

UNIFIEO - CENTRO UNIVERSITÁRIO FIEO
CURSO DE ENGENHARIA DA COMPUTAÇÃO

Livar Alves da Hora Filho
Victor Arpa Bock
Wagner Pinheiro
Willian Roberto Carareto

**PLATAFORMA GERADORA DE APLICAÇÕES
ONLINE, COLABORATIVA E EXTENSÍVEL DE ALTO
NÍVEL: CRUDFORGE**

Osasco
2013

Livar Alves da Hora Filho
Victor Arpa Bock
Wagner Pinheiro
Willian Roberto Carareto

PLATAFORMA GERADORA DE APLICAÇÕES ONLINE, COLABORATIVA E EXTENSÍVEL DE ALTO NÍVEL: CRUDFORGE

Trabalho apresentado como exigência parcial para obtenção do título de Engenheiro em Computação do curso de Engenharia da Computação do Unifieo - Centro Universitário FIEO, sob orientação do prof. Dr. Sandro Aparecido Ferraz.

Osasco
2013

FICHA CATALOGRÁFICA

Plataforma Geradora de Aplicações Online, Colaborativa e Extensível de Alto Nível: CrudForge /
Livar Alves da Hora Filho

São Paulo: 2013.

106f.; 30 cm.

Demais autores: Victor Bock, Wagner Pinheiro, Willian Roberto Carareto

Coordenação: Wagner Pinheiro.

Orientação: Profº Dr. Sandro Aparecido Ferraz.

Trabalho Final da Disciplina de Projeto Integrado de Graduação II –

Centro Universitário UNIFIEO, 2013.

Inclui anexo e bibliografia.

1. Engenharia. 2. Software. 3. Plataforma. 4. Colaborativa.

TERMO DE APROVAÇÃO

Livar Alves da Hora
Victor Arpa Bock
Wagner Pinheiro
Willian Roberto Carareto

PLATAFORMA GERADORA DE APLICAÇÕES ONLINE, COLABORATIVA E EXTENSÍVEL DE ALTO NÍVEL: CRUDFORGE

Trabalho de conclusão de curso aprovado como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação da faculdade Unifieo pela seguinte banca examinadora:

Coordenador do curso

Prof. Jean Marcos Laine

Banca examinadora

Professor Jonas Santiago de Oliveira
Membro

Professor Aristides Novelli Filho
Membro

Professor Valdomiro dos Santos
Membro

Osasco, dezembro de 2013

DECLARAÇÃO DE ÉTICA E RESPEITO AOS DIREITOS AUTORAIS

Declaro para os devidos fins, que a pesquisa foi elaborada pelos integrantes deste grupo e que não há, nesta monografia, cópias de publicações de trechos de títulos de outros autores sem a respectiva citação, nos moldes da NBR 10.520 de ago/2002.

Aluno: Livar Alves da Hora		Data
Aluno: Victor Arpa Bock de Freitas		Data
Aluno: Wagner Pinheiro		Data
Aluno: Willian Roberto Carareto		Data

DEDICATÓRIA

Dedicamos esse trabalho primeiramente as nossas famílias, que desde o início nos apoiaram e foram compreensíveis, mesmo nos momentos mais difíceis, nessa longa caminhada de cinco anos de curso. Também gostaríamos de agradecer àqueles que foram importantes em nossas vidas, mas que infelizmente nos foram subtraídos de nossa convivência, mas que de certa forma, de algum lugar, nos mandam energias e consequentemente nos motivam nessa etapa conclusa em nossas vidas.

Também não podemos deixar de dedicar esse trabalho a todos os companheiros de sala, que compartilharam e viveram momentos de alegrias e estiveram juntos nos momentos mais difíceis do curso, além dos professores que nos apoiaram em todas as etapas vencidas para chegar ao presente momento.

AGRADECIMENTOS

Ao professor orientador, aos professores que estiveram envolvidos direta ou indiretamente no desenvolvimento deste trabalho.

Um agradecimento especial às nossas famílias, pela confiança e motivação, além da compreensão em momentos de ausência devido à dedicação ao projeto.

Aos nossos amigos de sala, que juntos conseguimos passar por momentos de dificuldades para concluirmos essa etapa importante de nossas vidas.

À todos que colaboraram para a realização desse projeto.

EPÍGRAFE

"Tudo é mais simples do que você pensa,
e ao mesmo tempo, mais complexo do que
você imagina."

Johann Wolfgang von Goethe

RESUMO

Atualmente no mercado encontra-se uma grande variedade de soluções para softwares e sistemas, tanto para empresas, como para usuários finais de internet. Contudo, estes não tem uma grande flexibilidade para pessoas com pouco conhecimento ou com poucos recursos financeiros. Desta forma, faz com que pequenas empresas ou ainda usuários comuns, contratem serviços que não atendam todas as suas necessidades, com isso acabam adaptando-se ao serviço e não o contrário. De maneira geral, para adquirir um novo serviço, ou será custoso por ter que contratar terceiros, ou ficará preso em um serviço sem a possibilidade de realizar qualquer tipo de customização que for necessário. Levando isso em consideração, é proposta neste trabalho uma plataforma geradora de aplicativos online, colaborativo e extensível, e ainda que seja intuitiva e de baixo custo para o usuário final.

A proposta do projeto, apresentado neste documento, é uma plataforma que tem como principais conceitos: o *CRUD* (*Create, Read, Update, Delete* no banco de dados) e o gerador de aplicação *online*. A demonstração de viabilidade é realizada através de um protótipo, que implementa a arquitetura proposta utilizando estes dois conceitos, no qual o usuário pode definir *schemas*, e através dele o sistema faz o uso do gerador de aplicação *online*, e a partir disso ele pode cadastrar, visualizar, editar e deletar registros em um banco de dados. Ainda é possível a extensibilidade, onde o usuário pode editar um *schema*, adicionando ou removendo campos sem perder a integridade dos dados cadastrados. Como também é possível atuar de forma colaborativa, compartilhando com outros usuários as interfaces, e por consequência os dados, de propriedade do usuário dono do *schema*. A fim de validar este protótipo, foram propostos alguns objetivos específicos, que implementam 3 cenários : “Lista de Tarefas”, “Compartilhamento de coleção de DVDs” e “Contas a pagar e receber”. Sendo que estes cenários foram validados de acordo com os resultados esperados para o final do projeto.

Palavras-chaves: Geração automática, gerador de aplicação *online*, código, colaboração, pequenas e médias empresas, extensível, CRUD, Scaffold

ABSTRACT

Currently the market is a wide variety of solutions for software and systems for both businesses and end-users of the Internet. However, these do not have a lot of flexibility for people with little knowledge or with limited financial resources. In this way, causes small businesses or ordinary users, hire services that do not meet all your needs, with it eventually adapting to the service and not the opposite. In general, to acquire a new service, or will be costly to have to hire a third party, or be stuck in a service without the possibility of holding any kind of customization needed. Taking this into account, is proposed in this paper a generation platform application on-line collaborative and extensible, and that is intuitive and cost effective for the end user.

The project proposal, presented herein, is a platform whose main concepts: the CRUD (Create, Read, Update and Delete in the database) and Scaffold (automatic code generation based on the definition of a database schema). The feasibility demonstration is performed through a prototype that implements the proposed architecture using these two concepts, in which the user can define schemas, and through it the system does the Scaffold, and from this it can create, read, update and delete records in a database. Extensibility is also possible, where the user can edit a schema, adding or removing fields without losing the integrity of the registered data. How can also act collaboratively, sharing with other users interfaces, and consequently the data owned by the user who owns the schema. "Task List", "Sharing DVD Collection" and "accounts payable and receivable": In order to validate this prototype, some specific objectives that implement three scenarios were proposed. Since these scenarios validated with positive results at the end of the project.

Keywords: Automatic generation, scaffold, code, collaboration, small and medium enterprises, CRUD, schemas.

LISTA DE ABREVIATURAS E SIGLAS

CRUD - Create, Read, Update, Delete - Cadastrar, Listar, Atualizar, Apagar

SaaS - Software as a Service - Software como serviço

HTML - HyperTextMarkupLanguage - Linguagem de Marcação de Hipertexto

API - ApplicationProgramming Interface - Interface de Programação de Aplicativos

SGBD - Sistema de Gerenciamento de Banco de Dados

SQL - Structured Query Language - Linguagem de Consulta Estruturada

PHP - Hypertext Preprocessor

UML - Unified Modeling Language

IDE - Integrated Development Environment

ORM - Object Relational Mapping

MVC - Model-View-Controller

POO - Programação Orientado a Objetos

PME - Pequenas e Médias Empresas

ERP - Enterprise Resource Planning

ACL - Access Control List

ORM - Object Relational Mapping

IDE - Integrated Development Environment

XP - Extreme Programming

CSS - Cascading Style Sheets

GUI - Graphical User Interface

PaaS - Platform as a Service

SOA - Service Oriented Architect

EC2 - Elastic Compute Cloud

CASE - Computer-Aided Software Engineering

LISTA DE TABELAS

Tabela 1: Ficha para o caso de uso genérico.....	24
Tabela 2: Ficha do caso de uso ToDoList	27
Tabela 3: Ficha do caso de uso coleção de DVD	29
Tabela 4: Ficha do caso de uso Contas a pagar e receber	31
Tabela 5: Ficha do caso de uso Segurança	33
Tabela 6: Especificação Técnica - Instância t1.micro	52
Tabela 7: Custos de Instâncias	85
Tabela 8: Fluxo de Caixa.....	88

LISTA DE FIGURAS

Figura 1: Práticas da metodologia XP	12
Figura 2: Ciclo do TDD	14
Figura 3: Diagrama de componentes do backend	17
Figura 4: Mockup para a tela de design do CRUD	22
Figura 5: Mockup para a tela do CRUD em execução.....	22
Figura 6: Mockup para a tela de design de relatórios.....	23
Figura 7: Diagrama de caso de uso genérico 1	24
Figura 8: Caso de Uso ToDoList	26
Figura 9: Diagrama de caso de caso de uso para coleção de DVDs D	28
Figura 10: Diagrama de caso de uso Contas a pagar e receber	30
Figura 11: Diagrama de caso de uso Segurança	33
Figura 12: Diagrama de classe do Core	36
Figura 13: Diagrama de Sequência do Core	37
Figura 14: Diagrama de Sequência de Autenticação	38
Figura 15: Diagrama de Sequência de Autorização	39
Figura 16: Diagrama de Componentes do CRUD	40
Figura 17: Construtor e método do serviço Core	42
Figura 18: Método generateCrudAction, do Core Service	42
Figura 19: Método CleanCrudCache	43
Figura 20: Método generateEntity	44
Figura 21: Método setEntityOwner	44
Figura 22: Método updateSchema	45
Figura 23: Método generateCrud.....	46
Figura 24: Definição dos serviços no arquivo services.yml	47
Figura 25: Método fazendo o proxy para o retorno do objeto	47
Figura 26: Método getAclManager	48
Figura 27: Correção de bug do método addPermission	49
Figura 28: Utilização do método checkGrantedClass	50
Figura 29: Utilização do método checkGranted.....	51
Figura 30: Visualização da tela para acesso negado	51
Figura 31: Tela de cadastro de usuários	54

Figura 32: Listagem dos usuários.....	55
Figura 33: Tela de login	55
Figura 34: Criando um schema	56
Figura 35: Listagem dos schemas do usuário	57
Figura 36: Tela adicionando novos campos	57
Figura 37: Tela de cadastro de um novo campo	58
Figura 38: Tela de listagem dos campos de um schema.....	59
Figura 39: Tela onde é visualizado o botão para geração do CRUD.....	59
Figura 40: Tela principal do CRUD gerado pronto para utilização.....	60
Figura 41: Tela de cadastro de dados do CRUD.....	61
Figura 42: Tela de visualização do registro cadastrado.....	61
Figura 43: Tela com a listagem dos registros cadastrados.....	62
Figura 44: Tela para edição de um registro	63
Figura 45: Tela de listagem dos registros após a alteração	63
Figura 46: Tela com listagem dos campos cadastrados.....	64
Figura 47: Tela após a extensibilidade dos dados.....	65
Figura 48: Tela para compartilhamento do CRUD.....	66
Figura 49: Tela com a listagem dos compartilhamentos.....	67
Figura 50: Estrutura Analítica do Projeto	75
Figura 51: Cronograma TCC1	79
Figura 52: Cronograma TCC2	81
Figura 53: Infraestrutura escalável do AWS	90

SUMÁRIO

1. INTRODUÇÃO.....	1
1.1 Tema.....	1
1.2 Situação-Problema.....	2
1.3 Justificativa	2
2. OBJETIVOS	4
2.1 Objetivo Geral.....	4
2.2 Objetivos Específicos	4
3. MATERIAL E MÉTODOS	6
3.1 Engenharia de <i>Software</i>	6
3.2 UML	7
3.3 POO (Programação Orientada a Objetos)	7
3.4 MVC (Model-View-Controller)	8
3.5 CRUD (<i>Create, Read, Update, Delete</i>)	8
3.6 Scaffold	9
3.7 ACL - Access Control List	10
3.8 Engenharia Reversa	10
3.9 SOA	10
3.10 Metodologia XP	11
3.10.1 TDD (Test DrivenDevelopment)	12
3.11 Ferramentas.....	14
3.11.1 Google Drive.....	14
3.11.2 Controle de Versão: Git	15
3.11.3 GitHub	15
3.11.4 LucidChart	15
3.11.5 Astah	16
3.11.6 Eclipse e Netbeans IDE	16
3.12 Arquitetura	16
3.12.1 Ubuntu Server	17
3.12.2 Apache	18

3.12.3 PHP	18
3.12.4 MySQL.....	19
3.12.5 Symfony2.....	19
3.12.6 HTML.....	19
3.12.7 Javascript	20
3.12.8 JQuery	20
3.12.9 JQueryUI	20
3.12.10 AngularJS	21
3.12.11 Twitter Bootstrap	21
3.13 Mockups.....	21
3.14 Diagramas de Casos de Uso	23
3.14.1 Caso de Uso Geral	23
3.14.2 Caso de Uso Especifico.....	26
3.29.2.1 Caso de Uso: ToDoList	26
3.14.2.2 Caso de Uso: Compartilhamento de inventário.....	28
3.14.2.3 Caso de Uso: Contas a pagar e receber.....	30
2.14.3 Caso de Uso Segurança.....	33
3.15 Diagrama de Classe.....	35
3.16 Diagramas de Sequência.....	37
3.16.1 Geração do CRUD (Scaffold).....	37
3.16.2 Autenticação e Autorização.....	38
3.17 Diagrama de Componentes	40
3.18 Serviços	41
3.18.1 Core Service.....	41
3.18.2 Security Service.....	46
3.19 Referências de Hardware	51
3.19.1 Requisito Minimos	52
4. CAPÍTULO - APRESENTAÇÃO DOS RESULTADOS	53
4.1 Testes	53
4.2 Protótipo.....	54

CONCLUSÃO	68
REFERÊNCIAS	70
APÊNDICE I: EAP	75
APÊNDICE II: CICLO DE DESENVOLVIMENTO	77
APÊNDICE III - CRONOGRAMA	78
APÊNDICE IV: ESPECIFICAÇÃO TÉCNICA	83
APÊNDICE V: CUSTOS	85
APÊNDICE VI: PLANEJAMENTO FINANCEIRO	87
ANEXO I: INFRAESTRUTURA ESCALÁVEL AWS	90

1. INTRODUÇÃO

1.1 Tema

Nos dias atuais os principais requisitos exigidos pelo mercado em relação as *software houses* (organizações desenvolvedoras de software) são: a qualidade, o tempo e o custo.

O responsável por garantir os requisitos citados é o processo de desenvolvimento de *software*, sendo que este é dividido em etapas com a utilização de diversas ferramentas e metodologias de desenvolvimento. Porém, existem certos fatores que interferem no sucesso de um projeto.

Jones (1996) caracteriza as causas e sintomas mais pertinentes para falhas de projetos de *software*:

- Falta de entendimento das necessidades reais do cliente;
- Inabilidade de tratar mudanças dos requisitos;
- *Software* com alto custo de manutenção e de expansão;
- Baixa qualidade nos produtos de *software*;
- Descobrimiento tardio de sérios problemas no projeto;
- Desempenho inaceitável;
- Processo de liberação de versão não confiável;

Um dos fatores para o sucesso de um projeto está ligado à usabilidade que normalmente se refere à simplicidade e facilidade com que uma interface, um programa de computador ou um *website* pode ser utilizado.

Segundo Nielsen (1993), facilidade de aprendizagem é o mais importante atributo de usabilidade, pois está relacionado à primeira experiência que qualquer usuário tem com um sistema. Este fator é avaliado em função do tempo que o usuário demora em se tornar experiente na execução de suas tarefas. Outros atributos como eficiência, facilidade de lembrar, erros e satisfação subjetiva também devem ser levadas em consideração.

Tendo em vista este cenário, existem vários desafios associados à criação de uma plataforma capaz de suportar esses requisitos, tais como qualidade, o tempo e o custo.

1.2 Situação-Problema

Atualmente há várias soluções no mercado de *softwares* proprietários, assim como no mercado conhecido como *SaaS* (*Software* como serviço). Esses tipos de *softwares* possuem uma alta complexidade na seleção e na implantação, pois conta com uma grande abrangência das regras de negócios. Com isso existe a necessidade de contratação de consultores para que o projeto seja executado de forma eficiente, o que pode inviabilizar o investimento de pequenas empresas devido aos altos custos.

Outra opção encontrada no mercado é o *software* de prateleira, que por sua vez, trata-se daquele produzido em larga escala e de maneira uniformizada, não tendo qualquer diferencial de personalização para determinado usuário. Desse modo, o cliente necessita se adaptar ao *software*, como *por exemplo*, o *MS Access* (LIRAA, 2013).

Uma terceira via, é a implementação de sistemas com o apoio de plataformas geradoras de código com um alto nível de abstração como o *IBM Rational Rose*, o *MS Sharepoint*, o *PhpMaker* e o *UniPass*, porém, ainda há a necessidade de contratar terceiros com o conhecimento sobre os mesmos, além de um alto investimento.

1.3 Justificativa

Entende-se que não há a necessidade de pequenas empresas fazerem a aquisição de sistemas complexos, onde para o seu uso seria necessário o apoio de terceiros, o que poderia causar um custo elevado do projeto, tornando na maioria das vezes inviável. Além disso, alguns sistemas são muito complexos ao ponto de serem difíceis de entender e usar, com isso gera uma demora na aprendizagem e uma frustração por parte do usuário.

Levando em consideração esses aspectos técnicos, financeiros e usuais, o objetivo desse projeto é desenvolver uma plataforma geradora de aplicativos on-line para pequenas empresas, como também para o usuário final da internet, com uma interface amigável e intuitiva.

2. OBJETIVOS

2.1 Objetivo Geral

Desenvolver uma plataforma capaz de prover a qualquer usuário uma forma mais simples e intuitiva de gerenciar seus dados através de um sistema *online*. Mesmo o usuário não tendo um grande conhecimento sobre desenvolvimento de *softwares*, ele será capaz de modelar seu próprio sistema da maneira que lhe convir.

Basta o usuário fazer o *input* dos seus dados, definir o *schema*, que é a definição da estrutura desses dados inseridos e gerar o CRUD para fazer a sua gestão.

Dentro da plataforma o usuário poderá, através da criação de *schemas* e geração de CRUDs, criar a sua aplicação, apenas passando os seus dados como parâmetros de entrada, já que toda a parte de modelagem está em uma camada mais técnica, sendo executada embaixo da interface que o usuário terá acesso.

2.2 Objetivos Específicos

O objetivo é desenvolver uma plataforma que cria de forma automática um ambiente de controle e gerenciamento de pequenos negócios, no ambiente da internet. Com o intuito de exemplificar o uso do sistema, serão criados três cenários no ambiente:

1. **Lista de Tarefas (*To-Do List*):** o usuário poderá criar uma lista de tarefas que será compartilhada entre diversos usuários, para isso é necessário definir o *schema* através de uma interface amigável e o usuário poderá compartilhar com os seus colaboradores, através do ACL, uma lista de suas tarefas;
2. **Compartilhamento de DVDs:** o usuário poderá criar um sistema para cadastro de seus DVDs, e compartilhar com os seus amigos apenas com a permissão de leitura dos registros cadastrados;

3. **Contas a pagar e receber:** desenvolver um *CRUD* que implementa um controle de contas a pagar e receber básico, e que pode ser utilizado em qualquer empresa. O usuário pode estender esse *CRUD* adicionando campos que fazem sentido para a empresa dele ou removendo outros campos, de acordo com as necessidades.

3. MATERIAL E MÉTODOS

A arquitetura do projeto é um dos pontos relevantes a serem definidos neste projeto, portanto, neste capítulo são descritos os elementos de *softwares* e métodos que serão utilizados ao longo do projeto.

3.1 Engenharia de *Software*

A Engenharia de *Software* surgiu devido a necessidade do mercado de produzir *softwares* com um grau de qualidade elevado, há aproximadamente 5 décadas, no ano de 1963.

O processo de desenvolvimento de *software* muitas vezes é confundido com o ato de programar, o que causa assim uma distorção do que realmente é esse campo de atuação.

Essa abordagem é muitas vezes colocada devido ao fato de muitos profissionais começarem suas atividades através do conhecimento técnico centrado, ou seja, eles desenvolvem habilidades de raciocínio lógico, por meio de estrutura de dados e linguagens de programação. Desse ponto de vista não há nada de errado. Porém, se levarmos em consideração o fato de que os projetos aumentam gradativamente a sua complexidade, e conseqüentemente o número de problemas a serem resolvidos, essa prática, que tem uma visão específica, acaba não sendo mais a ideal. Dessa forma, este seria um bom caminho a ser seguido em situações simples e pontuais, como um único algoritmo que realiza operações simples, porém, não é indicado para a solução de problemas em grande escala ou com um grau de complexidade elevado.

Segundo Pressman (2011), a Engenharia de *Software* entra com a responsabilidade de organizar a complexidade de maneira que os projetos de *software* sejam estruturados corretamente, pois não envolve apenas uma solução específica, mas sim o projeto como um todo.

Em resumo, o principal objetivo da Engenharia de *Software* é oferecer as melhores práticas para o desenvolvimento de *software*, além de métodos para o gerenciamento dos projetos que serão aplicados em todas as escalas, desde a

concepção até a finalização, aumentando assim a qualidade do produto de *software*, além de aumentar a sua produtividade no processo de desenvolvimento.

Com foco na qualidade do desenvolvimento da plataforma descrita nesse documento, a Engenharia de *Software* busca os melhores meios e práticas para a viabilização do projeto (PRESSMAN, 2011).

Com a Engenharia de *Software* buscamos englobar a eficiência em diversos quesitos, por exemplo, as tecnologias utilizadas na elaboração do projeto, os recursos, a qualidade do produto final, através de técnicas para gerência, desenvolvimento, manutenção, entre outras atividades.

3.2 UML

UML (*Unified Modeling Language*) é a linguagem de modelagem adotada internacionalmente pela indústria de *software*, tendo como base a orientação a objetos. Através dela é possível fazer o desenvolvimento de diagramas de classes, de objetos, de casos de uso, entre outros. Esses diagramas são úteis para o entendimento e o desenvolvimento de um sistema (GUEDES, 2011). Através da UML, foram gerados alguns dos diagramas necessários para o entendimento lógico do projeto.

3.3 POO (Programação Orientada a Objetos)

Programação orientada a objetos é um conjunto de princípios, ideias, conceitos e abstrações utilizadas para o desenvolvimento de uma aplicação, ela fornece alguns benefícios como facilitar a manutenção de aplicações, diminuir a complexidade de desenvolvimento de sistemas, e promover o reaproveitamento de código (K19, 2013).

3.4 MVC (Model-View-Controller)

O paradigma MVC (*Model-View-Controller*) é utilizado para separar as camadas de modelo, visão e controle de um sistema. A camada de modelo contém classes que implementam as informações do domínio de negócios da aplicação, já na camada de visão, são definidas as regras de apresentação dos dados para o usuário, temos também a camada de controle, onde é processado as requisições realizadas pelo usuário.

Com a separação da aplicação em três camadas, obtêm-se uma série de vantagens ao desenvolvedor, uma delas é a de permitir reutilizar um mesmo objeto de modelo em diversas visualizações diferentes (RIVERA, 2013).

O MVC é de extrema importância para o presente projeto, pois o *Scaffold* gera as classes separando as camadas definidas no MVC e dessa forma provém uma estrutura escalável caso o usuário tenha necessidade de escalar o seu projeto para um sistema completo e desacoplado da plataforma do CRUDForge, além disso a manutenção do código fonte possui uma baixa complexidade, se comparada a outras arquiteturas, uma vez que seus pacotes são modulares e seu desenvolvimento é feito de maneira ágil.

