Documento de Projeto de Sistema

Marvin: núcleo

Vitória, ES

2024

Registro de Alterações:

Versão	Responsável	Data	Alterações
1.0	Vítor E. Silva Souza	10/12/2021	Plataforma, RNFs, arquitetura e Camada de Domínio do Problema (CDP).
2.0	Vítor E. Silva Souza	04/02/2022	Camada de Interface com o Usuário (CIU).
2.1	Vítor E. Silva Souza	08/02/2022	Correções na CIU.
3.0	Vítor E. Silva Souza	21/02/2022	Camada de Gerência de Dados (CGD).
3.1	Vítor E. Silva Souza	23/02/2022	Correções gerais, novo diagrama do JButler.

1 Introdução

Este documento apresenta o projeto (*design*) do sistema *Marvin: núcleo*. Trata-se do módulo central do sistema Marvin, responsável pelas funcionalidades básicas, como autenticação, cadastros, etc. Esta especificação foi construída aplicando-se técnicas de especificação dos requisitos não-funcionais, definição de arquitetura e modelagem de sistemas Web utilizando a abordagem FrameWeb (SOUZA, 2020).

Além desta introdução, este documento está organizado da seguinte forma: a Seção 2 apresenta a plataforma de software utilizada na implementação do sistema; a Seção 3 apresenta a especificação dos requisitos não funcionais (atributos de qualidade), definindo as táticas e o tratamento a serem dados aos atributos de qualidade considerados condutores da arquitetura; a Seção 4 apresenta a arquitetura de software; por fim, a Seção **??** apresenta os modelos FrameWeb que descrevem os componentes da arquitetura.

2 Plataforma de Desenvolvimento

Na Tabela 1 são listadas as tecnologias utilizadas no desenvolvimento da ferramenta, bem como o propósito de sua utilização.

Tabela 1 – Plataforma de Desenvolvimento e Tecnologias Utilizadas.

Tecnologia	**Versão**	**Descrição**	**Propósito**
Java	17	Linguagem de programação orientada a objetos e independente de plataforma.	Escrita do código-fonte das classes que compõem o sistema.
Jakarta EE Web Profile	9.1	Conjunto de especificações de APIs e tecnologias que são implementadas por programas servidores de aplicação.	Redução da complexidade do desenvolvimento, implantação e gerenciamento de aplicações Web a partir de seus componentes de infra-estrutura prontos para o uso.
Jakarta Enterprise Beans (EJB) Lite	4.0	API para construção de componentes transacionais gerenciados por *container*.	Implementação das regras de negócio em componentes distribuídos, transacionais, seguros e portáveis.
Jakarta Persistence (JPA)	3.0	API para persistência de dados por meio de mapeamento objeto/relacional.	Persistência dos objetos de domínio sem necessidade de escrita dos comandos SQL.
Jakarta Contexts and Dependency Injection (CDI)	3.0	API para injeção de dependências.	Integração das diferentes camadas da arquitetura.

Tecnologia	Versão	Descrição	Propósito
Jakarta Server Faces (JSF)	3.0	API para a construção de interfaces de usuários baseada em componentes para aplicações Web, seguindo o padrão MVC (*Model-View-Controller*).	Criação das páginas Web e sua comunicação com as classes Java.
Facelets	3.0	API para definição de decoradores (*templates*) integrada ao JSF.	Reutilização da estrutura visual comum às paginas, facilitando a manutenção do padrão visual do sistema.
PrimeFaces	10.0	Conjunto de componentes visuais JSF *open source*.	Reutilização de componentes visuais Web de alto nível.
AdminFaces	1.3	*Template* visual completo integrado ao Facelets/JSF e ao PrimeFaces.	Fornecimento do padrão visual do sistema.
JButler	2.0	Disponível em <https://github.com/dwws-ufes/jbutler/>, é um mini-framework para facilitar o desenvolvimento de aplicações Jakarta EE.	Fornecimento de superclasses prontas para entidades (classes de domínio) persistentes, DAOs e controladores/serviços básicos de cadastro (CRUD).
MySQL Server	8.0	Sistema Gerenciador de Banco de Dados Relacional gratuito.	Armazenamento dos dados manipulados pela ferramenta.
WildFly (Preview)	25.0	Servidor de Aplicações compatível com Jakarta EE 9.	Fornecimento de implementação das APIs citadas acima e hospedagem da aplicação Web, dando acesso aos usuários via HTTP.

Na Tabela 2 vemos os softwares que apoiaram o desenvolvimento de documentos e também do código fonte.

Tabela 2 – Softwares de Apoio ao Desenvolvimento do Projeto

Tecnologia	Versão	Descrição	Propósito
MacTeX	2021	Implementação do LaTeX.	Documentação dos requisitos e do projeto arquitetural do sistema.
TeXstudio	4.1	Editor de LaTeX.	Escrita da documentação do sistema, sendo usado o *template abnTeX*[1] adaptados para documentação de software.[2]
Visual Studio Code	1.62	Ambiente de desenvolvimento (IDE) multiplataforma.	Codificação da aplicação e interface com os sistemas de construção e de controle de versão.
Apache Maven	3.8	Ferramenta de gerência/construção de projetos de software.	Obtenção e integração das dependências do projeto e construção da aplicação para implantação no servidor de aplicação.

[1] <http://www.abntex.net.br>.
[2] <https://bitbucket.org/vitorsouza-ufes/latex-templates/>

Tecnologia	Versão	Descrição	Propósito
GitLab	14.4	Plataforma DevOps para gestão de projetos de software.	Controle de versão do código-fonte, gestão do time de desenvolvimento, implantação contínua.

