UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

GESTOR PSICOLOGIA: GERENCIAMENTO DE CONSULTÓRIOS DE PSICOLOGIA

FELIPE CORRÊA

BLUMENAU
2013

2013/2-12

FELIPE CORRÊA

GESTOR PSICOLOGIA: GERENCIAMENTO DE CONSULTÓRIOS DE PSICOLOGIA

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação— Bacharelado.

Prof. Jacques Robert Heckmann, Mestre - Orientador

BLUMENAU
2013

2013/2-12

GESTOR PSICOLOGIA: GERENCIAMENTO DE CONSULTÓRIOS DE PSICOLOGIA

Por

FELIPE CORRÊA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Jacques Robert Heckmann, Mestre – Orientador, FURB
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 06 de dezembro de 2013.

AGRADECIMENTOS

À minha família, que sempre me incentivou e criou as condições necessárias para a realização deste.

À minha namorada Carolina, que teve paciência, empenho e dedicação extraordinária para revisar este trabalho mesmo com prazos apertados.

Aos meus colegas, pelas sugestões e contribuições.

Ao meu orientador, Jacques Robert Heckmann, por ter acreditado na conclusão deste trabalho.

Aos professores do Departamento de Sistemas e Computação da Universidade Regional de Blumenau por suas contribuições durante os semestres letivos.

O único lugar onde o sucesso vem antes do trabalho é no dicionário.

Albert Einstein

RESUMO

Este trabalho foi realizado com o intuito de desenvolver um sistema para psicólogos independentes, que permita o gerenciamento de consultórios de psicologia, em uma plataforma *on-line* denominada Gestor Psicologia. O software foi desenvolvido para suprir carências apresentadas na utilização do sistema anterior, Sistema Administrativo Inteligente (SAI), usado para administrar pacientes de uma psicóloga. Para a apresentação de um serviço que atendesse as necessidades dos profissionais da área, fez-se necessário entender como funciona um consultório de psicologia; descobrir o funcionamento e o uso de um facilitador de pagamento; distribuir o software como serviço no modelo software *as a service* (SaaS) através da internet e utilizar *layout* responsivo para permitir acesso de dispositivos móveis. Para o desenvolvimento desta aplicação *web*, foram utilizados padrões de projeto e *frameworks* que permitiram uma melhor desenvoltura de trabalho e do próprio software, destacando-se ainda o fato de o código ser cuidadosamente trabalhado para evitar vulnerabilidades, demonstrando-se um sistema seguro. O Gestor Psicologia permite uma melhor forma de trabalho e organização dos profissionais do ramo da psicologia, que terão agora um prontuário eletrônico de fácil acesso, um sistema de agendamento de consultas e avisos, além de contar com informações estatísticas sobre seus pacientes e atendimentos.

Palavras-chave: Psicologia. Prontuário Eletrônico. Software *as a service*.

ABSTRACT

This paper was carried out in order to develop a system for independent psychologists, enabling the management of psychology offices, in an online platform called Gestor Psicologia. The software was developed to meet the needs presented using the previous system, Sistema Administrativo Inteligente (SAI) - used to manage patients of a psychologist. To achieve a service that could meet the needs of professionals in the field, it was necessary to understand how a psychology office works; discover the operation and use of a payment facilitator; distribute the software as a service in a software as a service (SaaS) model over the internet, and using responsive layout to allow access from mobile devices. For the development of this web application, it were used design patterns and frameworks that allowed a better ease of work and the software itself, highlighting also the fact that the code is carefully developed to avoid vulnerabilities, showing up a secure system. The Gestor Psicologia enables better way of working and organization of professionals in the field of psychology, who will now have an electronic medical record for easy access, a system of scheduling appointments and reminders, and statistical information about their patients and consultations.

Key-words: Psychology. Electronic medical records. Software as a service.

LISTA DE ILUSTRAÇÃO

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API - *Application Programming Interface*

DOM - *Document Object Model*

DRY - *Dont't Repeat Yourself*

FK - *Foreign Keys*

HTML - *HyperText Markup Language*

HTTP - *HyperText Transfer Protocol*

MVC - *Model-view-controller*

ORM - *Object-Relacional Mapping*

PHP - *Hypertext Preprocessor*

RF - Requisito funcional

PK - *Primary Keys*

RNF - Requisito não funcional

SaaS - Software *As A Service*

SAI - Sistema Administrativo Inteligente

SMS - *Short Message Service*

SQL - *Structured Query Language*

UC - *Use Case*

UML - *Unified Modeling Language*

URL - *Uniform Resource Locator*

XSS *Script - Cross-site scripting*

SUMÁRIO

1 INTRODUÇÃO

A psicologia, segundo Pontes (2005), está presente no cotidiano e no modo de viver de todos os seres humanos. A autora ressalta ainda que a capacidade de melhorar o comportamento humano será possível apenas por intermédio da compreensão da natureza humana. As pessoas ainda não conhecem bem os verdadeiros objetivos da psicologia, já que esta é uma ciência jovem. Sendo assim, o indivíduo ainda não percebeu o quanto pode ser beneficiado, profissional e socialmente, por um conhecimento maior nesta área.

No ano de 1985 o número de psicólogos graduados atingiu 102.865, sendo que este fato se deve ao aumento das instituições de ensino profissionais, obtendo crescimento exponencial a partir dos anos 70 (BASTOS; GOMIDE, 1989). Isso é comprovado analisando-se o desenvolvimento da área da psicologia no Brasil: em 1973 havia 3.500 mestres e 500 doutores e em 2003 esse número subiu para 27.630 e 8.045, respectivamente. Da mesma maneira a quantidade de cursos recomendados (637, em 1976; 2.993, em 2004) e de alunos matriculados (de 37.195 em 1987 para 112.314 em 2003) seguiu esta mesma tendência (SEIXAS; COELHO-LIMA; COSTA, 2010).

Com base nesse contexto, em 2010 verificou-se a necessidade de desenvolver um software para facilitar a administração dos pacientes de uma psicóloga[1], sendo este chamado de Sistema Administrativo Inteligente (SAI). No entanto, com o passar do tempo constatou-se que o mesmo não supria as necessidades para a administração completa dos pacientes.

Entre os principais problemas encontrados no software (SAI) e no cotidiano do psicólogo, estão: a dificuldade em controlar a agenda de pacientes e de mudar horários previamente agendados, pelo fato de os registros serem feitos em papel; a impossibilidade de saber o nível de absenteísmo dos pacientes nas sessões, pela falta de registro destas informações; a impossibilidade de gerar estatísticas dos pacientes, devido à falta de normalização no cadastro dos dados; e a indisponibilidade para vários psicólogos utilizarem o sistema, cada qual controlando seus próprios pacientes. Levando-se em conta os fatores citados, que limitam a desenvoltura do trabalho do psicólogo, justifica-se a decisão por aprimorar e agregar valor ao produto já existente, criando assim um novo sistema, o qual visa resolver a problemática não suprida pelo SAI.

O novo sistema se chamará “Gestor Psicologia” e proporcionará melhorias administrativas ao consultório, desde o agendamento de consultas à contratação de planos *on-*

[1] O nome da psicóloga não será exibido no presente documento devido a restrições legais no que tange a promoção ou propaganda do nome do psicólogo, o que é expressamente proibido na área da psicologia.

line, permitindo ganho de tempo com a agenda, que será gerada automaticamente com base nos horários pré-definidos dos pacientes e trará melhorias significativas no gerenciamento destes, através do cadastro centralizado e normalizado das informações. Além disso, possibilitará que vários psicólogos utilizem um só sistema *on-line*, cada qual controlando seus próprios pacientes e sua própria empresa. Sendo assim, permitirá a centralização de informações e gerenciamento das mesmas, com fácil acesso para o usuário final, proporcionando melhores condições para o desenvolvimento de seu trabalho.

Cabe ressaltar que este sistema será desenvolvido utilizando uma programação de nível elevado, ou seja, programando com padrões de projeto e *frameworks*, permitindo que o software possa ser facilmente modificado no futuro e que tenha um melhor desempenho para o cliente. Também será apresentada uma aplicação segura e confiável, já que vulnerabilidades que possam contribuir para futuros ataques serão evitadas e protegidas.

1.1 OBJETIVOS

O objetivo geral deste trabalho é o desenvolvimento de um sistema para psicólogos independentes, que permita o gerenciamento de consultórios de psicologia, em uma plataforma *on-line* denominada “Gestor Psicologia”. Para atingir este objetivo, os seguintes objetivos específicos são propostos:

a) entender como funciona um consultório de psicologia;
b) descobrir o funcionamento e o uso de facilitador de pagamento;
c) distribuir o software como serviço no modelo Software *As A Service* (SaaS) através da internet;
d) utilizar *layout* responsivo para permitir acesso de dispositivos móveis.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está organizado em quatro capítulos, iniciando deste primeiro, onde são apresentados a introdução, os objetivos e a descrição da estrutura do trabalho.

No segundo capítulo apresenta-se a fundamentação teórica, que busca explanar conceitos sobre a psicologia, o prontuário eletrônico, o SaaS, *design patterns* e *frameworks.* Também expõe informações sobre o sistema atual, além de apresentar trabalhos correlatos a este.

No terceiro capítulo relata-se o desenvolvimento do sistema, incluindo o levantamento de informações, as especificações dos requisitos, a implementação e os resultados obtidos.

Por fim, no quarto capítulo abordam-se as conclusões e extensões que podem vir a ser realizadas neste projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda assuntos a serem apresentados nas seções a seguir, que fundamentam o desenvolvimento do Gestor Psicologia, tais como o conceito de psicologia, SaaS, o sistema atual e os trabalhos correlatos.

2.1 PSICOLOGIA

O conceito de psicologia diverge dependendo do autor consultado. De acordo com Teles (1995), a psicologia estuda o comportamento humano e animal, e através do estudo científico do comportamento quer alcançar três objetivos: a descrição, a predição e o controle do comportamento.

> Procura compreender o Homem, seu comportamento, para facilitar a convivência consigo próprio e com o outro. Pretende fornecer-lhe subsídios para que ele saiba lidar consigo mesmo e com as experiências da vida. É, pois, a ciência do comportamento, compreendida esta em seu sentido mais amplo. (TELES, 1995, p. 9).

2.1.1 Avaliação psicológica

Desde os primórdios da psicologia vocacional, a avaliação psicológica constitui um instrumento para ajudar os indivíduos a tomarem suas escolhas (DUARTE, 2008). A avaliação psicológica é um processo complexo e multifacetado. O processo de avaliação começa muito antes do paciente ser apresentado à primeira situação de avaliação. Esse processo se inicia mesmo antes do primeiro contato, na entrevista inicial, com a pessoa a ser avaliada (TAVARES, 2012).

Ainda segundo Tavares (2012), o resultado está intimamente relacionado à qualidade da relação entre o avaliador e o sujeito avaliado, que será mediada pela maneira como o profissional utiliza a técnica em conjunto às habilidades interpessoais. Ao longo do processo avaliativo, é importante perceber os detalhes, pois todo e qualquer elemento pode vir a fazer parte das fantasias inconscientes do paciente, sendo que os elementos podem adquirir relevância, dependendo da natureza da angústia.

2.1.2 Acompanhamento psicológico

O tratamento psicológico na vida de um indivíduo não deve ser somente fruto de algum problema que veio a se desencadear a partir de algum conflito físico ou emocional. Ele contribui para uma melhor saúde, não somente como fator de cura, mas também de prevenção.

> A saúde não se resume à ausência de doença e ao bem-estar físico, mas um estado multidimensional que envolve três domínios: a saúde física, psicológica e social. A saúde física implica ter um corpo não apenas livre de doenças, mas também envolve hábitos relacionados ao comportamento e ao estilo de vida. Em seu outro domínio, a saúde social engloba as boas habilidades interpessoais, relacionamentos com amigos e família e atividades socioculturais. A saúde psicológica engloba, apesar das variações culturais, a capacidade de pensar de forma clara e objetiva, possuir uma auto-estima adequada e consciência de bem-estar. Nela pode-se incluir a criatividade, as habilidades intelectuais e a estabilidade emocional, caracteriza-se pela abertura às inovações e, ao mesmo tempo, pela presença de uma estrutura e funcionamento estável da personalidade. A psicologia na saúde tem como campo de pesquisa e de intervenção a interface dos três domínios, objetivando o estado completo de bem-estar físico, mental e social. (CAPITÃO; SCORTEGAGNA; BAPTISTA, 2005, p. 81).

Sendo assim, um tratamento psicológico é importante para se obter e manter um bem-estar em todos os campos da saúde. Contudo, é de suma importância o registro e acompanhamento de informações sobre o paciente, para comparações e constatações no decorrer do tratamento ou em futuras consultas.

2.1.3 Prontuário

Apesar da grande demanda de atendimentos psicológicos e a necessidade de conservação de algumas informações, ainda não existe um modelo padrão caracterizando informações relevantes ou não, para cada paciente ou caso, em relação ao tratamento psicológico. Um modelo de prontuário eletrônico para este fim poderia contribuir à padronização de informações obtidas, construindo a partir de opiniões e estudo uma melhor e mais satisfatória abordagem do paciente.

> Hoje o prontuário está presente no dia a dia de todos os profissionais que atuam em um hospital, unidade ambulatorial, posto de saúde, etc., contudo, o registro do psicólogo no prontuário ainda não é uma rotina em toda instituição de saúde; observa-se uma carência de divulgação de modelos de anotação em Psicologia e, especialmente, a ausência de um modelo padronizado de anotações, o que poderia

> facilitar a implementação de uma forma eficaz de registros, organizados temporal e funcionalmente, cujas informações dispersas estivessem inter-relacionadas para posterior recuperação, análise e interpretação, por meio de um denominador comum: o problema de saúde que motivou o tratamento. (ALMEIDA; CANTAL; COSTA JUNIOR, 2008, p. 433)

Diferentemente dos profissionais do ramo da psicologia, o Conselho Federal de Medicina através da Resolução nº 1.638/2002, "Define prontuário médico e torna obrigatória a criação da Comissão de Revisão de Prontuários nas instituições de saúde." (CONSELHO FEDERAL DE MEDICINA, 2012, p. 184). O Conselho Federal de Medicina também, por meio da Resolução nº 1.821/07 "Aprova as normas técnicas concernentes à digitalização e uso dos sistemas informatizados para a guarda e manuseio dos documentos dos prontuários dos pacientes, autorizando a eliminação do papel e a troca de informação identificada em saúde." (CONSELHO FEDERAL DE MEDICINA, 2007, p. 252), ou seja, legitima e normatiza o prontuário digitalizado, que não é considerado um prontuário eletrônico. Mas, desta maneira, já impulsionando a valorização da utilização das novas tecnologias para o desenvolvimento e rentabilidade do atendimento ao paciente.

Em 2012, o Conselho Federal de Medicina lançou uma cartilha sobre o Prontuário Eletrônico - que basicamente corresponde ao registro de dados antes feitos no papel, em meio eletrônico. Entre as informações disponíveis, ressalte-se que: "O prontuário em papel apresenta diversas limitações, sendo ineficiente para o armazenamento e organização de grande volume de dados, apresentando diversas desvantagens em relação ao prontuário eletrônico" (CONSELHO FEDERAL DE MEDICINA; SOCIEDADE BRASILEIRA DE INFORMÁTICA EM SAÚDE, 2012, p. 5).

Sendo assim, através de um prontuário eletrônico a informação está da melhor forma disponível ao profissional. Além disso, a informação em papel possui limitações:

> [...] possui baixa mobilidade e está sujeito a ilegibilidade, ambiguidade, perda frequente da informação, multiplicidade de pastas, [...], falta de padronização, dificuldade de acesso, fragilidade do papel e a sua guarda requer amplos espaços nos serviços de arquivamento. (CONSELHO FEDERAL DE MEDICINA; SOCIEDADE BRASILEIRA DE INFORMÁTICA EM SAÚDE, 2012, p. 5).

2.2 SOFTWARE AS A SERVICE (SAAS)

Software *As A Service* (SaaS), conhecido também por software como serviço, é basicamente uma forma de comercialização de software, onde o fornecedor do software se

responsabiliza por toda a estrutura necessária para a disponibilização do sistema (servidores, conectividade, cuidados com segurança da informação) e o cliente utiliza o software via internet, pagando um valor recorrente pelo uso.

Segundo Cambiucci (2009), o modelo de entrega de SaaS traz uma solução de software mais flexível e reutilizável, suportando diversos usuários sobre uma mesma infraestrutura configurável, oferecendo funcionalidades sob demanda.

2.3 FRAMEWORKS

Um *framework* é basicamente uma estrutura implementada "base", a qual pode ser desenvolvida para algo mais específico e maior. Segundo Minetto (2007), novos softwares podem ser mais facilmente desenvolvidos, já que vão dispor de uma coleção de códigos, técnicas, funções, métodos, classes, entre outros. Apesar de haver certa dificuldade ao iniciar a utilização de *frameworks*, já que possuem formas diferentes de programar, incluindo sintaxes diferentes, convenções de nomes de variáveis e arquivos, assim como de tabelas de banco de dados, as vantagens são muitas a médio e longo prazo.

Minetto (2007) afirma ainda, que a manutenção para programas que utilizam *framework* se torna mais fácil quando há a utilização dos mesmos, já que seguem um padrão, facilitando o entendimento da função proposta. Tarefas repetitivas também podem ser automatizadas, conceito conhecido como *Don't Repeat Yourself* (DRY), que significa não se repita. Para as diferentes tabelas que tenham que ter seus dados manipulados, as operações de inclusão, exclusão e alteração são praticamente iguais, assim, o código-fonte não precisa ser desenvolvido várias vezes e a criação dessas funções pode ser automatizada por ferramentas contidas no *framework*. "Além das já mencionadas, existem outras vantagens como separação de apresentação e lógica, facilidade de geração de testes automatizados e geração de documentação." (MINETTO, 2007, p. 18).

2.3.1 SILEX

Desenvolvido pela SensioLabs, o SILEX é um micro-*framework* para desenvolvimento de aplicações *web* utilizando a linguagem *Hypertext Preprocessor* (PHP). Segundo Silex (2013), basicamente, define os controladores e faz o mapeamento com as rotas em uma única etapa. Com base nos componentes do Symfony2, que é um *framework* em PHP, e do Pimple, que implementa um serviço de *Dependency Injection* permitindo a conexão

de todas as bibliotecas, o Silex oferece estrutura mais simples para a construção de aplicações em um único arquivo.

O SILEX se apresenta conciso, já que expõe uma *Application Programming Interface (*API) intuitiva e concisa; extensível, pois possui sistema de extensão baseado no micro *service-container* do Pimple, tornando mais fácil a utilização de bibliotecas de terceiros; e testável, onde, utilizando o HttpKernel, do Symfony2, que abstrai o *request* (solicitação/requisição) e o *response* (resposta), facilita o teste da aplicação e do *framework* (SILEX, 2013).

2.3.2 Propel ORM

O *Object-Relacional Mapping* (ORM), termo usado para mapeamento objeto-relacional, é uma técnica de programação onde utiliza-se um descritor de meta-dados para conectar o código orientado a objetos a um banco de dados (JANSSEN, 2013a). Ou seja, convertendo dados entre sistemas incapazes de coexistir dentro de banco de dados relacionais e linguagem orientada a objeto, o ORM cria para cada entidade do banco de dados (tabela), classes específicas para manipulação.

Segundo Janssen (2013a), alguns dos benefícios proporcionados pela utilização do ORM são: menos código escrito comparado com o código *Structured Query Language* (SQL) escrito manualmente; *cache* transparente dos objetos na camada de aplicação, melhorando a performance do sistema; uma solução simples para facilitar a manutenibilidade e aumentar a velocidade da aplicação.

O Propel é um ORM de código aberto para PHP, utilizando todos os principais conceitos de um ORM robusto: o padrão de projeto *Active Record*, validadores, comportamentos, herança de tabelas, engenharia reversa de um banco de dados existente e inicialização tardia (PROPEL, 2013). Desenvolvido para programadores que precisam manter o controle do seu código, permite que o banco de dados seja acessado através de um conjunto de objetos, também fornecendo uma API simples para armazenamento e recuperação dos dados. Propel (2013) afirma ainda que é um *framework* utilizado para abstrair a camada de persistência. Desta forma, ele possui uma estrutura orientada a objetos para manipular o banco de dados. Um dos grandes benefícios de utilizar o Propel ORM para abstração da camada de persistência é a possibilidade de migrar facilmente de um banco de dados para outro sem a necessidade de modificar toda a aplicação.

2.4 FACILITADORES DE PAGAMENTO

Os chamados facilitadores ou intermediadores de pagamento, segundo Maciel (2012), oferecem a micro e pequenas empresas o serviço de intermediar pagamentos. Eles disponibilizam aos empresários destes negócios meios para que os clientes dos *websites* tenham a possibilidade de diversas formas de pagamento, enquanto os contratantes do serviço terão garantias relacionadas ao capital, a não ocorrência de fraudes e a conciliação financeira simplificada.

Os custos envolvendo o serviço variam de 5,4% a 7% em relação às vendas por cartão de crédito e de 1,9% a 2,9% se tratando de vendas de débito e boleto, mas em compensação seu sistema é de fácil utilização. O proprietário do *website* cria uma conta no *site* do intermediário escolhido, integra-o ao seu *site* e pode começar a vender seu serviço imediatamente (MACIEL, 2012).

2.4.1 Moip

O Moip é um facilitador de pagamento que permite aos clientes que contratarem o serviço proporcionar as seguintes opções de pagamento: cartões de crédito, débito, boleto bancário, débito em conta e pelo celular, no seu *website*. Segundo Moip (2013), é uma empresa de gestão e intermediação de pagamentos, e entre seus serviços oferecidos destaca-se o *checkout* transparente, escolha muitas vezes feita para o melhor funcionamento e comodidade dos clientes.