3.5 CRUD (*Create, Read, Update, Delete*)

Um número cada vez maior de aplicações tem como objetivo permitir que um usuário execute as tarefas de Adicionar, Recuperar, Atualizar e Remover registros existentes em algum banco de dados (TIWARI, 2011), a esse tipo de implementação de interface dá-se o nome de *CRUD* (acrônimo do inglês para *Create, Read, Update, Delete*). Considerando que essa implementação de funcionalidade é atendida de forma simples e de baixo nível pelo banco de dados, com uma interface com poucos recursos e geralmente através de linha de comando, faz-se necessário a implementação

de uma plataforma sólida e com recursos avançados de controle de usuário, como por exemplo, uma matriz de responsabilidades (*RACI - Responsibility Matrix*), (DOW, TAYLOR, 2010), e mesmo com a possibilidade de implementação clara e testável das regras de negócios para atender a responsabilidade de tais *softwares*, e ainda considerando que os desenvolvedores atuais buscam uma forma de automatizar tal implementação (TIWARI, 2011), visto que é uma atividade repetitiva e entediante, conclui-se por inferência que existe um mercado a ser explorado para uma plataforma que execute essa função de geração de interfaces CRUD's, sendo esta a proposta do projeto CRUDForge.

Considerando que tal funcionalidade possui um grau de complexidade elevado, visto o alto nível de abstração necessário para que tal tarefa seja executada, segundo Ramos (2006), "A abstração consiste em focalizar os aspectos essenciais inerentes a uma entidade, ignorando propriedades acidentais [...] O uso apropriado de abstração permite que um mesmo modelo conceitual (orientação a objetos) seja utilizado para todas as fases de desenvolvimento de um sistema, desde sua análise até sua documentação" (RAMOS, 2006), sendo assim devemos escolher uma ferramenta que suporte esse alto nível de abstração, a qual deve simplificar o processo de geração de código orientada ao objeto.

3.6 Scaffold

O recurso de *scaffold* de aplicações é uma técnica que permite ao desenvolvedor definir e criar uma aplicação básica que possa inserir, selecionar, atualizar e excluir objetos. O *Scaffold* possibilita que os desenvolvedores definam como os objetos estão relacionados entre si além de como criar e destruir estas relações (POTENCIER, 2009).

3.7 ACL - Access Control List

O ACL - Access Control List (Lista de Controle de Acesso) é um recurso de sistema utilizado para fazer o controle de acesso a classes e suas instâncias (objetos) do sistema. Com ele é possível restringir e permitir o acesso de qualquer usuário a qualquer objeto do sistema, (POTENCIER, 2013).

É preciso fazer um mapeamento inicial do sistema para fazer a identificação de todos os objetos existentes no sistema, além de fazer a configuração do sistema que o ACL deverá utilizar.

3.8 Engenharia Reversa

A engenharia reversa nada mais é que um conjunto de atividades que permitem, a partir de uma solução de *software* existente, extrair todos os conceitos ali empregados. Esses conceitos podem ser padrões arquiteturais utilizados, diagramas de classes, a arquitetura do sistema, enfim, qualquer informação que contribua para um entendimento do sistema desenvolvido (DEV MEDIA, 2013).

A engenharia reversa foi utilizada nas classes responsáveis pelo *Scaffold*, pois a documentação dos exemplos de geração de CRUD apenas utilizando comandos em um terminal.

3.9 SOA

SOA (*Service Oriented Architecture*), é um conceito de Arquitetura Orientada a Serviço, ou seja, é uma arquitetura de sistemas no qual as funcionalidades ali existentes devem ser desenvolvidas e disponibilizadas em forma de serviço (ERL, 2009).

Uma das características do SOA é que a maioria desses serviços é disponibilizada através de um barramento de serviços (*ESB - Enterprise Service Bus*), ou seja, esse barramento disponibiliza os serviços através de interfaces, ou contratos (*WSDL - Web Service Description Language*), que são acessados através

de *Web Services*. Desta maneira, os objetos não são acessados diretamente no banco de dados, garantindo assim um maior nível de segurança.

Outro ponto a se destacar é o reaproveitamento de código, uma vez que um serviço disponibilizado no barramento de serviço pode ser acesso por outros serviços, ou seja, não há a necessidade de desenvolver um serviço para cada ação ou sistema, pois várias aplicações podem acessar o mesmo serviço, isso porque, um serviço que não é reutilizado não carrega particularidades ou regra de negócio específica (ERL, 2009).

3.10 Metodologia XP

Para este projeto, foi utilizado os conceitos disponíveis na metodologia ágil, mais especificamente a Programação extrema (do inglês eXtreme Programming), ou simplesmente XP com a definição de um ciclo de desenvolvimento com entregas parciais.

Segundo Teles (2005), XP é um processo de desenvolvimento de *software* voltado para:

- Projetos cujos requisitos são vagos e mudam com frequência;
- Desenvolvimento de sistemas orientados a objeto;
- Equipes pequenas, preferencialmente até 12 desenvolvedores;
- Desenvolvimento incremental (ou iterativo), onde o sistema começa a ser implementado logo no início do projeto e vai ganhando novas funcionalidades ao longo do tempo.

O XP é um processo de desenvolvimento que busca assegurar que o cliente receba o máximo de valor de cada dia de trabalho da equipe de desenvolvimento. Ele é organizado em torno de um conjunto de valores e práticas que atuam de forma harmônica e coesa para assegurar que o cliente sempre receba um alto retorno do investimento em *software* (TELES, 2005).

Segundo Beck (2004), os cinco valores fundamentais da metodologia XP são: comunicação, simplicidade, *feedback*, coragem e respeito. Esses valores no geral se referem ao relacionamento entre o cliente e a equipe.

De acordo com Beck (2004), as seguintes práticas devem ser seguidas na metodologia XP: Cliente Presente, Jogo do Planejamento, *Stand Up Meeting*, Programação em Par, Desenvolvimento Guiado por Testes, *Refactoring*, Código Coletivo, Código Padronizado, *Design* Simples, Metáfora, Ritmo Sustentável, Integração Contínua e *Releases* Curtos. Essas práticas representam aquilo que as equipes XP fazem diariamente (a *figura 1* apresenta todas as práticas do XP de forma resumida).

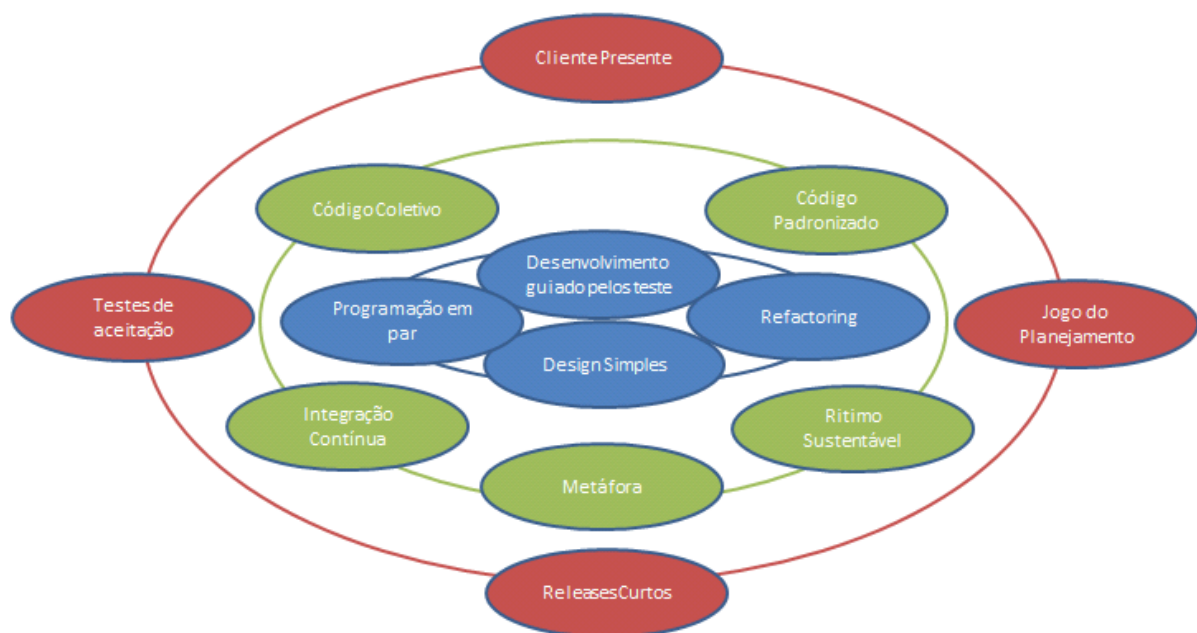


Figura 1: Práticas da metodologia XP

Além disso, uma equipe que utiliza o XP, normalmente é composta por pessoas que representam os seguintes papéis: Gerente de Projeto, *Coach*, Analista de Teste, Redator Técnico e Desenvolvedor (BECK, 2004). Desta forma, os papéis para esse projeto foram divididos seguindo este conceito, como pode ser visualizado no cronograma do projeto (APÊNDICE III).

3.10.1 TDD (Test Driven Development)

O TDD (Test Driven Development - Desenvolvimento orientado a teste) é parte da metodologia XP.

O TDD transforma o desenvolvimento, devido ao fato de antes de implementar o sistema, os testes são escritos. Desta forma, os testes são utilizados para facilitar no entendimento do projeto, segundo Freeman (2012) os testes são usados para clarear a ideia em relação ao que se deseja em relação ao código. A criação de testes unitários ou de componentes é parte crucial para o TDD. De acordo com Presmann (2011), “Os componentes individuais são testados para garantir que operem corretamente. Cada componente é testado independentemente, sem os outros componentes de sistema. Os componentes podem ser entidades simples, tais como funções ou classes de objetos, ou podem ser grupos coerentes dessas entidades”.

Mas não é só o teste unitário que vai trazer o sucesso a aplicação, é necessário testar o sistema como um todo, pois segundo Sommerville (2010), “Os componentes são integrados para compor o sistema. Esse processo está relacionado com a busca de erros que resultam das interações não previstas entre os componentes”.

Um sistema é um conjunto de unidades integradas, por este motivo é importante os testes unitários, para que se tenha uma visão em um nível mais baixo.

Não se pode deixar de testar a integração, ou seja, ao integrar dois ou mais componentes, deverá realizar testes para validar se a integração funciona. Projetos são passíveis de erro, principalmente no processo de montagem/integração de componentes.

Outro ponto importante no TDD é a redução do retrabalho em equipe, assim como a redução dos custos e maiores chances de sucesso ao final do projeto.

O Ciclo do TDD é resume-se a: é criado o teste -> Feita a codificação para passar no teste ->Refatorar o código (*figura 2*).

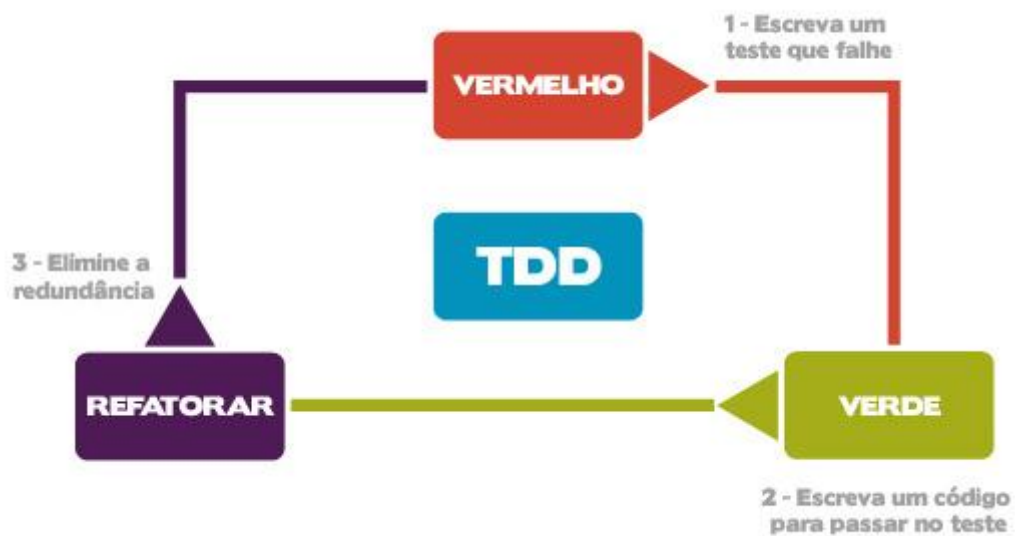


Figura 2: Ciclo do TDD

O Teste deve ser escrito antes da codificação, conceito este chamado de *test-first*, segundo Sommerville (2010), “a escrita de testes iniciais define implicitamente, que uma interface deve ser uma especificação do comportamento para a funcionalidade que está sendo desenvolvida”.

3.11 Ferramentas

Outro ponto importante para o projeto foi o uso de ferramentas CASE (*Computer-Aided Software Engineering*). Essa classificação abrange toda ferramenta baseada em computadores que auxiliam atividades de engenharia de software, desde a análise de requisitos e modelagem até programação e testes.

3.11.1 Google Drive

O *Google Drive* é um serviço *online* que permite o armazenamento de arquivos na nuvem do Google. O aplicativo é uma resposta da gigante americana aos programas do gênero, como *Dropbox* e *SkyDrive*. Com ele, é possível fazer o *upload* e acessar seus arquivos, incluindo vídeos, fotos, arquivos do *Google Docs* e PDFs (TECHTUDO, 2013). Além disso, é possível manipular os arquivos do *Google Docs* de forma colaborativa e em tempo real.

O *Google Drive* foi utilizado como repositório de documentos, nele foi possível fazer o acesso a todos os documentos utilizados no projeto, e realizando as alterações de forma colaborativa e online por todos os integrantes.

3.11.2 Controle de Versão: Git

O *Git* é um sistema de controle de versão, de código aberto, projetado para lidar com projetos muito grandes com rapidez e eficiência, mas também adaptado a pequenos repositórios pessoais. É especialmente popular na comunidade de código aberto, servindo como uma plataforma de desenvolvimento para projetos (OPENSUSE, 2013).

3.11.3 GitHub

GitHub é um Serviço de gerenciamento de projetos on-line que utilizam o controle de versionamento Git, ele possui uma ampla gama de recursos colaborativos, além de fornecer a possibilidade de criar repositórios privados para *software* proprietário (IMASTERS, 2013).

Além de ser utilizado no projeto para gerenciar o código fonte, ele também foi utilizado para o controle dos *tickets* de atividades a serem realizadas, sendo que foi definido um fluxo de trabalho (*apêndice II*) para que os integrantes implementassem as funcionalidades de forma ágil.

3.11.4 LucidChart

O LucidChart é capaz de criar fluxogramas, organogramas, *wireframes* de *sites*, projetos UML, mapas mentais, protótipos de *software*, e muitos outros tipos de diagramas. Além disso, esse *software* de diagramação permite aos usuários colaborar e trabalhar juntos em tempo real (LUCIDCHART, 2013). Desta forma, o LucidChart foi utilizado para criar diagramas e compartilhá-los diretamente no Google drive.

3.11.5 Astah

Astah é uma ferramenta para criação de diagramas UML, tais como diagrama de classe, casos de uso, entidade-relacionamento, diagrama de fluxo de dados e outras funcionalidades úteis à fase de especificação e projeto de um sistema (MARTINS, 2013). Devido a grande necessidade de diagramação do projeto, o Astah foi utilizado para melhor atender as necessidades de representação e entendimento dos artefatos criados no projeto, através de seus diagramas de representação.

3.11.6 Eclipse e Netbeans IDE

Para o desenvolvimento do projeto utilizamos a IDE Eclipse e o NetBeans, ambos são uma IDE *open-source*, que além de ser de fácil uso, suas tecnologias são baseadas a partir de *plugins*, o que nos possibilitou instalar o *plugin* de controle de versionamento Git, assim, passamos a ter um maior controle e agilidade sobre o projeto, pois o *plugin* oferece atalhos na IDE Eclipse, assim como o NetBeans, para realizar alterações no repositório, além de mostrar visualmente arquivos novos e atualizados (ECLIPSE, 2013), (NETBEANS, 2013).

3.12 Arquitetura

Nesta etapa, foi definida a infraestrutura web baseado na arquitetura SOA, onde é possível visualizar os *frameworks* utilizados na plataforma.

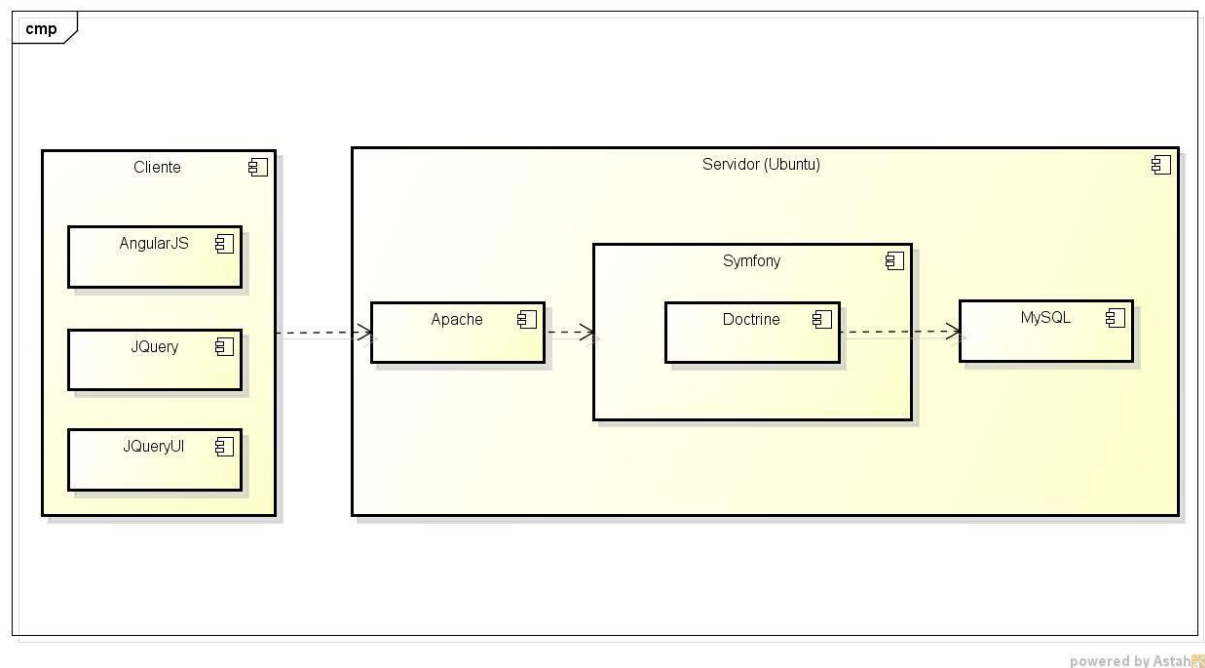


Figura 3: Diagrama de componentes do backend

Com o diagrama de componentes do *backend* (Figura 3), é possível visualizar toda a infraestrutura por trás do modelo cliente/servidor da plataforma.

Para que seja efetivado todo o processo de requisição de um usuário, foram utilizadas bibliotecas do lado cliente, como jQuery, jQueryUI e o AngularJS, que são responsáveis por exibirem mensagens, melhorar a parte visual da interface, e enviar formulários em formato JSON, ao servidor de requisições, o Apache, que é responsável por redirecionar ao Symfony2 as devidas solicitações feitas pelo usuário, desse modo então, são chamadas as funções do core da plataforma, responsáveis por executar a lógica de negócio, é nesse momento em que o Symfony2 chama as funções do Doctrine para que seja criado os CRUDs do usuário.

3.12.1 Ubuntu Server

O Ubuntu Server é a distribuição do sistema operacional Ubuntu destinada à servidores, onde não existe um ambiente gráfico pré-instalado. Seus programas, na maioria das vezes, são voltados para a administração de serviços de rede (JANG, 2008).

Uma vez que o projeto foi desenvolvido de maneira com que ele funcione totalmente virtualizado, ou seja, em nuvem, o Ubuntu Server foi utilizado, pois atende as principais características exigidas para o uso desse tipo de sistema, que é utilizar esses servidores da rede de forma simplificada, otimizada e com o objetivo de garantir seu alto desempenho, escalabilidade e agilidade.

3.12.2 Apache

O Apache é um servidor de páginas *web* que funciona nos principais sistemas operacionais, ele oferece opções de configuração, além de ser robusto, apresentar um alto desempenho, e seu código fonte é distribuído gratuitamente através da licença *Apache Software Foundation* (SOUZA, 2013).

A escolha do servidor *web Apache* no projeto, se deu ao fato dele ser utilizado como referência na documentação do *Symfony2*, além de ser de conhecimento técnico dos desenvolvedores do grupo.

3.12.3 PHP

PHP é uma linguagem *open source*, disponibiliza os benefícios da programação orientada a objetos (desde a versão 4), oferecendo as vantagens da reutilização de código e a facilidade na hora da manutenção (HTMLGOODIES, 2013).

A utilização da linguagem PHP no projeto foi escolhida por se uma linguagem sólida em aplicações web, um exemplo disso é a sua utilização no site *Facebook*, outro motivo é a de ser utilizada pelo *framework* *Symfony2*, na qual oferece as melhores práticas de desenvolvimento do mercado, como a programação orientada a objetos.

3.12.4 MySQL

O MySQL é um sistema gerenciador de banco de dados de código fonte aberto, que reúne características capazes de atender às necessidades dos mais variados tipos de usuários. Este produto tem sido utilizado como solução para desenvolvedores de sistemas, provedores de serviços, aplicações *Enterprise*, bem como para suportar aplicações livres (DUARTE, 2013).

3.12.5 Symfony2

O Symfony2 é um *framework* escrito em PHP, lançado em 2005, pela empresa *Sensio Labs*. Este *framework* oferece um ambiente estável de desenvolvimento, além de proporcionar componentes reutilizáveis e um conjunto de ferramentas para a aplicação das melhores práticas do mercado, para que o desenvolvedor se preocupe apenas com a lógica do negócio. Algumas das ferramentas disponibilizadas pelo *framework* Symfony2 são:

- tratamentos de requisições e respostas, de maneira simples e rápida;
- classes para o suporte a lógica de negócio;
- roteamento de páginas;
- validação de campos;
- suporte ao tratamento de formulários;
- esquema de validação e segurança;
- suporte a testes unitários e funcionais;

3.12.6 HTML

HTML (*Hyper Text Markup Language* - Linguagem de Marcação de Hipertexto) é uma linguagem de marcação utilizada no desenvolvimento de páginas *Web*. Através dessa linguagem é possível fazer toda a estruturação de uma página que será acessada online (NOVATEC, 2011).

Ela foi utilizada no desenvolvimento do projeto para melhor estruturar e disponibilizar aos usuários as informações que ele deseja, de maneira simples e clara.

3.12.7 Javascript

Linguagem *script* utilizada principalmente para manipulação de elementos HTML. Seu foco é fornecer dentro de estruturas HTML, níveis de interatividade com as páginas *WEB*, o que não é possível utilizando HTML puro. Graças a sua compatibilidade com a maioria dos navegadores modernos, é a linguagem de programação do lado do cliente mais utilizada (ALVAREZ, 2004).

3.12.8 JQuery

JQuery é uma biblioteca utilizada para o desenvolvimento de *javascripts* de maneira ágil afim de interagirem com uma página HTML. Essa biblioteca permite ao desenvolvedor atribuir eventos, criar efeitos, criar ou alterar elementos na página, entre outras funcionalidades (TAVARES, 2013). Sua utilização no projeto visa enriquecer as interfaces através do aumento no nível de interatividade com o usuário.

3.12.9 JQueryUI

JQueryUI é uma rica biblioteca de componentes gráficos para desenvolvimento *web*. Trata-se de um projeto de código aberto onde cada componente é construído de acordo com a arquitetura dirigida a eventos do jQuery (TAVARES, 2013).

3.12.10 AngularJS

AngularJS é um *framework* desenvolvido em JavaScript, que tem como principal diferencial a capacidade de aumentar o poder do HTML, facilitando o desenvolvimento de aplicações para a web (BRANAS, 2013).

3.12.11 Twitter Bootstrap

O Twitter Bootstrap é um padrão de interface que facilita na produtividade na criação de interfaces *web*, utilizando um *design* arrojado e flexível, eliminando processos trabalhosos e permitindo ao desenvolvedor focar no que está desenvolvendo (DEVMEDIA, 2013).