3 Requisitos Não Funcionais

A Tabela 3 apresenta a especificação dos requisitos não funcionais identificados no Documento de Especificação de Requisitos, os quais foram considerados condutores da arquitetura.

Para efeito de validação, considera-se que as medidas indicadas na tabela serão verificadas de acordo com seus critérios de aceitação durante um período de homologação do sistema junto aos seus principais *stakeholders*, após o qual será considerado entregue e em atendimento aos seus RNFs principais.

Tabela 3 – Especificação de Requisitos Não Funcionais.

RNF-1 – O sistema deve prover funções que atendam aos objetivos de seus usuários, fornecendo resultados corretos e precisos e facilitando tarefas que antes eram feitas com um grau menor de automação.	
Categoria:	Adequação Funcional
Tática / Tratamento:	Desenvolver o sistema utilizando técnicas para o aumento da confiabilidade, como tratamento de exceções, testes automatizados, revisão de código, bem como uso de abordagens ágeis que promovem proximidade com os *stakeholders* para que as funcionalidades entregues sejam o mais próximo possível das desejadas.
Medida:	Aplicação de pesquisa de satisfação com um grupo de 10 a 20 futuros usuários do sistema indicados em conjunto com os *stakeholders*, em que darão notas para as diferentes funcionalidades entregues pelo sistema.
Critério de Aceitação:	Média das avaliações na pesquisa de satisfação igual ou superior a 90%.

RNF-4 – O sistema deve ser fácil de aprender e operar, ser atrativo aos seus usuários e ajudá-los a evitar que cometam erros.	
Categoria:	Usabilidade
Tática / Tratamento:	Reutilizar componentes visuais (*templates*, *widgets*) bem estabelecidos, bem como observar padrões de interface amplamente utilizados e bem sucedidos.
Medida:	Observação de usuários em contato com o sistema pela primeira vez, a ser realizada juntamente com a pesquisa de satisfação mencionada na especificação do RNF-1, medição do tempo que o usuário demora para realizar cada tarefa e a quantidade de erros cometidos ao realizá-las. Inclusão de perguntas sobre a atratividade do sistema na pesquisa de satisfação.

Critério de Aceitação:	O tempo gasto por usuário novatos para realizar as tarefas não deve ser, em média, superior ao dobro do tempo gasto por um usuário treinado. O número de erros por minuto deve ser, em média, inferior a 1. A média das avaliações da atratividade do sistema deve ser igual ou superior a 80%.

RNF-7 – O sistema deve ser protegido contra o acesso de pessoas e sistemas não autorizados, bem como oferecer acesso aos(às) que estiverem autorizados(as).	
Categoria:	Segurança
Tática / Tratamento:	Utilizar um *framework* moderno de autenticação e autorização, observando as melhores práticas como, por exemplo, criptografia forte de senhas. Dedicar tempo para o estudo deste *framework* para sua adequada configuração e uso.
Medida:	Caso seja possível, reunir membros da comunidade acadêmica com experiência em segurança para tentar obter acesso não autorizado ao sistema. Durante a observação dos usuários mencionada na especificação do RNF-4, verificar que os mesmos possuem acesso as funções do sistema relativas aos seus perfis.
Critério de Aceitação:	Nenhum acesso não autorizado durante o testes de segurança. Todos os usuários possuem acesso a todas as funções relativas aos seus perfis durante a observação.

4 Arquitetura de Software

A Figura 1 mostra a arquitetura do sistema *Marvin: núcleo*. Ela é baseada numa combinação dos estilos arquitetônicos Camadas e Partições (FALBO, 2018).

O Marvin tem como base o pacote `br.ufes.inf.labes.marvin`, sob o qual encontram-se o subsistema núcleo — pacote `core` — e os demais subsistemas a serem integrados, representados pelo pacote fictício `outro-módulo`. O núcleo e os subsistemas do Marvin são as partições da arquitetura.

Cada partição é, então, subdividida em três camadas: (1) *Camada de Interface com o Usuário* (CIU), responsável pela interação com os usuários, exibindo os dados e capturando os estímulos externos ao sistema; (2) *Camada de Lógica de Negócio* (CLN), responsável pela representação dos elementos do domínio e implementação das funcionalidades do sistema; e (3) *Camada de Gerência de Dados* (CGD), responsável pela persistência dos objetos em bancos de dados relacionais.

Na CIU, aplica-se o padrão Modelo-Visão-Controlador (*Model-View-Controller* ou MVC) (BASS; CLEMENTS; KAZMAN, 2003), adaptado à plataforma Web, dividindo-a nos pacotes: `view` (visão), que contém as páginas Web e outros elementos visuais (imagens, folhas de estilo, scripts do lado do cliente, etc.); e `controller` (controle), que contém as classes controladoras, responsáveis por intermediar a interação entre os componentes da visão e os elementos da lógica de negócio (considerados a componente *Modelo* no