No *checkout* transparente, o cliente do serviço permanecerá no *site* em que faz o negócio durante todo o processo de pagamento, sem ser encaminhado a outras páginas (MOIP, 2013). Os dados dos clientes são encaminhados diretamente do navegador para o Moip, ou seja, sem passar pelos servidores do *site*, a segurança das informações não recaem sobre o mesmo. Permitindo a otimização de acordo com o ramo e as necessidades do cliente, utilizar o Moip – *checkout* transparente – proporciona uma melhor administração de pagamentos.

2.5 VULNERABILIDADE DE APLICAÇÕES WEB

Lipner e Howard (2005) destacam em seu trabalho, a necessidade de todo fornecedor de software analisar as ameaças à segurança do sistema e trabalhar para a contenção das

mesmas. Um software adequado deve proteger dados dos seus clientes, gerando confiança no serviço prestado.

Quando não há a devida proteção, cria-se um ambiente vulnerável e inseguro. Segundo Ceron et. al (2008), nos últimos anos, grande parte dos problemas relativos à segurança de software envolvem as aplicações disponíveis na internet. As invasões e danos decorrem, principalmente, da vulnerabilidade do software, que muitas vezes não tem as devidas precauções adotas para evitar atos mal intencionados, ou nem mesmo um código bem elaborado. Criando-se um código de forma que defenda destes problemas, muitas consequências podem ser evitadas.

2.5.1 Cross-Site Scripting

Hold (2006 apud Ceron et. al, 2008) realizou um estudo no qual cita o *Cross-Site Scripting* (XSS *Script*), assim como o SQL *injection*, entre os ataques que mais ocorrem devido a aplicações *web* vulneráveis. A Rede Segura (2012a) aponta que um ataque do gênero XSS *Script* se baseia na falha da validação de parâmetros de entrada de usuário, bem como de resposta do servidor.

Modificando estruturas do documento em *HyperText Markup Language* (HTML) visíveis do lado do cliente, causa danos aos usuários de um aplicativo vulnerável e ao seu desenvolvedor (REDE SEGURA, 2012a). Entre os problemas gerados podem haver roubos de sessão de usuários, redirecionamento de usuários a *site*s maliciosos e alteração do objeto *Document Object Model* (DOM) para captura de dados ou envio de *malware*.

2.5.2 SQL Injection

SQL *Injection* (Injeção de SQL) é uma técnica que visa explorar vulnerabilidades na utilização de códigos SQL e conseguir acesso não autorizado ao banco de dados da vítima. A Rede Segura (2012b) cita que a maior parte das aplicações *web* corporativas utilizam-se do banco de dados para manter dados tanto da empresa como de seus clientes. Nestes arquivos armazenados estão incluídas informações, das quais sua segurança é de grande importância. Para acessar o banco de dados, o formato mais utilizado é linguagem SQL.

Caso as instruções para a incorporação de dados fornecidos pelo desenvolvedor à linguagem SQL não sejam formuladas de maneira segura, corre-se o risco de o aplicativo ser vulnerável, dando oportunidade ao chamado SQL *injection*. Esta invasão consiste em um

usuário mal intencionado, que através da falha no sistema, reprogramará a comunicação entre o aplicativo e o banco de dados, podendo assim manipular ou extrair informações (REDE SEGURA, 2012b).

2.5.3 Session Hijacking

Session hijacking, também conhecido como sequestro de sessão, segundo Janssen (2013b), pode ocorrer quando, a partir do *logon*, a identificação do cliente é enviada a um servidor *web* e autenticada, mas a sessão de identificação encontra-se comprometida, ou seja, apresentando vulnerabilidade. Logo, um usuário mal intencionado consegue acesso não autorizado ao servidor *web*, implementa códigos *JavaScripts* maliciosos, cavalos de troia, ou causa outros danos prejudiciais aos dados e ao próprio usuário.

Há sempre muito cuidado por parte dos desenvolvedores ao se tratar de s*ession hijacking*. O perigo consiste também no fato que os *cookies HyperText Transfer Protocol* (HTTP), que são usados para sustentar a sessão de identificação de um *site*, podem ser pirateados por um invasor (JANSSEN, 2013b). Ou seja, baseado no XSS *Script* para roubar a sessão de um usuário que está com o *login* efetuado através da inserção de códigos *JavaScript*, que roubarão a sessão do usuário (*cookie*), permitirá que outro usuário mal intencionado, mesmo sem ter o *login* e senha do usuário roubado, consiga acessar a conta.

2.6 PADRÕES DE PROJETO

Padrões de projeto (*design patterns),* são soluções que permitem uma maior eficiência, quando se tratando da arquitetura de softwares, e são largamente utilizados, facilitando a resolução de problemas correntes em projeto de software (GAMMA et. al, 1994). Freeman e Freeman (2005) afirmam que, quando utilizado um padrão, outros desenvolvedores que precisarem trabalhar no projeto posteriormente, saberão como dar continuidade ao mesmo, permitindo mais facilidade no desenvolvimento do código. Pode-se adequar os padrões à necessidade do software, o que acresce de maneira significativa os benefícios que dispõe sua utilização.

Permitindo a construção de códigos que possuam boa qualidade de *design* orientado a objeto, os padrões de projeto propõem soluções para determinados contextos ou necessidades (FREEMAN; FREEMAN, 2005). Tem-se como exemplo a necessidade de acessar individualmente objetos de uma coleção, sem expor a implementação do conjunto, onde

determinado padrão pode permitir o encapsulamento da interação em uma classe separada. Existem diversos padrões de projeto, cada qual com suas características particularizadas e aplicações diferenciadas.

2.6.1 Active Record

Definido por Fowler (2006), o *Active Record* (registro ativo) é o responsável por armazenar e buscar dados no banco de dados, colocando a lógica de acesso a estes no objeto de domínio. Encapsulando linhas de tabela e visão, quanto ao acesso do banco de dados, todos os objetos sabem como ler os dados do banco de dados, bem como gravar os mesmos nele.

Em outras palavras, armazena dados em banco de dados relacionais. Fowler (2006) cita ainda que a estrutura de dados que irá compor o registro ativo deve corresponder à mesma composta pelo banco de dados, sendo que o tipo de atributo deve corresponder a entregue pela interface SQL. Se tratando ainda da classe registro ativo, tem-se as seguintes funções que são realizadas pelos métodos: de uma linha de conjunto de dados resultados de uma consulta SQL, é construída uma instância do registro ativo; para inserção posterior na tabela, há a construção de uma nova instância; encapsular consultas SQL e, pelos métodos de busca estáticos, retornar objetos do tipo registro ativo; inserir os dados do registro ativo no banco de dados, a partir da realização da atualização; leitura e gravação dos atributos; e, por fim, implementação de alguns fragmentos de lógica de negócio (FOWLER, 2006).

2.6.2 Padrão Factory Method, Padrão Multiton e a lazy initialization

O padrão *Factory Method* define uma interface para a criação de um objeto, permitindo às subclasses decidir qual classe instanciar (FREEMANN; FREEMANN, 2005). Sendo assim, o *Factory Method* permitirá a determinada classe deferir a instância para subclasses, adiando ela ou não. Já o padrão *Multiton*, segundo Carr (2012), permite que quando utilizado um determinado parâmetro novamente, ele obterá a mesma instância. Garantindo, desta forma, que um número limitado de instâncias de uma classe possa existir, de modo que especifica uma chave para cada instância permitindo que apenas um único objeto seja criado para cada uma destas chaves. Pode-se dizer ainda que o *Multiton* é constituído de um construtor privado e rastreia (mapeia) objetos criados, com uma chave privada, assim como seu construtor.

A *Lazy Initialization*, conhecida também por inicialização tardia, faz com que determinado objeto inicializado desta forma só seja criado quando for utilizado a primeira vez (MSDN, 2013). Ou seja, sua inicialização será adiada, contribuindo com a redução de requisitos de memória do programa, assim como irá melhorar o desempenho e evitar desperdício computacional.

Os padrões *Factory Method* e *Multiton* podem ser usados juntamente à *Lazy Initialization*. Sendo que, o *Factory Method* irá abstrair a criação dos objetos, o *Multiton* irá permitir buscar a mesma instância de um objeto já criado e o *Lazy Initialization* irá carregar o mais tarde possível determinada instância de um objeto.

2.6.3 Dependency Injection

Segundo Caprio (2005), a *Dependency Injection* (injeção de dependência) consiste em um padrão de projeto que objetiva atenuar o aumento de dependências quando há a reutilização de componentes e de ligações existentes entre eles, de forma que seja elaborada uma arquitetura coesa. A criação desta ligação acaba se tornando uma tarefa complicada, já que o aplicativo se torna cada vez maior e mais complexo, criando cada vez mais dependências entre si, por isso da necessidade de utilização deste padrão de projeto.

Permitindo a injeção de objetos em determinada classe em vez de confiar na classe para criar em si o objeto, o nível de acoplamento de diferentes módulos em um software se tornará menor (CAPRIO, 2005). De uma maneira geral, a *Dependency Injection* permite separar as diversas dependências existentes cada qual em seu próprio objeto, aumentando a reutilização de código e facilitando a manutenção futura do sistema.

2.6.4 Model-view-controller

O *Model-View-Controller* (MVC) é um conjunto de padrões agrupados, que, como seu próprio nome informa, é baseado no modelo, na visualização e em um controlador. Segundo Freeman e Freeman (2005), o controlador recebe dados solicitados pelo usuário e determina o significado dos mesmos para o modelo; o modelo possui todas as informações relativas à lógica do aplicativo, bem como o estado e demais dados; a visualização fornece a apresentação do modelo, ou seja, obtém o estado e os dados que devem ser exibidos do modelo.

Basicamente, pode-se dizer que há um ciclo quando se trata do MVC, onde o usuário faz uma solicitação ao aplicativo, esta é enviada ao controlador, que manipula o modelo para que o mesmo realize a ação. O modelo então informa à visualização da nova mudança de estado e a mesma fornece ao usuário a atualização de dados ou ação solicitada (FREEMAN; FREEMAN, 2005).

2.7 SISTEMA ATUAL

Atualmente o software SAI é disponibilizado *on-line* e apenas uma psicóloga possui permissão de acesso. Ele inclui módulos de cadastro de pacientes, consultas, relatório de informações de paciente, além de registros como sonhos, doenças, remédios e supervisões do profissional.

O controle dos horários dos pacientes é feito manualmente através de uma agenda em papel, dificultando uma visão ampla dos compromissos semanais e mensais. Além de impossibilitar a facilidade na alteração dos horários dos pacientes, pois devido a questões de segurança, a agenda fica somente no consultório.

O sistema atual não pode ser utilizado por outros psicólogos, devido ao fato do mesmo ter sido feito exclusivamente para um psicólogo. Isto impede a expansão do sistema e sua comercialização no modelo de software como serviço.

Desta forma, verificou-se a necessidade de criar um novo sistema, objetivando a reestruturação dos recursos existentes e a criação de novos módulos. De forma interligada, gerando assim otimização tanto em termos de organização de processos quanto em ganho de tempo. Esse novo sistema foi intitulado Gestor Psicologia.

2.8 TRABALHOS CORRELATOS

Pode-se citar como trabalhos correlatos o software livre GestorPsi (2004) desenvolvido pelo Instituto de Psicologia Comportamental de São Carlos e também o Insight (2006) desenvolvido pelo Grupo Insight.

O GestorPsi foi desenvolvido para atender todos os tipos de estabelecimentos de psicologia, sejam estes da área clínica ou não (GESTOR PSI, 2004). É um sistema *on-line* voltado para a gestão administrativa do estabelecimento de psicologia, contendo a gestão de serviços oferecidos, controle de funcionários, agenda para reuniões individuais ou em grupos

com reserva de equipamentos, gestão dos clientes com fila de espera, além de um prontuário técnico por paciente bastante abrangente.

Comparando o GestorPsi com o Gestor Psicologia, vê-se um foco diferente de mercado. Enquanto o GestorPsi se preocupa com a administração completa de qualquer estabelecimento, tornando-se assim mais abrangente e detalhado, o Gestor Psicologia objetiva facilitar o trabalho de psicólogos independentes, simplificando atividades como a administração de pacientes e a gestão da agenda.

Já o Insight foi desenvolvido para a plataforma *desktop* no idioma espanhol e permite gerenciar uma clínica de psicologia com alto nível de detalhes (INSIGHT, 2006). O sistema divide-se em quatro módulos principais: tarefas administrativas, gestão de pacientes, contabilidade e ferramentas.

De uma maneira geral o Insight permite a gestão completa de uma clínica de psicologia sem depender de outro software. Possibilita o cadastro de uma empresa, controle do fluxo de caixa, agendamento das consultas, múltiplos psicólogos e funcionários, geração de faturas para os pacientes e a administração do prontuário dos mesmos. Em contrapartida, o Gestor Psicologia se concentra em manter-se simples e prático, para psicólogos independentes que precisam de praticidade em seu dia-a-dia.

3 DESENVOLVIMENTO

Neste capítulo estão descritos as particularidades técnicas do sistema proposto, tais como a descrição, a apresentação dos requisitos funcionais e não funcionais, os diagramas de casos de uso e suas descrições, o diagrama de entidade relacionamento, a implementação, técnicas e ferramentas utilizadas, a operacionalidade da implementação e os resultados e discussão.

3.1 LEVANTAMENTO DE INFORMAÇÕES

O Gestor Psicologia é um sistema que permite o gerenciamento de um consultório psicológico, agregando o módulo de pacientes e agenda, além de possibilitar que vários consultórios utilizem um mesmo sistema, pelo fato da plataforma utilizada ser *web* no modelo de SaaS.

O desenvolvimento deste sistema é resultado da reformulação de um outro sistema chamado SAI que foi utilizado por mais de três anos e que possui uma série de deficiências. Desta forma, foi melhorado o módulo de pacientes, implementado uma módulo de agenda e criado um *site* que permite o cadastro de psicólogos.

O módulo de pacientes foi implementado para permitir pré-agendar as consultas e gerar automaticamente a agenda do psicólogo. Ainda no módulo de pacientes é possível gerenciar em uma só tela o prontuário do paciente, registrando as consultas, sonhos, doenças, remédios e supervisões de uma maneira mais simples que o sistema atual.

No módulo agenda o psicólogo tem uma visão clara por dia, semana ou mês dos seus eventos e suas consultas com pacientes. O módulo permite o cadastro de eventos com opções de recorrência, podendo ser configurados lembretes para o psicólogo e também para o paciente, via *e-mail* e *Short Message Service* (SMS), além de permitir categorizar os eventos por cor, facilitando a visualização da agenda.

É possível que um psicólogo se registre através do *site* do Gestor Psicologia, escolha um plano e, a partir disto possa cadastrar seus próprios pacientes e gerenciar seu consultório. Portanto, o Gestor Psicologia agregará vários consultórios totalmente independentes em um só sistema *on-line*, sendo que cada psicólogo poderá pagar um plano que dará acesso a determinados recursos. O cancelamento de planos e a troca da forma de pagamento poderão ser feitos a qualquer momento pelo psicólogo.

O sistema está hospedado em *cloud computing* compartilhado, usa a linguagem de programação PHP e *frameworks* Silex e Propel ORM, o banco de dados é o MySQL. A integração com as operadoras de telefonia celular é realizada através de *gateway* de SMS.

3.2 ESPECIFICAÇÃO DOS REQUISITOS

Nesta seção é apresentada a especificação do Gestor Psicologia, a qual foi modelada utilizando-se a ferramenta Enterprise Architect. O sistema foi desenvolvido seguindo a análise orientada a objetos, implementado de acordo com a proposta dos *frameworks* Silex e Propel ORM.

Utiliza-se a notação *Unified Modeling Language* (UML) para a criação do diagrama de casos de uso, classe e atividades.

3.2.1 Requisitos do sistema

O Quadro 1 apresenta os requisitos funcionais (RF) previstos para o sistema e sua rastreabilidade, ou seja, vinculação com os casos de uso (*use cases* - UC). O Quadro 2 lista os requisitos não funcionais (RNF) previstos para o sistema.

Quadro 1 - Requisitos funcionais

Requisitos Funcionais	**Caso de Uso**
RF01 – O visitante poderá escolher um plano para utilização do sistema.	UC01
RF02 – O visitante poderá criar uma conta para acessar o sistema.	UC02
RF03 – O sistema deverá permitir ao usuário efetuar *login* no sistema.	UC03
RF04 – O sistema deverá permitir ao usuário alterar a senha do *login*.	UC04
RF05 – O sistema deverá permitir ao usuário administrar os pacientes.	UC05
RF06 – O sistema deverá permitir ao usuário administrar o prontuário de um paciente.	UC06
RF07 – O sistema deverá permitir ao usuário administrar os sonhos de um paciente.	UC07
RF08 – O sistema deverá permitir ao usuário administrar as doenças de um paciente.	UC08
RF09 – O sistema deverá permitir ao usuário administrar os remédios	UC09

de um paciente.	
RF10 – O sistema deverá permitir ao usuário administrar as supervisões de um paciente.	UC10
RF11 - O sistema deverá permitir ao usuário administrar as opções de como nos conheceu de um paciente.	UC11
RF12 - O sistema deverá permitir ao usuário administrar as opções de encaminhado por de um paciente.	UC12
RF13 - O sistema deverá permitir ao usuário administrar as opções de escolaridade de um paciente.	UC13
RF14 - O sistema deverá permitir ao usuário administrar as opções de profissão de um paciente.	UC14
RF15 - O sistema deverá permitir ao usuário administrar as opções de estado civil de um paciente.	UC15
RF16 - O sistema deverá permitir ao usuário administrar as opções de religião de um paciente.	UC16
RF17 – O sistema deverá permitir ao usuário visualizar o relatório de informações de um paciente.	UC17
RF18 – O sistema deverá permitir ao usuário visualizar o relatório de visão geral dos pacientes.	UC18
RF19 – O sistema deverá permitir ao usuário administrar uma agenda.	UC19
RF20 – O sistema deverá permitir ao usuário administrar os eventos da agenda.	UC20
RF21 – O sistema deverá permitir ao usuário configurar lembretes para cada evento da agenda.	UC20
RF22 – O sistema deverá permitir ao usuário administrar as categorias de eventos da agenda.	UC21
RF23 – O sistema deverá permitir ao usuário mudar o tema de exibição.	UC22
RF24 – O sistema deverá permitir ao usuário administrar outros usuários de sua conta.	UC23
RF25 – O sistema deverá permitir ao usuário mudar de plano.	UC24
RF26 – O sistema deverá permitir ao usuário cancelar sua conta.	UC25

Quadro 2 - Requisitos não funcionais

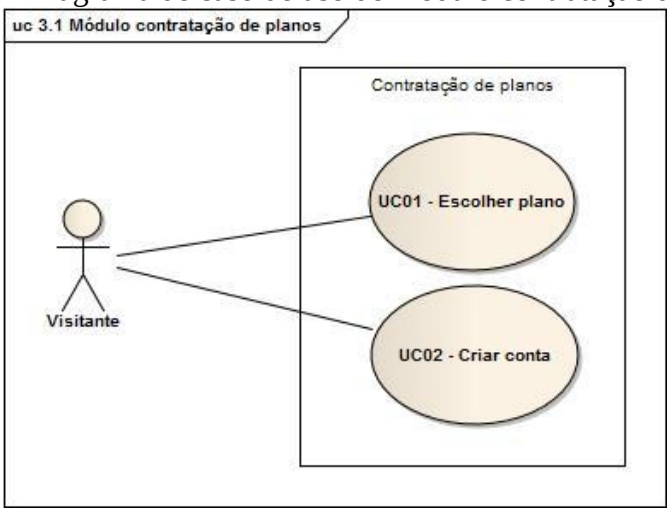
Requisitos Não Funcionais
RNF01 – O sistema deve utilizar senhas de acesso para o controle seguro da aplicação.
RNF02 – O sistema deve ser desenvolvido na linguagem PHP.
RNF03 – O sistema deve ser desenvolvido utilizando o ambiente NetBeans 7.3 IDE, seguindo o modelo de arquitetura em camadas *Model, View* e *Controller* (MVC).
RNF04 – O sistema deve ser desenvolvido para a plataforma Linux.
RNF05 – O sistema deve utilizar o banco de dados MySQL a partir da versão 5.5 em diante.
RNF06 – O sistema deve ser desenvolvido utilizando a tecnologia de *layout* responsivo (que se ajusta à tela do usuário).

3.2.2 Diagrama de casos de uso

Esta subseção apresenta os diagramas de casos de uso do sistema para melhor exemplificação das atividades exercidas. Jacobson (1992 apud MACORATTI, 2004) define caso de uso como um "documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo". As descrições dos principais casos de uso estão apresentadas no Apêndice A.