3.13 Mockups

Para guiar o desenvolvimento das interfaces gráficas (*GUI*), foram definidos alguns *mockups* (maquete virtual das interfaces gráficas), que mais tarde foram implementados utilizando-se HTML e *widgets* da biblioteca JQuery-UI.

No *mockup* da figura 4 está definido um guia para a implementação da interface gráfica responsável pelo cadastramento dos campos existente em uma interface CRUD (*Schema*). Nela é possível identificar os itens que compõem essa interface (*menu*, grid para cadastro dos campos do *Schema* e botões para as outras funções do sistema), além de definir sua possível posição na implementação de tal interface gráfica.

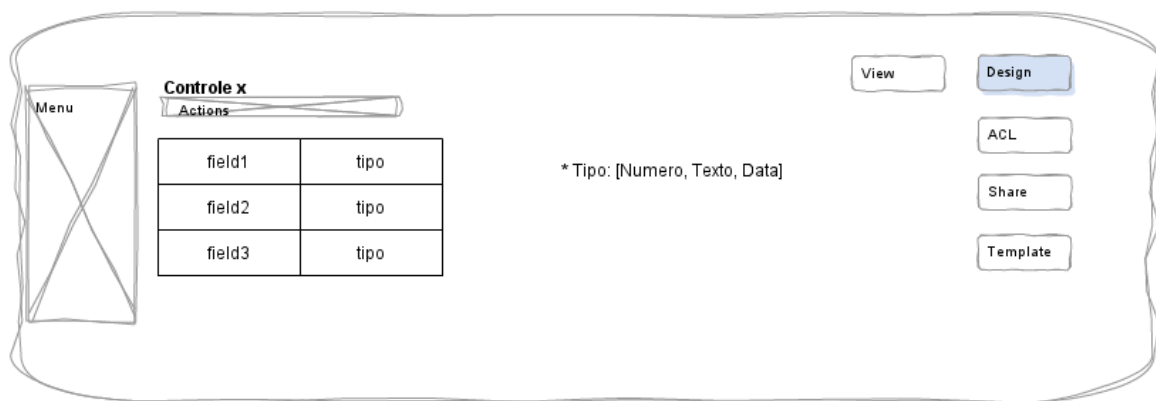


Figura 4: Mockup para a tela de design do CRUD

Outro *mockup* definido, fornece um guia para a implementação da interface para visualização dos dados cadastrados em um CRUD, apresentados em um *grid* (figura 5).

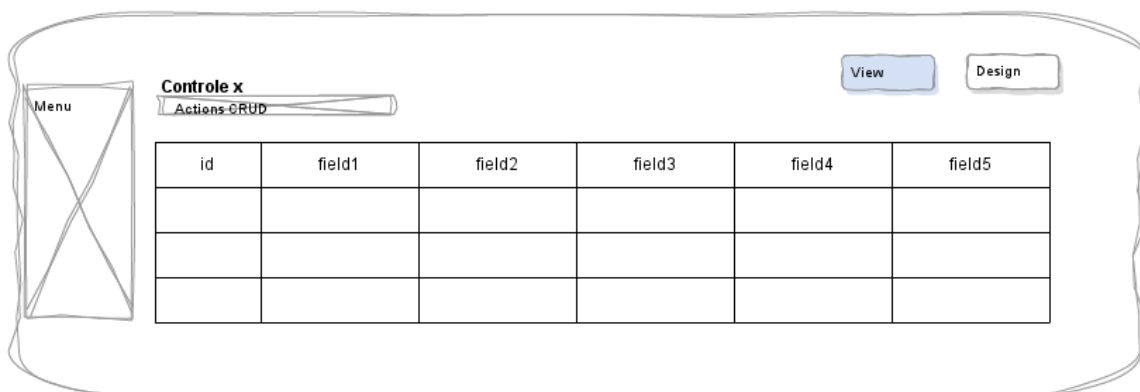


Figura 5: Mockup para a tela do CRUD em execução

A plataforma CRUDForge futuramente deve possuir uma interface gráfica para definição de *templates* para serem utilizados com o acoplamento dos dados cadastrados nas interfaces CRUD (figura 6).

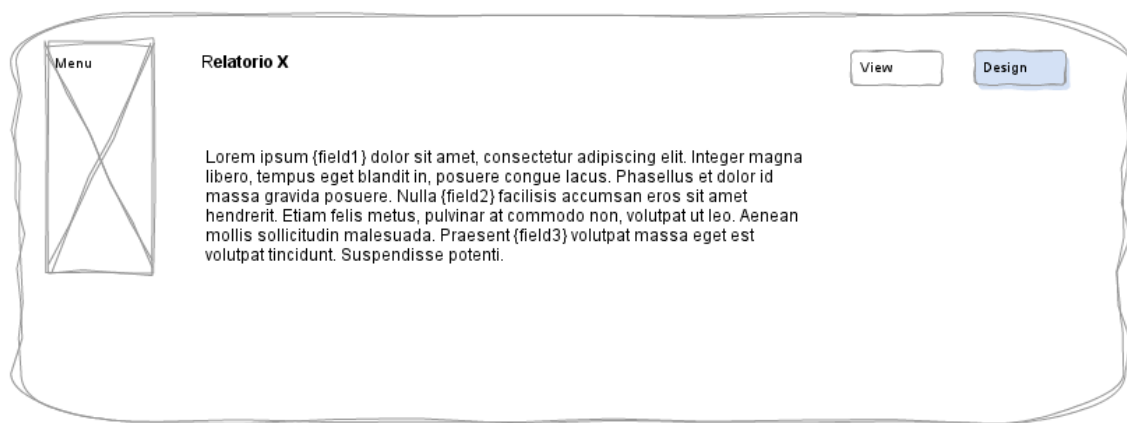


Figura 6: Mockup para a tela de design de relatórios

3.14 Diagramas de Casos de Uso

De acordo com Cockburn (2005), os casos de uso descrevem o comportamento do sistema sob diversas condições conforme o sistema responde a uma requisição de um dos *stakeholders*, chamado ator primário.

Deste modo, para identificar e validar os requisitos do sistema. Foram descritos os casos de uso, onde, o cliente tem a possibilidade de personalizar o sistema de acordo com as necessidades do seu negócio, fazendo assim com que o sistema utilize-se do *scaffold* para gerar as interfaces que serão utilizadas pelo usuário.

3.14.1 Caso de Uso Geral

Através do diagrama de caso de uso (*figura 7*), é demonstrada uma utilização genérica do CRUDForge.

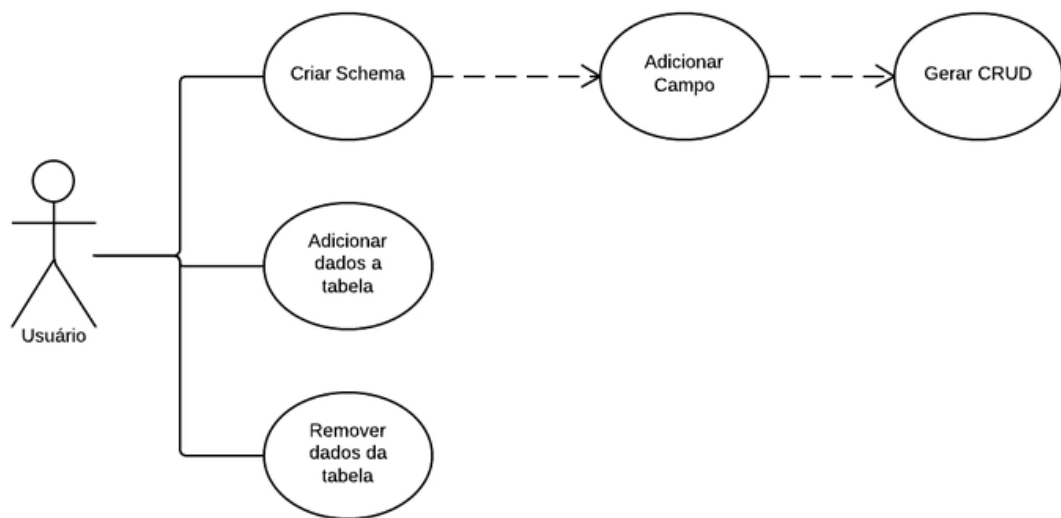


Figura 7: Diagrama de caso de uso genérico 1

O caso de uso genérico consiste em um primeiro momento criar um *schema*, logo após adicionar os campos desejados e por fim gerar *CRUD*.

Após a criação da tabela, o usuário pode adicionar dados a tabela, através das interfaces geradas pelo *scaffold*.

Tabela 1: Ficha para o caso de uso genérico

Objetivo	Gerar o CRUD
Atores	Usuário
Pré-Condição	Usuário adiciona um novo <i>schema</i>
Ativação	O caso de uso começa quando o usuário aciona o comando “Novo Schema”
Fluxo de Eventos	<p>FLUXO NORMAL:</p> <ol style="list-style-type: none"> 1. O usuário preenche o nome do <i>schema</i>. 2. O usuário aciona o comando “create” 3. O CRUDFORGE gera o <i>schema</i>. 4. O usuário aciona o comando “ver campos” 5. O usuário aciona o comando “novo campo” 6. O usuário preenche o nome do campo e o tipo. 7. O usuário aciona o comando “create” 8. O usuário aciona o comando “Voltar a seleção dos <i>schemas</i>” 9. O usuário aciona o comando “Gerar CRUD” do <i>schema</i>

	desejado
Requisitos não funcionais	
Diagrama de atividades :	Diagrama de atividades Criação do <i>Schema</i>
Interface Gráfica (GUI):	

3.14.2 Caso de Uso Específico

Nesse item são apresentados os diagramas, para os objetivos específicos, definidos no *capítulo 2.2*.

3.29.2.1 Caso de Uso: *ToDoList*

Através do diagrama de caso de uso (figura 8), é demonstrada a utilização de um dos objetivos propostos, a lista de tarefas.

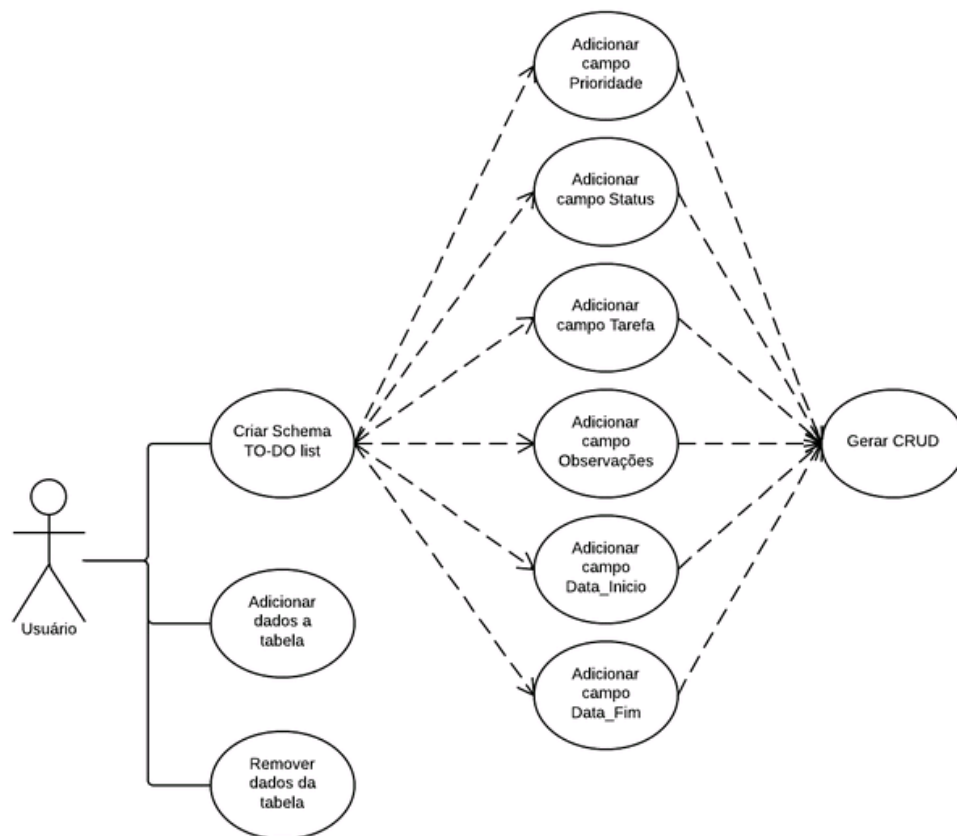


Figura 8: Caso de Uso *ToDoList*

Nesse caso de uso é necessário em um primeiro momento criar o *schema ToDoList*, logo após adicionar os campos *Prioriedade*, *Status*, *Tarefa*, *DataInicio*, *DataFim* e por fim gerar *CRUD*.

Tabela 2: Ficha do caso de uso ToDoList

Objetivo	Gerar o CRUD
Atores	Usuário
Pré-Condição	Usuário adiciona um <i>schema</i> ToDoList
Ativação	O caso de uso começa quando o usuário aciona o comando "Novo Documento"
Fluxo de Eventos	<p>FLUXO NORMAL:</p> <ol style="list-style-type: none"> 1. O usuário preenche o nome do <i>schema</i> com "ToDoList" 2. O usuário aciona o comando "create" 3. O CRUDFORGE cadastra o <i>schema</i> 4. O usuário aciona o comando "ver campos" 5. O usuário aciona o comando "novo campo" 6. O usuário preenche o nome do campo com "Prioridade" e o tipo como 'texto' 7. O usuário aciona o comando "create" 8. O usuário aciona o comando "novo campo" 9. O usuário preenche o nome do campo com "Status" e o tipo como "texto" 10. O usuário aciona o comando "create" 11. O usuário aciona o comando "novo campo" 12. O usuário preenche o nome do campo com "Tarefa" e o tipo como "texto" 13. O usuário aciona o comando "create" 14. O usuário aciona o comando "novo campo" 15. O usuário preenche o nome do campo com "Observações" e o tipo como 'texto' 16. O usuário aciona o comando "create" 17. O usuário aciona o comando "novo campo" 18. O usuário preenche o nome do campo com "DataInicio" e o tipo como 'texto' 19. O usuário aciona o comando "create" 20. O usuário aciona o comando "novo campo" 21. O usuário preenche o nome do campo com "DataFim" e o tipo como 'texto' 22. O usuário aciona o comando "create" 23. O usuário aciona o comando "Voltar a seleção dos <i>schemas</i>" 24. O usuário aciona o comando "Gerar CRUD" do <i>schema</i> desejado

Requisitos não funcionais	
Diagrama de atividades :	Diagrama de atividades Criação do <i>SchemaToDoList</i>
Interface Gráfica (GUI):	

Após a criação da tabela, o usuário pode adicionar dados a tabela ou até mesmo adicionar novos campos, se desejar.

3.14.2.2 Caso de Uso: Compartilhamento de inventário

Através do diagrama de caso de uso (figura 9), é demonstrada a utilização de um dos objetivos propostos, o DVDs.

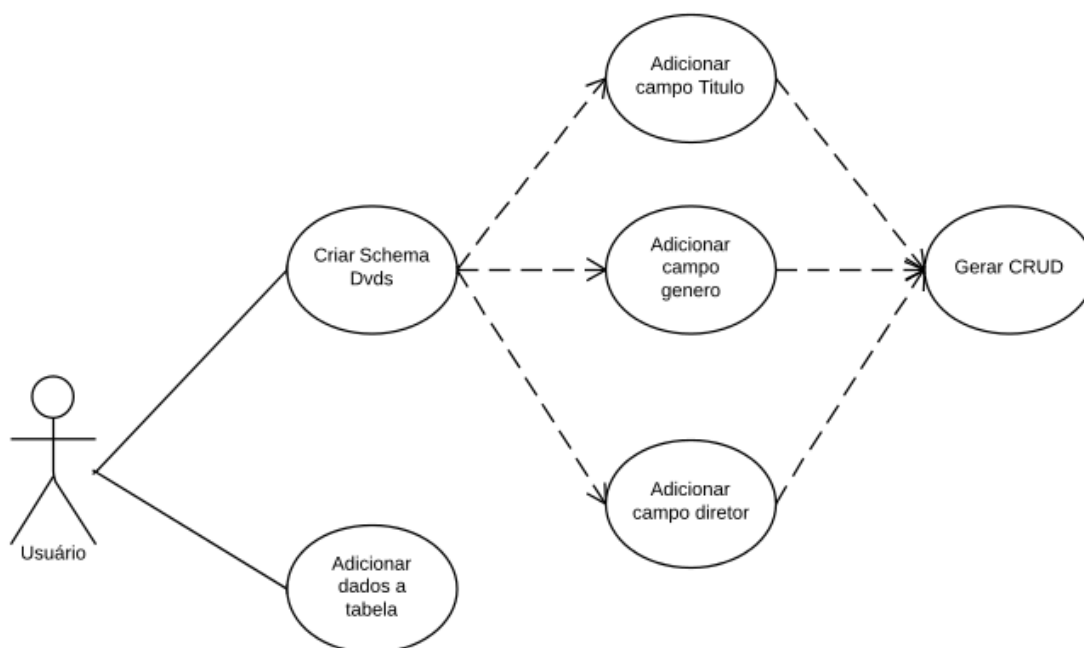


Figura 9: Diagrama de caso de caso de uso para coleção de DVDs D

Nesse caso de uso, é necessário em um primeiro momento criar o *schema DVDs*, logo após adicionar os campos Título, Gênero, Diretor e por fim gerar *CRUD*.

Tabela 3: Ficha do caso de uso coleção de DVD

Objetivo	Gerar o <i>CRUD</i>
Atores	Usuário
Pré-Condição	Usuário criando um <i>schema</i> DVD
Ativação	O caso de uso começa quando o usuário aciona o comando "Novo Documento"
Fluxo de Eventos	<p>FLUXO NORMAL:</p> <ol style="list-style-type: none"> 1. O usuário preenche nome do <i>schema</i> com "Dvds" 2. O usuário aciona o comando "create" 3. O CRUDFORGE cadastra o <i>schema</i> 4. O usuário aciona o comando "ver campos" 5. O usuário aciona o comando "novo campo" 6. O usuário preenche o nome do campo com "Título" e o tipo como "texto" 7. O usuário aciona o comando "create" 8. O usuário aciona o comando "novo campo" 9. O usuário preenche o nome do campo com "Gênero" e o tipo como "texto" 10. O usuário aciona o comando "create" 11. O usuário aciona o comando "novo campo" 12. O usuário preenche o nome do campo com "Diretor" e o tipo como "texto" 13. O usuário aciona o comando "create" 14. O usuário aciona o comando "Voltar a seleção dos <i>schemas</i>" 15. O usuário aciona o comando "Gerar CRUD" do <i>schema</i> desejado
Requisitos não funcionais	
Diagrama de atividades :	Diagrama de atividades Criação do <i>Schema</i> "Dvds"
Interface Gráfica (GUI):	

Após a criação da tabela, o usuário pode adicionar dados a tabela ou até mesmo adicionar novos campos, se desejar.

3.14.2.3 Caso de Uso: Contas a pagar e receber

Através do diagrama de caso de uso (figura 10), é demonstrada a utilização de um dos objetivos propostos, o objetivo “Contas a pagar e receber”.

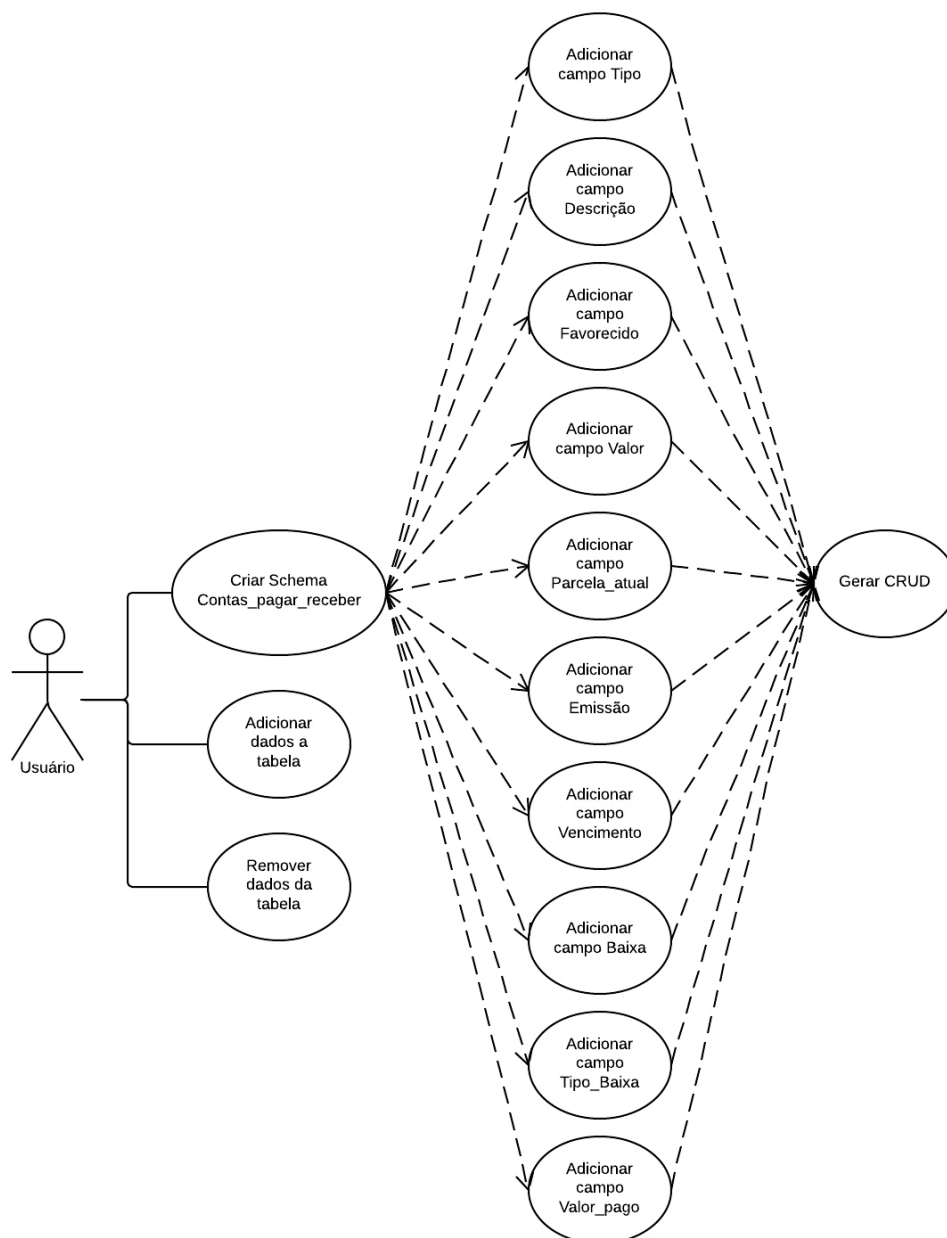


Figura 10: Diagrama de caso de uso Contas a pagar e receber

Nesse caso de uso, é necessário em um primeiro momento criar o *schema* *ContasPagarReceber*, logo após adicionar os campos Tipo, Descrição, Favorecido, Valor, ParcelaAtual, Emissão, Vencimento, Baixa, TipoBaixa, ValorPago e por fim gerar *CRUD*.