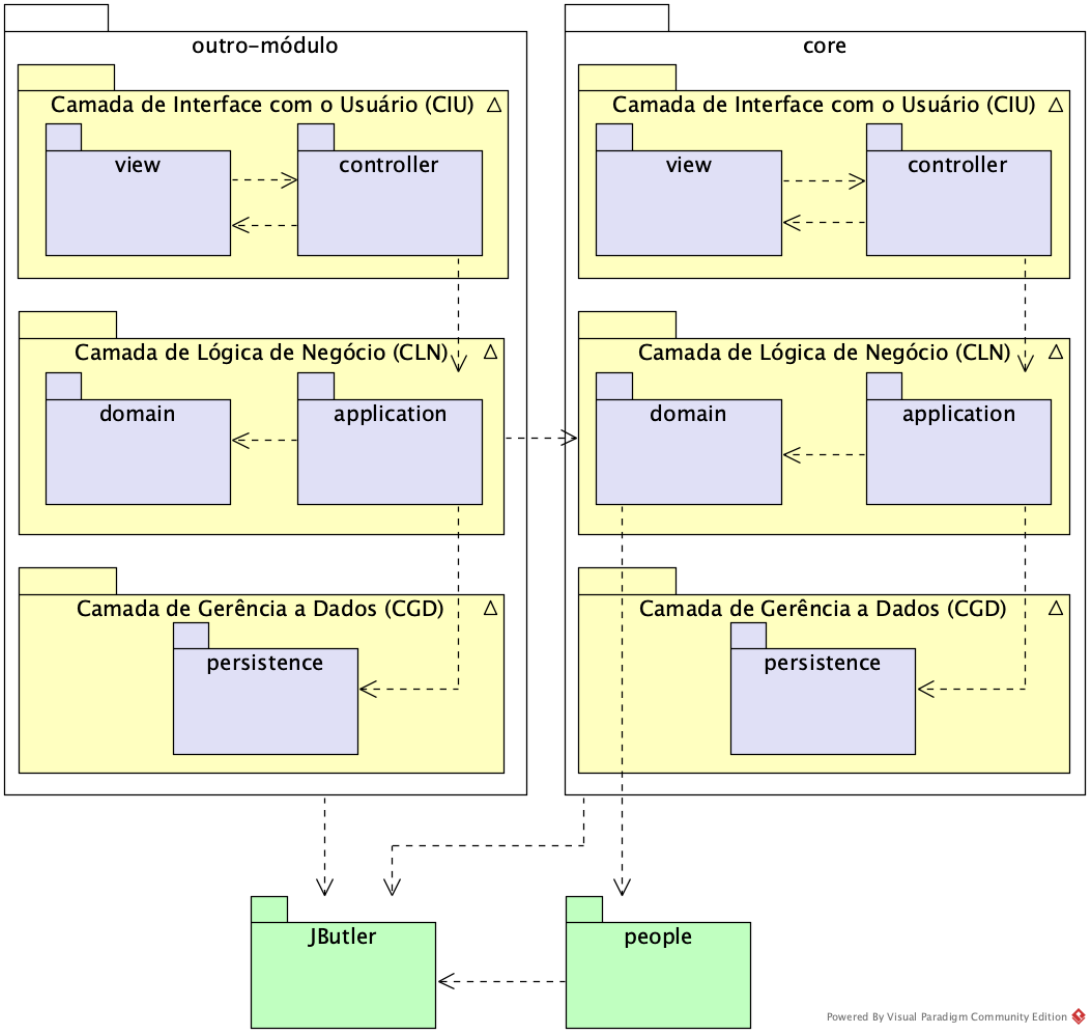

Figura 1 – Arquitetura de Software do *Marvin: núcleo*.

padrão MVC). Visão e controle são interdependentes, enquanto `controller` depende unilateralmente do pacote `application`, mantendo a lógica de negócio independente da interface com o usuário.

Na CLN, aplica-se o padrão Camada de Serviço (*Service Layer*) (FOWLER, 2002), dividindo-a nos pacotes: `domain` (domínio), que contém as classes que representam os elementos do domínio do problema (identificados nos modelos estruturais da fase de requisitos); e `application` (aplicação), que contém classes que implementam as funcionalidades do sistema (representadas nas estórias de usuário e casos de uso na fase de requisitos). Apesar da separação das funcionalidades do sistema no pacote aplicação, deve-se atentar para que o sistema não tenha um modelo de domínio anêmico (FOWLER, 2003). Para implementar suas funções, a aplicação depende do domínio, pois manipula seus objetos, e da persistência, para armazená-los no banco de dados.

Na CGD, aplica-se o padrão Objeto de Acesso a Dados (*Data Access Object* ou DAO) (BAUER; KING, 2006), contendo um único pacote, `persistence` (persistência), cujas classes são responsáveis pelas operações de persistência (utilizando mapeamento

objeto/relacional) de uma única classe de domínio cada.

Os pacotes da Figura 1 fazem referência aos pacotes Java utilizados na implementação do sistema, porém relacionam-se com as diferentes componentes descritas por Falbo (2018), conforme a Tabela 4.

Tabela 4 – Pacotes do Marvin e os componentes equivalentes em (FALBO, 2018).

Pacote	Componente
`view`	Componente de Interação Humana (CIH)
`controller`	Componente de Controle de Interação (CCI)
`application`	Componente de Gerência de Tarefas (CGT)
`domain`	Componente de Domínio do Problema (CDP)
`persistence`	Componente de Gerência de Dados (CGD)

As relações entre os diversos subsistemas e o núcleo do Marvin devem, a princípio, ser feitas entre camadas de lógica de negócio, dependência esta representada no diagrama arquitetônico da Figura 1. Desta forma, elementos de domínio de outros módulos podem se referir aos do núcleo, bem como serviços de outros módulos podem chamar métodos de serviço do núcleo. Apesar deste planejamento, outras dependências podem ser definidas pelos módulos de acordo com suas necessidades, devidamente justificadas. As dependências com o núcleo, no entanto, devem ser sempre unidirecionais, permanecendo o núcleo independente de outros módulos do Marvin.