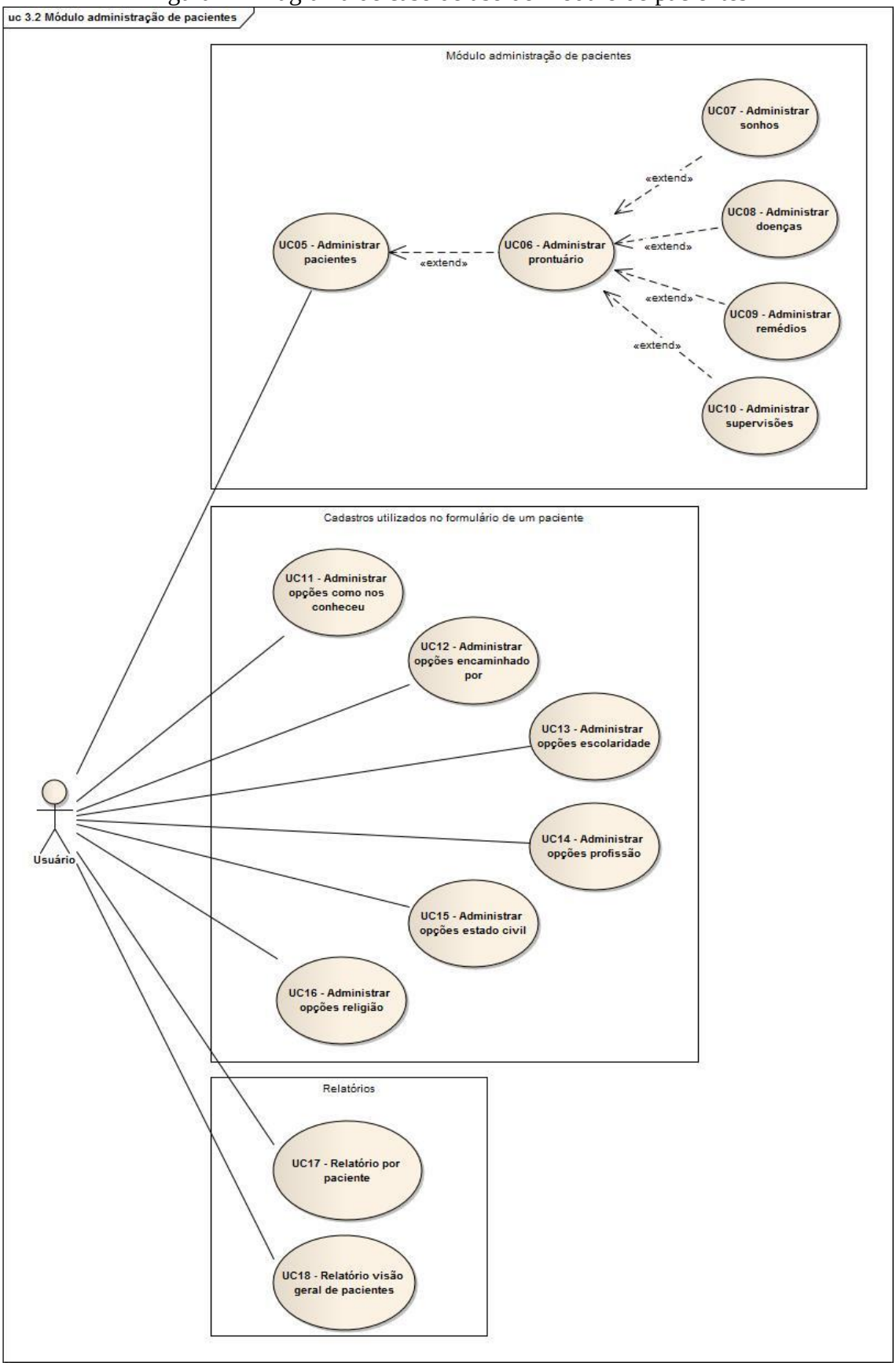
Na Figura 1 mostra-se como é o processo de criação de conta de um visitante, que pode acessar o *site*, escolher um plano e cadastrar-se para ter acesso ao sistema.

Figura 1 - Diagrama de caso de uso do módulo contratação de planos



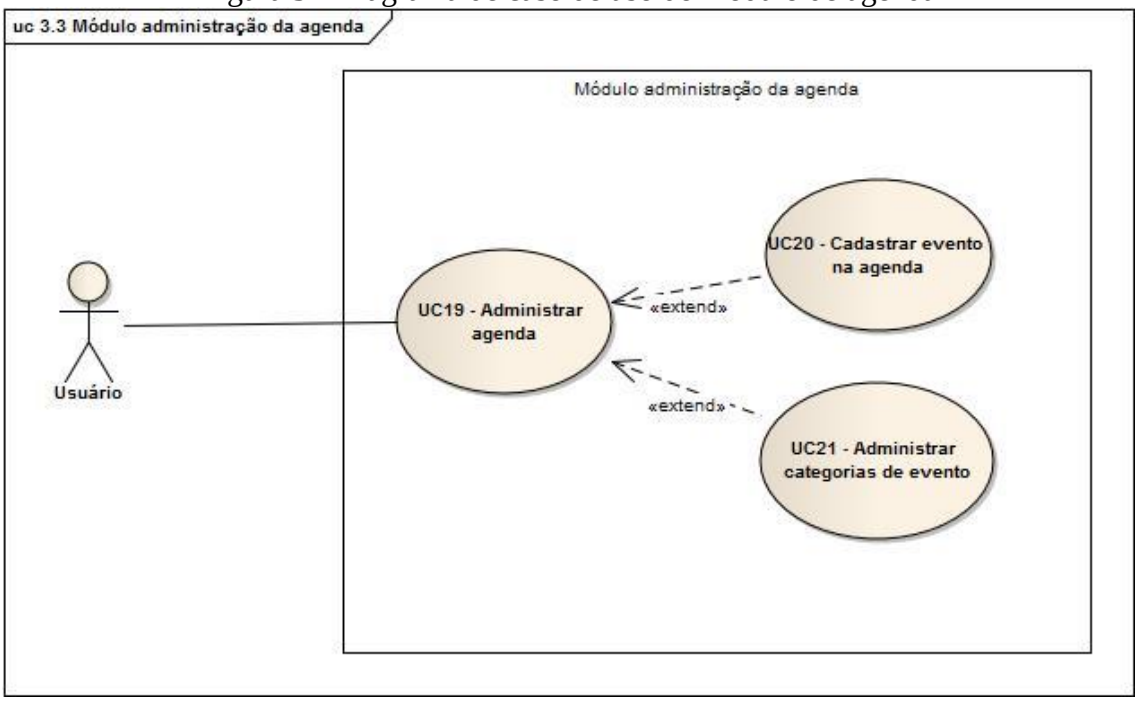
Tem-se, na Figura 2, o intuito de mostrar o funcionamento do módulo de pacientes, com seus cadastros e relatórios. O usuário já tendo acessado o sistema, poderá administrar seus pacientes e gerenciar os prontuários destes (tela única que permite o cadastro de consultas, sonhos, doenças, remédios e supervisões). Para permitir a normalização dos dados e a geração de relatórios, é possível administrar os dados utilizados no formulário de cadastro de um paciente (como nos conheceu, encaminhado por, escolaridade, profissão, estado civil e religião). Há dois relatórios possíveis no módulo de pacientes, o relatório por paciente, que é um listagem das informações de um paciente específico e o relatório de visão geral de pacientes, que permite uma visão holística dos pacientes atendidos pelo psicólogo.

Figura 2 - Diagrama de caso de uso do módulo de pacientes



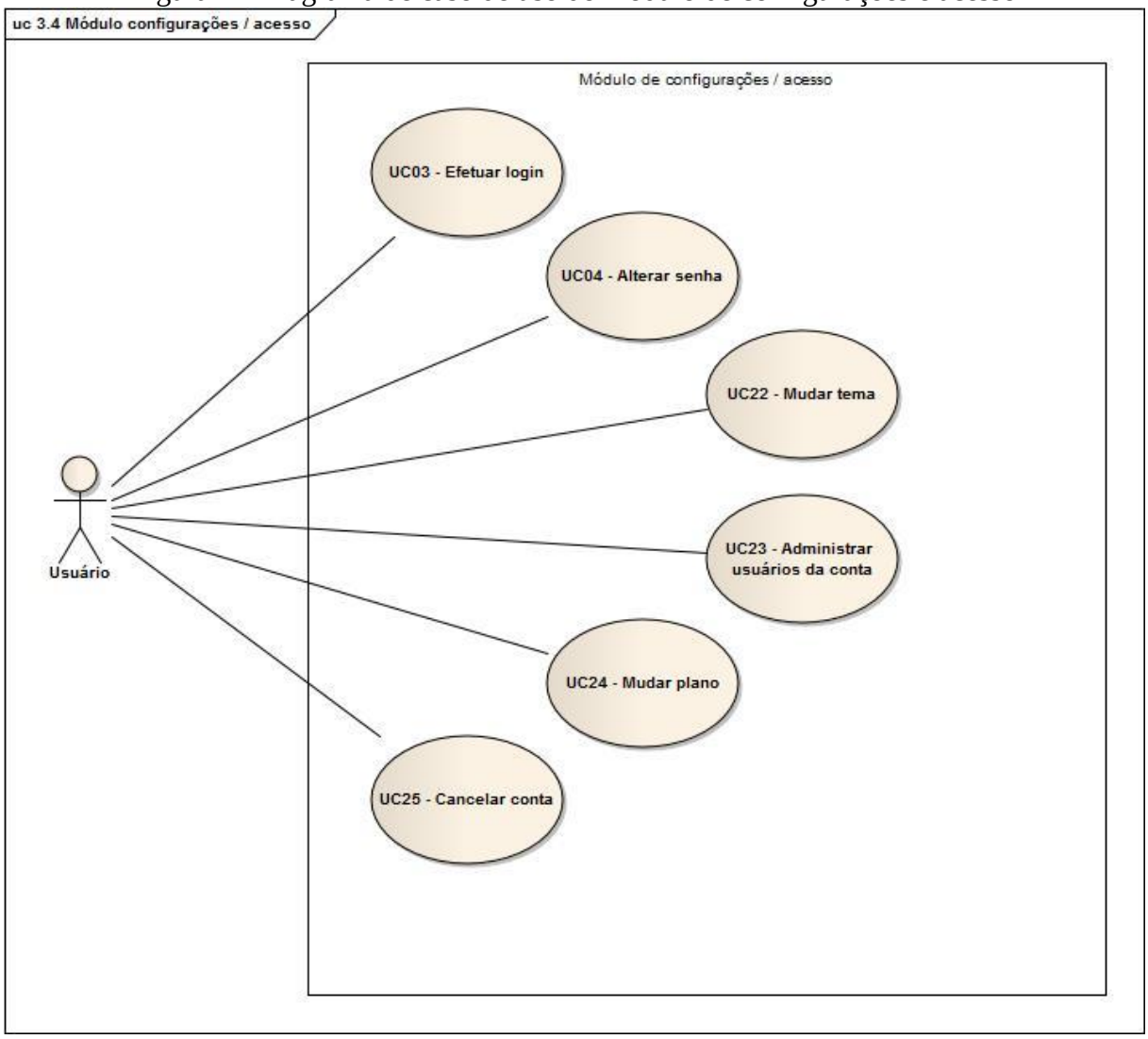
É possível, na Figura 3, entender o módulo de agenda do sistema. O usuário já tendo acessado o sistema, poderá administrar sua agenda, visualizando os eventos por dia, semana ou mês e filtrando por categoria. Além disso, será possível cadastrar novos eventos e também criar novas categorias.

Figura 3 - Diagrama de caso de uso do módulo de agenda



Na Figura 4, mostra-se o módulo de configurações e acesso. O usuário poderá autenticar-se através do *login* e realizar diversas ações como alterar sua senha, mudar o tema de apresentação do sistema, administrar outros usuários, além de mudar de plano e também cancelar a conta atual.

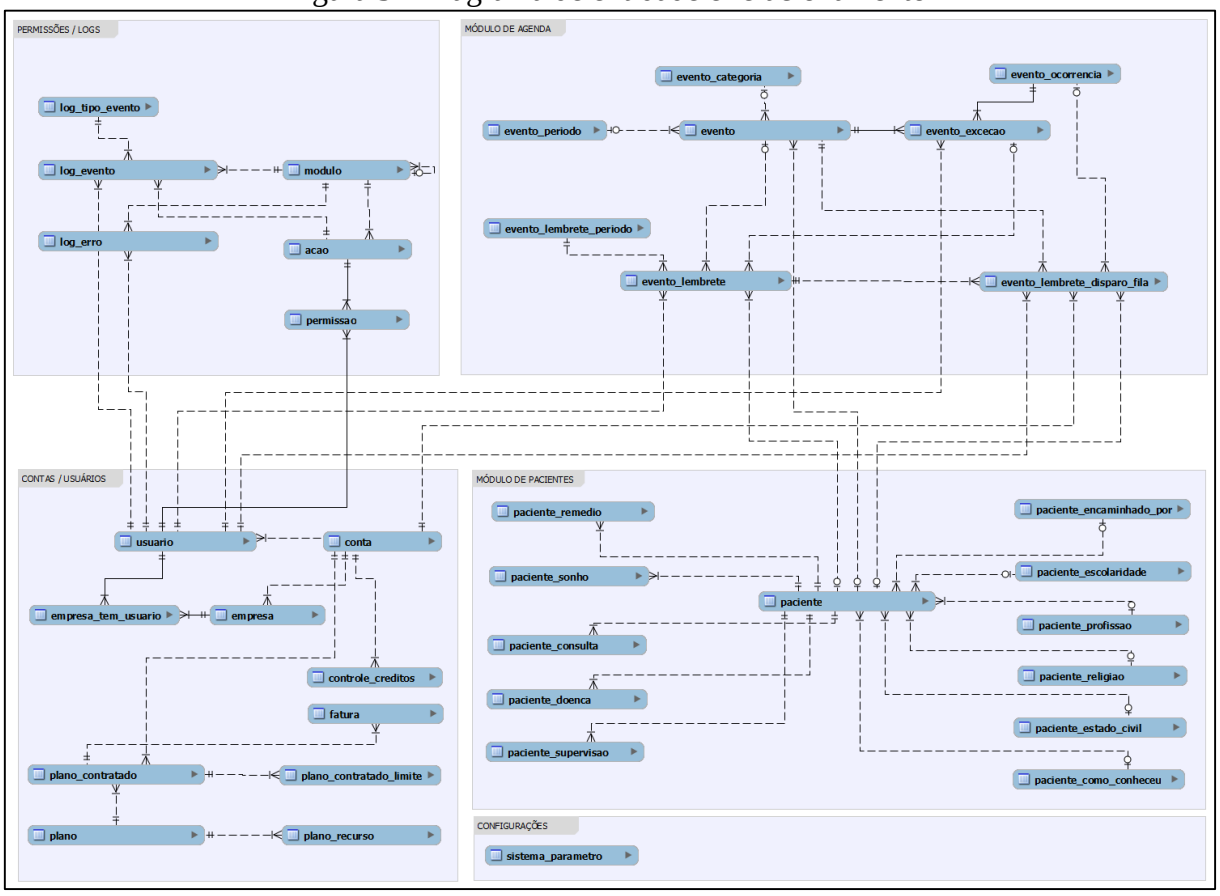
Figura 4 - Diagrama de caso de uso do módulo de configurações e acesso



3.2.3 Modelo Entidade Relacionamento

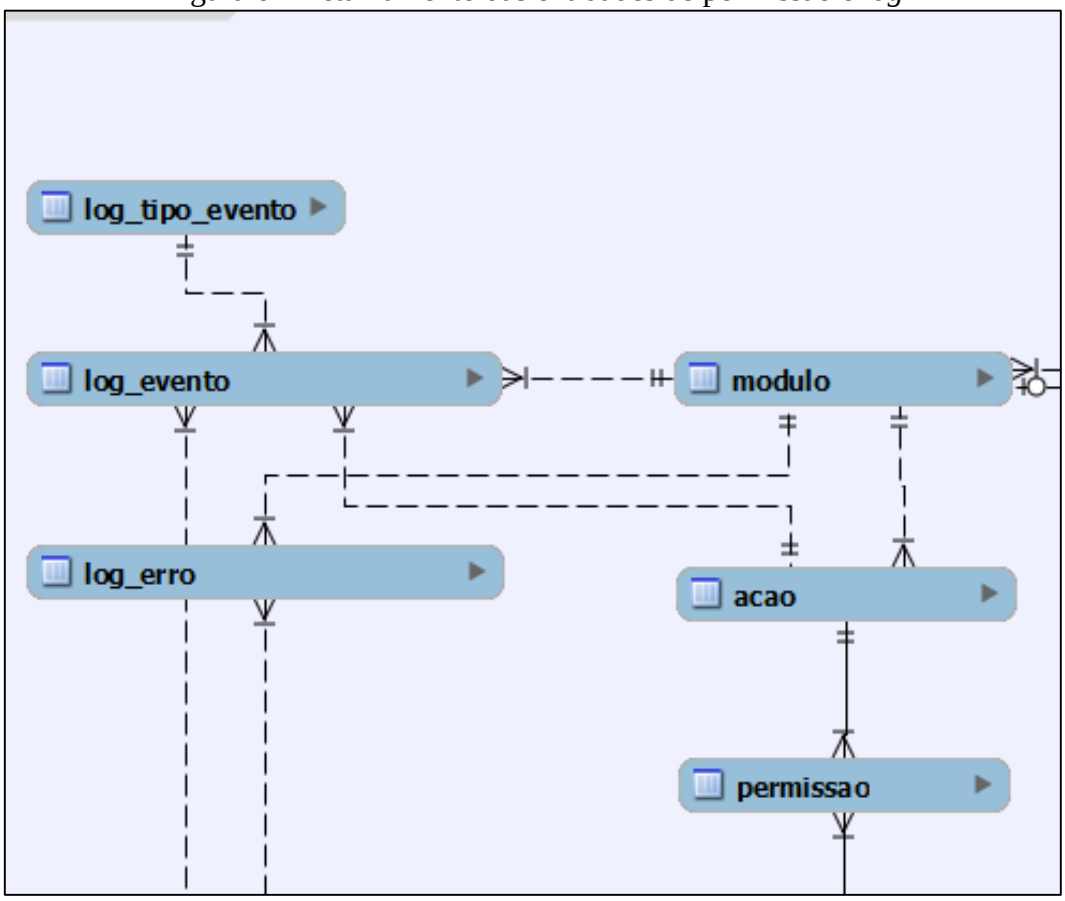
Na Figura 5 é mostrado o Modelo Entidade Relacionamento do sistema desenvolvido. São apresentadas as entidade de permissão e *log*, as entidades das contas e usuários e também dos módulos de agenda e pacientes. O dicionário de dados das principais entidades está descrito no Apêndice B.

Figura 5 - Diagrama de entidade e relacionamento



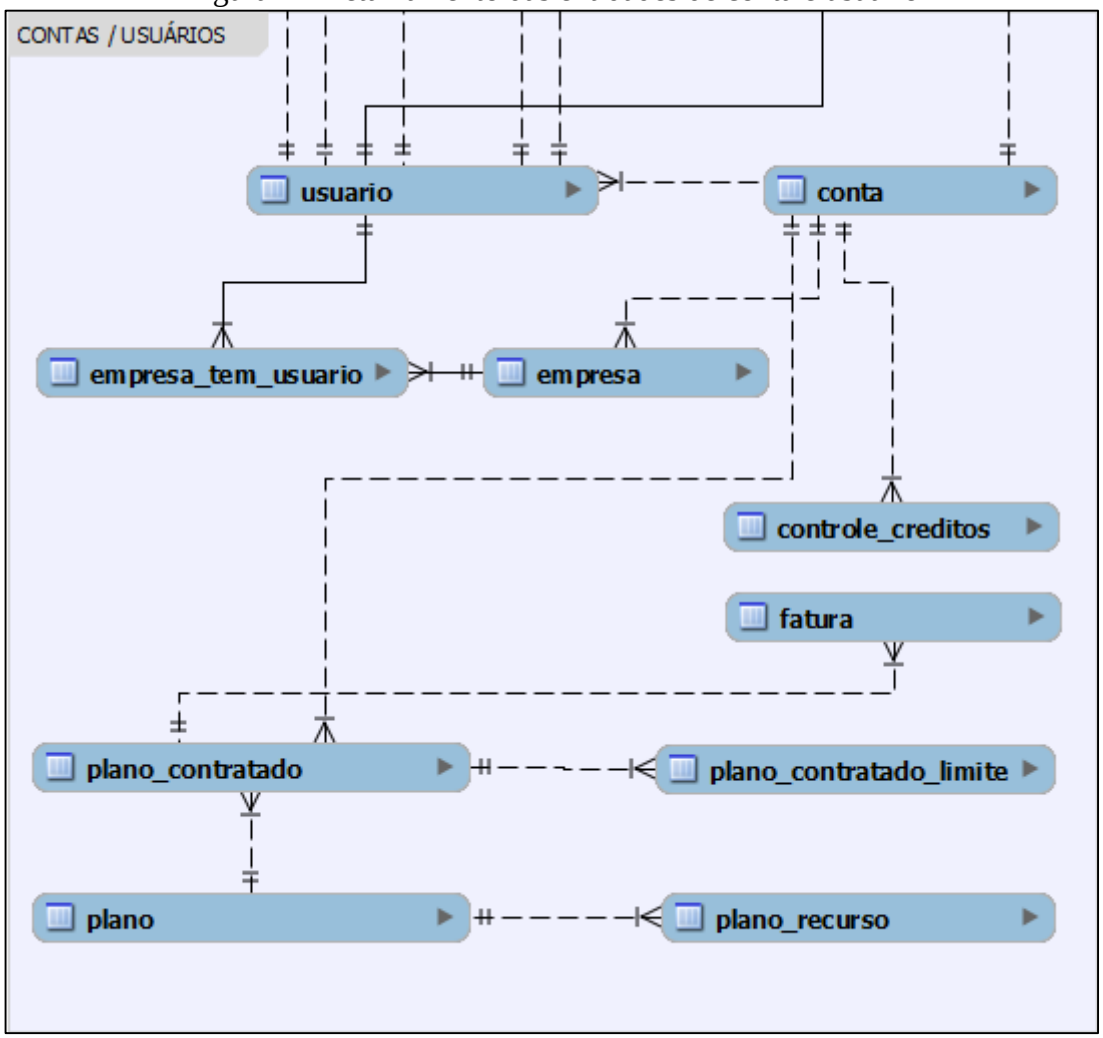
Pode-se visualizar na Figura 6 o controle de *logs* e permissões. No que tange a parte de *logs*, todas as ações de todos os usuários são registradas em *log*, inclusive os possíveis erros de autenticação e erros de sistema. Sobre as permissões, da forma como o sistema foi modelado é possível restringir para cada usuário o acesso às ações dos módulos existentes.