Tabela 4: Ficha do caso de uso Contas a pagar e receber

Objetivo	Gerar o CRUD
Atores	Usuário
Pré-Condição	Usuário adiciona um <i>schema</i> Contas a pagar e receber
Ativação	O caso de uso começa quando o usuário aciona o comando "Novo Documento"
Fluxo de Eventos	<p>FLUXO NORMAL:</p> <ol style="list-style-type: none"> 1. O usuário preenche nome do <i>schema</i> com "ContasPagarReceber" 2. O usuário aciona o comando "create" 3. O CRUDFORGE cadastra o <i>schema</i> 4. O usuário aciona o comando "ver campos" 5. O usuário aciona o comando "novo campo" 6. O usuário preenche o nome do campo com "Tipo" e o tipo como "texto" 7. O usuário aciona o comando "create" 8. O usuário aciona o comando "novo campo" 9. O usuário preenche o nome do campo com "Descricao" e o tipo como "texto" 10. O usuário aciona o comando "create" 11. O usuário aciona o comando "novo campo" 12. O usuário preenche o nome do campo com "Favorecido" e o tipo como "texto" 13. O usuário aciona o comando "create" 14. O usuário aciona o comando "novo campo" 15. O usuário preenche o nome do campo com "Valor" e o tipo como "número" 16. O usuário aciona o comando "create" 17. O usuário aciona o comando "novo campo" 18. O usuário preenche o nome do campo com "parcelaatual" e o tipo como "numero" 19. O usuário aciona o comando "create" 20. O usuário aciona o comando "novo campo"

		<p>21. O usuário preenche o nome do campo com "Emissao" e o tipo como "texto"</p> <p>22. O usuário aciona o comando "create"</p> <p>23. usuário aciona o comando "novo campo"</p> <p>24. O usuário preenche o nome do campo com "Vencimento" e o tipo como "texto"</p> <p>25. O usuário aciona o comando "create"</p> <p>26. O usuário aciona o comando "novo campo"</p> <p>27. O usuário preenche o nome do campo com "Baixa" e o tipo como "texto"</p> <p>28. O usuário aciona o comando "create"</p> <p>29. O usuário aciona o comando "novo campo"</p> <p>30. O usuário preenche o nome do campo com "Tipobaixa" e o tipo como "texto"</p> <p>31. O usuário aciona o comando "create"</p> <p>32. O usuário aciona o comando "novo campo"</p> <p>33. O usuário preenche o nome do campo com "Valorpago" e o tipo como "texto"</p> <p>34. O usuário aciona o comando "create"</p> <p>35. O usuário aciona o comando "Voltar a seleção dos <i>schemas</i>"</p> <p>36. O usuário aciona o comando "Gerar CRUD" do <i>schema</i> desejado</p>
Requisitos funcionais	não	
Diagrama de atividades :	de	Diagrama de atividades Criação do <i>Schema</i> Contas a pagar e receber
Interface Gráfica (GUI):		

Após a criação da tabela, o usuário pode adicionar dados a tabela ou até mesmo adicionar novos campos, se desejar.

2.14.3 Caso de Uso Segurança

Através deste diagrama de caso de uso (figura 11), é demonstrada a segurança e o compartilhamento do CRUD.

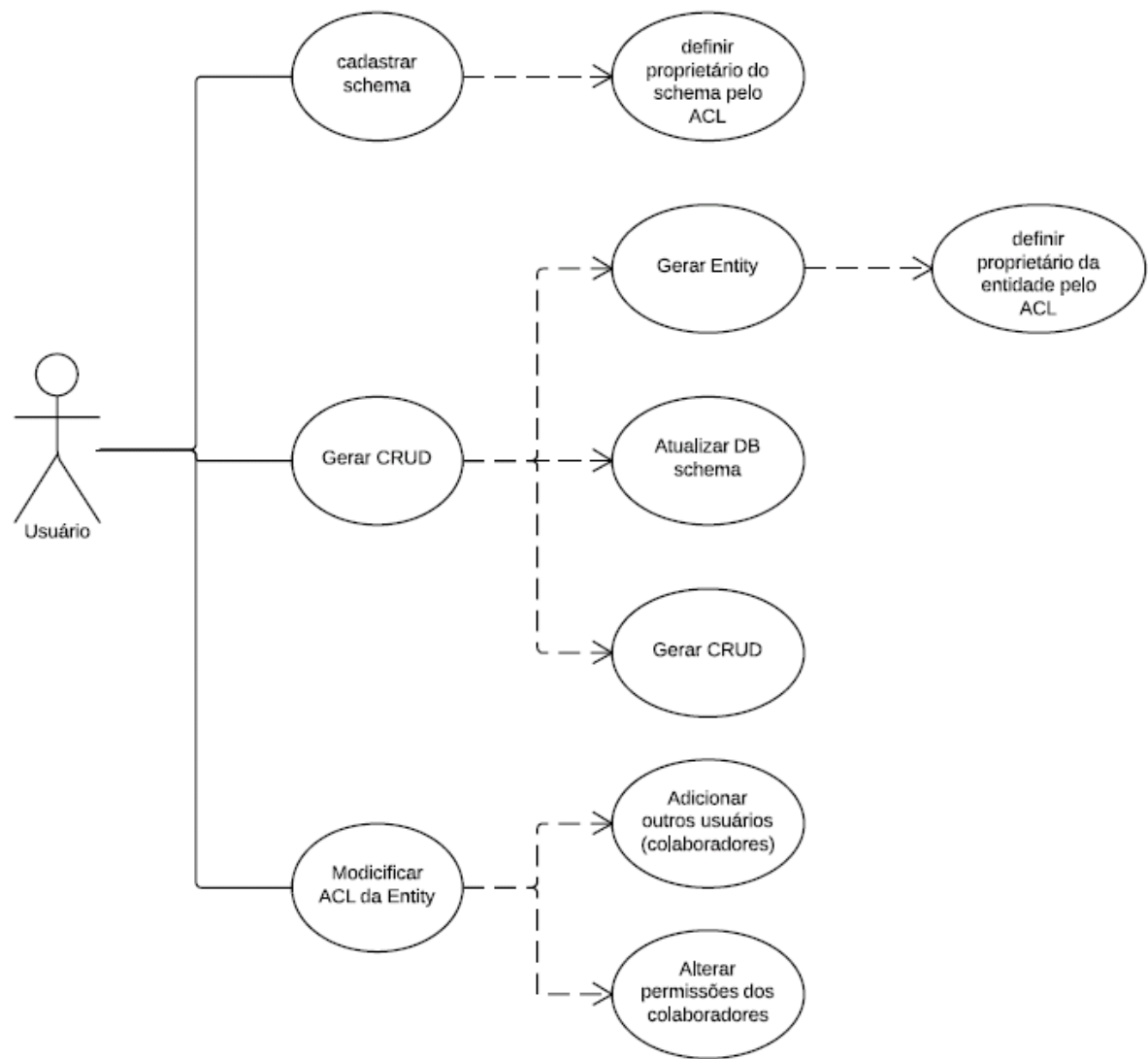


Figura 11: Diagrama de caso de uso Segurança

Tabela 5: Ficha do caso de uso Segurança

Objetivo	Gerar o CRUD
Atores	Usuário

Pré-Condição	Usuário já gerou o <i>schema</i>
Ativação	O caso de uso começa quando o usuário aciona o comando “Compartilhar”
Fluxo de Eventos	<p>FLUXO NORMAL:</p> <ol style="list-style-type: none"> 1. O usuário preencheo nome do usuário no campo "email" 2. O usuário define as permissões dos colaboradores 3. O usuário aciona o campo “convidar”
Requisitos não funcionais	
Diagrama de atividades	Diagrama de atividades Criação do <i>Schema</i>
Interface Gráfica (GUI)	

3.15 Diagrama de Classe

O *framework* symfony2 possui um *bundle* (pacote de *software* que utiliza o *framework* para implementar certa funcionalidade - POTENCIER, 2013), que implementa o *scaffold*, ou seja, a geração das classes MVC (*Model-View-Controller*), de forma interativa através de um aplicativo de console. Uma das finalidades desse projeto é a de fazer a engenharia reversa nas classes responsáveis pela geração de código, abstraindo todos os parâmetros solicitados pelo aplicativo de console e sintetizando essa lógica no *bundle* do CRUDForge. Para que tal objetivo fosse satisfeito, foram utilizados como ferramentas os diagramas de classe (*figura 12*) e sequência (*figura 13*) da UML.

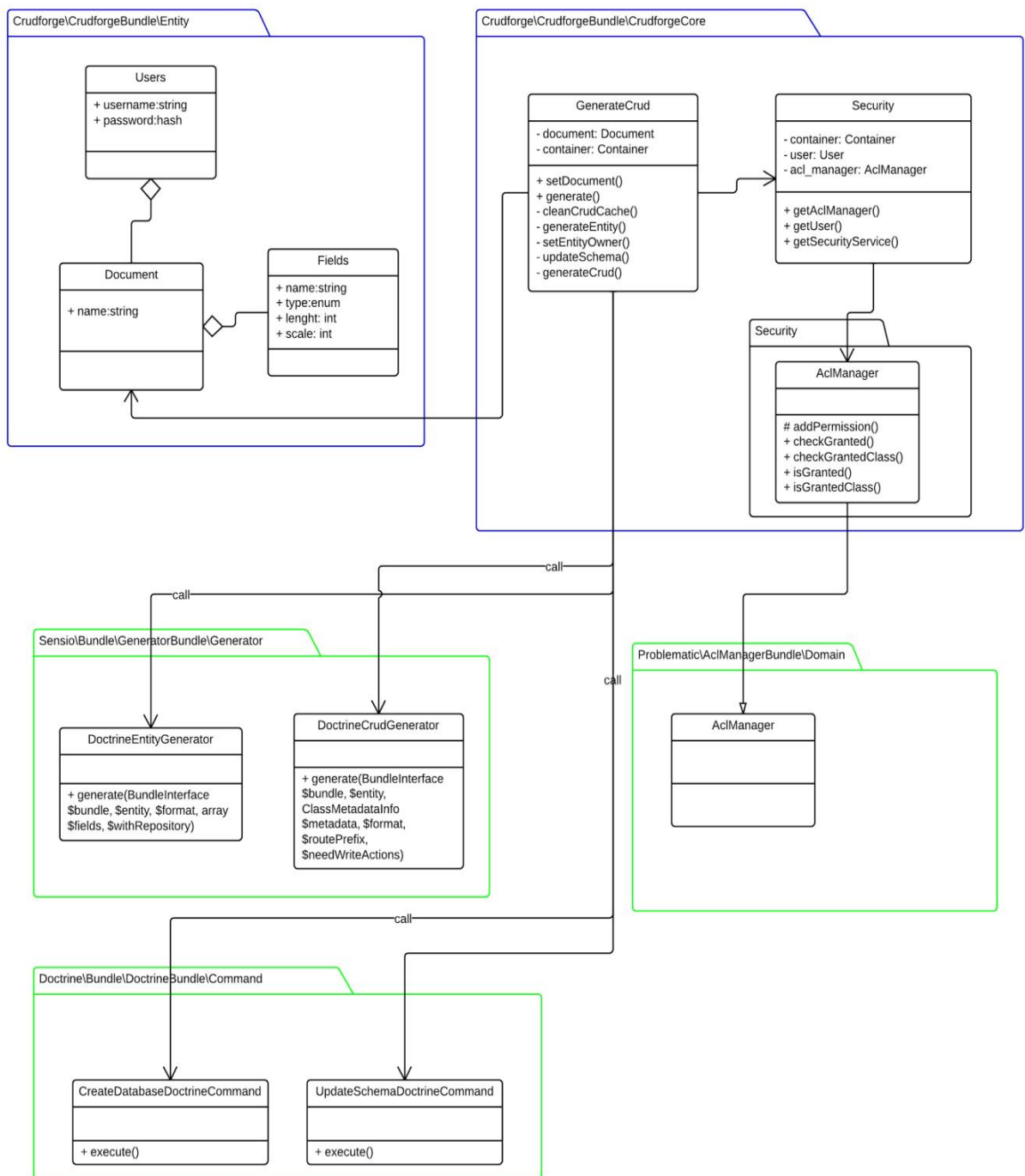


Figura 12: Diagrama de classe do Core

3.16 Diagramas de Sequência

3.16.1 Geração do CRUD (Scaffold)

Na fase de planejamento, foi especificado um diagrama de sequência da classe *GenerateCrud* (figura 13) responsável pela implementação do serviço principal do projeto, o seu *core*. Neste diagrama são visualizados as principais funções e componentes do *framework* utilizados na geração das classes pertinentes ao CRUD e demais tarefas auxiliares para implementação da arquitetura MVC, sendo que essa sequência será iniciada em algum determinado momento após a definição do *schema* por parte do usuário ou pelo próprio sistema.

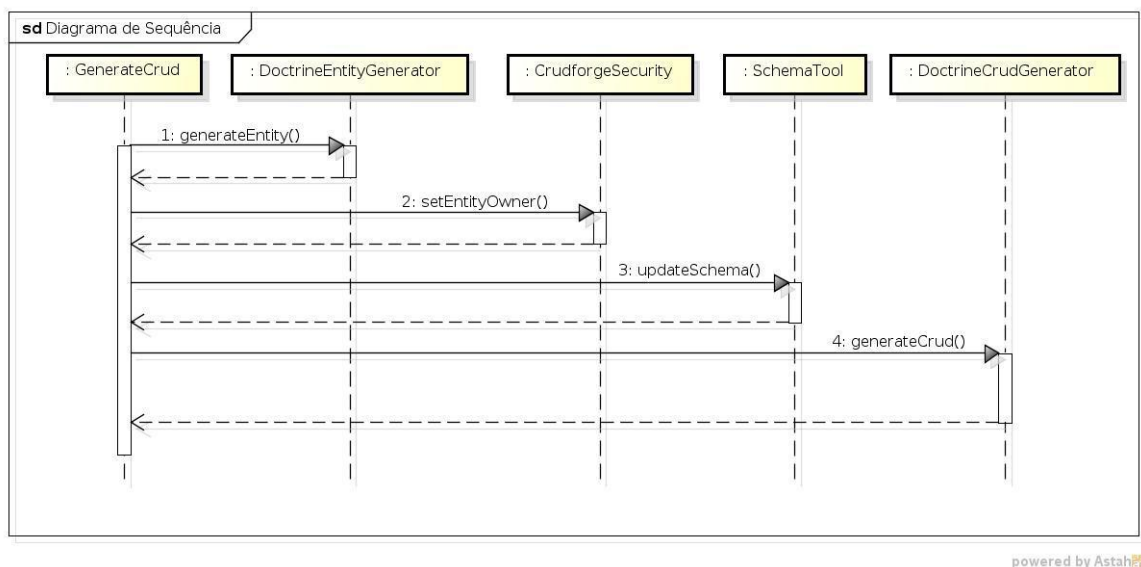


Figura 13: Diagrama de Sequência do Core

Sequência de ações para gerar o CRUD:

1. *generateEntity*: a sequência inicia-se com a geração da classe de entidade, contendo as propriedades e métodos utilizados pelo ORM;
2. *setEntityOwner*: logo em seguida é atribuído um dono a classe do tipo *Entity* do *schema* criado pelo usuário, para que seja iniciado o ACL (*Access ControlList*) do CRUD;

3. *updateSchema*: após a criação da classe *Entity* e associa-las ao usuário, o *schema* é criado ou atualizado na base de dados;
4. *generateCrud*: finaliza a sequência com a geração das classes pertencentes as camadas de *Controller* e *View* do CRUD, baseados na classe *Entity* do *Schema* definido pelo usuário;

3.16.2 Autenticação e Autorização

Com o diagrama de sequência de autenticação (*figura 14*), é possível visualizar a etapas de autenticação, que ocorrem a partir do momento em que o usuário seleciona a página inicial do sistema para logar-se, ou quando requisita uma URL que está protegida, sendo redirecionado para a página de *login*.

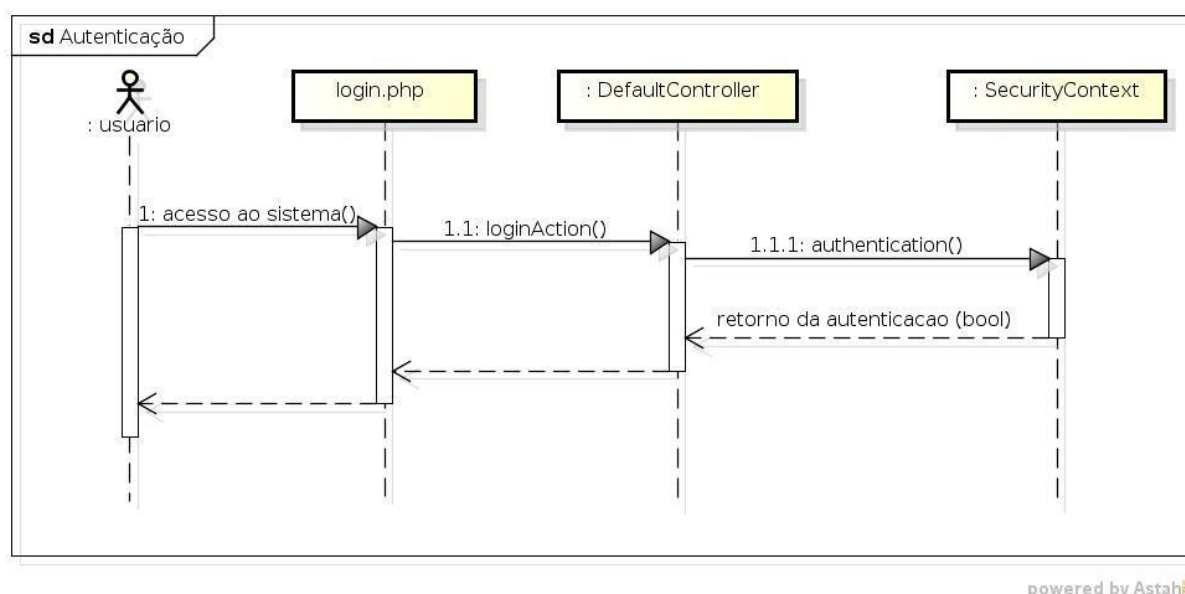
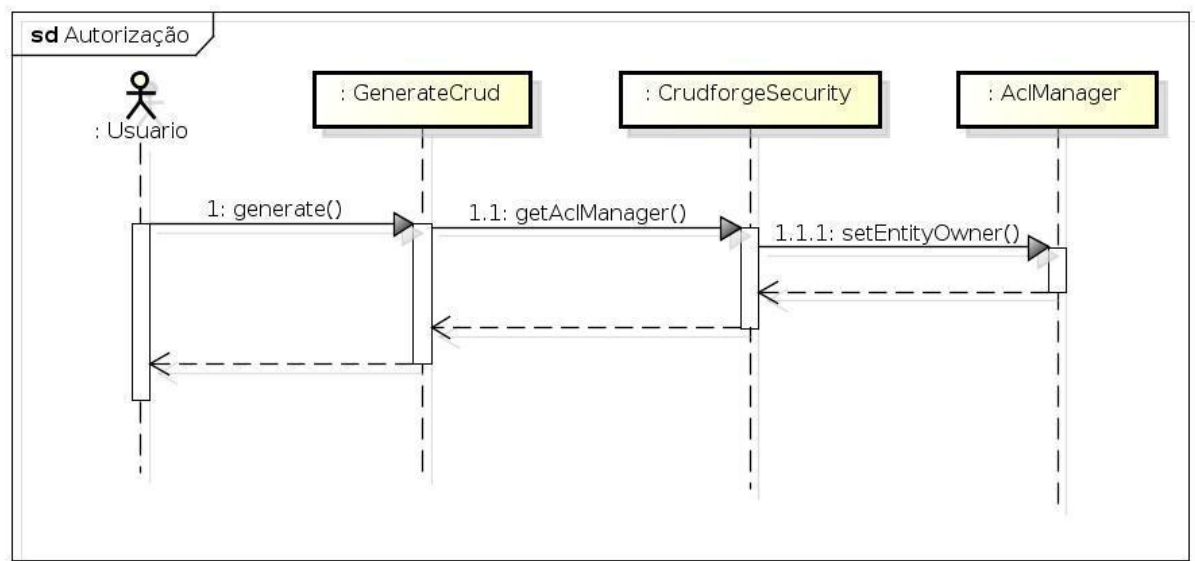


Figura 14: Diagrama de Sequência de Autenticação

O diagrama de sequência de autorização (*figura 15*) ilustra o momento em que ao gerar uma nova entidade o sistema define o usuário dono da entidade.



powered by Astah

Figura 15: Diagrama de Sequência de Autorização

3.17 Diagrama de Componentes

Através do diagrama de componentes (*figura 16*), é possível visualizar as camadas e as classes geradas pela plataforma através do *scaffold*, no momento em que o usuário utilizar a função gerar *CRUD*, após a definição do *schema*.

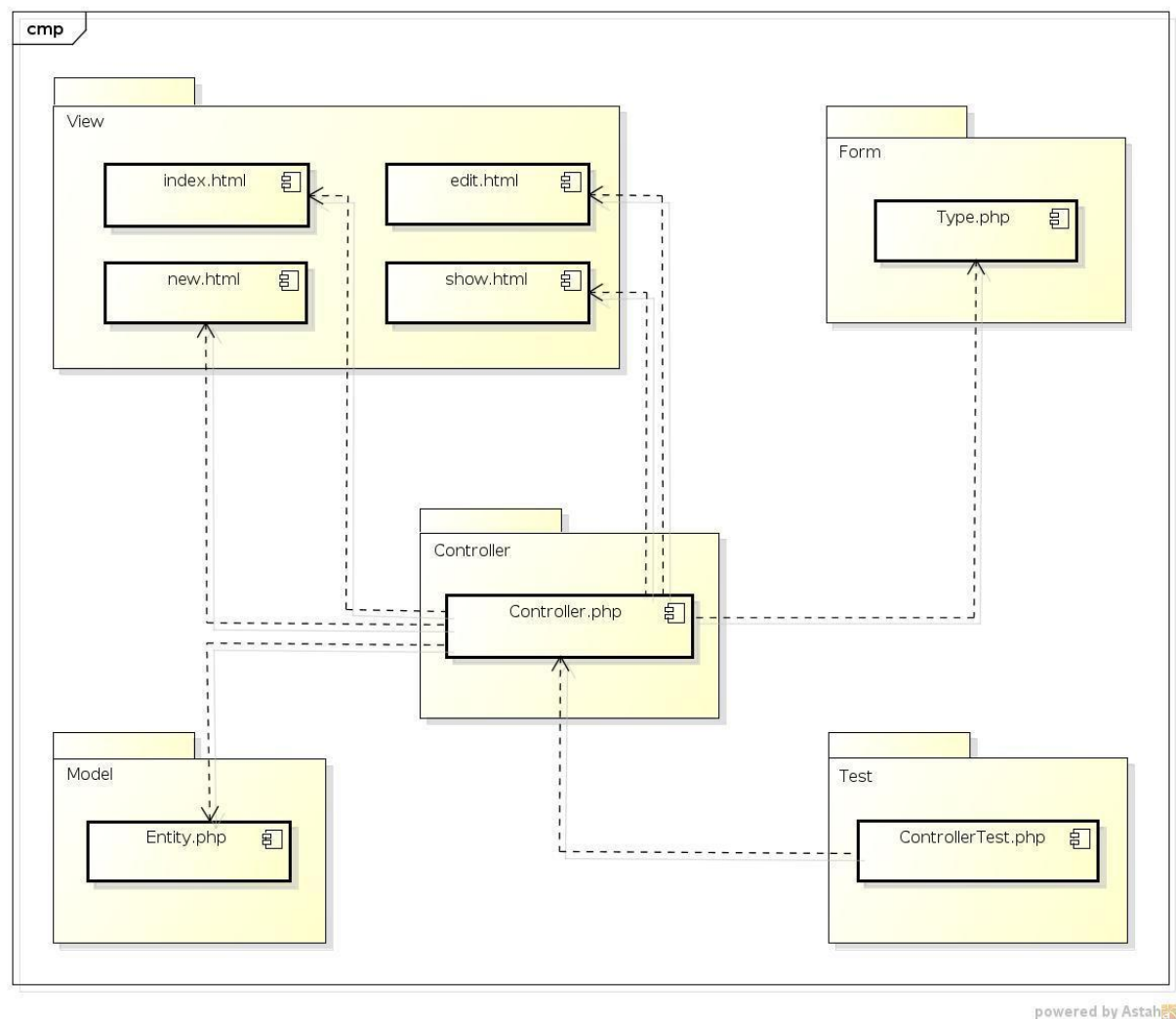


Figura 16: Diagrama de Componentes do CRUD

3.18 Serviços

Ao utilizar o *framework* Symfony2 na implementação dos *controllers*, por vezes as mesmas tarefas básicas são reescritas por diversas vezes, como por exemplo o redirecionamento, encaminhamento e renderização de *templates*, nesse contexto utiliza-se as funcionalidades para definição de *services* (serviços), onde uma mesma tarefa pode ser escrita uma vez e reutilizada em contextos diferentes de acordo com a implementação de *controllers* diferentes (POTENCIER, 2013).

Para que sejam implementadas as funcionalidades de geração de interfaces CRUD, faz-se necessário a implementação de dois serviços com o apoio do *framework* Symfony2. O primeiro serviço ficará responsável pela implementação do *core* (núcleo) da plataforma, nele são utilizadas as classes responsáveis pela geração de todas as classes pertencentes ao MVC (*model-view-controller*) de cada CRUD (*Scaffold*). O Segundo serviço ficará responsável pela segurança do sistema, atuando com o auxílio de uma classe *Proxy* para as funções de sessão de usuário e autenticação do sistema, e além disso, utilizará uma terceira classe que fornece uma implementação do ACL (*Access Control List*) para o sistema.