Por fim, a Figura 1 representa, também, dois utilitários utilizados pelo *Marvin: núcleo*: a componente de domínio do núcleo utiliza classes genéricas que representam informações de pessoas (nome, data de nascimento, telefones, etc.), presentes no pacote `people`. Já o pacote `JButler` representa o mini-*framework* já listado na Tabela 1, que provê superclasses prontas para entidades (classes de domínio) persistentes, DAOs e controladores/serviços básicos de cadastro (CRUD). Como tais superclasses são utilizadas nas três camadas de cada partição, a dependência é registrada como sendo da partição inteira com o JButler.

5 Projeto dos Componentes da Arquitetura

Conforme mostrado na Figura 1, o *Marvin: núcleo* é dividido em 3 camadas, as quais são descritas em detalhes nesta seção.

5.1 Camada de Lógica de Negócio

A Figura 2 mostra o projeto da Componente de Domínio do Problema (pacote `domain`) do *Marvin: núcleo*. Devido à sua dependência com o pacote `people`, o mesmo também é detalhado na figura.

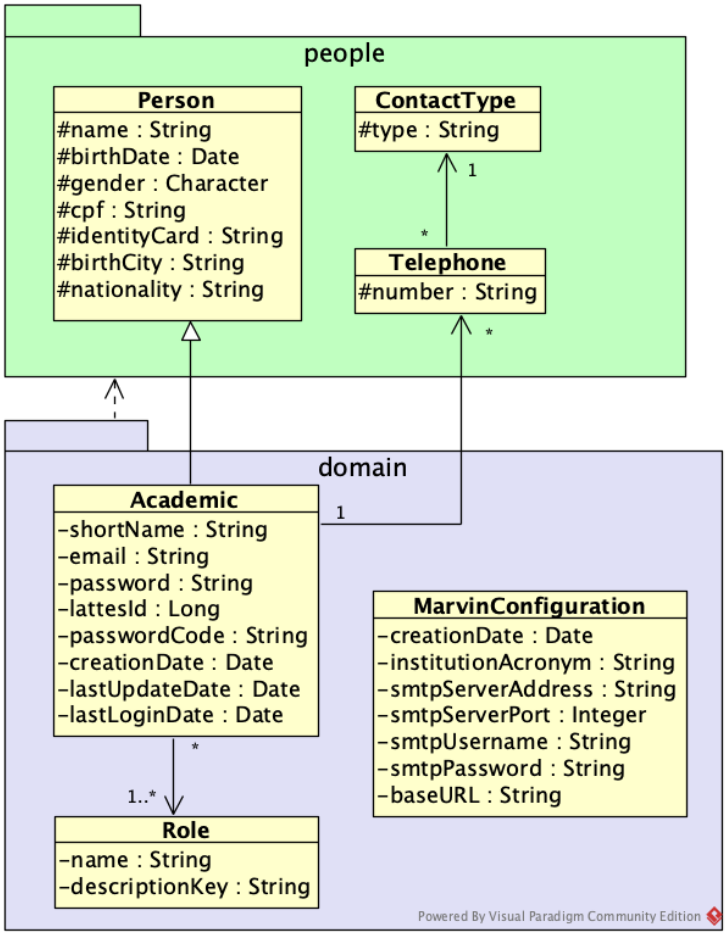

Figura 2 – Projeto da Componente de Domínio do Problema (`domain`) do *Marvin: núcleo*.

Vítor: Descrição das diferenças entre este modelo e o modelo feito na fase de requisitos pendente, pois o modelo de requisitos ainda não foi desenhado.

Conforme já ilustrado na Figura 1, o *Marvin: núcleo* utiliza o utilitário JButler. Especificamente na CDP, todas as classes do pacote `domain` herdam da classe `PersistentObjectSupport` do JButler, representada na Figura 3. Foi escolhido representar as classes do JButler em um diagrama separado e não representar as relações de herança para não poluir muito o diagrama da Figura 2.

A classe `PersistentObjectSupport` traz atributos e métodos úteis para toda classe persistente. Alguns destes elementos foram separados numa outra superclasse, `DomainObjectSupport`. Ambas as classes possuem interfaces que declaram seus respectivos contratos, de modo a permitir o reuso do JButler mesmo para quem não queira usar o mecanismo de herança, como feito no Marvin.

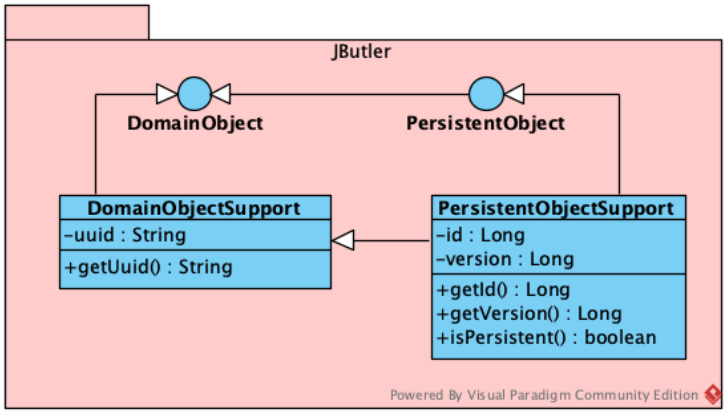

Figura 3 – Classes do utilitário JButler utilizadas pelo projeto da Componente de Domínio do Problema (`domain`) do *Marvin: núcleo*.