Figura 6 - Detalhamento das entidades de permissão e *log*



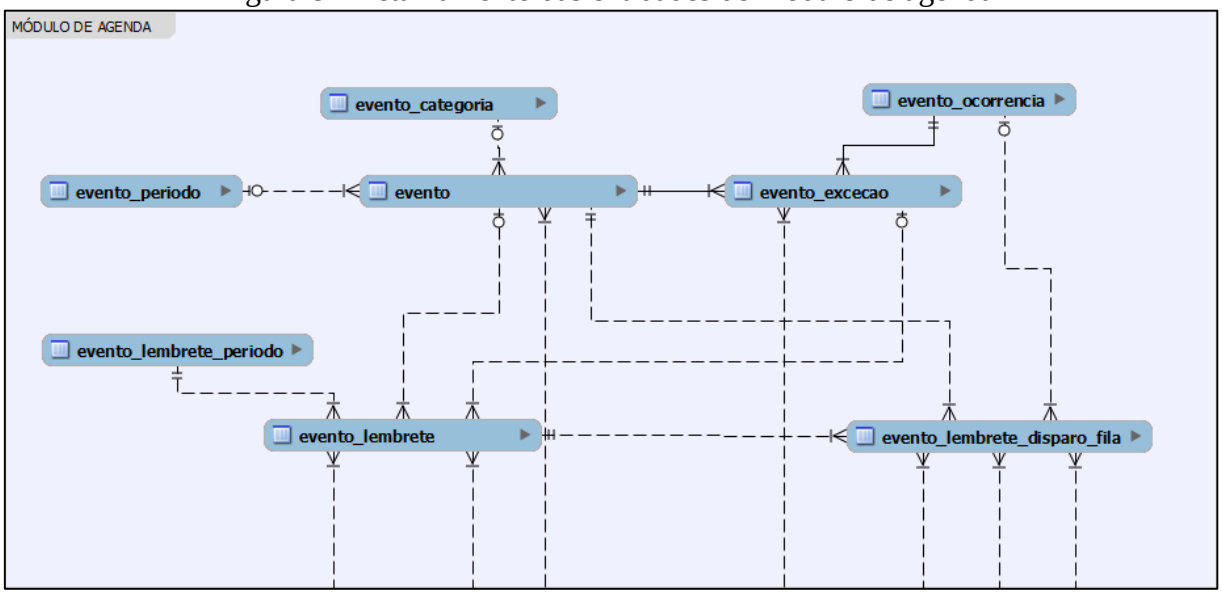
É detalhado a estrutura de entidades existente para comportar um sistema no modelo SaaS, na Figura 7. Uma conta pode estar vinculada a um único plano ativo, sendo que há um histórico de todos os planos já contratados por uma conta. Além disso, uma conta pode ter várias empresas vinculadas e um usuário (psicólogo) pode trabalhar para mais de uma empresa. Existe também um controle de limites para o plano contratado, que é utilizado pelo sistema para verificar as restrições do plano atual.

Figura 7 - Detalhamento das entidades de conta e usuário



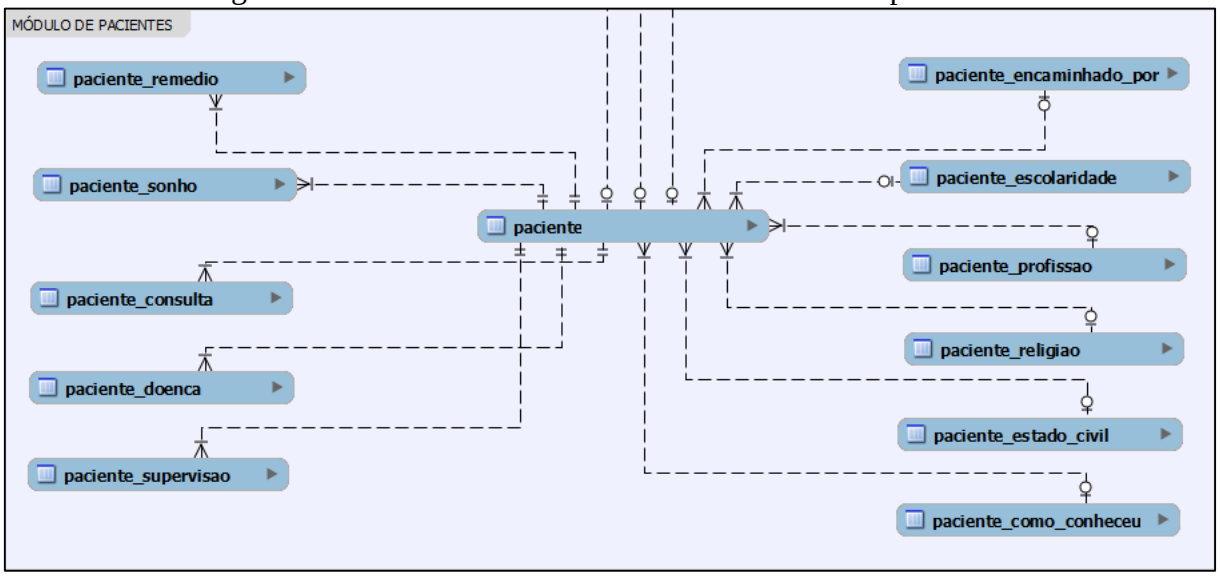
Na Figura 8 mostra-se as entidades existentes no módulo de agenda. Um evento pode estar vinculado a uma categoria e pode ter recorrências (por dia, semana, mês e ano), sendo que o usuário pode criar exceções para determinada ocorrência do evento. Ao mesmo tempo, é possível criar lembretes por *e-mail* ou SMS para o evento, sendo que todos os avisos gerados a partir dos lembretes são criados dinamicamente e colocados em uma entidade de fila de envio.

Figura 8 - Detalhamento das entidades do módulo de agenda



As entidades do módulo de pacientes estão apresentadas na Figura 9. A um paciente pode ser vinculado consultas, remédios, sonhos, doenças e supervisões, além disso, por questões de estatística, as seguintes informações de um paciente são normalizadas: encaminhado por, escolaridade, profissão, religião, estado civil e como nos conheceu.

Figura 9 - Detalhamento das entidades do módulo de pacientes



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento do sistema, foi utilizada a linguagem de programação PHP juntamente com os *frameworks* Silex e Propel ORM, banco de dados MySQL, alguns padrões de projeto de software, técnicas de prevenção de vulnerabilidades e técnicas para *layout* responsivo.

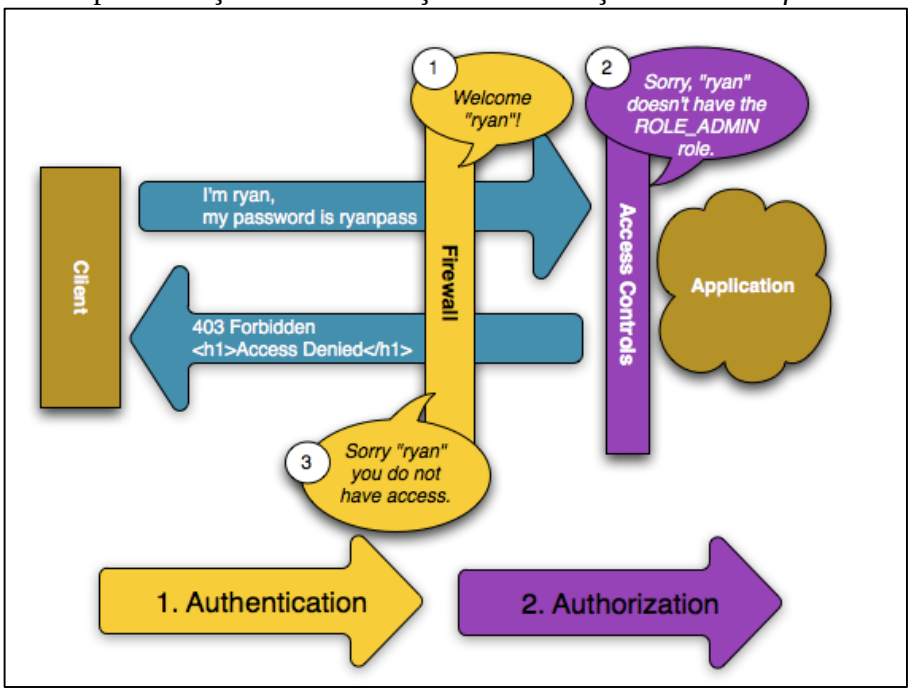
3.3.1.1 Framework Silex

O sistema foi desenvolvido utilizando o *framework* Silex em conjunto com o *framework* Propel ORM, para criar uma aplicação seguindo o padrão MVC. Desta forma, o Silex se preocupa com as camadas de controle e visualização, enquanto o Propel ORM se preocupa em abstrair a camada de modelo em um interface totalmente orientada a objetos.

O conjunto de ferramentas disponibilizadas pelo Silex ajudaram, entre outras coisas, a padronizar a autenticação e a autorização dos usuários, criação de *cache* automatizado das páginas, padronização e criação de *Uniform Resource Locator* (URLs) amigáveis, implementação de formulários seguindo um modelo uniforme de criação de campos e validação, além de permitir reaproveitar o código de componentes na camada de visualização da aplicação.

Na Figura 10 é possível entender o funcionamento da autenticação e autorização implementados pelo Silex, para tratar o acesso dos usuários. Enquanto na Figura 11 demonstra-se como é a criação de URLs amigáveis na aplicação exemplificando através de um trecho de código. Mostra-se, na Figura 12, a criação padronizada de formulários através de uma estrutura orientada a objetos, além disso, é possível entender os métodos de validação e solicitação dos dados preenchidos no formulário.

Figura 10 - Implementação de autenticação e autorização utilizando *firewalls* e regras



Fonte: Symfony (2013).

Figura 11 - Criação de URLs amigáveis e padronização no recebimento de parâmetros

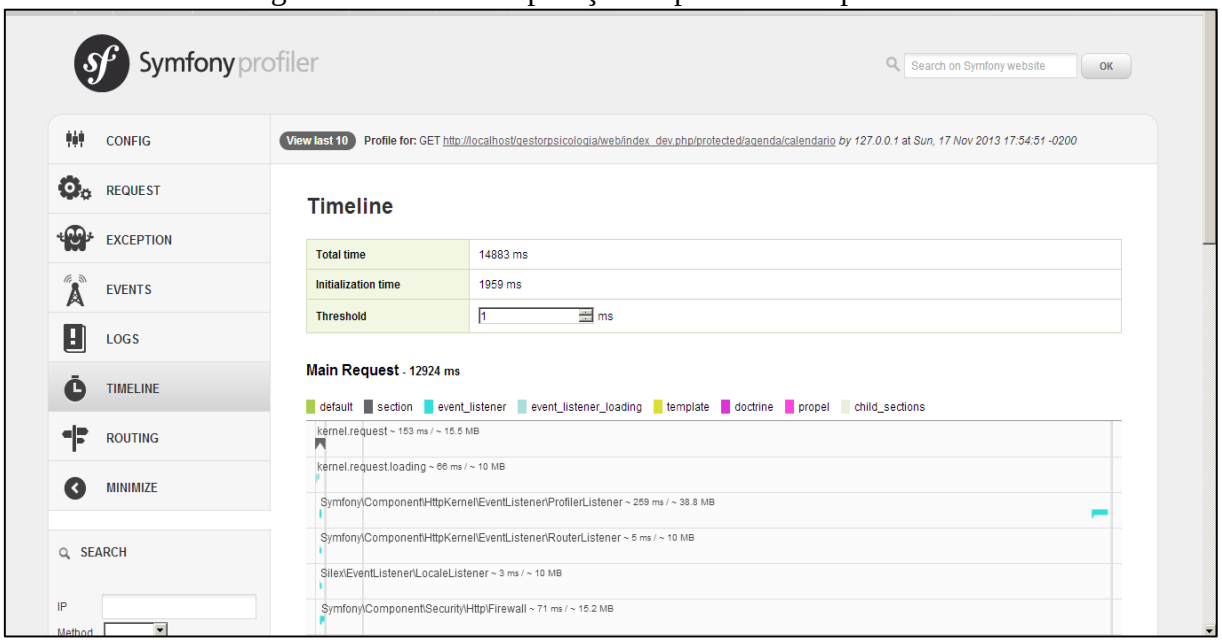
```php
<?php

use Symfony\Component\Form\FormError;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Validator\Constraints as Assert;
use GestorPsicologiaSistema\Models\LogEventoPeer;
use GestorPsicologiaSistema\Models\LogTipoEventoPeer;
use GestorPsicologiaSistema\Models\EventoCategoria;
use GestorPsicologiaSistema\Models\EventoCategoriaPeer;
use GestorPsicologiaSistema\Models\EventoCategoriaQuery;

$sistema->match('/agenda/categoria/{acao}/{id}', function(Request $request, $acao, $id) use ($app)
{
    $isNomeDisabled = false;

    if ($acao == 'cadastrar')
    {
        $predefinido = array(
        );
    }
    elseif ($acao == 'editar')
    {
        $id = decrypt($id, true);

        /* @var $objeto GestorPsicologiaSistema\Models\EventoCategoria */
        $objeto = EventoCategoriaQuery::create()
                    ->filterById($id)
                    ->filterByContaId(_conta_id())
                    ->filterByUsuario(_usuario())
                    ->findOne();
```

Figura 12 - Implementação de formulários de forma organizada em toda a aplicação

```php
$form = $app['form.factory']->createBuilder('form', $predefinido)
        ->add('Nome',            'text', array('label' => 'Nome:', 'disabled' => $isNomeDisabled))
        ->add('Cor',            'text', array('label' => 'Cor:'))
        ->getForm();

$erros = array();

if ($request->isMethod('POST'))
{
    $form->bind($request);

    if ($form->isValid() && empty($erros))
    {
        $formData = $form->getData();

        $objeto = isset($objeto) ? $objeto : new EventoCategoria;
        $objeto->setByArray($formData);
        $objeto->setContaId(_conta_id());
        $objeto->setUsuario(_usuario());
        if ($objeto->isNew())
            $objeto->setCorFundo(null); // Setando cor para utilizar a validação do Propel

        // Pegando cores da paleta selecionada
        if ($formData['Cor'] && ($cor = strtoupper($formData['Cor'])) && array_key_exists($cor, EventoCate
        {
            $objeto->setCorFundo(EventoCategoriaPeer::$colorsPalette[$cor]['cor_fundo']);
            $objeto->setCorBorda(EventoCategoriaPeer::$colorsPalette[$cor]['cor_borda']);
            $objeto->setCorTexto(EventoCategoriaPeer::$colorsPalette[$cor]['cor_texto']);
        }
```

Além disso, o Silex possui ótimos recursos para depurar a aplicação em ambiente de desenvolvimento, tendo uma ferramenta exclusiva que contém informações úteis acerca da página que foi carregada, conforme é demonstrado através da Figura 13, e também um *log* de todos os erros que acontecem.

Figura 13 - Tela de depuração disponibilizada pelo Silex



3.3.1.2 Framework Propel ORM

O Propel ORM foi utilizado para abstrair a camada de persistência e disponibilizar uma forma orientada a objetos para intermediar a comunicação com o banco de dados. Como consequência, as tarefas de inserir, atualizar, deletar e consultar dados foram simplificadas. As classes da camada de persistência são criadas a partir das tabelas que o Propel ORM identifica no banco de dados. Na Figura 14 exemplifica-se a criação de uma consulta ao banco de dados, passando informações de ordenação, filtros e gerando paginação, além da possibilidade de utilizar condições, como o *if* dentro da própria consulta, para manter a legibilidade do código criado. Na Figura 15 têm-se uma demonstração da exclusão de registros no banco de dados utilizando o Propel ORM.

Figura 14 - Exemplo de consulta utilizando o Propel ORM

```
// URLs
$paginacao_url = change_url_query_string($request->getUri(), null, array('pagina'), array('pagina' => ''));
$ordenacao_url = change_url_query_string($request->getUri(), null, array('ordenacao', 'pagina'), array('ordenacao' => ''));

list($ordenacao_col, $ordenacao_ordem) = get_parametros_ordenacao(new PacienteDoenca, $ordenacao, $ordenacao_padrao);

$objetos_listagem = PacienteDoencaQuery::create()
            ->_if($ordenacao_col)
            ->orderBy($ordenacao_col, $ordenacao_ordem)
            ->_endif()

            ->_if($busca)
            ->filterByNome("%{$busca}%")
            ->_endif()

            ->filterByPacienteId($paciente_id)
            ->filterByContaId(_conta_id($app))
            ->paginate($pagina, _param($app, 'QTD_REGISTROS_POR_PAGINA'));

LogEventoPeer::gravarEvento(LogTipoEventoPeer::TIPO_VISUALIZAR, 7, 13, "PacienteId: {$paciente_id}");
```

Figura 15 - Exemplo de exclusão utilizando o Propel ORM

```
try
{
    $deletados = PacienteDoencaQuery::create()
                    ->filterById($table['row'])
                    ->filterByContaId(_conta_id())
                    ->filterByPaciente($paciente)
                    ->delete();

    LogEventoPeer::gravarEvento(LogTipoEventoPeer::TIPO_REMOVER, 7, 16, "PacienteId:{$paciente_id} Id:" . impl

    $app['session']->getFlashBag()->add('success', 'Registro(s) excluídos(s) com sucesso.');
}
catch (\Exception $e)
{
    if ($app['debug']) {
        return new Response($e->__toString());
    } else {
        $app['session']->getFlashBag()->add('error', 'Oops! Parece que algo de errado aconteceu, por favor, te
    }
}
```

Este *framework* auxilia também a aumentar a segurança da comunicação com o banco de dados, pois por padrão ele evita ataques de SQL *Injection* e trata as entradas de dados de acordo com o tipo de campo especificado no banco de dados.

Além de tudo isso, o Propel possui seu próprio conjunto de validações dos campos que é utilizado em conjunto com o Silex para fazer a validação dos formulários, conforme é exibido na Figura 16.

Figura 16 - Criação de validações de entrada no Propel ORM

```
<table name="evento" phpName="Evento" idMethod="native">
    <column name="ID" phpName="Id" type="INTEGER" size="10" sqlType="int(10) unsigned" primaryKey="true" au
    <column name="DATA_INICIO" phpName="DataInicio" type="TIMESTAMP" required="true"/>
    <column name="DATA_TERMINO" phpName="DataTermino" type="TIMESTAMP" required="true"/>
    <column name="DIA_INTEIRO" phpName="DiaInteiro" type="BOOLEAN" size="1" required="false" defaultValue="
    <column name="TITULO" phpName="Titulo" type="VARCHAR" size="100" required="true"/>
    <column name="LOCAL" phpName="Local" type="VARCHAR" size="100" required="false"/>
    <column name="DESCRICAO" phpName="Descricao" type="VARCHAR" size="400" required="false"/>
    <column name="RECORRENCIA_CONTAGEM" phpName="RecorrenciaContagem" type="INTEGER" required="true"/>
    <column name="PERIODO_FREQUENCIA" phpName="PeriodoFrequencia" type="INTEGER" required="true" defaultVal
    <column name="EVENTO_PERIODO_ID" phpName="EventoPeriodoId" type="INTEGER" size="10" sqlType="int(10) un
    <column name="IS_USAR_DATA_TERMINO" phpName="IsUsarDataTermino" type="BOOLEAN" size="1" required="false
    <column name="IS_EVENTO_SISTEMA" phpName="IsEventoSistema" type="BOOLEAN" size="1" required="false" def
    <column name="EVENTO_CATEGORIA_ID" phpName="EventoCategoriaId" type="INTEGER" size="10" sqlType="int(10
    <column name="PACIENTE_ID" phpName="PacienteId" type="INTEGER" size="10" sqlType="int(10) unsigned" req
    <column name="CONTA_ID" phpName="ContaId" type="INTEGER" size="10" sqlType="int(10) unsigned" required=
    <column name="USUARIO_ID" phpName="UsuarioId" type="INTEGER" size="10" sqlType="int(10) unsigned" requi
    <column name="DATA_CRIACAO" phpName="DataCriacao" type="TIMESTAMP" required="false"/>
    <column name="DATA_ATUALIZACAO" phpName="DataAtualizacao" type="TIMESTAMP" required="false"/>

    <validator column="TITULO">
        <rule name="required" message="O campo Título é obrigatório."/>
    </validator>
    <validator column="DATA_INICIO">
        <rule name="required" message="O campo Data de Início é obrigatório."/>
    </validator>
    <validator column="DATA_TERMINO">
        <rule name="required" message="O campo Data de Término é obrigatório."/>
    </validator>
    <validator column="PERIODO_FREQUENCIA">
```

3.3.1.3 Padrões de projeto

De modo a criar um sistema robusto e de alto desempenho, foram utilizados alguns padrões de projeto, dentre os quais destacam-se:

a) o *Active Record*, que permite ao Propel ORM manter a persistência das consultas e salvar os objetos através de uma interface orientada a objetos;
b) o *Lazy Initialization* que permite ao Propel ORM buscar apenas as informações desejadas no momento da requisição, consumindo menos recursos. Além disso, o *Lazy Initialization* foi utilizado também para buscar as configurações do usuário, sendo que os parâmetros vão sendo carregados à medida que são solicitados;

c) o *Dependency Injection* é utilizado no Silex para interligar os diversos componentes existentes de uma maneira organizada e mantendo o reaproveitamento de código. Como exemplo, na Figura 17 é possível verificar que através do método *share*() o componente *Twig* ficará disponível para utilização em toda a aplicação;

d) o MVC é utilizado para organizar a aplicação em três camadas, sendo que o Silex fica responsável pela lógica de negócio na camada de controle e pela exibição das informações na camada de visualização, enquanto o Propel ORM é utilizado para controlar a camada de modelo.