3.18.1 Core Service

Como já mencionado, o serviço do *core* do sistema consiste na geração das classes pertencentes ao CRUD seguindo o padrão de arquitetura MVC. Esse serviço é definido através do arquivo *services.yml* no qual faz o apontamento para a classe *GenerateCrud*, e define-se os parâmetros para a injeção de dependência implementados nessa classe (*figura 17*).

```

48     public function __construct(ContainerInterface $container) {
49         $this->container = $container;
50         $this->bundle = $this->container->get('kernel')->getBundle($this->bundle_name);
51     }
52
53     public function setDocument(Document $document){
54         $this->document = $document;
55         $this->entity_name = $this->document->getName();
56     }
57

```

Figura 17: Construtor e método do serviço Core

Ao cadastrar um novo *schema*, o serviço do *core* é chamado fazendo a injeção de dependência (*Dependency Injection*) do *Container* utilizado pela classe *DocumentController*, o qual através do método *generateCrudAction()* (figura 18) é responsável pela definição do *schema* atual como dependência da geração do CRUD através da chamada do método *generate()* existente no *core*.

```

216     /**
217      * Generate CRUD.
218      *
219      * @Route("/{id}/generateCrud", name="generate_crud")
220      * @Template()
221      */
222     public function generateCrudAction(Request $request, $id){
223
224         $em = $this->getDoctrine()->getManager();
225         $entity = $em->getRepository('CrudforgeBundle:Document')->find($id);
226
227         if (!$entity) {
228             throw $this->createNotFoundException('Unable to find Document entity.');
```

Figura 18: Método *generateCrudAction*, do Core Service

Ao executar o método *generate()*, serão realizadas as seguintes operações (definidas no diagrama de sequência da figura 13):

Limpeza do Crud Existente (método protegido *cleanCrudCache()* - figura 19): nesse método é realizado a remoção de qualquer classe gerada anteriormente pelo sistema;

```
66     protected function cleanCrudCache(){
67
68         $bundle_root = $this->bundle->getPath();
69         $entity = $this->entity_name;
70         $files = array(
71             "{$bundle_root}/Controller/{$entity}Controller.php",
72             "{$bundle_root}/Entity/{$entity}.php",
73             "{$bundle_root}/Form/{$entity}Type.php",
74             "{$bundle_root}/Tests/Controller/{$entity}ControllerTest.php",
75             "{$bundle_root}/Resources/views/{$entity}/edit.html.twig",
76             "{$bundle_root}/Resources/views/{$entity}/index.html.twig",
77             "{$bundle_root}/Resources/views/{$entity}/new.html.twig",
78             "{$bundle_root}/Resources/views/{$entity}/show.html.twig",
79         );
80
81         foreach($files as $file){
82             if(file_exists($file)){
83                 unlink($file);
84             }
85         }
86
87         //{$bundle_root}/Resources/config/routing.yml
88     }
```

Figura 19: Método CleanCrudCache

1. Geração da classe entidade (método protegido *generateEntity()* - figura 20): para cada *schema* é gerado uma classe que implementa o ORM (*Object Relational Mapping*) através de anotações (especificadas pelo componente *Doctrine* do *framework* *Symfony2*, alocados no *namespace* *Doctrine\ORM\Mapping*) e da geração das propriedade e métodos dessa classe, que mais tarde será persistida no banco de dados através da criação de uma tabela;

```

90     protected function generateEntity(){
91         /* verifica se arquivo existe */
92         $generator = new DoctrineEntityGenerator($this->container->get('filesystem'), $this->container->get('doctrine'));
93
94         $fields = array();
95         foreach($this->document->getFields() as $field){
96             $fields[$field->getName()] = array('fieldName' => $field->getName(), 'type' => $field->getType(), 'length' => $
97         }
98
99         /**
100          * @todo: replace entity if exists
101          * see: EntityGenerator->setRegenerateEntityIfExists()
102          * função: set
103          */
104         $generator->generate($this->bundle, $this->entity_name, $this->entity_format, array_values($fields), $this->entity_
105
106     }

```

Figura 20: Método generateEntity

- Definição do Dono da Entidade (método protegido *setEntityOwner()* - figura 21): para efeito de segurança, toda classe do tipo *Entity* ao ser gerada, deve ter um dono atribuído para que seja iniciado o ACL (*Access Control List*) do CRUD, dessa forma mais tarde será possível criar regras de permissão para outros usuários colaboradores; nesse método é feito a chamada do serviço de segurança e injetado como dependência o *Container* utilizado pelo serviço do Core, para que seja chamado o método *setClassPermission()* da classe *AclManager* (a qual implementa alguns métodos customizados especificamente para este projeto e que foi estendida da classe *AclManager* pertencente ao bundle *ProblematicAclManagerBundle*), e com isso é definido a permissão de *Owner* para o usuário atual nessa classe *Entity*;

```

123     protected function setEntityOwner(){
124         $security = new CrudforgeSecurity($this->container);
125         $aclManager = $security->getAclManager();
126         $entityClass = $this->container->get('doctrine')->getEntityNamespace($this->bundle_name).'\'.$this->entity_name;
127         $aclManager->setClassPermission($entityClass, MaskBuilder::MASK_OWNER);
128     }
129

```

Figura 21: Método setEntityOwner

- Atualização do *schema* do banco de dados (método protegido *updateSchema()* - figura 22): após definida a classe *Entity* do *schema* e definido o seu ACL, é necessário persistir esse *schema* no banco de dados através da criação (ou alteração) de uma tabela, para isso é chamado o

serviço do componente *Doctrine* do *framework*, e feito a leitura do *schema* do banco de dados através de anotações, propriedades e métodos definidos na classe de entidade do CRUD gerado no item 2; esse método apesar de ser bem simples mas poderoso, teve um alto grau de esforço despendido para chegar em sua implementação, visto que originalmente o *framework* *Symfony2*, não possuem uma API documentada para esse conjunto de tarefas, sendo que a geração de interfaces CRUD somente é realizada através de comandos executados em um terminal, e portanto houve a necessidade de fazer a engenharia reversa desses comandos até se chegar nas classes responsáveis pela geração do CRUD existentes no *framework*;

```
111     protected function updateSchema(){
112         $em = $this->container->get('doctrine')->getManager();
113         $schemaTool = new SchemaTool($em);
114         $metadatas = $em->getMetadataFactory()->getAllMetadata();
115         $schemaTool->updateSchema($metadatas);
116     }
```

Figura 22: Método *updateSchema*

4. Geração das classes das camadas de *controllere view* do *CRUD* (método protegido *generateCrud()* - figura 23): por fim serão gerados as classes pertencentes as camadas de *controller* e *view* do *CRUD* baseados na classe *Entity* do *schema* definido pelo usuário e que seguem um esqueleto pré-definido residente na pasta do *bundle* (no caminho “*./Resources/skeleton/crud*”); assim como no item 4, foi usado de forma intensa a engenharia reversa na implementação desse método, por não haver documentação para a *API* responsável pelo *Scaffold*, existente apenas via linha de comando; nesse método também são atualizadas as rotas utilizadas para navegação entre os *controllers* do lado do *client* (navegador do usuário);

```

133     protected function generateCrud(){
134
135         $prefix = 'user/' . $this->document->getUser()->getId() . '/' . $this->getRoutePrefix($this->entity_name);
136         $entityClass = $this->container->get('doctrine')->getEntityNamespace($this->bundle_name).'\\'.$this->entity_name;
137         $factory = new MetadataFactory($this->container->get('doctrine'));
138         $metadata = $factory->getClassMetadata($entityClass)->getMetadata();
139
140         $generator = new DoctrineCrudGenerator($this->container->get('filesystem'), realpath( __DIR__.'../Resources/skelet
141         $generator->generate($this->bundle, $this->entity_name, $metadata[0], $this->entity_format, $prefix, $this->crud_wi
142
143         $formGenerator = new DoctrineFormGenerator($this->container->get('filesystem'), realpath( __DIR__.'../Resources/sk
144         $formGenerator->generate($this->bundle, $this->entity_name, $metadata[0]);
145
146         //$this->getContainer()->get('filesystem')->mkdir($bundle->getPath().'/Resources/config/');
147         $routing = new RoutingManipulator($this->bundle->getPath().'/Resources/config/routing.yml');
148         try {
149             $ret = $routing->addResource($this->bundle->getName(), $this->entity_format, '/' . $prefix, 'routing/'.strtolower
150         } catch (\RuntimeException $exc) {
151             $ret = false;
152         }
153
154     }

```

Figura 23: Método generateCrud

Os componentes gerados pelo serviço *core* podem ser visualizados no diagrama de componentes (Figura 16).

3.18.2 Security Service

Para a camada de segurança foi implementado um serviço específico para o controle da sessão de usuário, autenticação e autorização (através de *Roles* e *ACL*). Assim como o *Core Service*, o serviço foi configurado no arquivo “services.yml” (Figura 24), definindo-se a classe a ser utilizada (“Crudforge\CrudforgeBundle\CrudforgeCore\CrudforgeSecurity”) e os parâmetros para a injeção de dependências (nesse caso o componente *Container* utilizado nos *Controllers*).

```

1  ## YAML Template.
2  ---
3  parameters:
4      crudforge_generate_crud.class: Crudforge\CrudforgeBundle\CrudforgeCore\GenerateCrud
5      crudforge_security.class: Crudforge\CrudforgeBundle\CrudforgeCore\CrudforgeSecurity
6
7  services:
8      crudforge.core:
9          class:      "%crudforge_generate_crud.class%"
10         arguments: ['@service_container']
11      crudforge.security:
12          class:      "%crudforge_security.class%"
13          arguments: ['@service_container']

```

Figura 24: Definição dos serviços no arquivo *services.yml*

A classe *CrudforgeSecurity* atua segundo o *Design Pattern* (padrão de design) *Proxy*, servindo como ponte para o serviço *Security Context* disponibilizado pelo *framework*, e retornando os objetos necessários para o controle da sessão do usuário (Figura 25), autenticação e autorização baseado em *ACL* (Figura 26).

```

28  /**
29   * Get actual user authenticated
30   * @return \Symfony\Component\Security\Core\User\User
31   */
32  public function getUser(){
33      if(!$this->user){
34          $this->user = $this->container->get('security.context')->getToken()->getUser();
35      }
36      return $this->user;
37  }
38

```

Figura 25: Método fazendo o proxy para o retorno do objeto


```

39  /**
40  * Get AclManager
41  * @return AclManager
42  */
43  public function getAclManager(){
44      if(!$this->aclManager){
45          $aclProvider = $this->container->get('security.acl.provider');
46          $securityContext = $this->container->get('security.context');
47          $objectIdentityRetrievalStrategy = $this->container->get('security.acl.object_identity_retrieval_strategy');
48          $this->aclManager = new AclManager($aclProvider,$securityContext,$objectIdentityRetrievalStrategy);
49      }
50      return $this->aclManager;
51  }
52

```

Figura 26: Método getAclManager

O serviço *Security Context*, fornece uma *API* para a utilização de *ACL*, mas seguindo o princípio *DRY (Don't Repeat Yourself)* da engenharia de *software* e arraigado no conceito de *bundle* do *framework symfony*, foi inserido como dependência do projeto o *bundle ProblematicAclManager*, o qual fornece uma implementação mais amigável da *API* para o *ACL*, contando com uma implementação de métodos que simplifica o uso do *ACL* no projeto.

Ao utilizar o *ProblematicAclManager*, foi encontrado um *bug* (erro) no método *AddPermission()*, dessa forma foi estendido a classe *AclManager* e corrigido este método (figura 26, linha 19), além disso foram adicionados outros 4 métodos para facilitar a verificação da autorização nas classes e nos objetos (figura 27 - linha 39 a 57).

```

1  <?php
2  namespace Crudforge\CrudforgeBundle\CrudforgeCore\Security;
3
4  use Problematic\AclManagerBundle\Domain\AclManager as ProblematicAclManager;
5  use Symfony\Component\Security\Acl\Domain\ObjectIdentity;
6  use Symfony\Component\Security\Core\Exception\AccessDeniedException;
7
8
9  /**
10   * Description of AclManager
11   *
12   * @author wagner
13   */
14  class AclManager extends ProblematicAclManager{
15
16      /**
17       * {@inheritdoc}
18       */
19      protected function addPermission($domainObject, $field, $mask, $securityIdentity = null, $type = 'object', $replace_exis
20          if(is_null($securityIdentity)){
21              $securityIdentity = $this->getUser();
22          }
23          $context = $this->doCreatePermissionContext($type, $field, $securityIdentity, $mask);
24          $oid = $this->getObjectIdentityRetrievalStrategy()->getObjectIdentity($domainObject);
25
26          //fixes setClassPermission without $domainObject
27          if(is_null($oid) && $type=='class'){
28              $oid = new ObjectIdentity('class', $domainObject);
29          }
30
31          $acl = $this->doLoadAcl($oid);
32          $this->doApplyPermission($acl, $context, $replace_existing);
33
34          $this->getAclProvider()->updateAcl($acl);
35
36          return $this;
37      }
38
39      public function isGrantedClass($attributes, $class){
40          $object = new ObjectIdentity('class', $class);
41          return $this->getSecurityContext()->isGranted($attributes, $object);
42      }
43
44      public function isGranted($attributes, $object = null) {
45          return (parent::isGranted($attributes, $object) || $this->isGrantedClass($attributes, get_class($object)));
46      }
47
48      public function checkGrantedClass($attributes, $class){
49          if(!$this->isGrantedClass($attributes, $class)){
50              throw new AccessDeniedException();
51          }
52      }
53
54      public function checkGranted($attributes, $object){
55          if(!$this->isGranted($attributes, $object)){
56              throw new AccessDeniedException();
57          }
58      }
59  }
60  }

```

Figura 27: Correção de bug do método addPermission

A utilização do AclManager para a verificação das permissões de um usuário em uma interface *CRUD* é realizada fazendo o *get* desse componente e verificando a permissão da entidade através do método *checkGrantedClass* (verifica permissão apenas na classe - figura 28, linha 79), ou *checkGranted* (verifica a permissão na classe e no objeto - figura 29, linha 178), caso o usuário não possua permissão será

gerado uma exceção (*figura 30*), que em futuras implementações fornecerá uma mensagem mais amigável para o usuário. Essa divisão em dois métodos, deve ser repensada em implementações futuras, através do *refactoring* da classe *AcIManager*, com uma melhor utilização da orientação ao objeto. Para o protótipo desse projeto, foi realizada a divisão em dois métodos, visto que para algumas ações, a autorização é realizada em cima da classe, visto que, por exemplo, no caso de inserção de dados, não existe ainda uma instância da classe para que seja verificado o *ACL*. Enquanto que em outros casos, por exemplo, na edição de dados, a autorização é realizada em cima de um objeto (instância da classe), no qual além de ser verificado o *ACL* para o mesmo, também é realizada uma herança do *ACL* da classe (*figura 27 - linha 45*).

```
69 | /**
70 |  * Displays a form to create a new livros entity.
71 |  *
72 |  * @Route("/new", name="user_4_livros_new")
73 |  * @Template()
74 |  */
75 | public function newAction()
76 | {
77 |     $em = $this->getDoctrine()->getManager();
78 |     $this->get('crudforge.security')->getAcIManager()
79 |         ->checkGrantedClass('CREATE', $em->getRepository('CrudforgeBundle:livros')->getClassName());
80 |
81 |     $entity = new livros();
82 |     $form = $this->createForm(new livrosType(), $entity);
83 |
84 |     return array(
85 |         'entity' => $entity,
86 |         'form' => $form->createView(),
87 |     );
88 | }
89 |
```

Figura 28: Utilização do método *checkGrantedClass*

```

121 | /**
122 |  * Displays a form to edit an existing livros entity.
123 |  *
124 |  * @Route("/{id}/edit", name="user_4_livros_edit")
125 |  * @Template()
126 |  */
127 | public function editAction($id)
128 | {
129 |     $em = $this->getDoctrine()->getManager();
130 |
131 |     $entity = $em->getRepository('CrudforgeBundle:livros')->find($id);
132 |
133 |     if (!$entity) {
134 |         throw $this->createNotFoundException('Unable to find livros entity.');
```

Figura 29: Utilização do método `checkGranted`

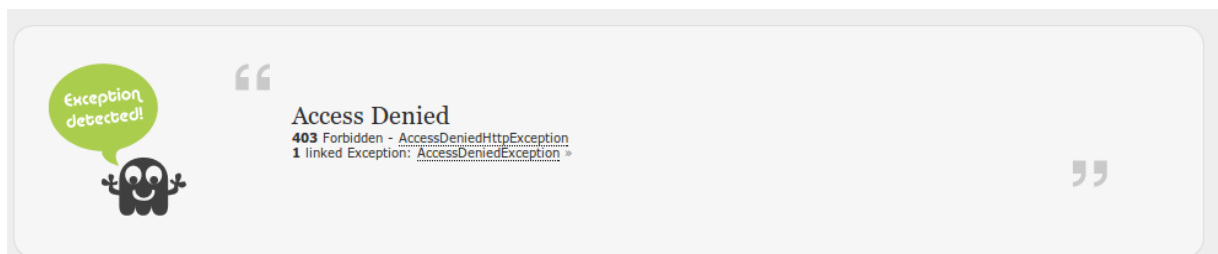


Figura 30: Visualização da tela para acesso negado

3.19 Referências de Hardware

O sistema aqui descrito foi concebido para ser uma aplicação em nuvem (*cloud computing*), ou seja, será armazenado em ambiente web.

Um desses ambientes é o Amazon EC2 (*Elastic Compute Cloud*), uma plataforma que permite ao usuário a alocação de computadores virtuais para rodarem suas próprias aplicações online.

3.19.1 Requisito Mínimos

Para a realização do projeto foi utilizada a micro instância *t1.micro* da *Amazon*. Ela funciona como um servidor virtual, onde nela são executados os aplicativos. Ela é fornecida através do *AWS (Amazon Web Services)*, que é uma plataforma disponibilizada pela Amazon nos moldes do *PaaS (Platform as a Service)* com o objetivo de oferecer uma plataforma de infraestrutura escalável e de fácil gerenciamento online.

Essa micro instância é utilizada para testes de servidores dentro da estrutura AWS ou para projetos com baixos requisitos de *hardware*, uma vez que apresentam uma quantidade reduzida de recursos de processamento e memória (*tabela 6*). Porém quando ciclos adicionais são necessários, é possível aumentar a capacidade de processamento e memória, alterando a definição do tipo de instância utilizada, ou ainda utilizando outros recursos para escalonamento da infraestrutura (*anexo I*) como os serviços:

1. **AWS Auto Scaling**: permite expandir ou reduzir a capacidade do Amazon EC2 automaticamente, de acordo com condições pré-definidas;
2. **AWS Elastic Load Balancing**: distribui automaticamente o tráfego de entrada dos aplicativos em várias instâncias do EC2;
3. **Amazon ElastiCache**: serviço web que torna fácil implantar, operar e escalar um cache na memória na nuvem;

Tabela 6: Especificação Técnica - Instância *t1.micro*

Família de Instâncias	Tipo de Instâncias	Arquitetura do Processador	vCPU	ECU	Memória	Armazenamento da Instância	Otimização disponível para EBS	Desempenho de Rede
Microinstâncias	t1.micro	32 bits/64 bits	1	Variável	0,615	Somente EBS	-	Muito baixo

4. CAPÍTULO - APRESENTAÇÃO DOS RESULTADOS

A seção de análise dos resultados consiste na definição das dimensões que representam o objeto de análise desse trabalho. As informações foram originadas da avaliação da plataforma CRUDForge e foram relevantes para alcançar os objetivos propostos para esse trabalho.

4.1 Testes

Ao longo do processo de desenvolvimento da plataforma, visando manter a qualidade e a confiabilidade da aplicação, foram criados testes unitários e funcionais, a fim de verificar a existência de novos erros ao longo do processo de desenvolvimento.

Os testes Unitários foram utilizados para verificar o comportamento das principais funções do core da aplicação, já os testes funcionais, foram utilizados para verificar a integração das camadas MVC.

O framework Symfony2, se integra com uma biblioteca chamada PHPUnit, disponibilizando uma ótima ferramenta de testes, possibilitando executar os testes unitários e funcionais através de linha de comandos em um terminal, tornando a realização de testes simples e ágeis (POTENCIER, 2013).

Teste 1 - Teste unitário das funções da classe `GenerateCrud.php`

Foi criado para este teste, uma classe chamada *GenerateCrudTest.php*, com o objetivo de testar as principais funções do core da aplicação, nessa classe foram criados os seguintes métodos para:

- verificar a limpeza de cache, afim de procurar a existência de vestígios do último *schema* criado pelo usuário;
- testar a criação de *schemas*, afim de verificar possíveis erros ocasionados pela criação dos pacotes;
- testar a criação das classes, afim de verificar o sucesso na geração do CRUD;

Teste 2 - Teste funcional de Autenticação de usuário.

Esse teste foi criado na classe *CrudforgeSecurityTest.php*, com o objetivo de verificar se o usuário foi autenticado no sistema, caso não estivesse, seria verificado também, se ele seria redirecionado para a pagina de *login*.

4.2 Protótipo

Para este projeto foi definido um protótipo, no qual a criação dos usuários será apenas permitida pelo administrador desta plataforma (*figura 31*).

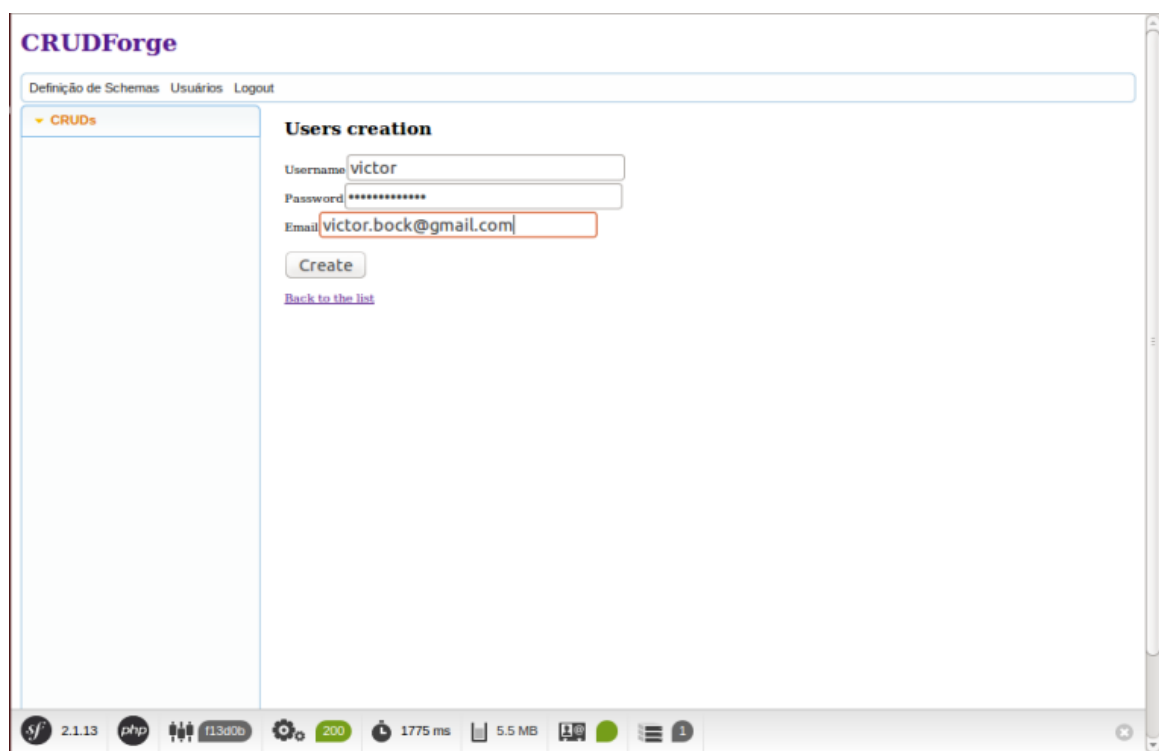


Figura 31: Tela de cadastro de usuários

Como também foi definido que a edição e exclusão dos usuários (*figura 32*), será apenas permitida pelo administrador desta plataforma.