Por fim, a Componente de Gerência de Tarefas (pacote `application`) será apresentada juntamente com os elementos da Camada de Interface com o Usuário, na Seção 5.2.

5.2 Camada de Interface com o Usuário

A Figura 4 mostra o projeto da Componente de Interação Humana (pacote `view`) e Componente de Controle de Interação (pacote `controller`) do *Marvin: núcleo*. Devido à sua dependência com a Componente de Gerência de Tarefas (pacote `application`), o mesmo também é representado na figura.

No modelo, elementos que usam o estereótipo *boundary* da UML representam as páginas Web, enquanto os elementos que usam o estereótipo *control* representam classes Java que atuam como controladoras (as que possuem sufixo Controller são *beans* do JSF, já as que possuem o sufixo Servlet são *servlets* Java puros). As páginas Web representam a *View* no padrão MVC e encontram-se no pacote `view`, enquanto os controladores representam o *Controller* do MVC e encontram-se no pacote `controller`. Em conjunto, compõem o *front-end* da aplicação.

Por sua vez, as classes marcadas com o estereótipo *interface* são interfaces Java que representam a porta de entrada do componente *Model* do MVC e fazem parte do pacote `application`. Neste pacote, encontram-se também implementações para cada uma destas interfaces (não representadas no diagrama). Juntamente com a Componente de Domínio do Problema (pacote `domain`) e Componente de Gerência de Dados (pacote `persistence`), compõem o *back-end* da aplicação.

Os controladores possuem associação unidirecional com as interfaces de serviço e, seguindo a arquitetura especificada, estas últimas não conhecem os primeiros, mantendo o *back-end* independente do *front-end*. As relações bidirecionais entre controladores e páginas Web representam a ligação (*binding*) feita pelo JSF entre *tags* presentes nas

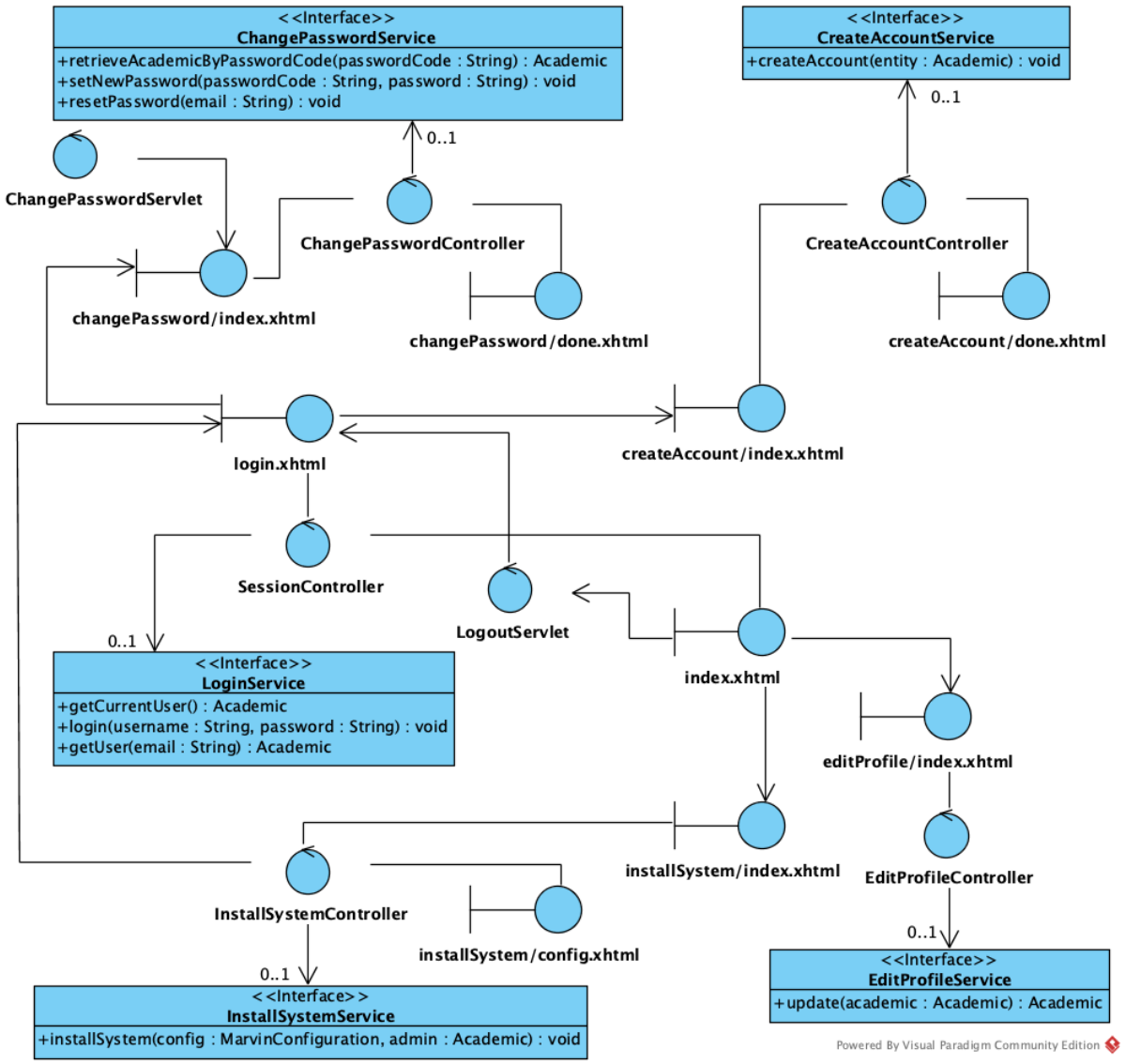

Figura 4 – Projeto da Camada de Interface com o Usuário (`view` e `controller`) do *Marvin: núcleo*, juntamente com a Componente de Gerência de Tarefas (`application`).