Figura 17 - Exemplo de *Dependency Injection*

```
class TwigServiceProvider implements ServiceProviderInterface
{
    public function register(Application $app)
    {
        $app['twig.options'] = array();
        $app['twig.form.templates'] = array('form_div_layout.html.twig');
        $app['twig.path'] = array();
        $app['twig.templates'] = array();

        $app['twig'] = $app->share(function ($app) {
            $app['twig.options'] = array_replace(
                array(
                    'charset'          => $app['charset'],
                    'debug'            => $app['debug'],
                    'strict_variables' => $app['debug'],
                ), $app['twig.options']
            );

            $twig = new \Twig_Environment($app['twig.loader'], $app['twig.options']);
            $twig->addGlobal('app', $app);
            $twig->addExtension(new TwigCoreExtension());

            if ($app['debug']) {
                $twig->addExtension(new \Twig_Extension_Debug());
            }
```

3.3.1.4 Técnicas de prevenção de vulnerabilidades

No desenvolvimento de sistemas para *web* alguns cuidados com vulnerabilidades conhecidas precisam ser tomados para protegê-las contra ataques. Por isso, as seguintes implementações foram feitas:

a) tratar todas as informações exibidas na camada de visualização de modo a evitar o XSS *Script*. Conforme é exibido na Figura 18, um código *JavaScript* malicioso foi inserido no nome de um paciente, porém, como havia tratamento, o ataque não teve êxito;

b) evitar inserção de códigos maliciosos em comandos SQL, em um tipo de vulnerabilidade conhecida como SQL *Injection*. O Propel ORM utiliza uma forma de parametrizar todas as informações externas enviadas ao banco de dados, permitindo desta forma impedir a utilização da técnica de SQL *Injection*;
c) impedir o sequestro de sessões ou *Session Hijacking*. Tendo como exemplo a Figura 18, caso o atacante conseguisse roubar o *cookie* do usuário autenticado, ele poderia ter total acesso a conta do usuário de uma máquina remota. Para evitar isso, existe um parâmetro chamado *httponly* que permite que o *cookie* seja acessado apenas pela máquina do próprio usuário, não ficando disponível para acesso em máquinas remotas. Desta forma, as chances de acontecer o sequestro de sessão diminuem consideravelmente. Na Figura 19 é exibido a saída do comando *session_get_cookie_params* do PHP, mostrando que o parâmetro *httponly* está ativo.

Figura 18 - Exemplo de ataque utilizando XSS Script

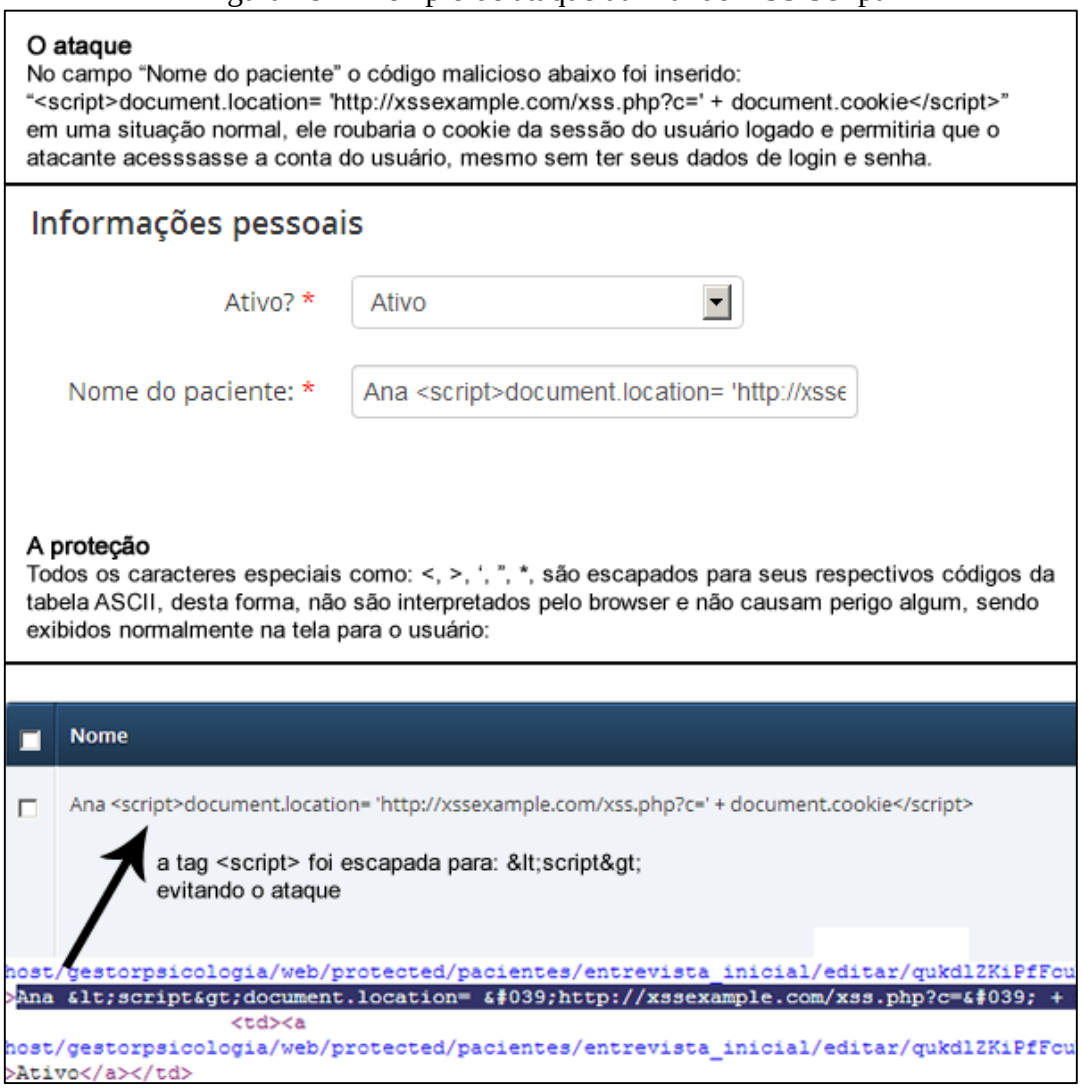


Figura 19 - Comprovação de que o parâmetro *httponly* está ativo

3.3.1.5 Ferramentas para layout responsivo

De modo a permitir o desenvolvimento de uma aplicação que se ajuste aos diversos dispositivos móveis, foi utilizado um recurso conhecido como *media queries* que é aplicado na estilização das páginas da camada de visualização, permitindo configurar que na resolução de um *tablet*, por exemplo, determinados recursos sejam ocultados ou tenham tamanhos

diferentes. Além disso, foi utilizada a biblioteca *Twitter Bootstrap* que permite criar o *layout* baseado em colunas, desta forma, todos os elementos da página *web* ajustam sua largura de acordo com a resolução da tela do dispositivo. Na Figura 20 tem-se um exemplo de configurações específicas dos elementos para resoluções de até 767 *pixels*. Já na Figura 21 é possível visualizar um elemento HTML que possui a classe span8, fazendo com que a largura deste elemento seja proporcional a oito colunas independentemente da resolução da tela do usuário. Através das Figuras 22 e 23 pode-se visualizar a adaptação nas diversas resoluções de dispositivos móveis.

Figura 20 - Exemplo de configurações específicas para resoluções de tela de até 767 *pixels*

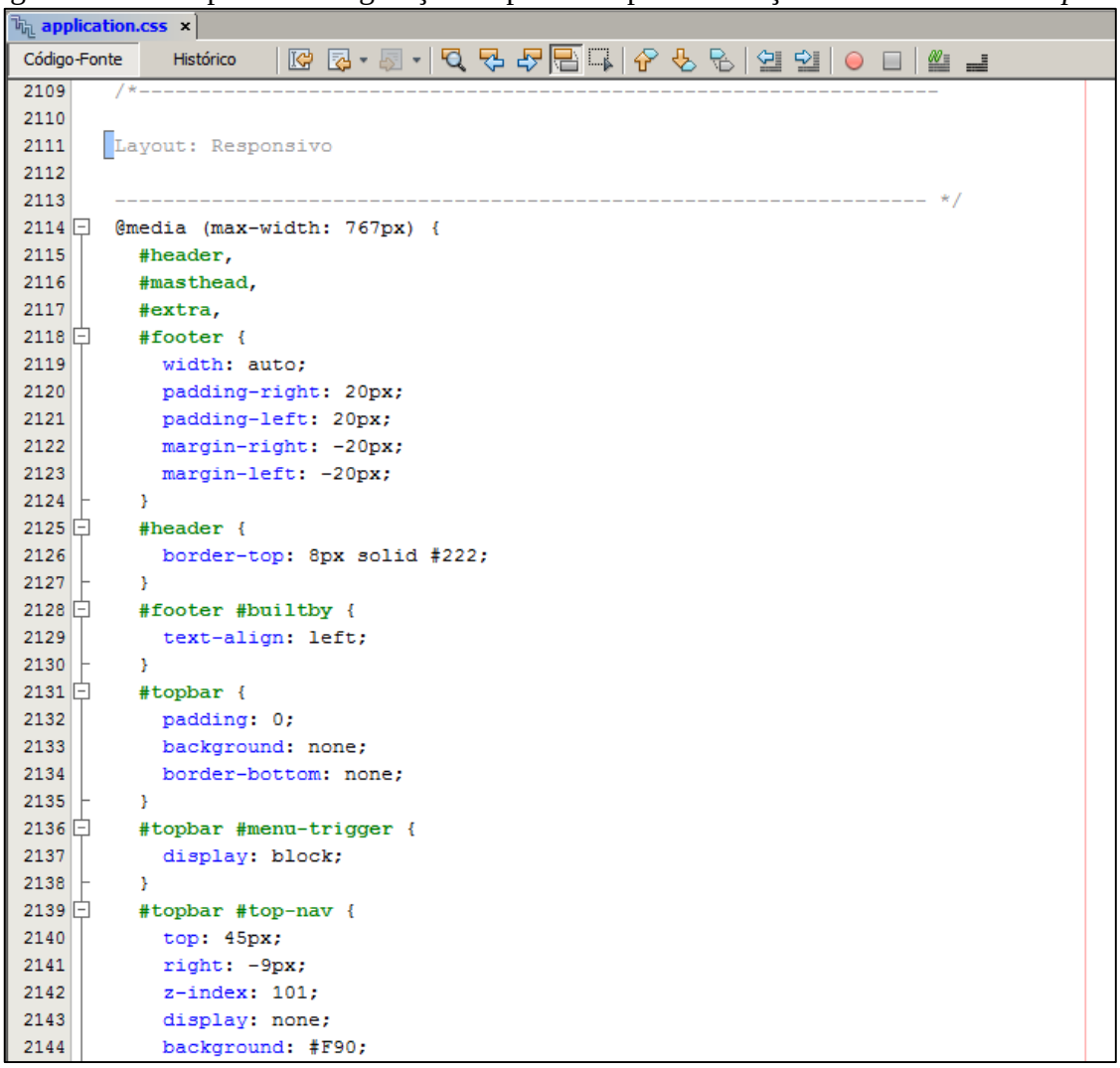
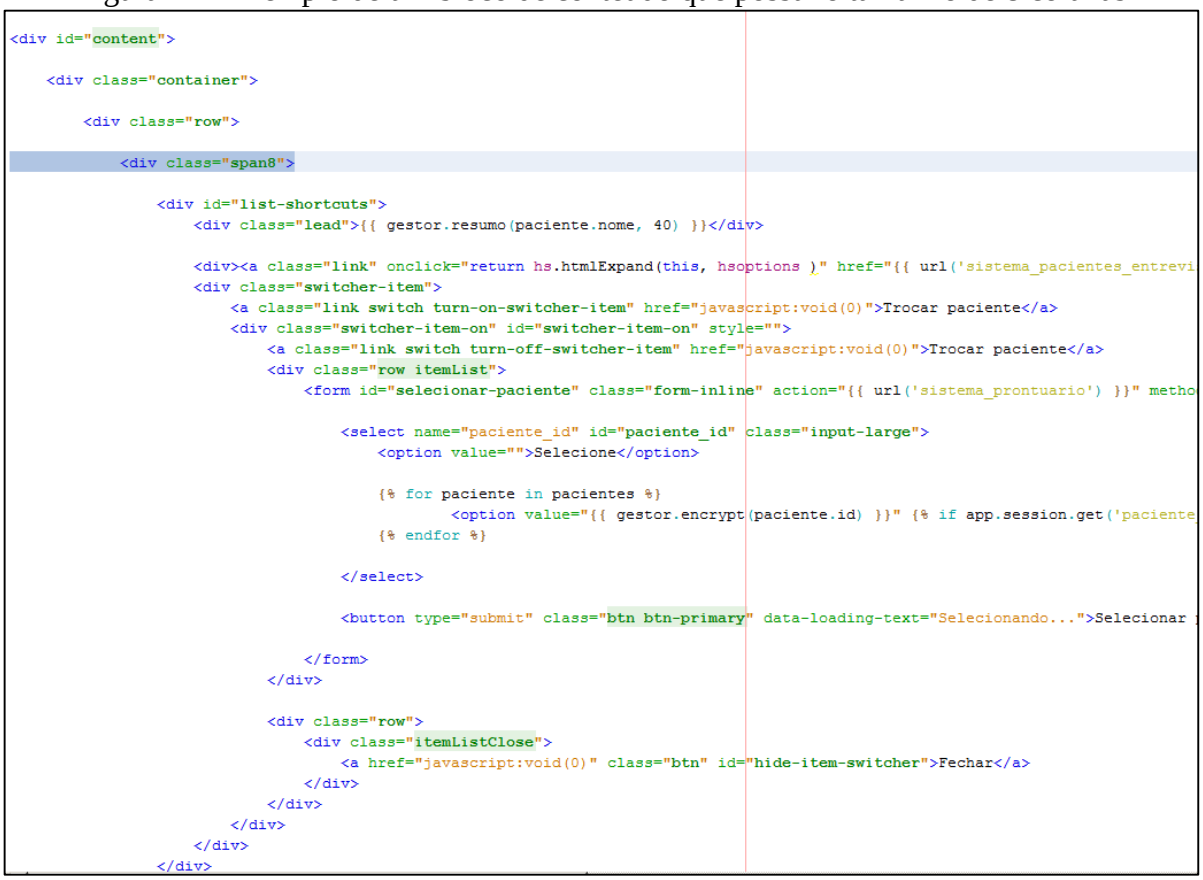


Figura 21 - Exemplo de um bloco de conteúdo que possui o tamanho de 8 colunas



```
<div id="content">

    <div class="container">

        <div class="row">

            <div class="span8">

                <div id="list-shortcuts">
                    <div class="lead">{{ gestor.resumo(paciente.nome, 40) }}</div>

                    <div><a class="link" onclick="return hs.htmlExpand(this, hsoptions )" href="{{ url('sistema_pacientes_entrevi
                    <div class="switcher-item">
                        <a class="link switch turn-on-switcher-item" href="javascript:void(0)">Trocar paciente</a>
                        <div class="switcher-item-on" id="switcher-item-on" style="">
                            <a class="link switch turn-off-switcher-item" href="javascript:void(0)">Trocar paciente</a>
                            <div class="row itemList">
                                <form id="selecionar-paciente" class="form-inline" action="{{ url('sistema_prontuario') }}" metho

                                    <select name="paciente_id" id="paciente_id" class="input-large">
                                        <option value="">Selecione</option>

                                        {% for paciente in pacientes %}
                                            <option value="{{ gestor.encrypt(paciente.id) }}" {% if app.session.get('paciente
                                        {% endfor %}

                                    </select>

                                    <button type="submit" class="btn btn-primary" data-loading-text="Selecionando...">Selecionar

                                </form>
                            </div>

                            <div class="row">
                                <div class="itemListClose">
                                    <a href="javascript:void(0)" class="btn" id="hide-item-switcher">Fechar</a>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
```

Figura 22 - Exemplo de visualização em um *tablet*

Figura 23 - Exemplo de utilização em um celular

3.3.2 Operacionalidade da implementação

Nesta subseção apresentam-se as principais telas do *site* e do sistema com uma apresentação sobre suas funcionalidades, bem como trechos de códigos relevantes para o entendimento de algumas funcionalidades.

Primeiramente tem-se a tela inicial do *site*, onde os visitantes poderão acessar para conhecer melhor as funcionalidades do Gestor Psicologia, conforme a Figura 24.

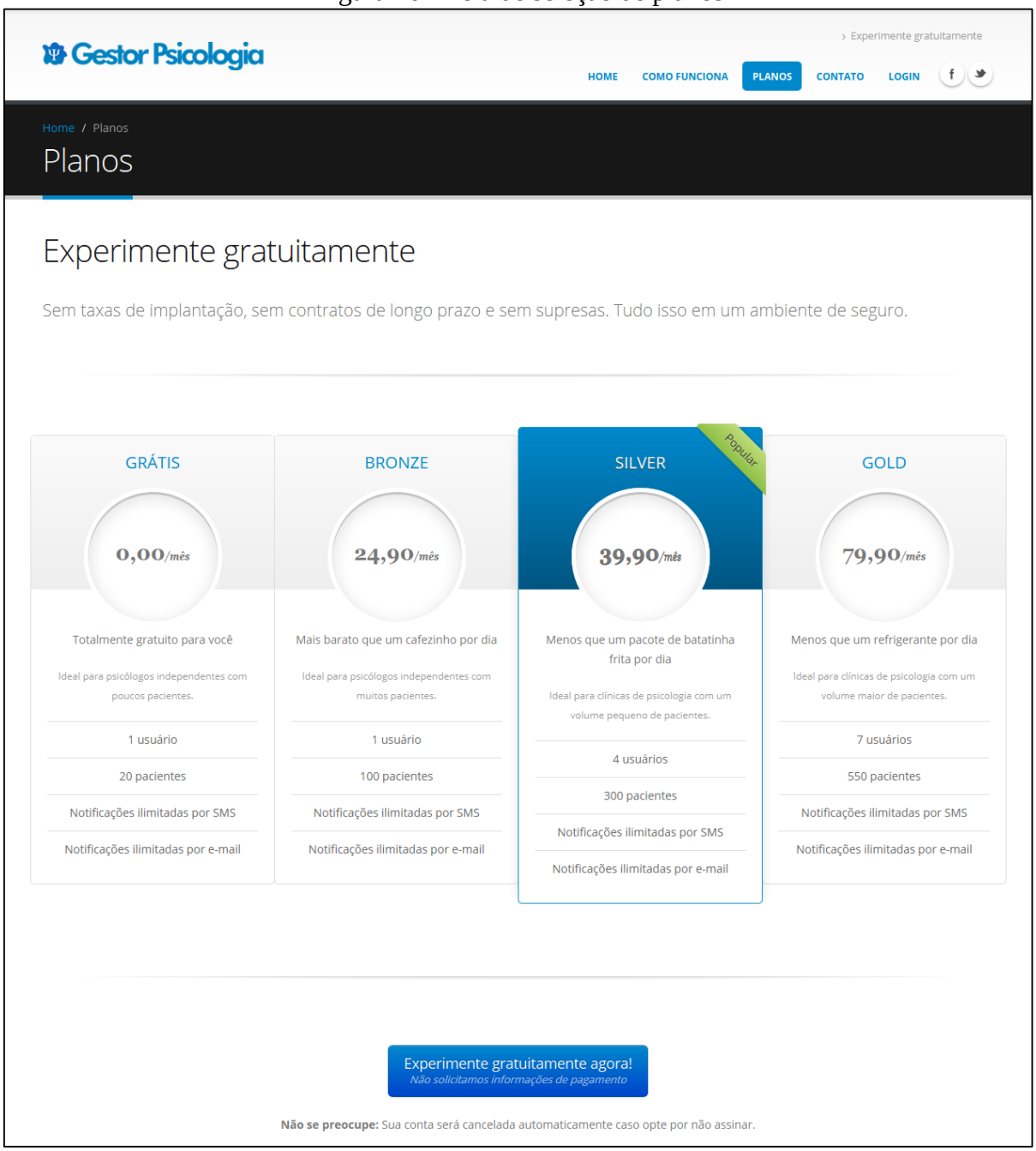
Figura 24 - Tela inicial do *site*

Ainda no *site* é possível acessar a tela de como funciona, onde o visitante encontrará todos os recursos do sistema explicados de forma detalhada. Na Figura 25 apresenta-se a tela de como funciona do *site*.