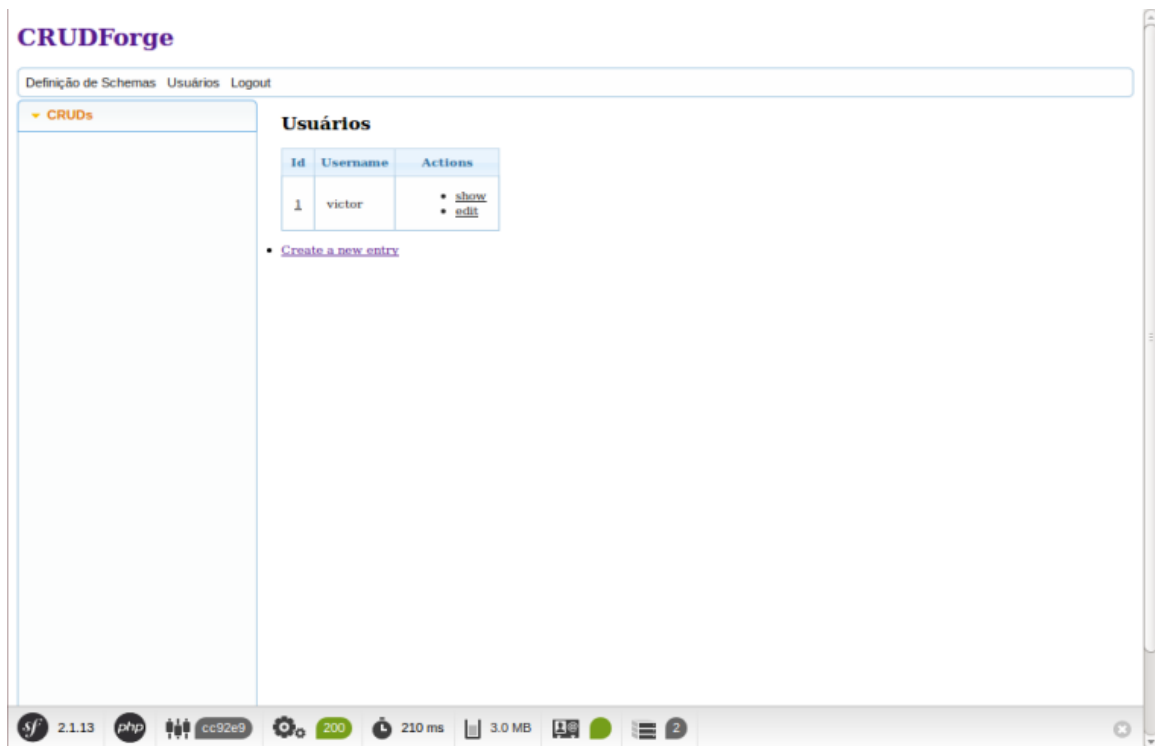


Figura 32: Listagem dos usuários

Para a autenticação dos usuários cadastrados, foi implementado uma tela de login (figura 33).

CRUDForge

Username:

Password:

Figura 33: Tela de login

Após o *login* ser efetuado com sucesso, o usuário pode criar novos *schemas* (figura 34).

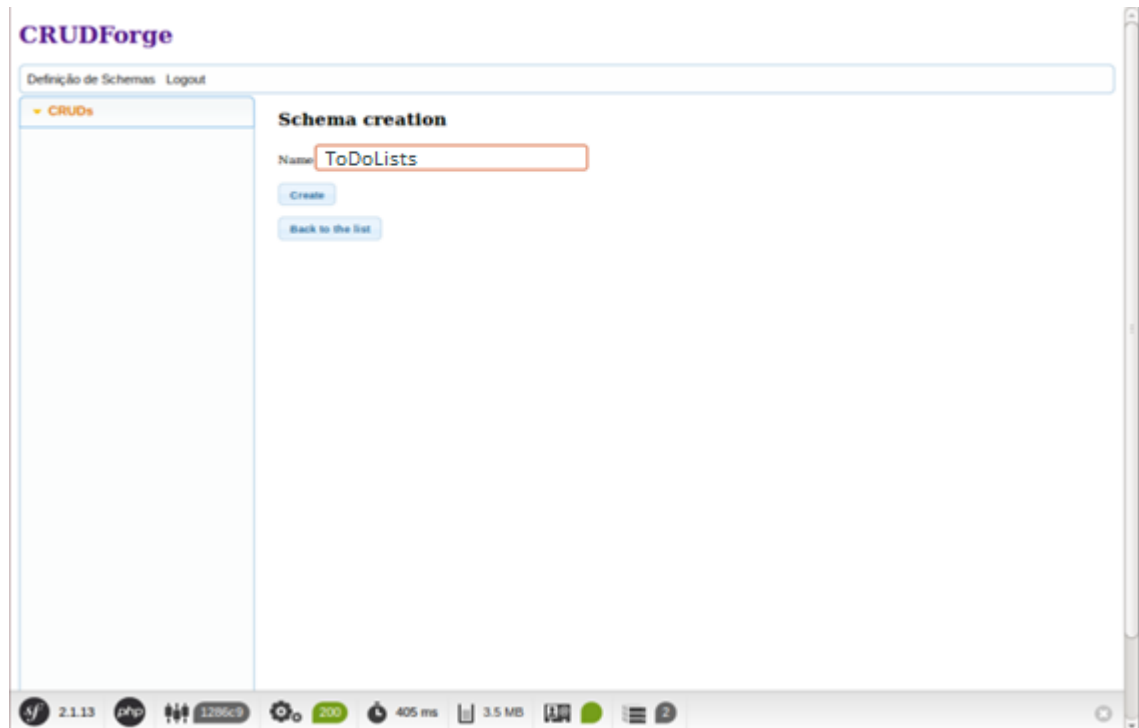


Figura 34: Criando um schema

Após a definição do *schema*, esse registro pode ser alterado, como também pode voltar a lista, onde é possível adicionar campos ao novo *schema* (figura 35).

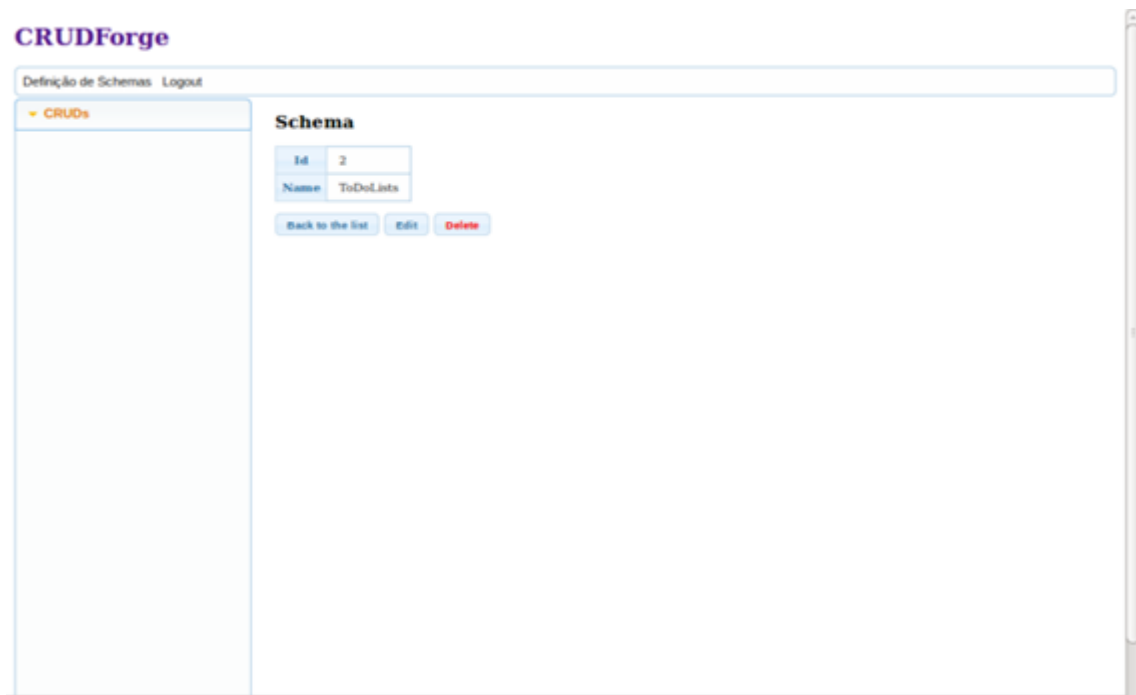


Figura 35: Listagem dos schemas do usuário

É possível visualizar a tela onde adiciona os novos campos na figura 36.

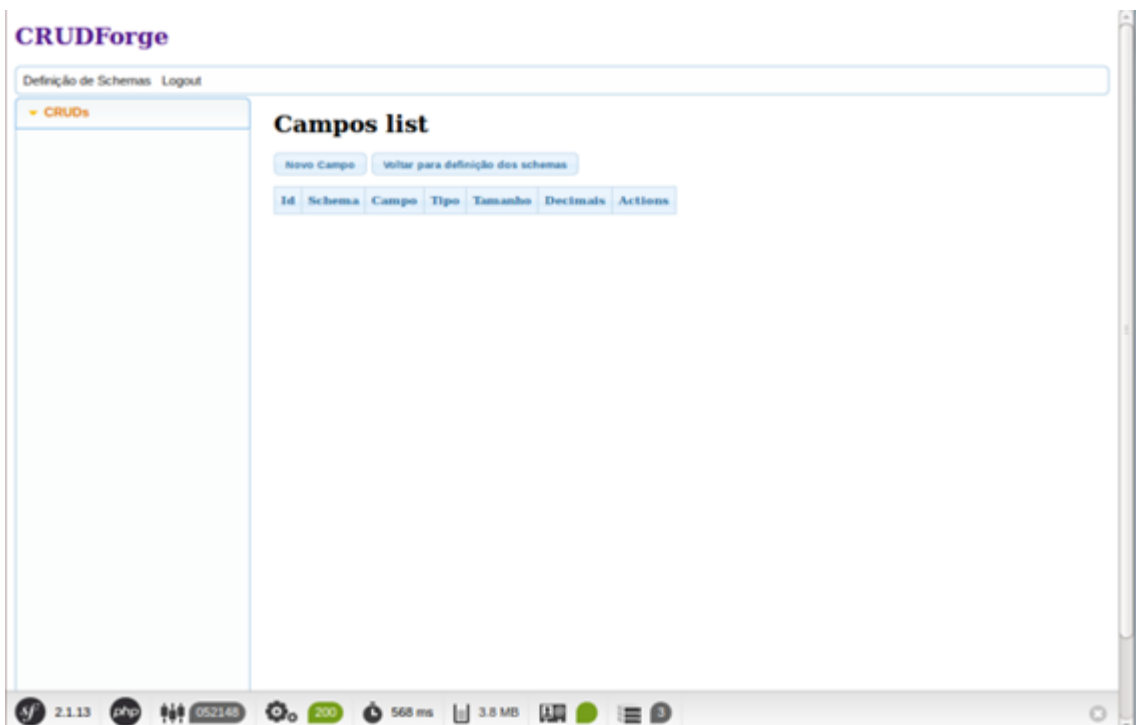
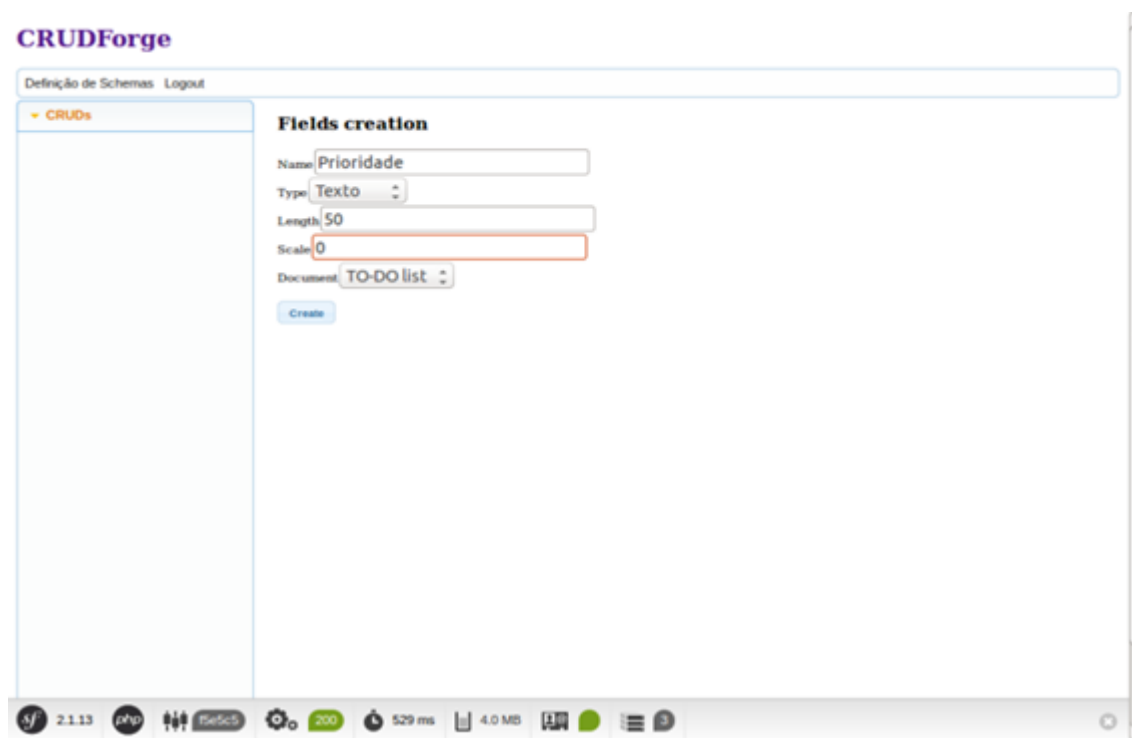


Figura 36: Tela adicionando novos campos

Os usuários podem definir o nome do campo, tipo, tamanho, escala e o documento (figura 37). Porém para o futuro pretendemos apenas deixar nome e tipo, desta forma facilitando o uso.



The screenshot displays the 'CRUDForge' application interface. At the top, there's a header with 'Definição de Schemas' and 'Logout'. Below this, a sidebar on the left shows a tree view with 'CRUDs' expanded. The main content area is titled 'Fields creation'. It contains a form with the following fields: 'Name' (text input with 'Prioridade'), 'Type' (dropdown menu with 'Texto'), 'Length' (text input with '50'), 'Scale' (text input with '0'), and 'Document' (dropdown menu with 'TO-DO list'). A 'Create' button is located at the bottom of the form. The bottom of the image shows a system tray with various icons and status information like '2.1.13', 'php', '529 ms', and '4.0 MB'.

Figura 37: Tela de cadastro de um novo campo

É possível observar, o *schema* após a criação de alguns campos na figura 38.

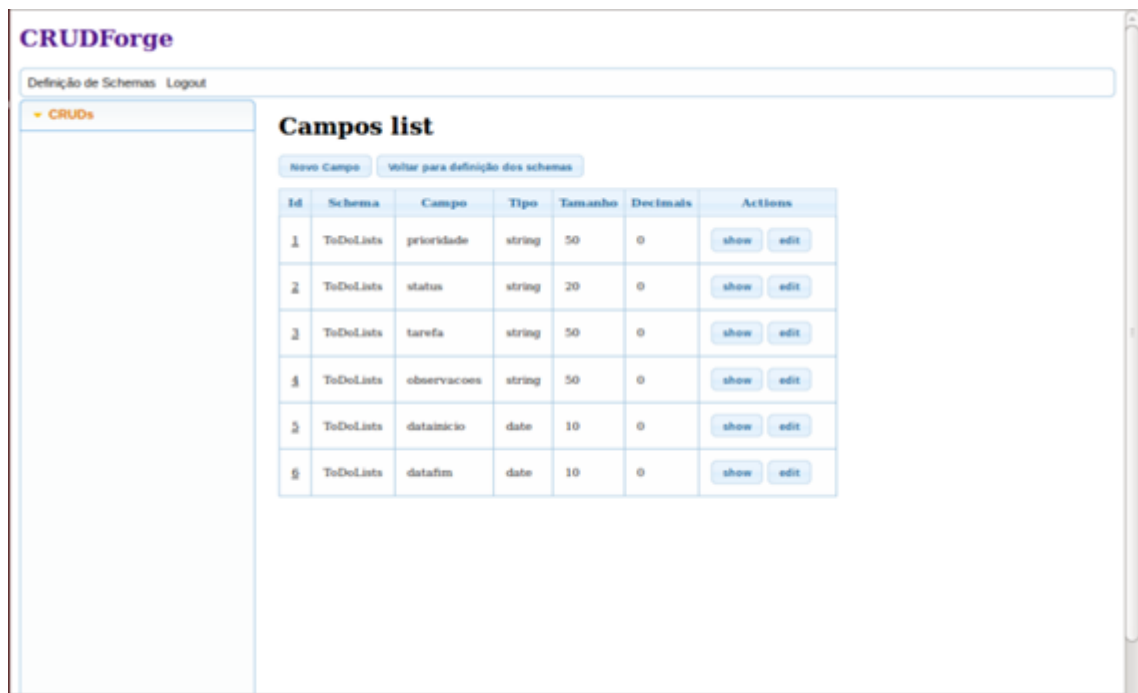


Figura 38: Tela de listagem dos campos de um schema

Após todos os campos serem criados, o usuário pode selecionar o botão “gerar CRUD” que está na figura 39 e desta forma o CRUD será gerado.

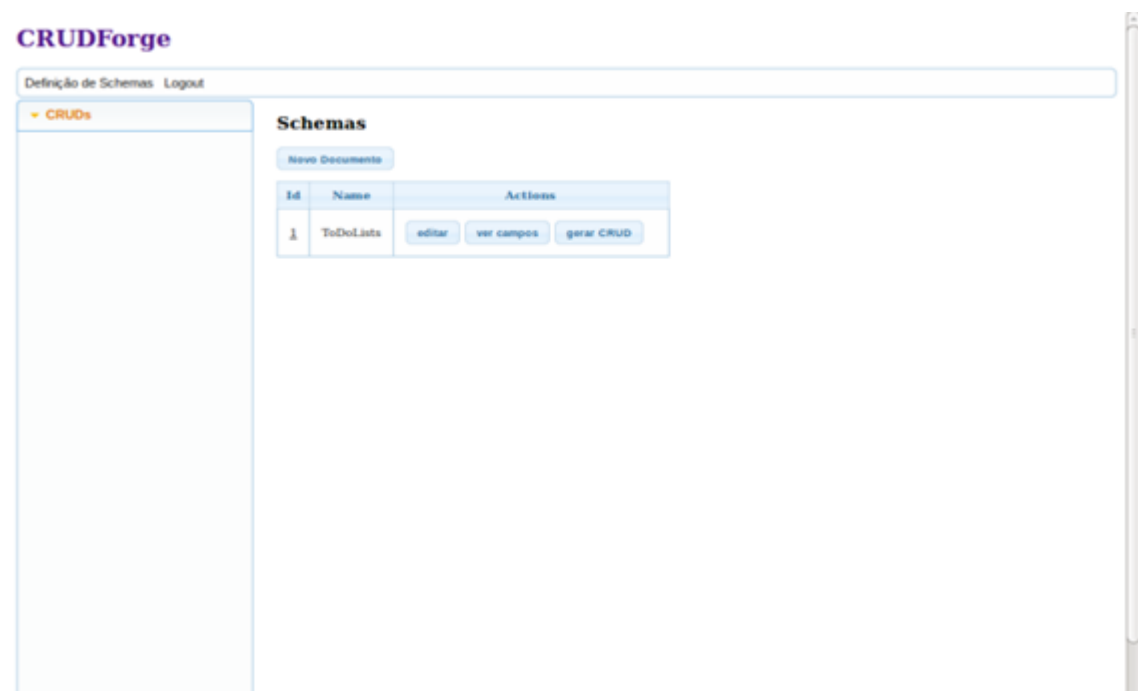


Figura 39: Tela onde é visualizado o botão para geração do CRUD

É possível visualizar o CRUD ToDoList gerado e com os campos que foram definidos pelos usuário na *figura 40*.

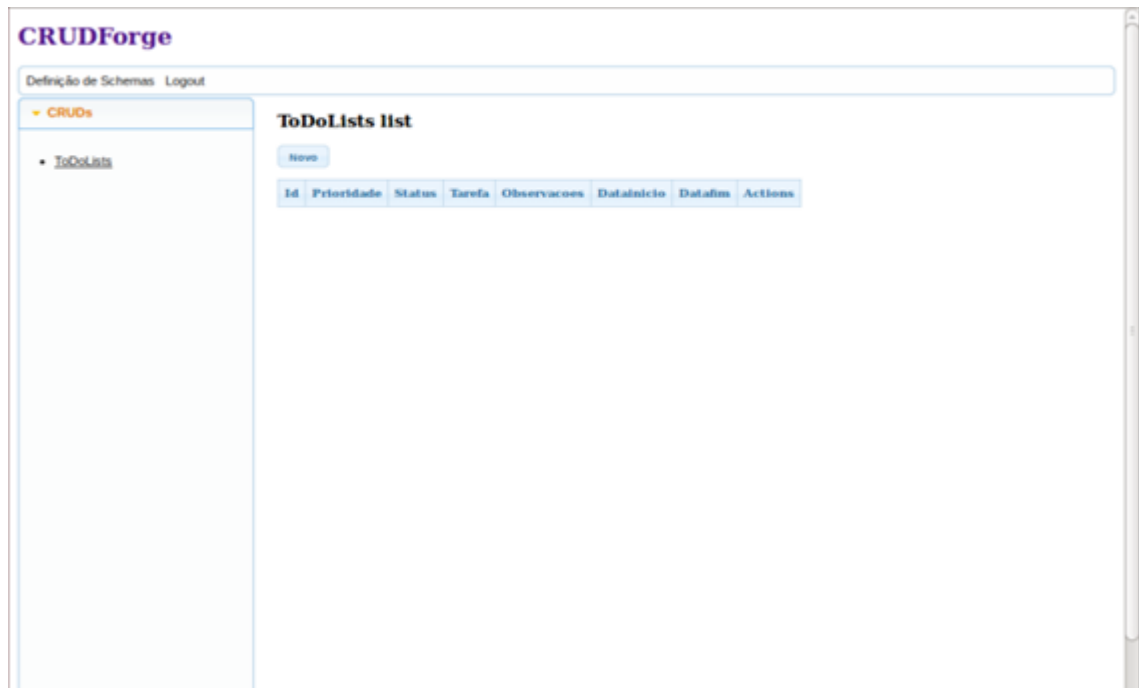


Figura 40: Tela principal do CRUD gerado pronto para utilização

Após o CRUD ser gerado é possível adicionar dados (*figura 41*).

CRUDForge

Definição de Schemas Logout

▼ CRUDs

- [ToDoLists](#)

ToDoLists creation

Prioridade:

Status:

Tarefa:

Observacoes:

Datainicio:

Datafim:

Figura 41: Tela de cadastro de dados do CRUD

É possível o usuário visualizar os dados gerados (figura 42), como também pode editar ou deletar esses dados.

CRUDForge

Definição de Schemas Logout

▼ CRUDs

- [ToDoLists](#)

ToDoLists

Id	1
Prioridade	Alta
Status	Novo
Tarefa	TCC
Observacoes	Considerações finais
Datainicio	2013-01-02 00:00:00
Datafim	2013-12-14 00:00:00

Figura 42: Tela de visualização do registro cadastrado

É demonstrada a inserção dos dados (*figura 43*).

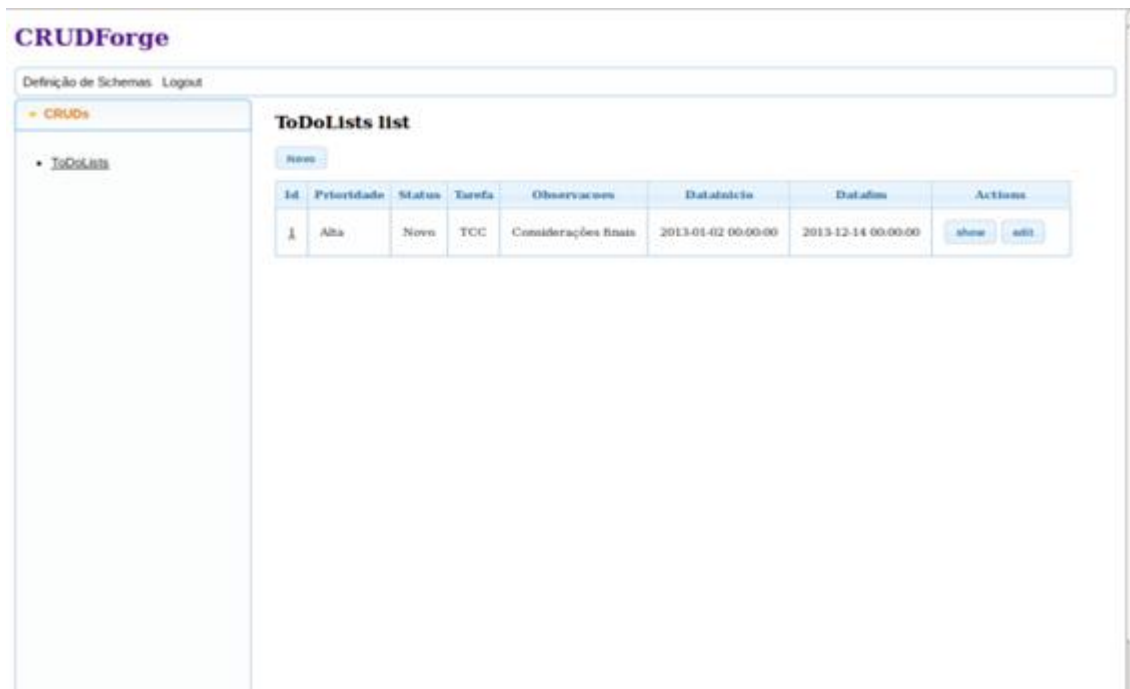


Figura 43: Tela com a listagem dos registros cadastrados

É demonstrado a possibilidade de edição dos dados (*figura 44*).