páginas XHTML (ex.: campos de formulário) e atributos das classes controladoras: os dados trafegam nas duas direções. Associações unidirecionais entre um controlador e uma página Web indicam um redirecionamento, ou seja, como resultado de alguma operação, o controlador redireciona a requisição para a nova página, porém não troca dados com ela. O mesmo vale para *servlets*, também associados com páginas Web de forma unidirecional. Por fim, associações unidirecionais entre páginas Web representam links entre elas que, similar ao redirecionamento, efetuam uma nova requisição pela página de destino.

Cardinalidades são representadas nas associações entre controladores e serviços e apenas na extremidade que é navegável (do lado do serviço). Demais associações não apresentam suas cardinalidades, pois não envolvem classes, mas sim páginas Web, não havendo sentido na especificação da cardinalidade nestes relacionamentos.

A seguir, para fins didáticos, explicamos com mais detalhes as diferentes funciona-

lidades deste subsistema:

- **Página inicial**: ao ser acessado pela primeira vez, o sistema exibe a página index.xhtml, que neste momento possui link apenas para installSystem/index.xhtml (*Instalar Sistema*). Após o procedimento de instalação do sistema, o *template* de decoração AdminFaces passa a redirecionar o(a) acadêmico(a) para a página de *Login*. Uma vez autenticado, o(a) acadêmico(a) pode acessar novamente a página inicial, porém verá apenas os links para editProfile/index.xhtml (*Editar Perfil*) e LogoutServlet (*Logout*), visto o sistema já ter sido instalado;

- **Instalar Sistema**: a página installSystem/index.xhtml pede alguns dados necessários para instalação do sistema e os envia para InstallSystemController. Se válidos, ele prossegue para installSystem/config.xhtml para mais alguns dados de configuração, retornando para o controlador. Se novamente válidos, é chamado o método InstallSystemService.installSystem() e, em caso de sucesso, o(a) acadêmico(a) é redirecionado para login.xhtml (*Login*);

- **Login**: a página de login apresenta o formulário que pede e-mail e senha do(a) acadêmico(a), bem como links para createAccount/index.xhtml (*Criar Conta*) e changePassword/index.xhtml (*Mudar Senha (Esqueci minha Senha)*). Se preenchido o formulário de login, é chamado o método SessionController.login() e, em caso de sucesso, o(a) acadêmico(a) é levado à página index.xhtml, que puxa dados do(a) acadêmico(a) por meio do SessionController em LoginService.getCurrentUser();

- **Criar Conta**: a página createAccount/index.xhtml pede alguns dados do(a) acadêmico(a) para poder criar uma nova conta, enviando-os para CreateAccountController. Se válidos, ele chama o método CreateAccountService.createAccount() e, em caso de sucesso, mostra a página createAccount/done.xhtml com uma mensagem indicando que um e-mail foi enviado para o endereço informado para validação do cadastro (mesmo procedimento de troca de senha a partir de link/código, descrita a seguir);

- **Mudar Senha (Esqueci minha Senha)**: a página changePassword/index.xhtml pede o e-mail do(a) acadêmico(a) e o envia para ChangePasswordController. O controlador chama o método ChangePasswordService.resetPassword(), que vai atribuir um código (`passwordCode`) ao(à) acadêmico(a) que deseja mudar a senha e enviar este código para ao e-mail cadastrado no sistema, embutido num link. Ao seguir este link, ele acessa o ChangePasswordServlet, que direciona novamente para changePassword/index.xhtml e, sendo o código válido, recupera o objeto `Academic` a partir do código por meio do controlador, que usa o método retrieveAcademicByPasswordCode() do serviço. A página Web agora pede uma nova senha ao(à) acadêmico(a), envia ao controlador, que enfim chama o método resetPassword() e, em caso de sucesso,

mostra a página changePassword/done.xhtml, com uma mensagem informativa de que a senha foi modificada;

- **Editar Perfil**: a página editProfile/index.xhtml mostra os atuais dados do(a) acadêmico(a) e permite que ele(a) os modifique e envie para o EditProfileController. Se válidos, ele chama o método EditProfileService.update(), retornando ao mesmo formulário com uma mensagem de sucesso;
- **Logout**: quando o LogoutServlet é acessado via index.xhtml, a sessão do(a) acadêmico é invalidada (destruída) e ele(a) é redirecionado(a) para login.xhtml.

5.3 Camada de Gerência de Dados

A Figura 5 mostra o projeto da Componente de Gerência de Dados (pacote `persistence`) do *Marvin: núcleo*. Devido à sua dependência com o utilitário `JButler`, o mesmo também é demonstrado na figura.