Figura 25 - Tela de como funciona do *site*

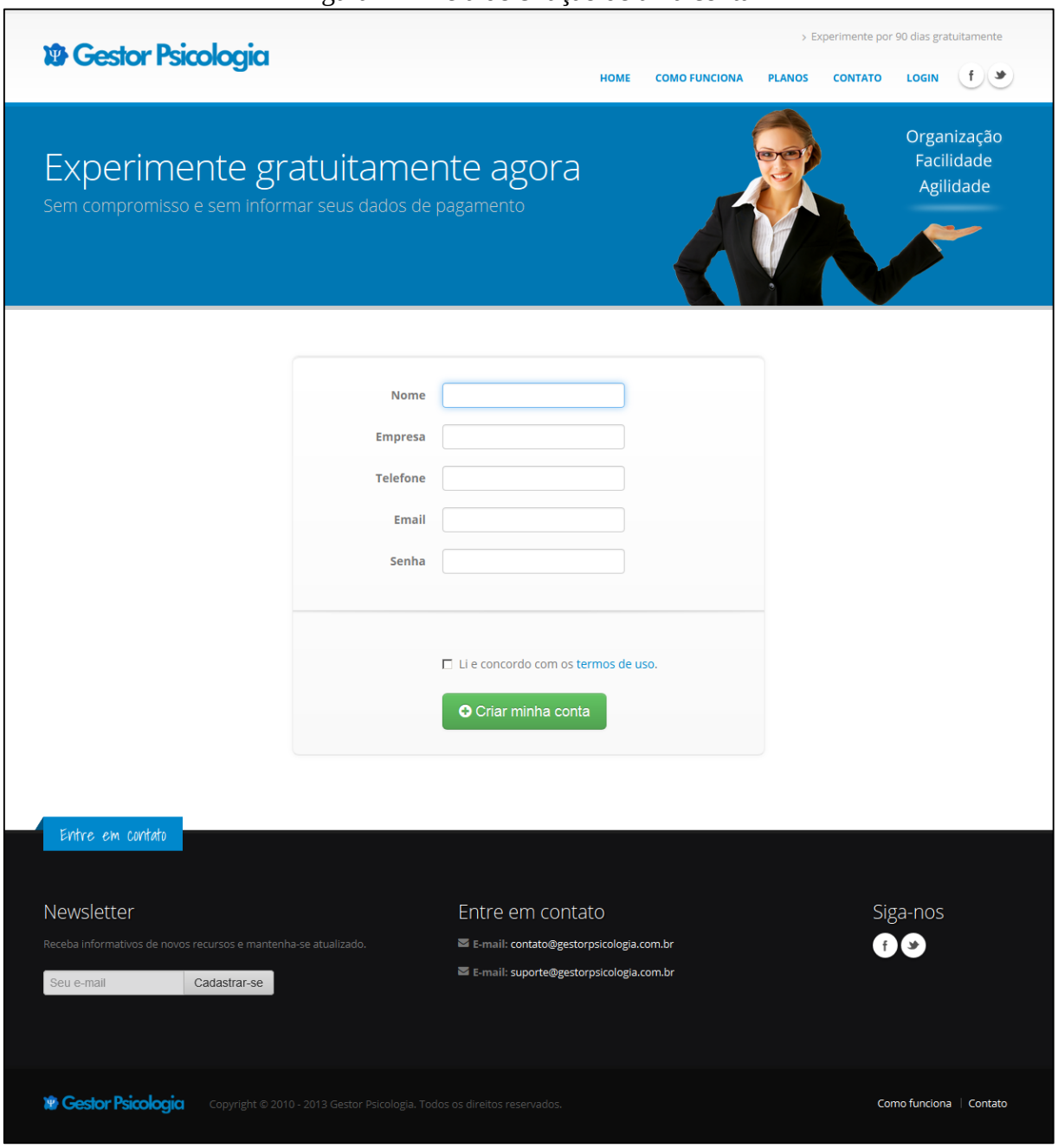
Através da Figura 26 que mostra a tela de seleção de planos do *site*, o visitante poderá optar por testar o sistema gratuitamente por um período, atendendo o requisito funcional RF01.

Figura 26 - Tela de seleção de planos



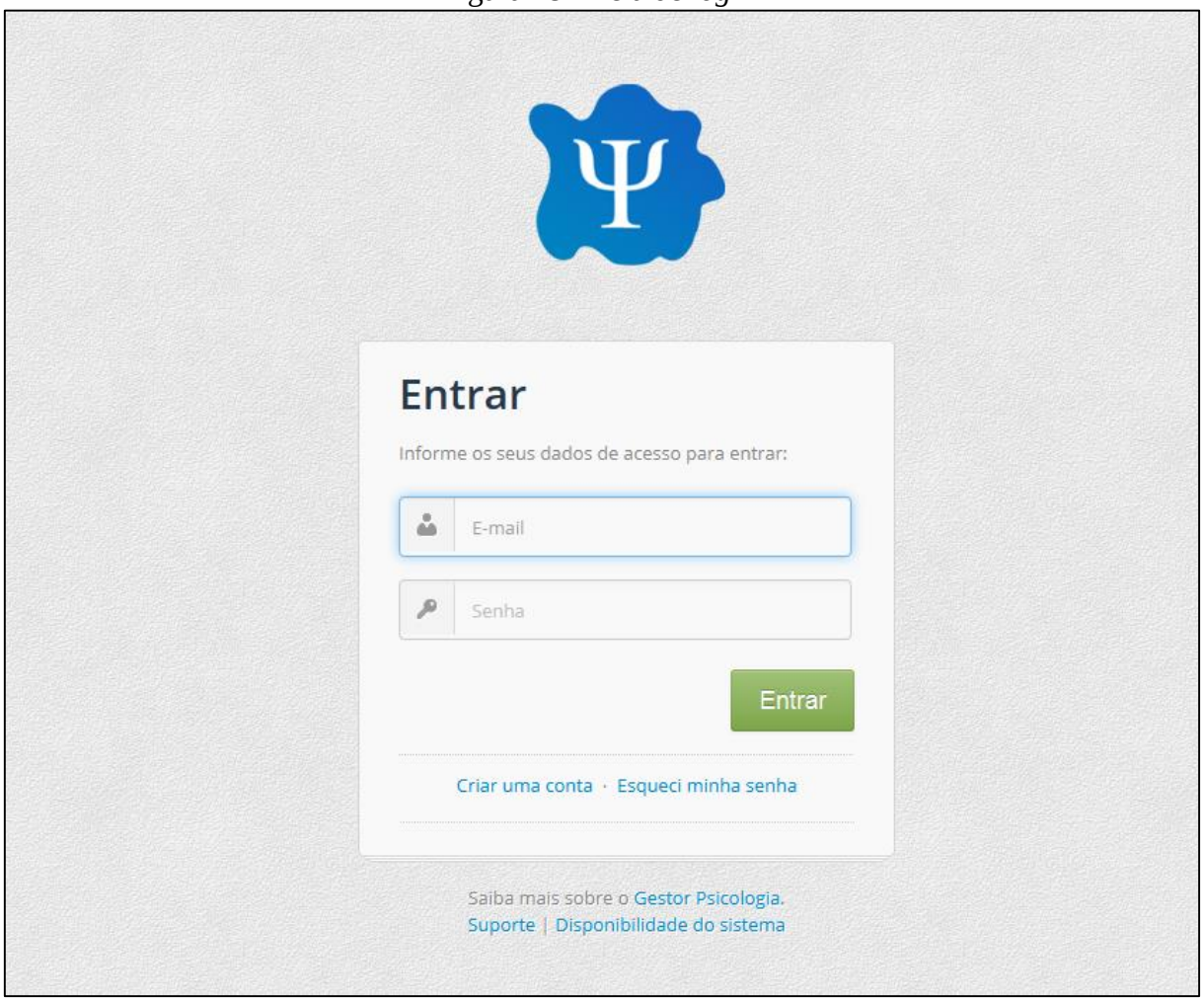
Após o visitante clicar no botão “Experimente gratuitamente agora” da Figura 26, então é exibida a tela de criação de uma conta conforme demonstração na Figura 27, atendendo o requisito funcional RF02.

Figura 27 - Tela de criação de uma conta



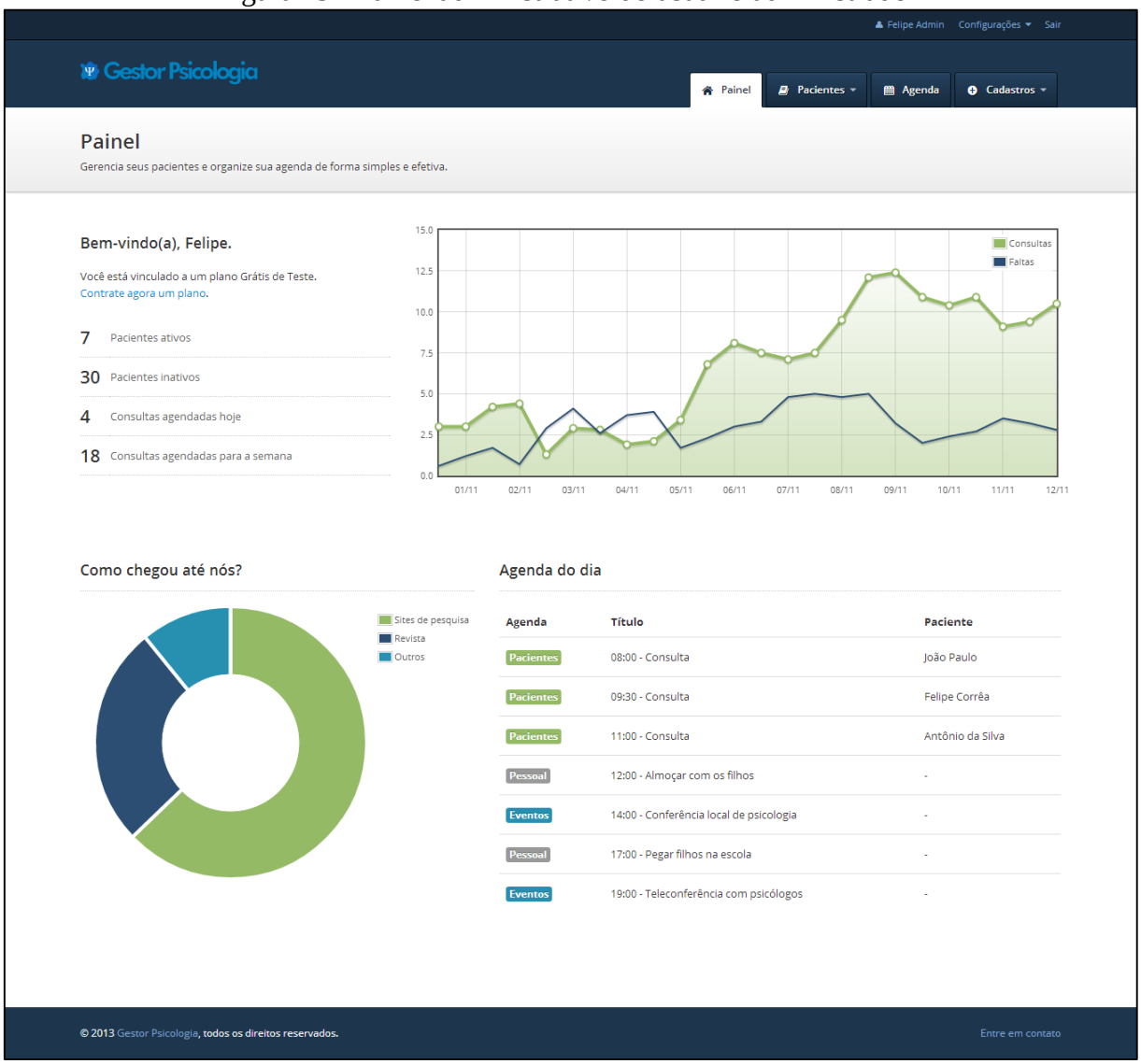
Assim que a criação da conta é finalizada, o usuário pode fazer o *login* para acessar o sistema. Cada usuário possuirá restrições de recursos com base no plano escolhido. Na Figura 28 é exibida a tela de *login* atendendo ao requisito funcional RF03.

Figura 28 - Tela de *login*



Após efetuar a entrada no sistema, é apresentado um painel onde são exibidos informações e gráficos relevantes para o psicólogo. Na Figura 29 tem-se o painel administrativo de um usuário administrador, contendo informações estatísticas de suas consultas, como os pacientes chegaram até o psicólogo e também a agenda de compromissos do dia. Neste ponto, permitindo a contratação de planos, a criação de contas e o *login* através da internet, atende-se o segundo objetivo específico do sistema, distribuindo-se o software como serviço no modelo SaaS.

Figura 29 - Painel administrativo do usuário administrador



As Figuras 30 e 31 atendem ao requisito funcional RF05 de que um usuário poderá administrar os pacientes. Na Figura 30 é possível visualizar em forma de lista todos os pacientes do usuário que está utilizando o sistema. Já na Figura 31 tem-se o formulário de entrevista inicial, que é o meio pelo qual um novo paciente é cadastrado ou editado.

Figura 30 - Tela de listagem de pacientes do sistema

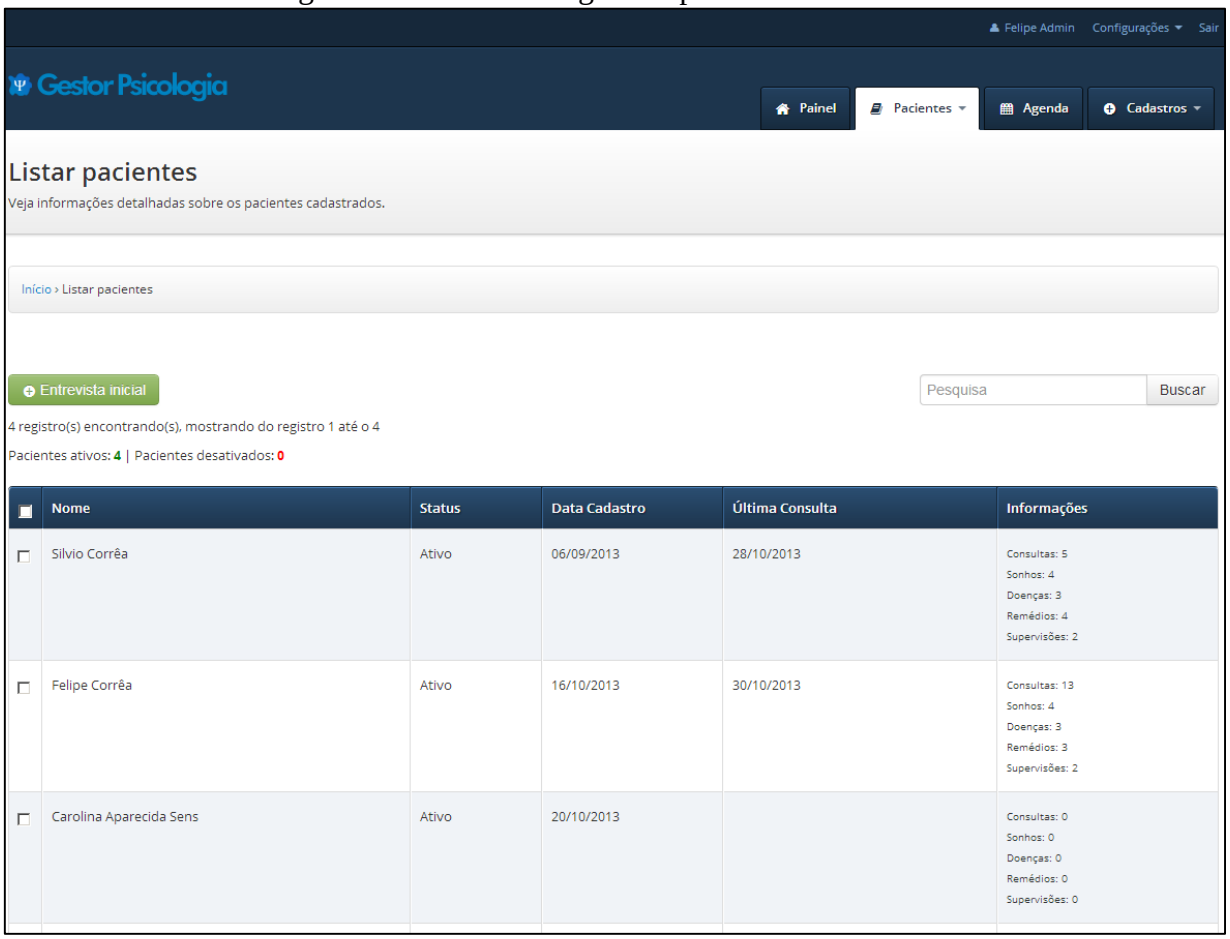
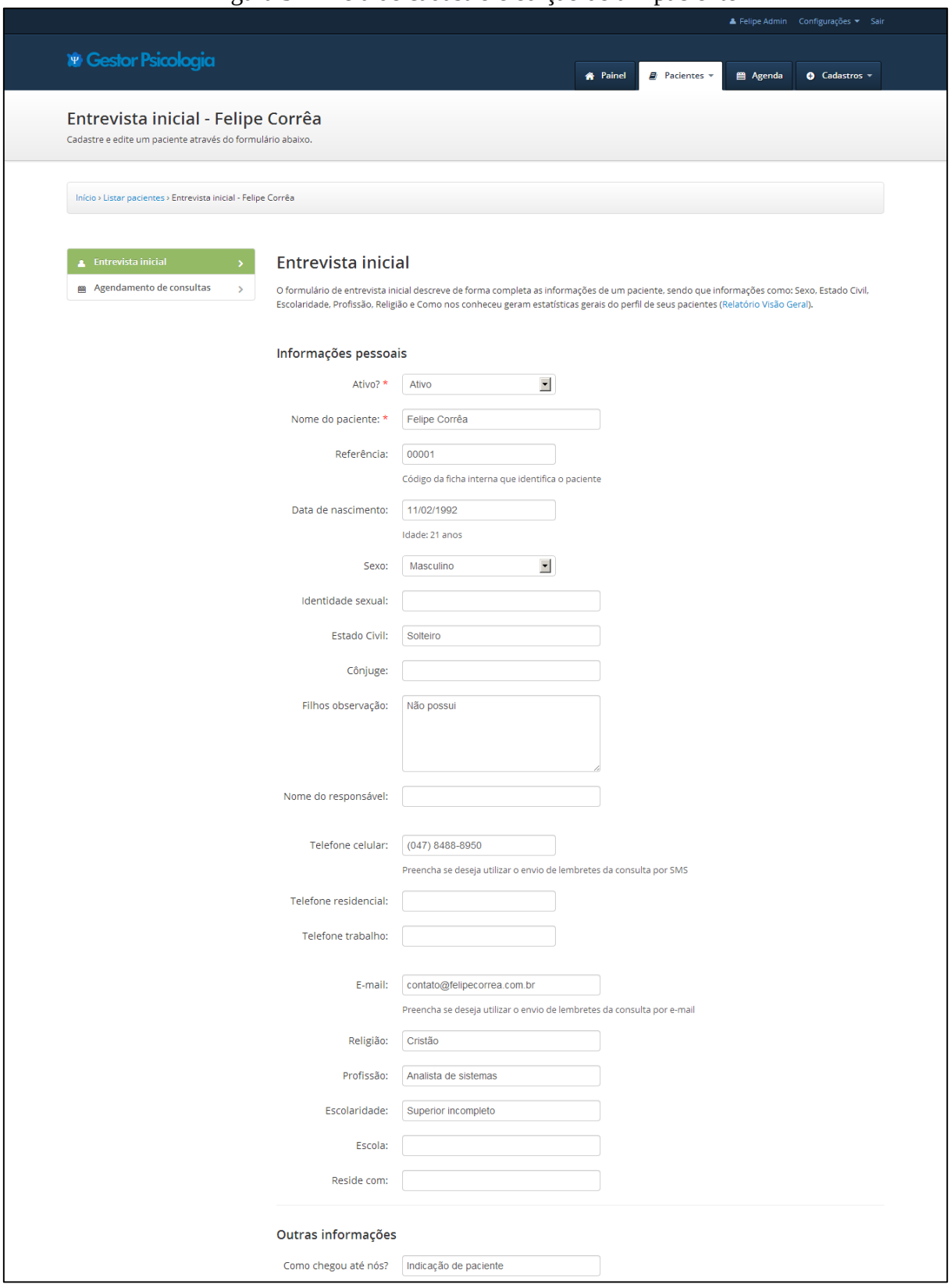


Figura 31 - Tela de cadastro e edição de um paciente



A Figura 32 mostra o trecho de código para validar e salvar as informações vindas do formulário de Entrevista inicial, desta forma, os dados são recebidos do formulário, depois

são inseridos em um objetivo do tipo paciente. As demais informações normalizadas de Como nos conheceu, Encaminhado Por, Escolaridade, Profissão, Estado Civil e Religião são salvas e vinculadas ao objeto paciente, ao final, é utilizado o método *validateWithExternalErros* para verificar no *framework* Propel ORM se o objeto paciente ou alguma das informações normalizadas possuem erros, caso tudo esteja correto, então os dados são persistidos no banco de dados, é criada uma mensagem amigável de sucesso para o usuário e o evento de cadastro ou atualização é salvo no log de eventos.

Figura 32 - Trecho de código da camada de controle da página entrevista inicial

```
94
95          if ($request->isMethod('POST'))
96          {
97              $form->bind($request);
98
99              if ($form->isValid() && empty($erros))
100             {
101                 $arrForm = $form->getData();
102
103                 try {
104                     $objeto = isset($objeto) ? $objeto : new Paciente;
105                     $objeto->setByArray($form->getData());
106                     $objeto->setContaId(_conta_id());
107                     $objeto->setUsuario(_usuario());
108                     $objeto->setEmpresaId(_empresa_id());
109
110                     // Vinculando opções informadas através de texto (busca os respectivos objetos ou
111                     $objeto->setPacienteComoConheceu(PacientePeer::vincularOpcao('PacienteComoConheceu
112                     $objeto->setPacienteEncaminhadoPor(PacientePeer::vincularOpcao('PacienteEncaminhad
113                     $objeto->setPacienteEscolaridade(PacientePeer::vincularOpcao('PacienteEscolaridade
114                     $objeto->setPacienteProfissao(PacientePeer::vincularOpcao('PacienteProfissao', $ar
115                     $objeto->setPacienteEstadoCivil(PacientePeer::vincularOpcao('PacienteEstadoCivil',
116                     $objeto->setPacienteReligiao(PacientePeer::vincularOpcao('PacienteReligiao', $arrF
117
118                     if ($objeto->validateWithExternalErrors($erros))
119                     {
120                         $objeto->save();
121                         $app['session']->getFlashBag()->add('success', "Paciente '{$objeto->getNome()}
122
123                         if ($acao == 'cadastrar')
124                             LogEventoPeer::gravarEvento(LogTipoEventoPeer::TIPO_CADASTRAR, 11, 89, "Id
125                         else
126                             LogEventoPeer::gravarEvento(LogTipoEventoPeer::TIPO_ATUALIZAR, 11, 31, "Id
127
128                         return $app->redirect($app['url_generator']->generate('sistema_pacientes_entre
129                     }
130                 }
                    catch (\Exception $e)
132                 {
133                     $app['session']->getFlashBag()->add('error', "Oops! Parece que um erro grave acont
134                 }
135             }
136
```

No prontuário de um paciente é possível editar as informações do mesmo, administrar consultas, sonhos, doenças, remédios e supervisões. Nas Figuras 33, 34 e 35 mostra-se a tela de prontuário de um paciente, além da listagem e edição de um sonho. Desta forma, os seguintes requisitos funcionais foram atendidos: RF06, RF07, RF08, RF09, RF10.

