código, haja visto que cada padrão tem a sua própria pasta – e nela são definidas as classes.

Já a normalização excessiva do banco de dados causou um grande número de consultas. Apenas na tela inicial, são feitas quatro consultas. A longo prazo e com um banco de dados bastante povoado, essas consultas gerariam um tempo de resposta longo.

Uma possível solução seria a criação de Views. Views em banco de dados podem ser consideradas tabelas virtuais que também são armazenadas em cache. Elas também podem ser implementadas com algumas restrições de obtenção de dados. Porém esse método deve ser usado com cautela, já que se haver muitas inserções, a View toda terá que ser recriada.

6. Conclusão

Nesse trabalho foi feito um estudo em padrões de projeto no desenvolvimento de um estudo de caso de um sistema gestor de repúblicas estudantis. Para tanto, foi realizada na fase de concepção um estudo detalhado da especificação de toda arquitetura junto com o detalhamento da implementação. Dessa implementação foi feito o estudo dos padrões e seu comportamento no ambiente em que foi proposto.

Muitos desenvolvedores ainda insistem em não utilizar padrões de projeto. Esse problema também ocorre na Engenharia de Software como um todo, o que resulta em projetos com deadlines comprometidos e profissionais desgastados.

As técnicas aqui citadas foram testadas em um ambiente web, mas podem ser utilizadas em qualquer ambiente. É apenas preciso fazer algumas adaptações tendo cuidado com o contexto que será aplicado.

A contribuição desse artigo foi apresentar o uso dessas técnicas que produzem aplicações com qualidade que certamente sobreviverão no mercado por muito tempo.

Referências Bibliográficas

MORAES, CLAUDIA C. A.; MIRANDA, BRUNA P. Repúblicas estudantis: a tradição como potencialidade turística em Ouro Preto (MG). In: Anais do XXVI Simpósio Nacional de História – ANPUH, São Paulo, julho 2011. Disponível em: http://www.snh2011.anpuh.org/resources/anais/14/1300932593_ARQUIVO_REPUBLICASESTUDANTIS.pdf. Acesso em 08/11/2018.

Gamma, E.; Helm, R.; Johnson R.; Vlissides, J. Padrões de Projetos: Soluções reutilizáveis de software orientados a objetos. Bookman, 2000.

David Garlan and Mary Shaw; An Introduction to Software Architecture, Janeiro 1994. Disponível em: http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf. Acesso em 12/12/2018.

FOWLER, M: Patterns of Enterprise Application Architecture. 1st Edition. Addison-Wesley Professional, (November 15, 2002).

ZANSAVIO, CARLOS H. L: grep. Disponível em: https://github.com/tccdocarlos/grep

He:labs: Scrum: metodologia ágil para gestão e planejamento de projetos. Disponível em: https://www.desenvolvimentoagil.com.br/scrum/

KILOV, H. Business Specifications: The Key to Succesful Software Engineering. Prentice Hall, 1998.