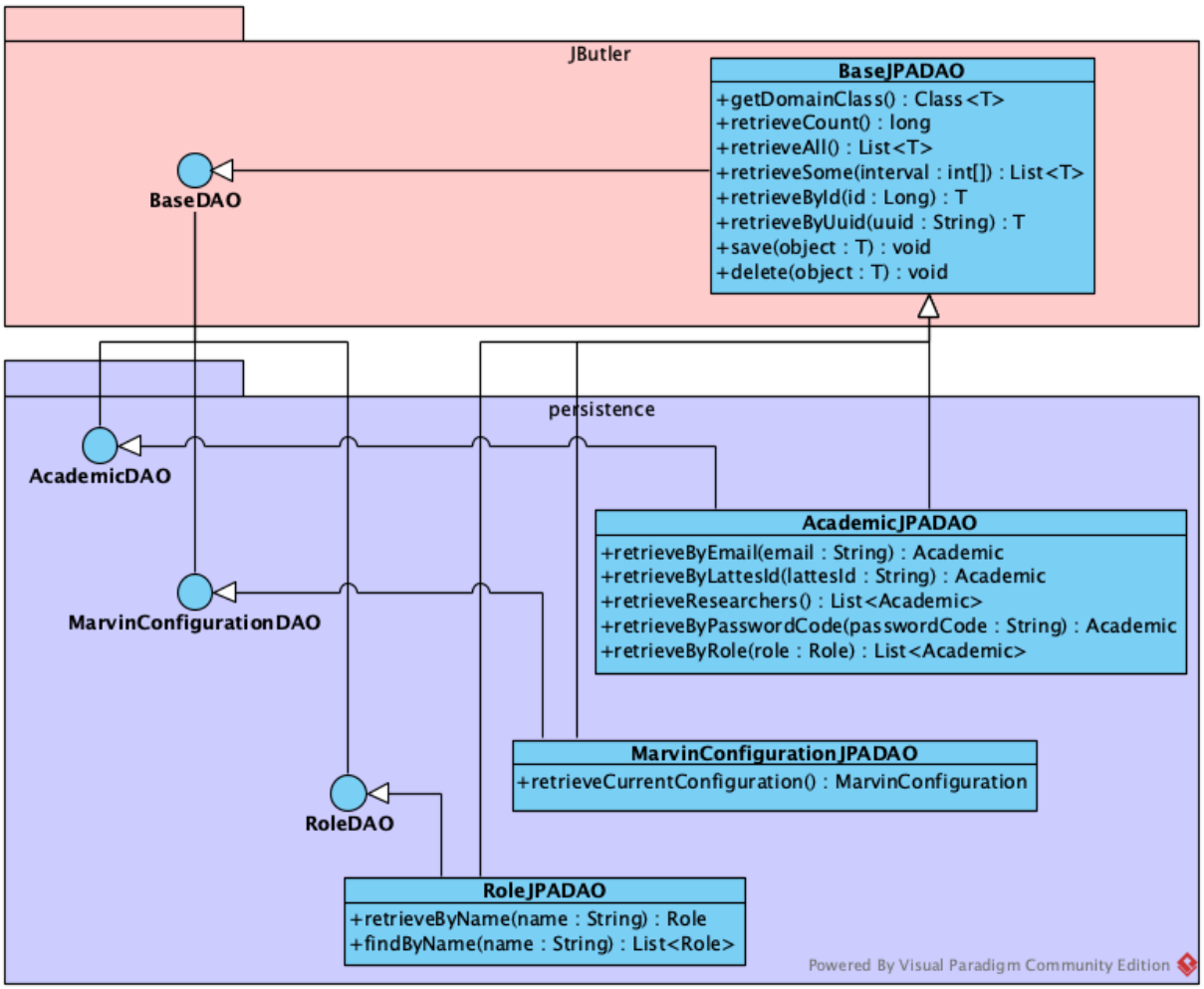

Figura 5 – Projeto da Componente de Gerência de Dados (`persistence`) do *Marvin: núcleo*.

O diagrama mostra duas hierarquias: à esquerda, uma hierarquia de interfaces com `BaseDAO` do JButler na raiz; à direita, uma hierarquia de classes com `BaseJPADAO` do JButler, que implementa `BaseDAO`, na raiz. No pacote `persistence` do *Marvin: núcleo* há uma classe DAO que implementa a respectiva interface DAO para cada classe de domínio do pacote `domain` (vide Figura 2). As interfaces declaram as operações que cada DAO oferece, enquanto as classes implementam os métodos declarados.

O JButler oferece as operações básicas de persistência, que podem ser implementadas de forma genérica, como recuperar a quantidade de objetos persistentes, recuperar todos os objetos ou apenas um intervalo, recuperar objetos a partir de seus identificadores, salvar um objeto como persistente ou excluir um objeto (fazê-lo deixar de ser persistente). Já as classes e interfaces do *Marvin: núcleo* oferecem funcionalidades específicas para as classes de domínio:

- **`AcademicJPADAO`**: recuperar um acadêmico a partir do seu e-mail, ID do currículo Lattes ou código de senha (recebido como parte do procedimento de recuperar senha); ou listar os acadêmicos que são pesquisadores (i.e., possuem ID Lattes) ou que possuem um determinado papel;
- **`MarvinConfigurationJPADAO`**: recuperar a configuração Marvin atual, ou seja, a que foi registrada mais recentemente;
- **`RoleJPADAO`**: recuperar um papel a partir do seu nome exato ou encontrar papeis a partir de parte do nome.

Conforme mencionado na Seção 4, as instâncias das classes de domíno do *Marvin: núcleo* são persistidas por meio de mapeamento objeto/relacional, ou seja, utilizando um Sistema Gerenciador de Banco de Dados Relacional (SGDBR). A Figura 6 mostra o diagrama relacional com o projeto do banco de dados do sistema.