Figura 33 - Tela de prontuário de um paciente

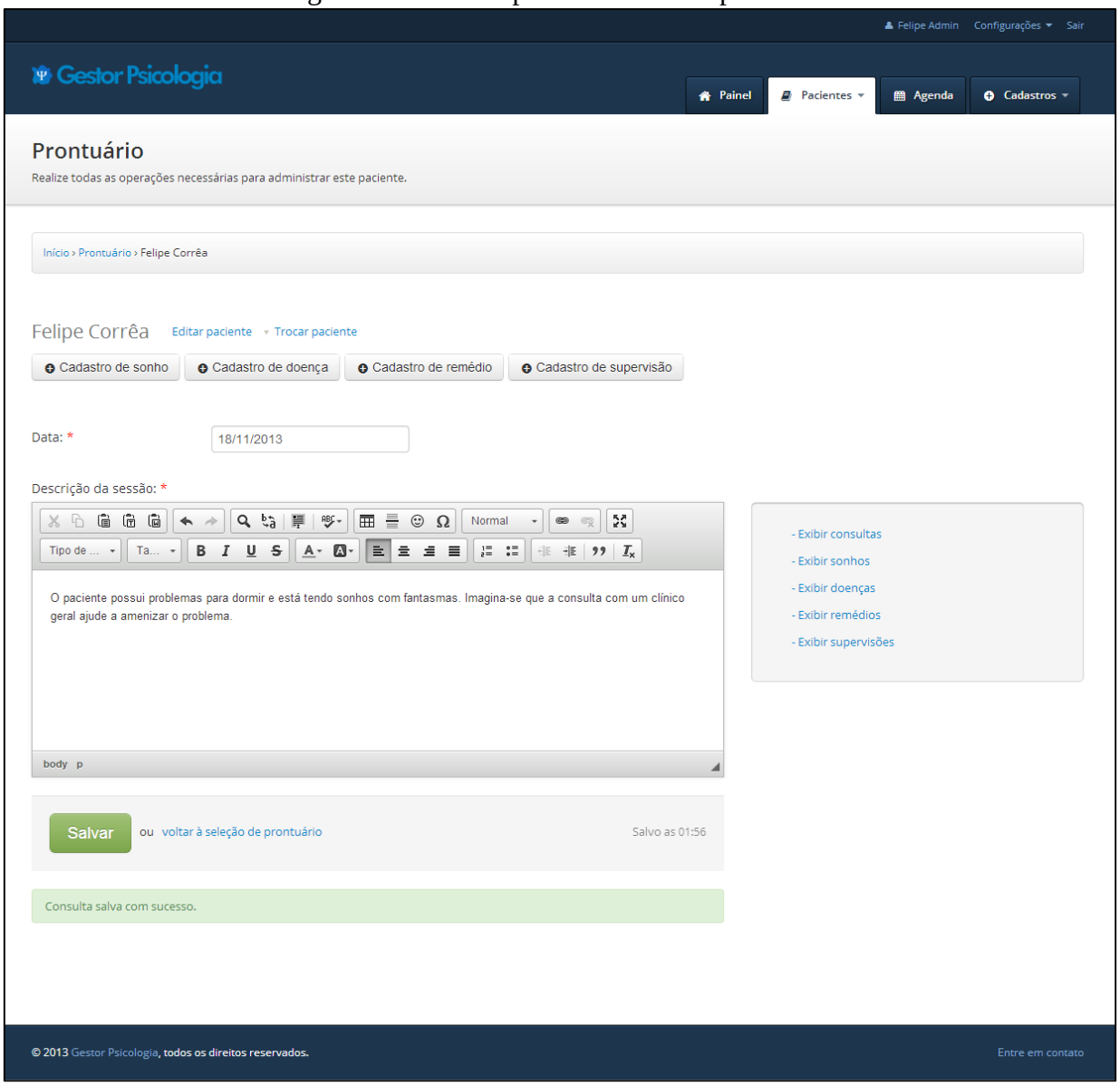


Figura 34 - Tela de listagem de sonhos no prontuário

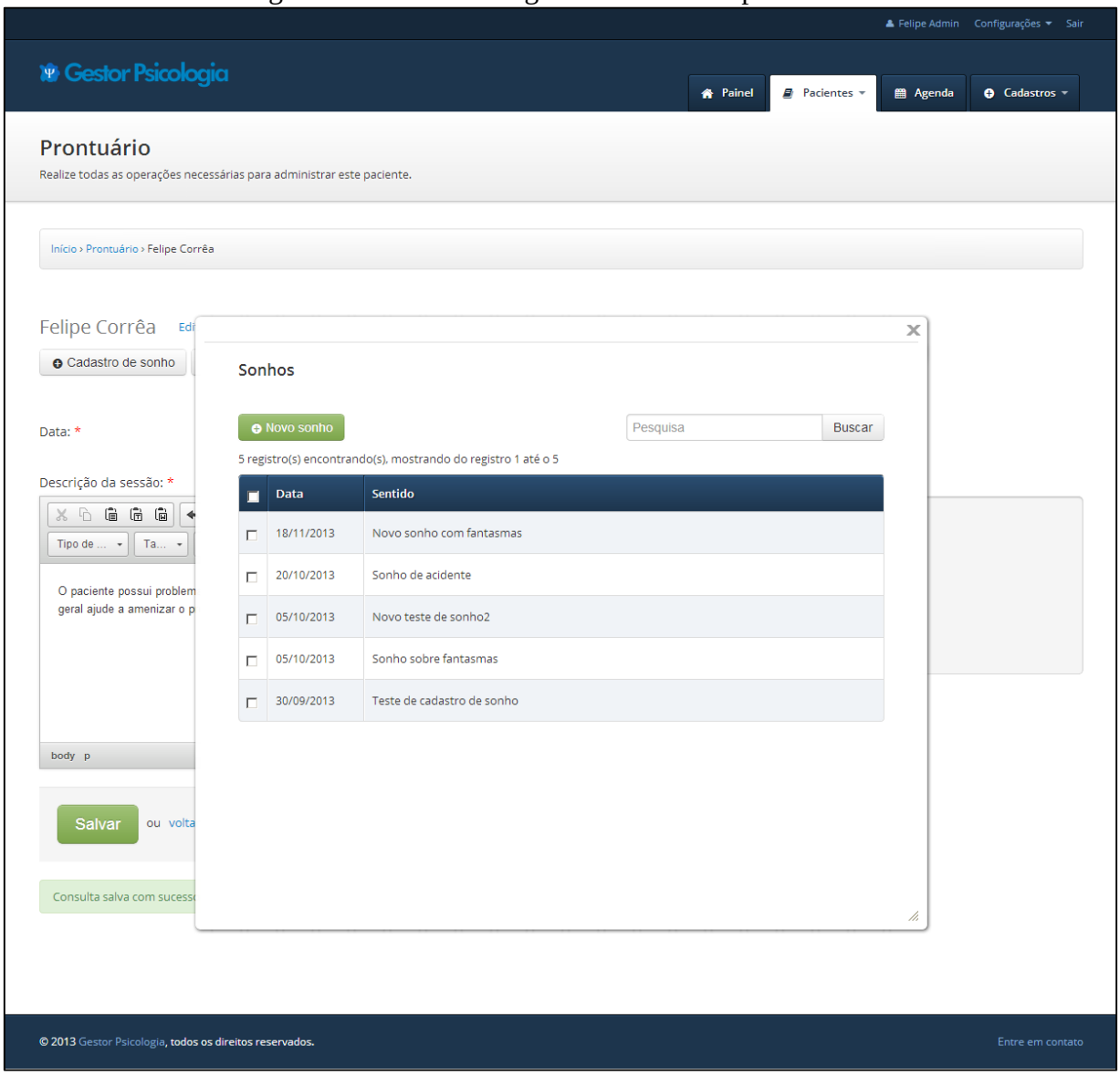
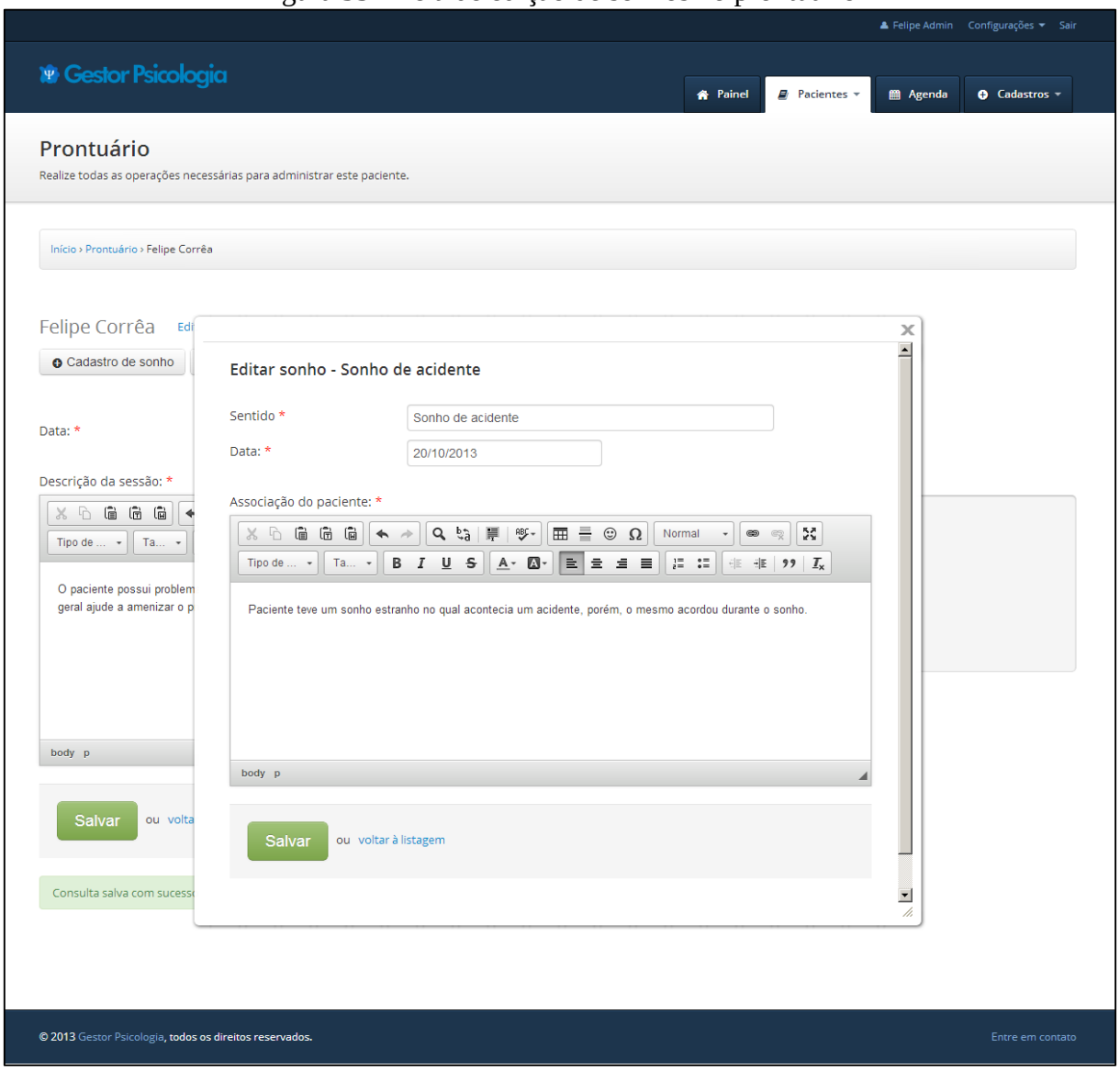


Figura 35 - Tela de edição de sonhos no prontuário



As Figuras 36 e 37 demonstram que o requisito funcional RF11 foi atendido, permitindo ao usuário visualizar as informações de um paciente através de um relatório. Na Figura 36 configura-se o relatório das informações que serão exibidas e na Figura 37 mostra-se os resultados solicitados.

Figura 36 - Configuração do relatório por paciente

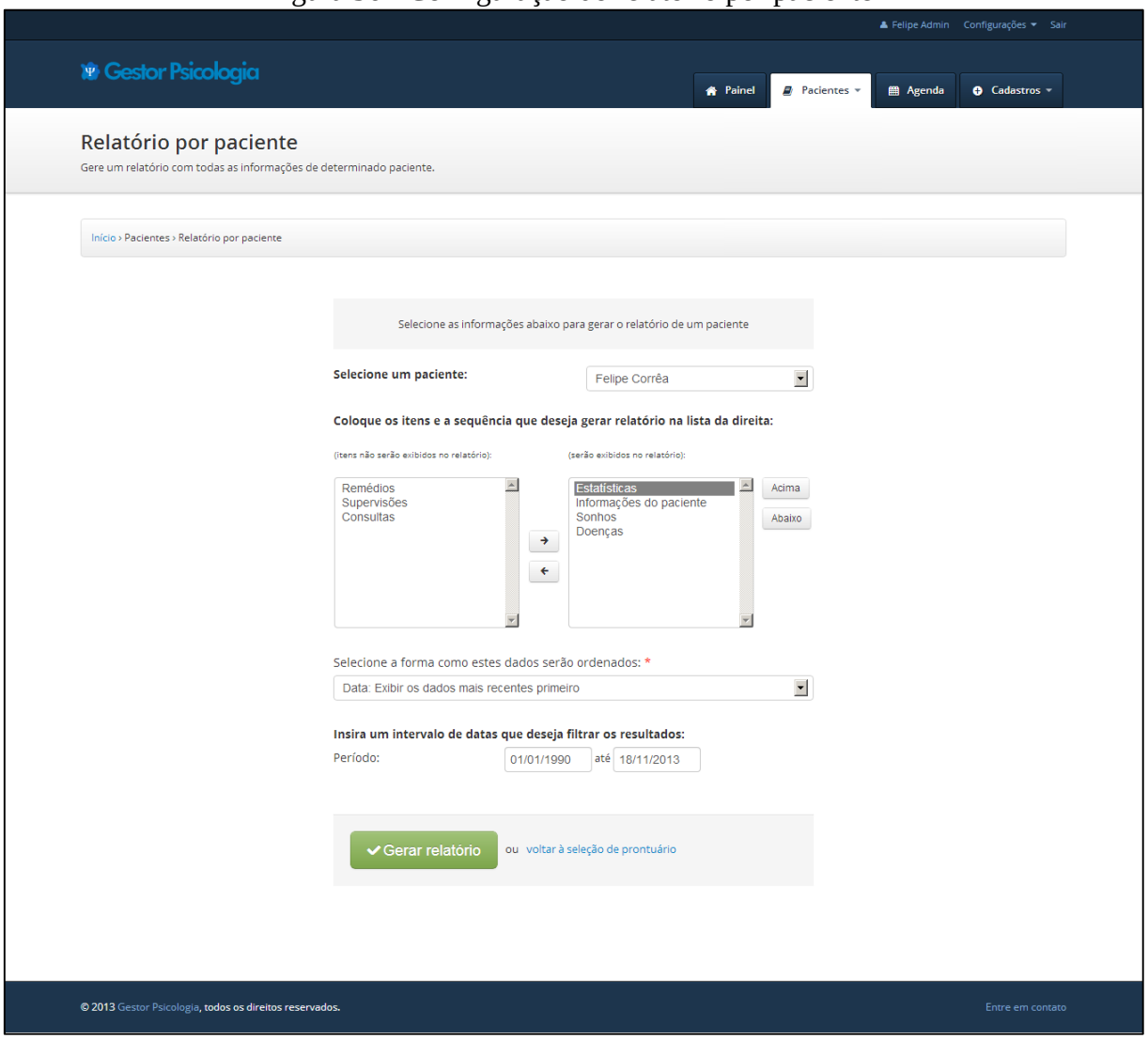
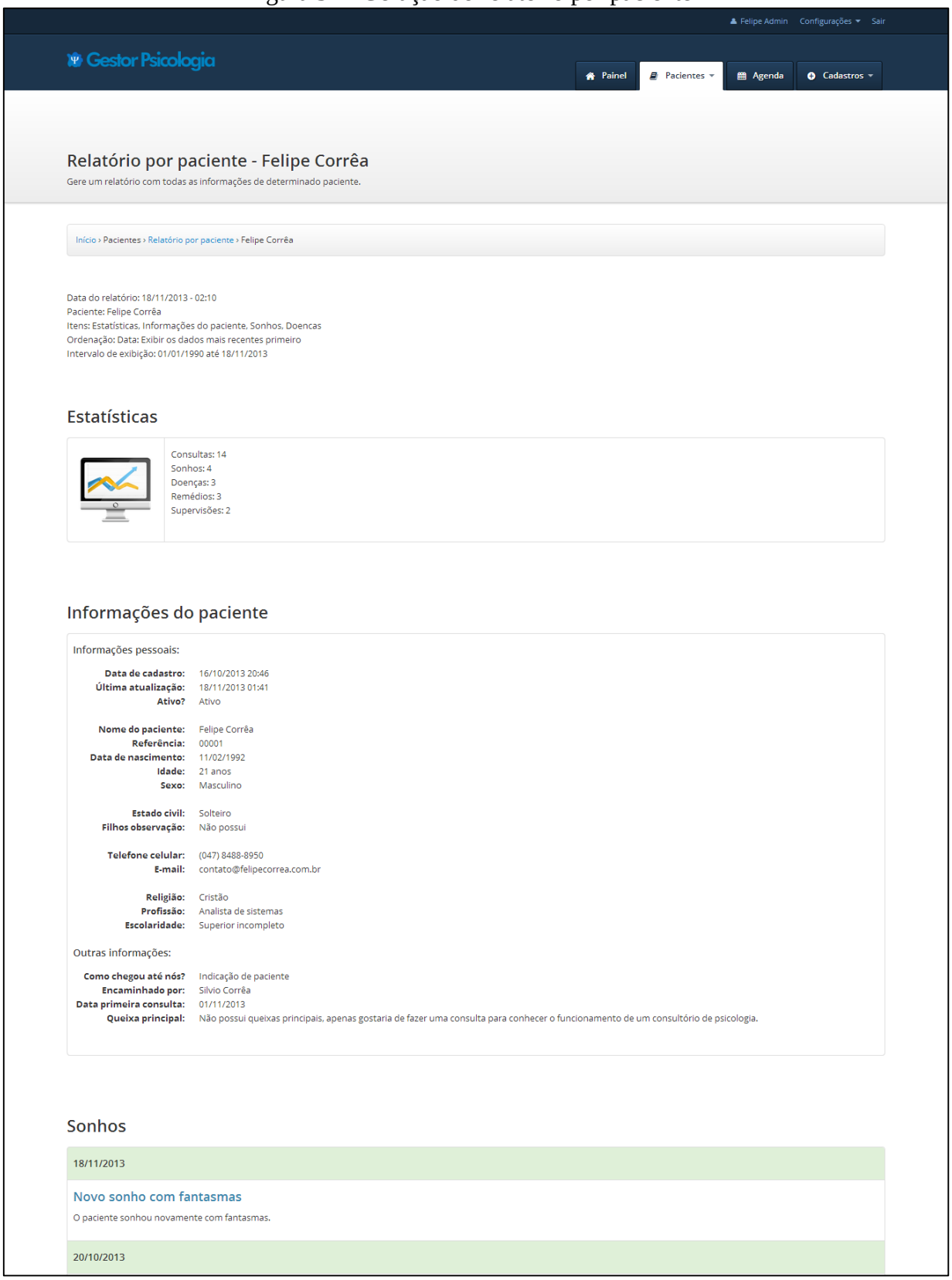


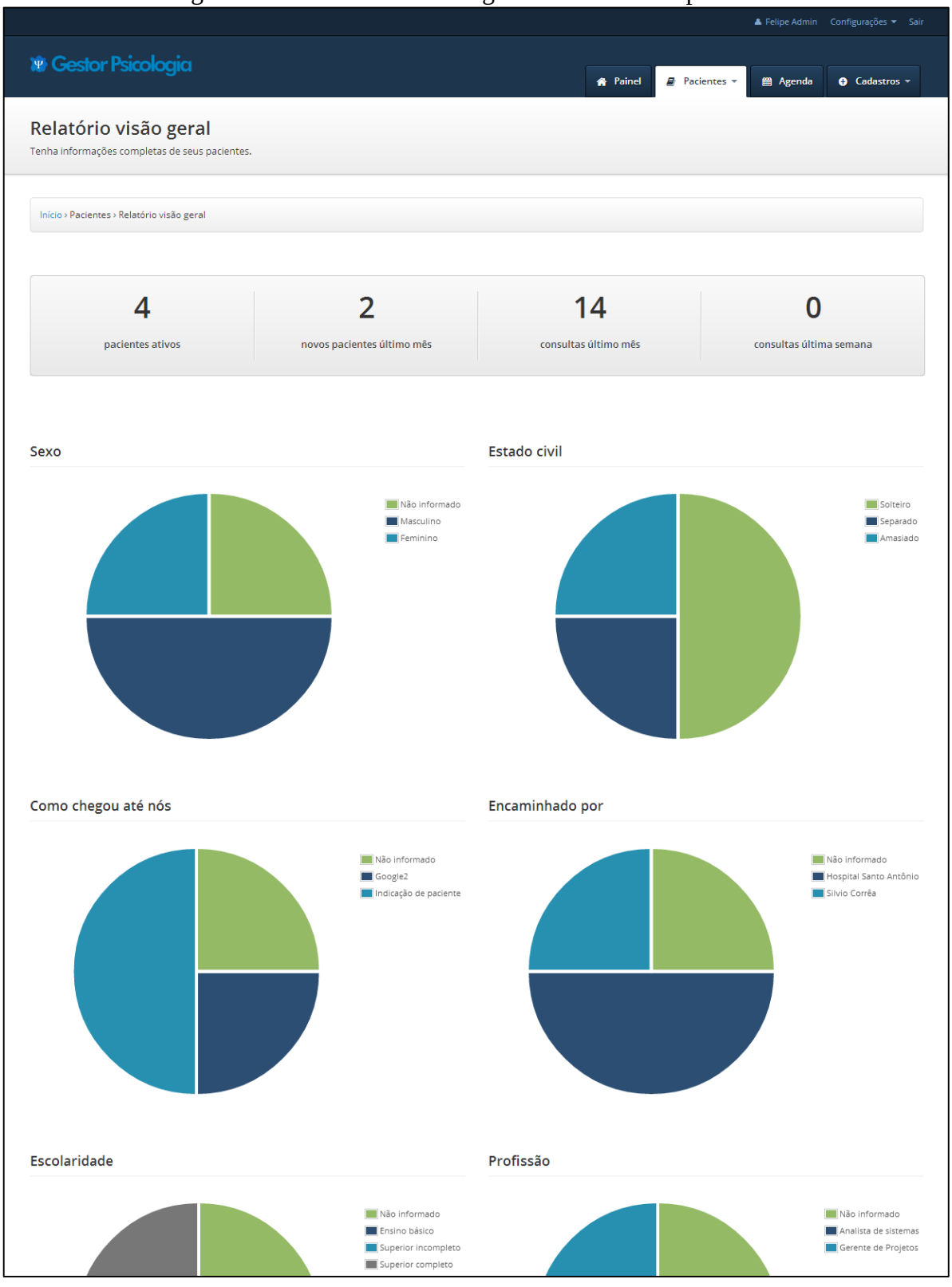
Figura 37 - Geração do relatório por paciente



No relatório de visão geral do sistema apresentado através da Figura 38, é possível visualizar de forma gráfica informações estatísticas sobre os pacientes no que tange o sexo,

estado civil, como chegou até nós, encaminhado por, escolaridade, profissão e religião. Isso é possível através do cadastro normalizado destas informações entre os pacientes de uma conta. Através da Figura 38 atende-se o requisito funcional RF12.