CRUDForge

Definição de Schemas Logout

▼ CRUDs

- [ToDoLists](#)

ToDoLists edit

Prioridade

Status

Tarefa

Observacoes

Datainicio

Datafim

Figura 44: Tela para edição de um registro

É demonstrada a alteração dos dados nos campos “Observacoes” e “Datafim” (figura 45).

CRUDForge

Definição de Schemas Logout

▼ CRUDs

- [ToDoLists](#)

ToDoLists list

Id	Prioridade	Status	Tarefa	Observacoes	Datainicio	Datafim	Actions
<u>1</u>	Alta	Novo	TCC	Alteração da data fim	2013-01-02 00:00:00	2013-12-13 00:00:00	<input type="button" value="show"/> <input type="button" value="edit"/>

Figura 45: Tela de listagem dos registros após a alteração

A fim de demonstrarmos a extensibilidade, adicionamos um novo campo no CRUD gerado e com dados (figura 46), além disso, foi alterado o campo “Observacao” para “Obs”.

The screenshot shows the CRUDForge web application. On the left, a sidebar contains a 'CRUDs' section with a link to 'ToDoLists'. The top navigation bar includes 'Definição de Schemas' and 'Logout'. The main content area is titled 'Campos list' and features a 'Novo Campo' button and a 'Voltar para definição dos schemas' button. Below these is a table listing the fields for the 'ToDoLists' schema.

Id	Schema	Campo	Tipo	Tamanho	Decimals	Actions
1	ToDoLists	prioridade	string	50	0	show edit
2	ToDoLists	status	string	20	0	show edit
3	ToDoLists	tarefa	string	50	0	show edit
4	ToDoLists	obs	string	50	0	show edit
5	ToDoLists	datainicio	date	10	0	show edit
6	ToDoLists	datafim	date	10	0	show edit
7	ToDoLists	Ref	string	50	0	show edit

Figura 46: Tela com listagem dos campos cadastrados

Como podemos observar na figura 47, os dados não foram perdidos ou corrompidos, mesmo após um dos campos ter o seu nome alterado, os dados permaneceram intactos.

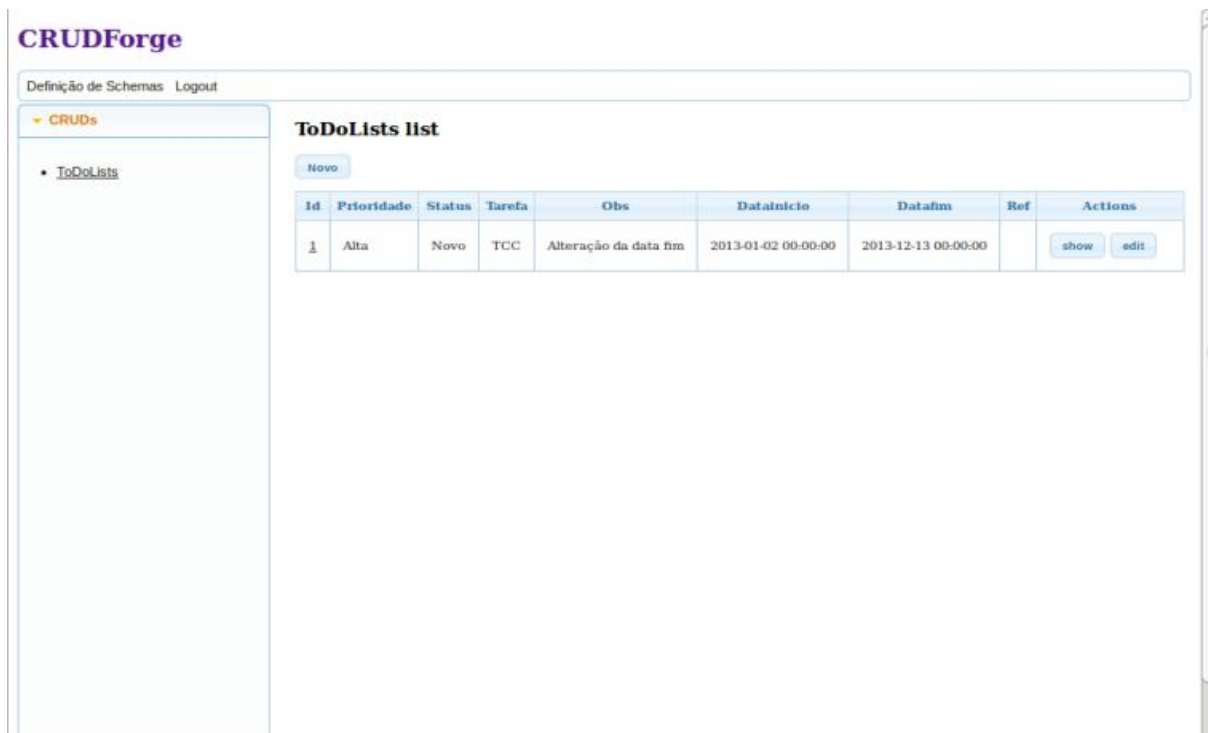


Figura 47: Tela após a extensibilidade dos dados

É possível visualizar a tela de compartilhamento (figura 48), no qual o usuário pode definir com quem quer compartilhar o CRUD e definir o modo de acesso do usuário escolhido.

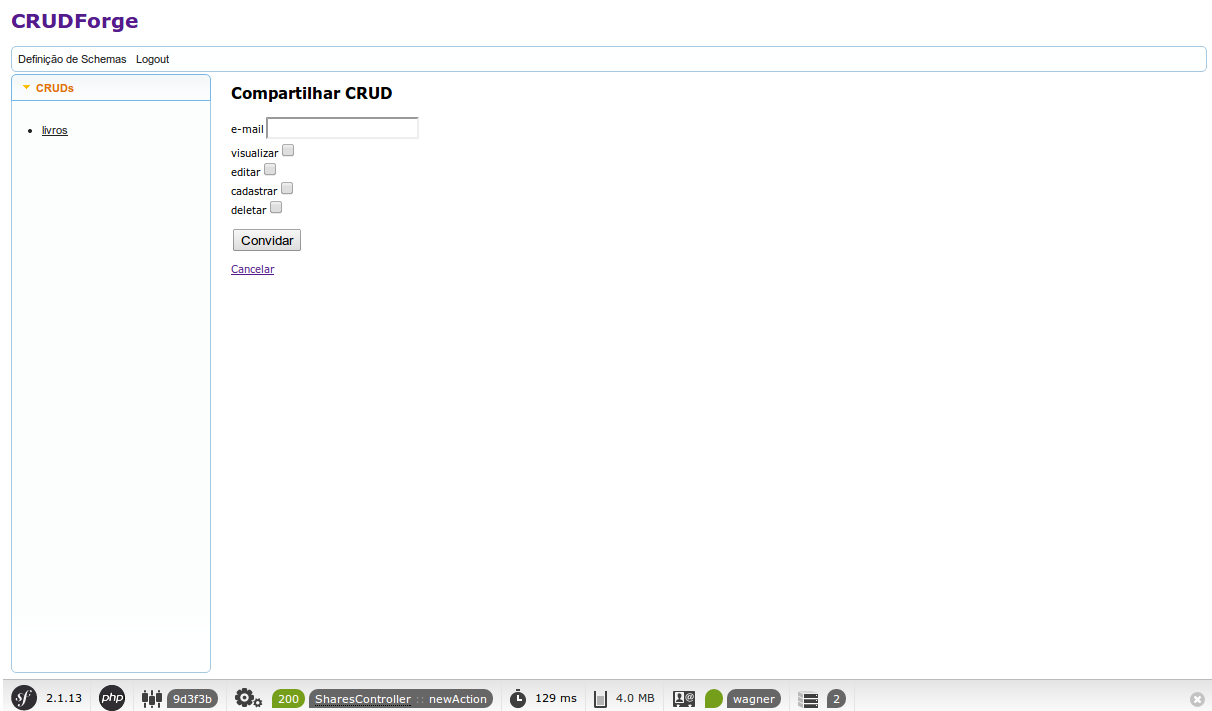


Figura 48: Tela para compartilhamento do CRUD

São demonstradas as permissões que foram dadas, de modo que este outro usuário só poderá visualizar e editar as informações no CRUD, não podendo cadastrar nenhuma informação e nem apagar (figura 49).

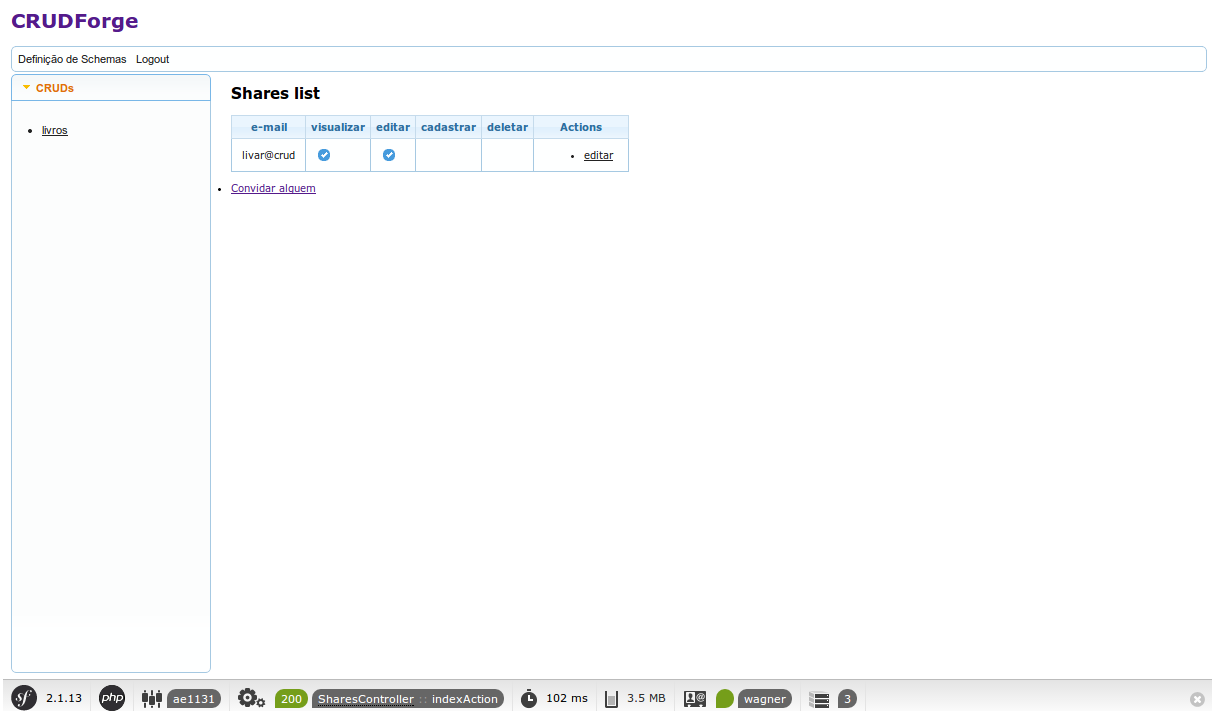


Figura 49: Tela com a listagem dos compartilhamentos

Contudo, segundo Mendes (2011), a habilidade de criar os dados desejados é chamada de extensibilidade. Ou seja, quando um código não é útil para uma nova aplicação, fazemos uso do reuso para revisões e adaptações, o que pode tornar o software sujeito a erros. Porém demonstramos a integridade da plataforma.

Com isso foi demonstrado que o CRUDForge consegue ser uma plataforma para gestão de dados de forma colaborativa e extensível de alto nível.

CONCLUSÃO

Tendo como foco principal o fornecimento de um sistema na nuvem que faça o *scaffold* de acordo com as especificações do usuário em um *schema*, o projeto cumpriu com os objetivos esperados, visto que, foram comprovadas as funcionalidades especificadas no projeto e validados os objetivos específicos.

O próximo passo para o sucesso do projeto seria uma análise pormenorizada sobre investimentos (Apêndice V) e retornos necessários para tornar o produto comercializável, trabalho esse que foi feito de modo simplificado ao analisar um planejamento financeiro para o projeto (Apêndice VI). Assim como será necessário implementar um refinamento da usabilidade, fluxo de trabalho e design das interfaces mais amigáveis.

Uma dificuldade foi trabalhar com o *framework Symphony2*, devido ao fato de ser uma ferramenta nova e com documentação escassa, sendo assim em muitos momentos tivemos que fazer a engenharia reversa para descobrir como funcionava a API de certos componentes. Outro desafio durante o projeto foi trabalhar com um grupo de pessoas com diferentes habilidades e deficiências, sendo que com a troca de experiências, foram superados tais desafios. Concluindo, foram superadas as dificuldades técnicas, humanas e deixamos o caminho para uma possível comercialização do produto.

O objetivo geral do trabalho foi desenvolver um sistema *online* que trabalha em cima de uma plataforma em nuvem, e cujo foco principal é permitir ao usuário inserir e gerar seus códigos de acordo com a sua necessidade.

Para isso foram implementadas todas as tecnologias aqui descritas, de maneira com que viabilizasse de maneira eficiente a implantação do sistema, atendendo ao escopo proposto.

O protótipo produzido atendeu as expectativas esperadas, uma vez que os três cenários propostos (*To-Do List*, DVDs e Contas a Pagar e Receber) foram concebidos e implementados de maneira eficaz, realizando os pontos principais do projeto: geração de código, definição de *schemas*, colaboração entre os usuários e extensibilidade, possibilitando assim fazer a adição de novos campos nos *schemas* já criados sem a perda dos dados que já foram gerados, como consta na

especificação desse documento, servindo como base de aprendizado para futuras implementações.

Como qualquer projeto, foram encontradas inúmeras dificuldades para a concepção e realização desse trabalho, sendo elas de âmbito técnico, gerencial, funcionais e até mesmo pessoais.

Entre tantas dificuldades podemos destacar a utilização do *framework Symphony2*, uma vez que a ferramenta possui uma documentação que não atende todas as necessidades para o desenvolvimento do projeto, sendo necessária fazer uma Engenharia Reversa para um melhor entendimento sobre o funcionamento da API de alguns componentes.

Outro ponto importante a ser considerado, e muitas vezes difícil de ser entendido, foi o alinhamento entre que foi proposto e os conhecimentos técnicos (sejam eles teóricos ou por experiência de trabalho) do envolvidos no projeto. Tal ponto foi previamente estudado antes do *start* do projeto, para evitar conflitos internos que poderiam ocorrer caso houvesse um mau gerenciamento das atividades propostas, além da má distribuição das tarefas.

Além disso, próximo passo para o sucesso do projeto seria uma análise pormenorizada sobre investimentos (Apêndice V) e retornos necessários para tornar o produto comercializável, trabalho esse que foi feito de modo simplificado ao analisar um planejamento financeiro para o projeto (Apêndice VI). Assim como será necessário implementar um refinamento da usabilidade, fluxo de trabalho e design das interfaces mais amigáveis.

Apesar de todas as dificuldades encontradas e superadas, das expectativas propostas e atingidas e das melhorias a serem implementadas, entendemos que a conclusão desse projeto não deve ficar restrita academicamente. Há um mercado que poderá ser explorado com as futuras incrementações a serem feitas, porém não foram possíveis devido ao foco do projeto, ao escopo e ao tempo para o seu desenvolvimento.

REFERÊNCIAS

ALVAREZ, A; **O que é javascript**. Disponível em:

<<http://www.criarweb.com/artigos/184.php>>. Acesso em: 29 de nov. 2013.

BECK, K.; **Programação Extrema (XP) Explicada**. Editora Bookman, 2004.

BOWLER, T.; VANCER, W.; **Symfony 1.3 Web Application Development**. Editora Packt Publishing Ltd, 2009.

BRANAS, R.; **AngularJS: Criando Integrações com Spring MVC e Hibernate**.

Disponível em: <<http://www.devmedia.com.br/angularjs-criando-integracoes-com-spring-mvc-e-hibernate/29295#ixzz2m4hRiADw>>. Acesso em: 29 de nov. 2013.

BRETERNITZ, V. **A SELEÇÃO DE SISTEMAS ERP (ENTERPRISE RESOURCE PLANNING) PARA PEQUENAS E MÉDIAS EMPRESAS**.

COCKBURN, A.; **Escrevendo Casos de Uso Eficazes**. Editora Bookman, 2005.

CRUZ, T. **Uso e desuso de sistemas de Workflow: Porque as organizações não conseguem obter retorno com investimentos em projetos de Workflow**. Editora E-papers.

DEVMEDIA. **Trabalhando com engenharia reversa**. Disponível em

<<http://www.devmedia.com.br/trabalhando-com-engenharia-reversa-revista-engenharia-de-software-magazine-59/28203#ixzz2m4o4O9fq>>. Acesso em: 29 de nov. 2013.

DOW, W.; TAYLOR, B. **Project Management Communications Bible**. Ed. John Wiley & Sons, 2010. 1272 páginas.

DUARTE, E.; **Artigo SQL Magazine 37 - MySQL**. Disponível em: <<http://www.devmedia.com.br/artigo-sql-magazine-37-mysql/6890>>. Acesso em: 04 de dez. 2013.

ECLIPSE; **The Eclipse Foundation**. Disponível em <<http://www.eclipse.org/>>. Acesso em: 25 de nov. 2013.

ERL, T.; **SOA - Princípios de Design de Serviço**. Editora Pearson.

MENDES, A.; **Introdução à Programação Orientada a Objetos Com C++**. Editora Elsevier Brasil, 2011

FUSCO, J.P.A. **Tópicos emergentes em engenharia de produção - vol 02**.

Google Drive; **Google Drive: tenha até 5 Gb para armazenar seus arquivos em nuvem**. Disponível em <<http://www.techtudo.com.br/tudo-sobre/s/google-drive.html>>, acesso em: 29 de nov. 2013

GUEDES, T.A. GILLEANES; **UML2 Uma abordagem prática**. Editora Novatec

FREEMAN, Steve. Pryce, Nat. **Desenvolvimento de Software Orientado a objetos, Guiado por Testes**. Rio de Janeiro: Alta Books, 2012.

IMASTERS; **GitHub ultrapassa a marca de três milhões de usuários**. Disponível em: <<http://imasters.com.br/noticia/github-ultrapassa-a-marca-de-tres-milhoes-de-usuarios>> Acesso em 25 de nov. 2013.

JANG, MICHAEL; **Ubuntu Server Administration**. Editora Osborne. 2008.

JONES, C., **Patterns of Software Systems Failure and Success**. London: International Thompson Computer Press, 1996

K-19; **Orientação a Objetos em Java**. Disponível em:

<<http://www.k19.com.br/downloads/apostilas/java/k19-k11-orientacao-a-objetos-em-java>> . Acesso em 25 de nov. 2013.

LADEIA, B.; **Home Office ainda é tabu para empresas brasileiras**. Disponível em:

<<http://exame.abril.com.br/gestao/noticias/pratica-de-home-office-ainda-tabu-nas-empresas-brasileiras?page=1>>. Acesso em: 29 de nov. 2013.

LEITE, M.; **Técnicas de Programação - Uma Abordagem Moderna**. Editora Brasport.

LINS, S. **Desafios Sistêmicos: Lições aprendidas por consultores e executivos que vivenciaram a implementação de sistemas**. Editora E-papers.

LIRA E ASSOCIADOS ADVOCACIA. Disponível em:

<<http://www.liraa.com.br/conteudo/2421/>>. acesso em: 29 de nov. 2013.

Manifesto Ágil. Varios autores.; disponível em: <<http://manifestoagil.com.br/>>, acesso em: 25 mar. 2012;

MARTINS, D. M. S.; **Projeto de Software com Astah**. Disponível em:

<http://www.devmedia.com.br/projeto-de-software-com-astah*-engenharia-de-software-30/18442#ixzz2m4lnzW7f>. Acesso em: 29 de nov. 2013.

NETO, O. M. **Entendendo e dominando o Java - 3a edição**. Ed. Universo dos Livros. 416 páginas.

NETBEANS; **WelcometoNetBeans**. Disponível em: <<https://netbeans.org/>>. Acesso em: 25 de nov. 2013.

NIEDERAUER, J.; **Desenvolvendo Websites com PHP**. Editora Novatec. 2011.

NIELSEN, J., **Usability Engineering**. Academic Press, Cambridge, MA. 1993.

OPENSUSE; **GIT**. Disponível em: <<http://pt.opensuse.org/Git>> Acesso em: 29 de nov. 2013.

POREBSKI, B.; PRZYSTALSKI, K.; NOWAK, L.; **Building PHP Applications with Symfony, CakePHP, and Zend Framework**. Editora John Wiley and Sons, 2011.

POTENCIER, F.; **Symfony 2.3 - The Book**. Sensio Sa, 2013.

POTENCIER, F.; **Practical Symfony 1.3 & 1.4 for Doctrine**. Sensio Sa, 2009.

POMPILHO, S. **Análise Essencial Guia Prático de Análise de Sistemas**. Rio de Janeiro: Ed Ciência Moderna Ltda, 1995.

PRESSMAN, S. R.; **Engenharia de Software 7 - Uma Abordagem Profissional**. Editora AMGH Editora. 2011

RAMOS, R.A.; **Treinamento Prático em UML**. Editora Universo dos Livros. 2006.

RIVERA, M. C. M.; **MVC**. Disponível em: <<http://www.devmedia.com.br/mvc/12202>>. Acesso em: 29 de nov. 2013.

SILVA, S. M.; **HTML 5 - A Linguagem de Marcação que Revolucionou a Web**. Editora Novatec, 2011.

SPARKS, D. E.; HAGIU, A.; SCHMALENSEE, R.L.; **Invisible Engines: How Software Platforms Drive Innovation And Transform Industries**.

STEPHENS, R.; **Beginning Database Design Solutions**. Ed. John Wiley & Sons, 2010. 400 páginas.

SENSIO LABS. **Howto use Access ControlLists (ACLs)**. Disponível em: <<http://symfony.com/doc/current/cookbook/security/acl.html>>. Acesso em: 29 de nov. 2013.

SOMMERVILLE, Ian. Engenharia de software. São Paulo:Person, 2010.

SOUZA, A.; **Servidor Web Apache, o que esperar?** Disponível em: <<http://www.devmedia.com.br/servidor-web-apache-o-que-esperar/7096#ixzz2mINZtFpw>>. Acesso em: 29 de nov. 2013.

TAVARES, J; NETO, J.B.R.; HOFFMANN, S.C; **Sistemas de Gestao Integrados**. SENAC SAO PAULO. 2008.

TAVARES, LUIS ANTONIO; **JQueryUI na prática**. Disponível em: <<http://www.devmedia.com.br/jquery-ui-na-pratica-revista-java-magazine-96/22542#ixzz2m4ekgy3U>>. Acesso em: 29 de nov. 2013.

TELES, V.M.; **Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade**. Editora Novatec. 2005

TIWARI, S.; **Professional BlazeDS: Creating Rich Internet Applications with Flex and Java**. Ed. John Wiley & Sons, 2011. 384 páginas.

TURBAN, E.; WETHERBE, J.C.; MCLEAN, J. **Tecnologia Da Informacao Para Gestao**. Bookman.

ZANINOTTO, F.; POTENCIER, F.; **The Definitive Guide to Symfony**. Editora Apress, 2007.

APÊNDICE I: EAP

A EAP (Estrutura Analítica de Projeto) é um dos meios mais simples para se entender a estrutura como um todo do projeto. Seu principal objetivo é quebrar o projeto em pequenas partes e com isso ter um melhor controle, o que o torne mais fácil de compreender.