No diagrama, retângulos representam as tabelas, com seus nomes na parte superior e suas colunas logo abaixo. Relações entre as tabelas são representadas por meio da sintaxe *Crow's Foot*. As colunas das tabelas são especificadas com um ícone, o nome e o tipo, dentre os possíveis tipos de coluna do SGBDR MySQL, especificado como banco de dados a ser utilizado neste projeto na Seção 2. O ícone representa outras características das colunas, a saber:

- Chave amarela: chave primária da tabela;
- Chave vermelha: chave primária da tabela que é, ao mesmo tempo, chave estrangeira referenciando outra tabela;
- Losango pintado (azul): coluna que não pode ter valores nulos;

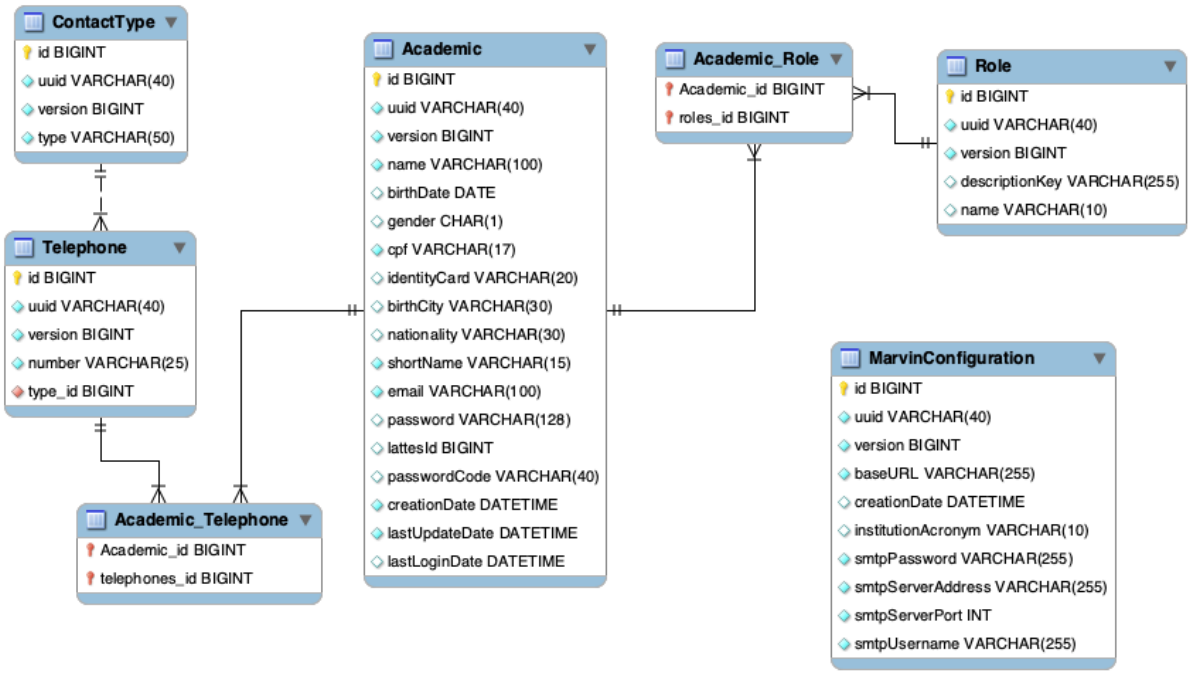

Figura 6 – Diagrama Relacional do *Marvin: núcleo*.

- Losango vazio (branco): coluna que pode ter valores nulos.

O projeto do banco de dados possui uma tabela para cada classe do pacote `domain` (vide Figura 2), com exceção da hierarquia entre `Academic` e `Person`, pois foi escolhida a estratégia de tabela única para toda a hierarquia. Além disso, duas tabelas associativas — `Academic_Role` e `Academic_Telephone` — a primeira necessária para representar a associação muitos-para-muitos entre `Academic` e `Role`, e a segunda como escolha de projeto para representar a associação um-para-muitos entre `Academic` e `Telephone`.

Referências

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 2. ed. [S.l.]: Addison Wesley, 2003. Citado na página 5.

BAUER, C.; KING, G. *Java Persistence with Hibernate*. [S.l.]: Manning, 2006. ISBN 9781932394887. Citado na página 6.

FALBO, R. A. *Projeto de Sistemas de Software - Notas de Aula*. [S.l.], 2018. Disponível em: <http://www.inf.ufes.br/~vitorsouza/falbo/Notas_Aula_Projeto_Sistemas_Falbo_2018.pdf>. Citado 2 vezes nas páginas 5 e 7.

FOWLER, M. *Patterns of Enterprise Application Architecture*. 1. ed. [S.l.]: Addison-Wesley, 2002. ISBN 9780321127426. Citado na página 6.

FOWLER, M. *AnemicDomainModel, https://www.martinfowler.com/bliki/AnemicDomainModel.html (last access: December 10, 2021)*. 2003. Citado na página 6.

SOUZA, V. E. S. The FrameWeb Approach to Web Engineering: Past, Present and Future. In: ALMEIDA, J. P. A.; GUIZZARDI, G. (Ed.). *Engineering Ontologies and Ontologies for Engineering*. 1. ed. Vitória, ES, Brazil: NEMO, 2020. cap. 8, p. 100–124. ISBN 9781393963035. Disponível em: <http://purl.org/nemo/celebratingfalbo>. Citado na página 2.