Pensando nisso, a EAP do projeto foi projetada como especificada abaixo:

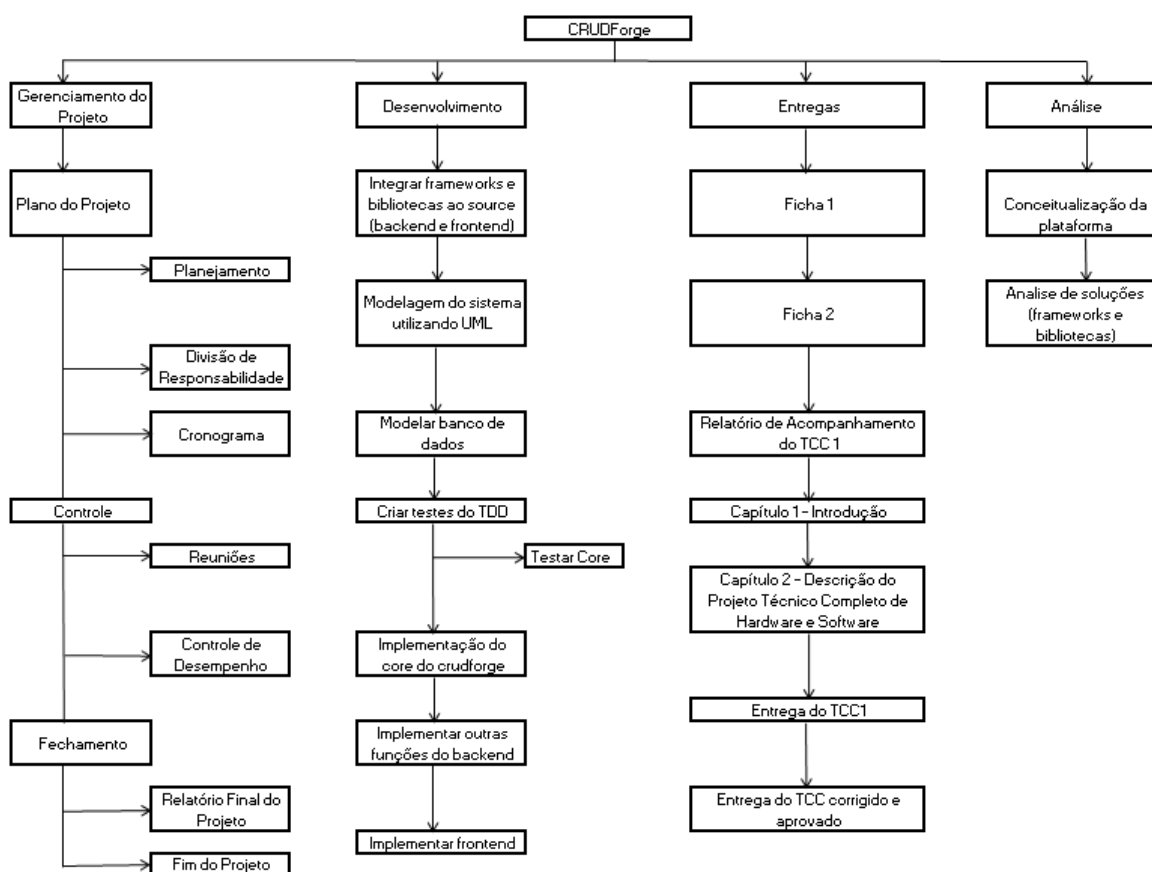


Figura 50: Estrutura Analítica do Projeto

Pode-se observar o projeto dividido em quatro frentes principais (Gerenciamento do Projeto, Desenvolvimento, Entregas e Análise), nas quais cada uma possui suas fases divididas em sub-frentes.

A frente Gerenciamento do Projeto consiste em representar as fases relacionadas a gestão do projeto, seu foco é mais administrativo, cuidando assim que gerir todas as outras frentes do projeto.

Já o Desenvolvimento envolve questões de âmbito técnico, necessários para a construção do projeto. Suas especificações vão desde a modelagem, até implementações dos componentes do sistema.

As questões relacionadas a parte burocrática do projeto, ou seja, questões que envolvem documentação do projeto, são tratadas na frente de Entrega. Seu principal objetivo é obter o controle de toda a documentação gerada pelo projeto, e que tem por obrigatoriedade a realização das entregas dos mesmos.

Por fim temos a frente de Análise, talvez a mais simples de todas. Nela são feitas atividades relacionadas a análise da plataforma e das soluções necessárias para a sua idealização.

Todas essas fases compõem o CrudForge.

APÊNDICE II: CICLO DE DESENVOLVIMENTO

Início das atividades por um desenvolvedor

1. clonar o repositório;
2. configurar o seu usuário global do git;
3. instalar backend (existe script para instalação no Ubuntu);

Ciclo por Tarefa

Execução:

1. escolher uma tarefa (Issue) atribuída para o seu usuário no github;
2. marcar esta tarefa com o label 'working';
3. fazer PULL do repositório se envolver modificação do source;
4. desenvolvimento
5. sempre comentar o desenvolvimento da tarefa a medida que ocorre;
6. incluir nos comentários da tarefa as fontes de pesquisa como por exemplo os links para os artigos e ou livros usados para resolver a tarefa;
7. se for pertinente incluir anotações na documentação do TCC;
8. implementar funcionalidade no source se aplicável;
9. fazer o commit das alterações para o working-copy quando achar necessário, usando o seguinte padrão de mensagem: '* #X Descrição da implementação' (sendo X o numero do ticket relacionado);

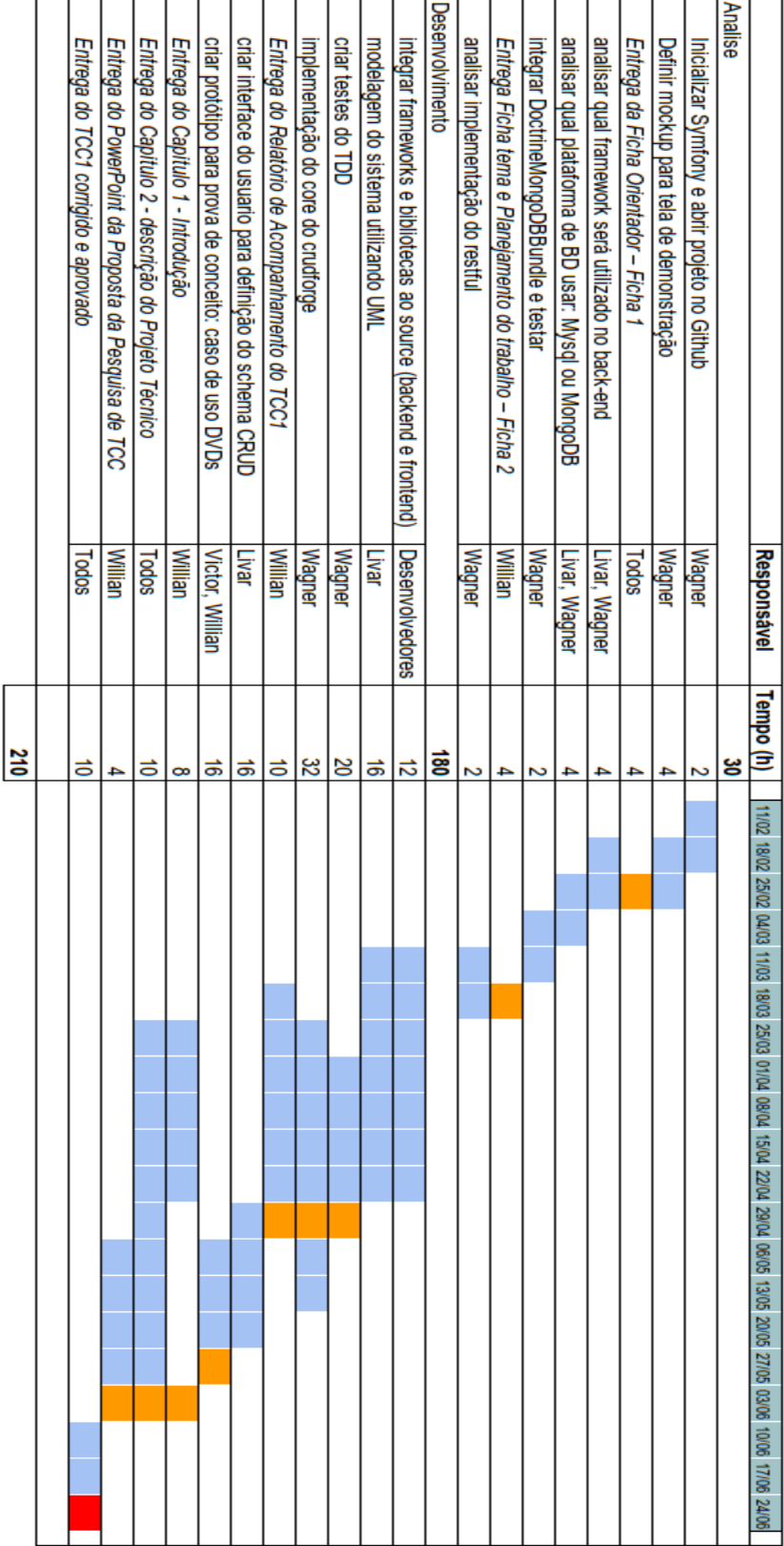
Finalização da execução:

1. fazer PUSH do repositório;
2. Remover tag 'working' da tarefa;
3. Fechar a tarefa;

APÊNDICE III - CRONOGRAMA

Baseado nas entregas exigidas para a realização do projeto, o cronograma utilizado foi dividido em duas partes, Cronograma TCC1 (*figura 51*) e Cronograma TCC2 (*figura 52*), ambos foram planejados respeitando as datas limites impostas como padrão para cada entrega de acordo com o calendário da faculdade.

CRONOGRAMA TCC 1: Projeto CRUDForge



Atribuições:
Livar: Desenvolvedor Backend e Frontend
Victor: Desenvolvedor Backend e Testes
Willian: Documentação e Testes
Wagner: Arquitetura e Desenvolvimento



Figura 51: Cronograma TCC1

O Cronograma TCC1 atende as necessidades de desenvolvimento e entregas previstas para os primeiros cinco meses de projeto, ou seja, de fevereiro de 2013 até junho de 2013.

Foi necessário o cumprimento de todas as fases nos seus devidos tempos para a continuidade do trabalho no Cronograma TCC2 (*Figura 52*), pois o mesmo é totalmente dependente das atividades anteriores.

Ambos os cronogramas do trabalho foram divididos em duas frentes: Análise e Desenvolvimento. Além disso, essas duas frentes são relacionadas aos responsáveis por suas tarefas, pelo tempo de desenvolvimento e pelas datas das entregas.

As tarefas são as mesmas estruturadas na Estrutura Analítica de Projeto (Vide Apêndice I), nesse caso estão destinadas aos envolvidos no tempo de trabalho e de entrega pré-definidos.

Sendo assim, a descrição das tarefas foram especificadas, ou seja, basta apenas fazer a associação das atividades descritas com os envolvidos no projeto, como no cronograma descrito.

Com ele é possível visualizar três informações relevantes para o projeto: datas, fases e responsáveis.

Suas duas frentes estão definidas como Desenvolvimento e Implantação. Essas frentes, de âmbito técnicas, foram definidas como pontos focais do projeto, devido ao fato do mesmo ter sido iniciado de maneira com que questões técnicas se tornassem o carro chefe do projeto, ou seja, o primeiro passo do projeto foi a realização de atividades técnicas.

Essas frentes foram definidas de acordo com o que Teles (2005) sugere, que todo projeto em XP seja dividido em releases e iterações, de modo que:

Releases são módulos do sistema que geram um valor bem definido para o cliente.

Iterações, por sua vez, são períodos de tempo de poucas semanas (em torno de duas, em média) no qual a equipe implementa um conjunto de funcionalidades acordado com o cliente.

Desta forma assegura que a equipe esteja sempre trabalhando naquilo que é o mais importante.

APÊNDICE IV: ESPECIFICAÇÃO TÉCNICA

Para que os objetivos sejam alcançados, foi utilizado a ferramenta *webGithub*, com o propósito de fazer o gerenciamento das atividades do projeto, no qual a mesma provê as funcionalidades de gerenciamento de *tickets*, *wikis* e controle de versão utilizando *git*.

Para que o gerenciamento de *tickets* funcionasse, foram utilizados os conceitos disponíveis na metodologia XP, como o *refactoring* e o código coletivo.

Já que não existe uma pessoa responsável por uma parte do código, cada desenvolvedor tem acesso a qualquer parte do sistema e possui liberdade para alterá-la a qualquer momento.

Esta prática tem como consequência um código revisado por diversas pessoas e caso algo não esteja claro, o mesmo será alterado para que torne-se legível (*refactoring*).

Após as definições anteriores, foram definidos alguns *mockups* para as telas de demonstração do projeto.

Após a análise de certos aspectos como a escolha do *framework*, o projeto foi inicializado utilizando o *framework* *Symfony2*, com a persistência dos dados no banco de dados *MySQL*.

A fase de desenvolvimento foi feita utilizando a linguagem de programação *PHP* (um acrônimo recursivo para "PHP: Hypertext Preprocessor"), uma linguagem de *script* de código fonte livre e de uso geral, muito utilizada e especialmente criada para o desenvolvimento de aplicações Web (Php.net, 2005), o que é o caso do *CRUDForge*. A modelagem dos diagramas de classe, e diagramas de caso de uso foram desenvolvidas utilizando a *UML* (Unified Modeling Language) com o apoio de diversas ferramentas (*LucidChart* e *Astah*).

Posteriormente houve a definição do licenciamento do sistema *CRUDForge* para em seguida ser feito o desenvolvimento de uma versão beta de interface do usuário, com o intuito de definir o *schema* *CRUD* do projeto, junto com a criação do *script* para a instalação do *back-end*.

O site do projeto foi desenvolvido utilizando a linguagem de programação PHP, muito comum no desenvolvimento de sistemas *web*, junto aos testes do *crudgenerator*, com o MySQL e a criação dos CSS para o protótipo do *CRUD*.

APÊNDICE V: CUSTOS

Afim de colocar em uso a plataforma projetada, será utilizado o serviço AWS da Amazon, através da configuração de uma instância t1.micro, com um custo mensal de \$ 5,14 ao mês (tabela 7). Dessa forma, será possível comercializar a plataforma seguindo o modelo SaaS, inicialmente tendo como limite 15 usuários de testes, e depois escalando a infraestrutura de acordo com a demanda.

Tabela 7: Custos de Instâncias

Amazon AWS - comparação de instâncias

Estratégia	instance	vcors	ECU	ram	storage (GB)	custo	ip estatico	custo	db data set (GB)	custo	intra-transfer	custo
micro - testes	t1.micro	1	>2	0,6	8	\$ 21,29	0	\$ 0,00	2	\$ 0,00	0	\$ 0,00
micro - completo	t1.micro	1	>2	0,6	8	\$ 21,29	1	\$ 3,66	2	\$ 0,00	0	\$ 0,00
micro - completo com separação db	t1.micro	1	>2	0,6	8	\$ 21,29	1	\$ 3,66	5	\$ 1,37	5	\$ 0,10
small - completo	m1.small	1	1	1,7	160	\$ 58,56	1	\$ 3,66	>10	\$ 0,00	0	\$ 0,00
medium - completo	c1.medium	2	5	1,7	350	\$ 146,40	1	\$ 3,66	>10	\$ 0,00	0	\$ 0,00

* atualizado em 03/07/2013

** Custo mensal link Embratel 1M R\$ 962,00

Estratégia	data transfer in	custo	data transfer out	custo	usuários	vazão	custo on-demand	desconto (free tier)	custo final (mês)	Reserva 1 ano	
										primeiro mês	outros 11 meses
micro - testes	5	\$ 0,00	5	\$ 1,00	10-15	4,5/min	\$ 22,29	-\$ 21,28	\$ 1,01	\$ 31,00	\$ 11,72
micro - completo	5	\$ 0,00	5	\$ 1,00	10-15	4,5/min	\$ 25,95	-\$ 21,28	\$ 4,67	\$ 31,00	\$ 15,39
micro - completo com separação	5	\$ 0,00	5	\$ 1,00	?	?	\$ 27,42	-\$ 22,28	\$ 5,14	\$ 31,00	\$ 16,75
small - completo	5	\$ 0,00	5	\$ 1,00	?	?	\$ 63,22	\$ 0,00	\$ 63,22	\$ 124,00	\$ 50,51
medium - completo	5	\$ 0,00	5	\$ 1,00	>15-x	?	\$ 151,06	\$ 0,00	\$ 151,06	\$ 267,00	\$ 101,75

A Amazon fornece como incentivo ao uso de computação em nuvem, com um nível de uso gratuito, interessante para quem está iniciando nessa área. Esse nível de uso gratuito é conhecido como *Free Usage Tier*, e fornece serviços de infraestrutura por um tempo pré-estabelecido.

Como foi utilizado o suporte de computação em nuvem EC2 (Elastic Compute Cloud), as especificações gratuitas desse serviço no Free Usage Tiger englobam

750 horas por mês de microinstâncias para os sistemas operacionais Linux e Microsoft, durante o período de 12 meses, (Amazon, 2013).

APÊNDICE VI: PLANEJAMENTO FINANCEIRO

O principal foco do CRUDForge é atender a pequenas e médias empresas, e com isso, foi especificado na planilha Fluxo de Caixa, os possíveis investimentos e retorno com a implementação do CRUDForge.

Fluxo de Caixa

mensalid. por usuário simples: R\$ 15,00
mensalid. por usuário ERP: R\$ 199,00

razão p.a.: 10
razão p.a.: 5

	mês 1	mês 2	mês 3	mês 4	mês 5	mês 6	mês 7	mês 8	mês 9	mês 10	mês 11	mês 12	Total	média	%
num. users pagantes	0	0	0	10	20	30	40	50	60	70	80	90	90		
num. users ERP	0	0	0	0	0	5	15	25	35	45	55	65	65		
num. funcionarios	0	0	0	0	0	0	1	1	1	2	2	2	2		
Recebimento	0	0	0	150	300	1.445	3.585	5.725	7.865	10.005	12.145	14.285	55.505	4.625	100%
Custos Variáveis	11	11	11	24	36	130	494	669	845	1.220	1.396	1.571	6.420	535	11,6%
Custo AWS	11	11	11	11	11	11	200	200	200	400	400	400	1.868	156	3,4%
Impostos	0	0	0	12	25	118	294	469	645	820	996	1.171	4.551	379	8,2%
Desvios (Est.+Fin.)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,0%
Margem Contrib.	-11	-11	-11	126	264	1.315	3.091	5.056	7.020	8.785	10.749	12.714	49.085	4.090	88,4%
Custos Fixos	0	0	0	0	0	0	3.000	4.000	5.000	8.000	9.000	10.000	39.000	3.250	70,3%
Pessoal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,0%
Ocupação	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,0%
Desp Administr.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,0%
Pró-Labore	0	0	0	0	1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	36.000	3.000	64,9%
Terceiros	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,0%
Outros	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,0%
Resultado	(11)	(11)	(11)	126	264	1.315	91	1.056	2.020	785	1.749	2.714	10.085	840	18,2%
fundo inicial	160	149	137	126	252	516	1.831	1.922	2.978	4.998	5.782	7.531	10.245		
fundo final	149	137	126	252	516	1.831	1.922	2.978	4.998	5.782	7.531	10.245			

CMV	Fornecedores em geral, fretes, etc.
Desvios Est. e Fin	Desvios de estoque (diferenças em inventários) + Desvios Financeiros (cheques sem fundos e outros problemas de pagamento)
Pessoal	Salários Fixos, Férias, 13o., vale transporte, uniforme, etc.
Ocupação	Aluguel, segurança, manutenção predial, seguro, limpeza, material de limpeza, etc.
Despesas Adm	Água, luz, telefones, internet, etc.
Terceiros	Software, contabilidade, consultorias, etc.
Pró-Labore	Salário dos sócios - se estes trabalharam na loja
Outros	Itens que não puderam ser enquadrados acima (o ideal é que não existam!)

Tabela 8: Fluxo de Caixa

O CRUDForge pode ser utilizado de maneira gratuita para clientes que desejam utilizar até 1000 registros. Ultrapassando esse limite será cobrado uma taxa mensal de R\$199,00 por usuário ERP, ou seja, corporações (pessoa jurídica), ou R\$ 15,00 por usuário simples (pessoa física).

A estimativa de progressão aritmética é que a cada mês ingressem 10 novos usuários de ERP e 5 novos usuários simples. Caso essa estimativa se confirme, cobrando as devidas taxas especificadas na planilha, em um ano é possível com que dois, dos quatro envolvidos no projeto, sejam considerados sócios com pró-labores, ou seja, tenham um ganho de R\$5.000 mensais.

ANEXO I: INFRAESTRUTURA ESCALÁVEL AWS

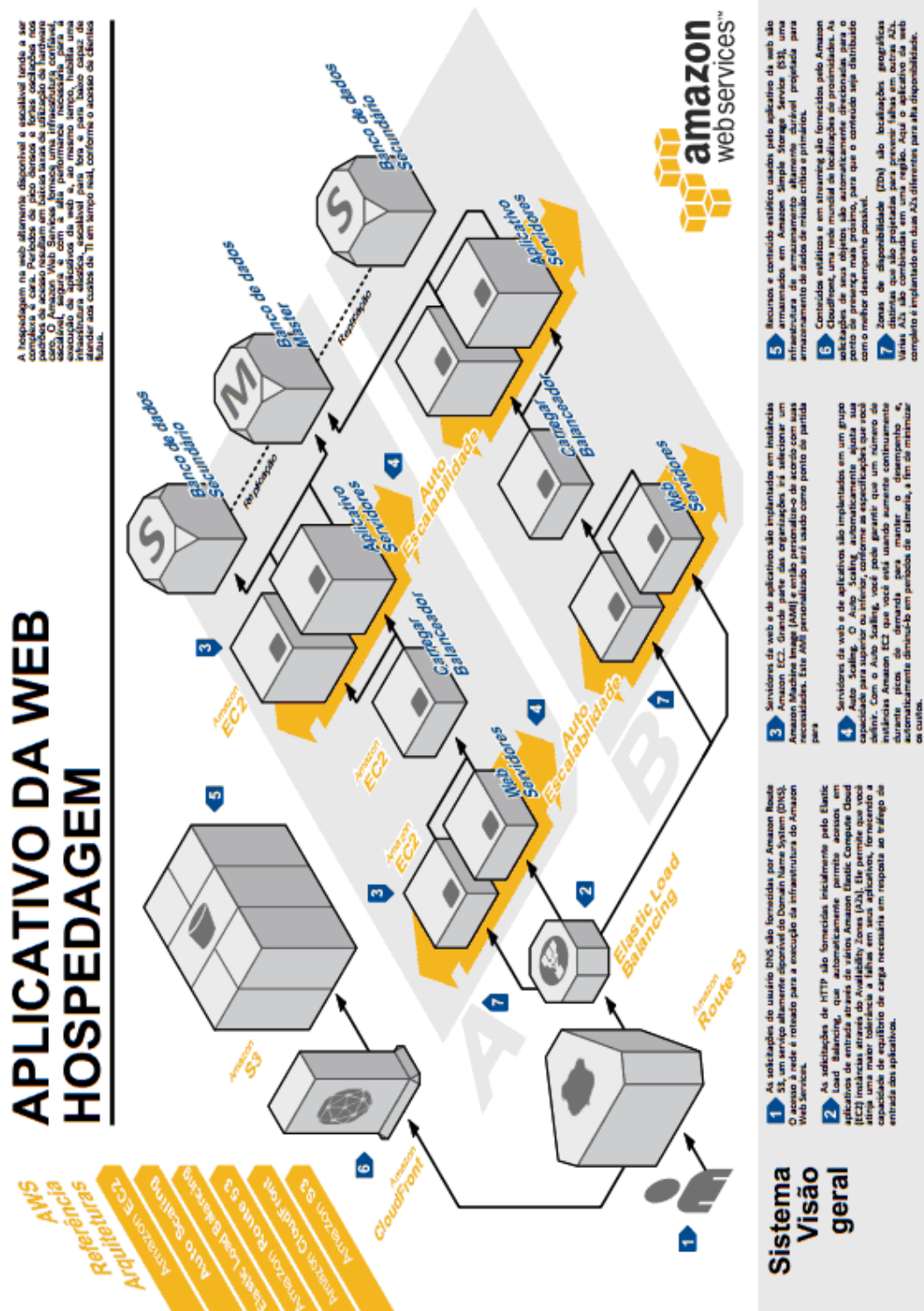


Figura 53: Infraestrutura escalável do AWS