Figura 38 - Relatório de visão geral do módulo de pacientes



No módulo de agenda é possível visualizar em um calendário todos os eventos cadastrados pelo usuário, podendo alternar entre os modos de visualização por mês, por semana e por dia. Na Figura 39 é exibido o calendário no modo de visualização por mês, já na Figura 40 o modo de exibição foi modificado para mostrar a semana. Através das Figuras 39 e 40 atende-se ao requisito funcional RF13.

Figura 39 - Módulo de agenda mostrando todos os eventos do mês

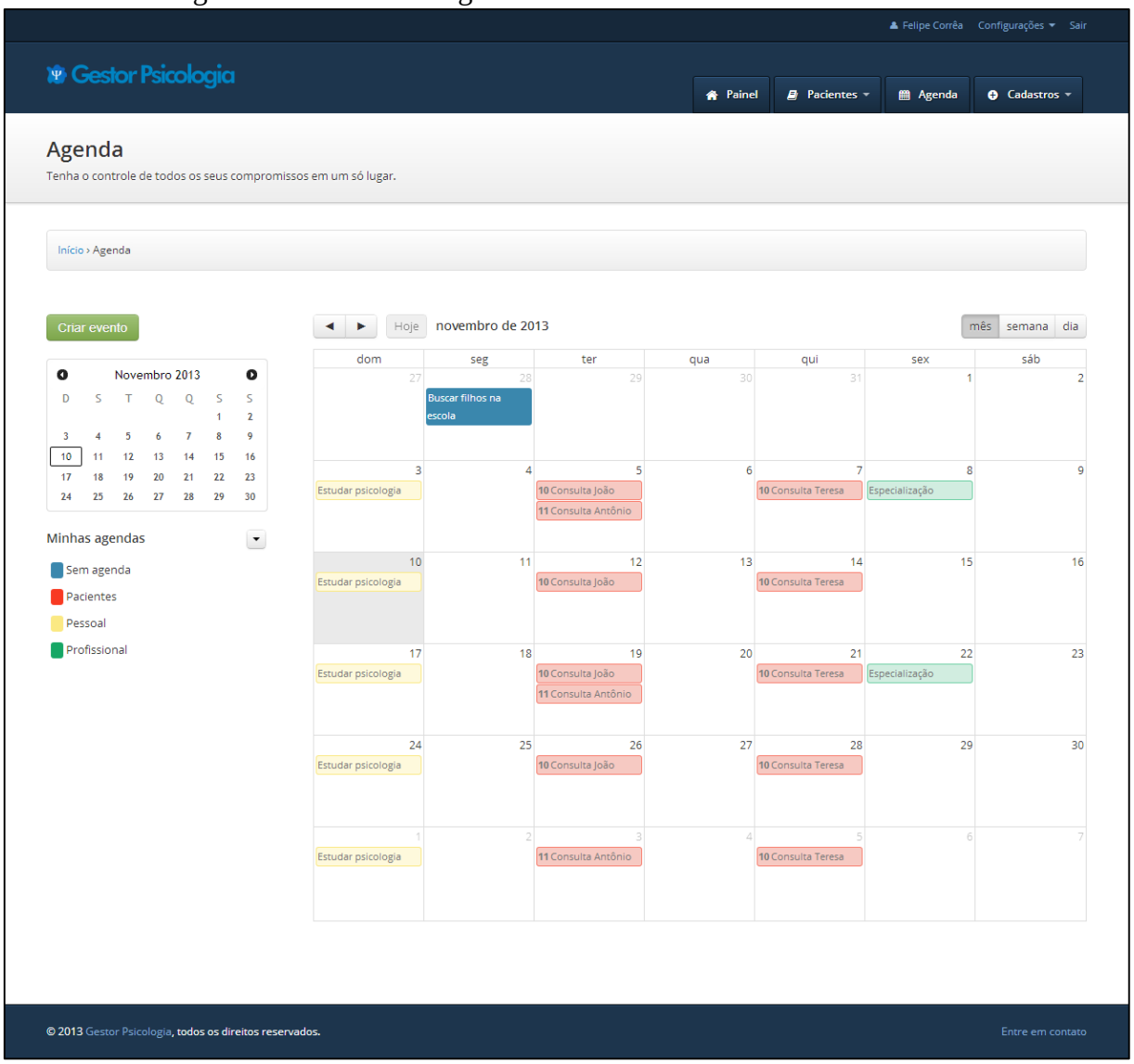
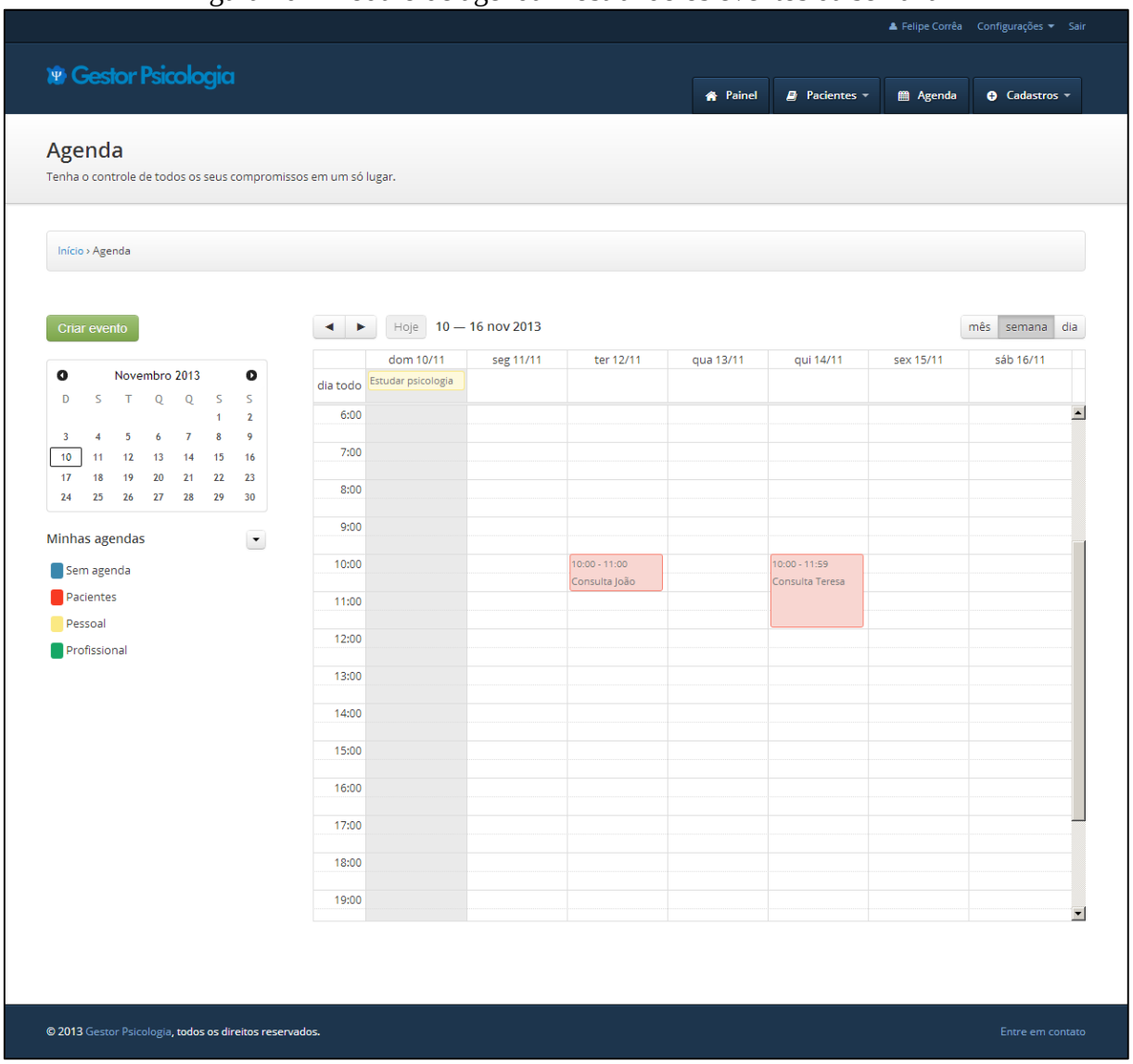


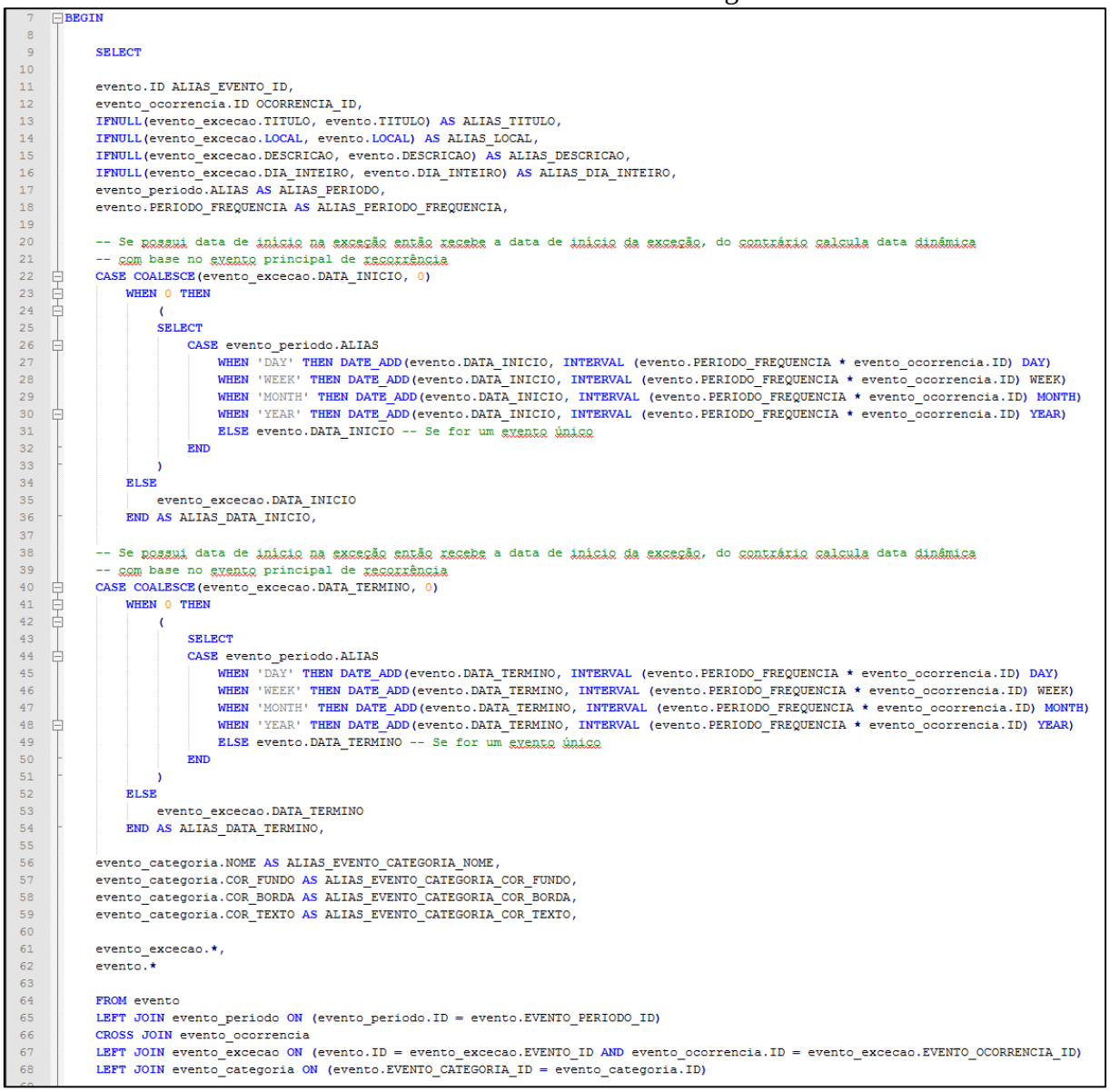
Figura 40 - Módulo de agenda mostrando os eventos da semana



Na Figura 41 é mostrado o treco de código da *Store Procedure* criada no banco de dados MySQL responsável por calcular os eventos que deverão ser exibidos no calendário de um usuário. Basicamente a *Store Procedure* lista todos os eventos de determinado usuário e, para cada evento, junta todas as informações necessárias para a exibição, como: data e hora de início, data e hora de término, categoria e cores do evento. O mais complexo na lógica desenvolvida é que os eventos que possuem recorrência são calculados dinamicamente. Sendo assim, para saber, por exemplo, a data de início e fim da terceira ocorrência de um evento que possui repetição semanal, é necessário somar à data inicial do evento a multiplicação entre a frequência de repetição e o número da recorrência. Além disso, é verificado se uma determinada ocorrência de um evento possui exceção – que nada mais é do que alguma

informação modificada especificamente para uma ocorrência de um evento – e, caso possua, então utiliza a informação da exceção.

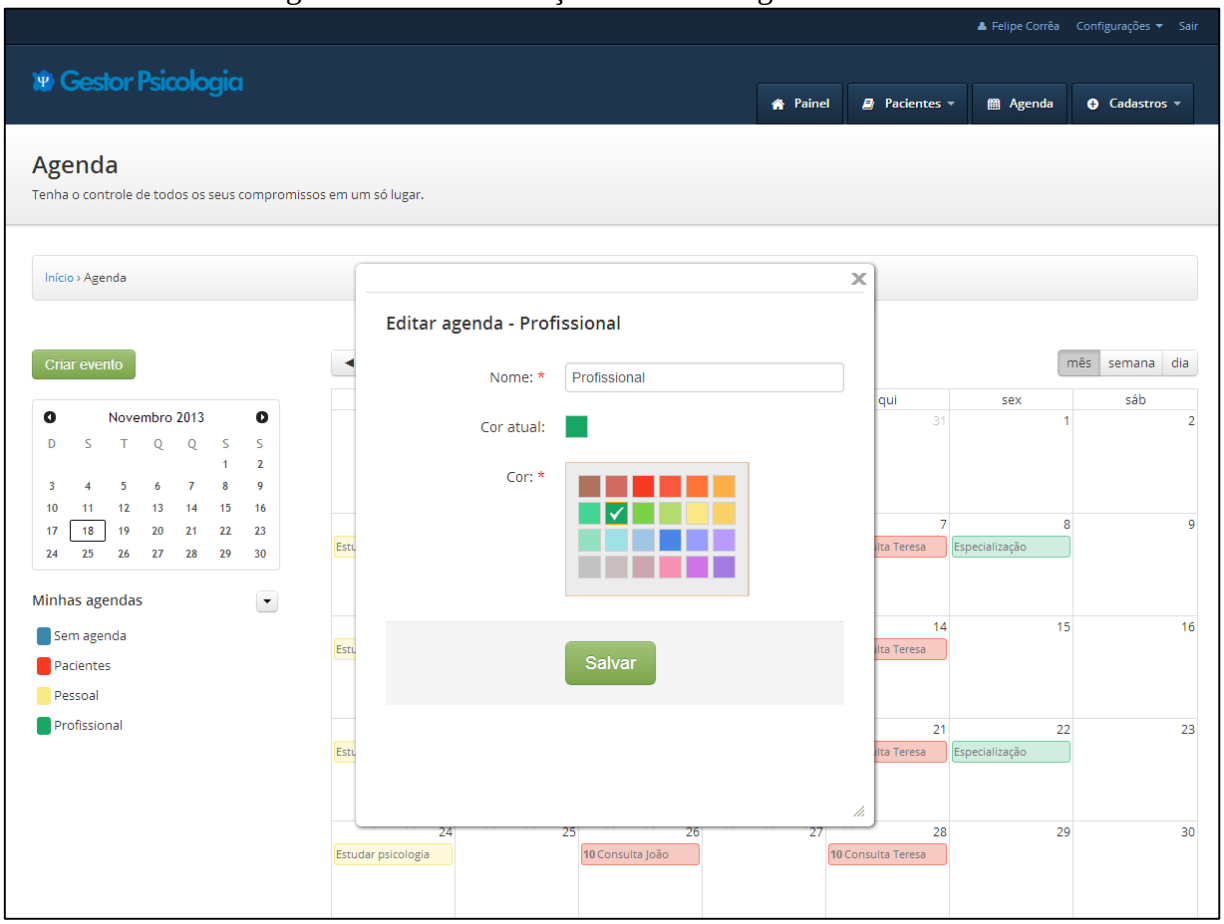
Figura 41 - Trecho de código da *Store Procedure* criada no MySQL que calcula os eventos exibidos no calendário do módulo agenda



```
BEGIN

    SELECT

    evento.ID ALIAS_EVENTO_ID,
    evento_ocorrencia.ID OCORRENCIA_ID,
    IFNULL(evento_excecao.TITULO, evento.TITULO) AS ALIAS_TITULO,
    IFNULL(evento_excecao.LOCAL, evento.LOCAL) AS ALIAS_LOCAL,
    IFNULL(evento_excecao.DESCRICAO, evento.DESCRICAO) AS ALIAS_DESCRICAO,
    IFNULL(evento_excecao.DIA_INTEIRO, evento.DIA_INTEIRO) AS ALIAS_DIA_INTEIRO,
    evento_periodo.ALIAS AS ALIAS_PERIODO,
    evento.PERIODO_FREQUENCIA AS ALIAS_PERIODO_FREQUENCIA,

    -- Se possui data de início na exceção então recebe a data de início da exceção, do contrário calcula data dinâmica
    -- com base no evento principal de recorrência
    CASE COALESCE(evento_excecao.DATA_INICIO, 0)
        WHEN 0 THEN
            (
            SELECT
                CASE evento_periodo.ALIAS
                    WHEN 'DAY' THEN DATE_ADD(evento.DATA_INICIO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_ocorrencia.ID) DAY)
                    WHEN 'WEEK' THEN DATE_ADD(evento.DATA_INICIO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_ocorrencia.ID) WEEK)
                    WHEN 'MONTH' THEN DATE_ADD(evento.DATA_INICIO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_ocorrencia.ID) MONTH)
                    WHEN 'YEAR' THEN DATE_ADD(evento.DATA_INICIO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_ocorrencia.ID) YEAR)
                    ELSE evento.DATA_INICIO -- Se for um evento único
                END
            )
        ELSE
            evento_excecao.DATA_INICIO
        END AS ALIAS_DATA_INICIO,

    -- Se possui data de início na exceção então recebe a data de início da exceção, do contrário calcula data dinâmica
    -- com base no evento principal de recorrência
    CASE COALESCE(evento_excecao.DATA_TERMINO, 0)
        WHEN 0 THEN
            (
                SELECT
                CASE evento_periodo.ALIAS
                    WHEN 'DAY' THEN DATE_ADD(evento.DATA_TERMINO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_ocorrencia.ID) DAY)
                    WHEN 'WEEK' THEN DATE_ADD(evento.DATA_TERMINO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_ocorrencia.ID) WEEK)
                    WHEN 'MONTH' THEN DATE_ADD(evento.DATA_TERMINO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_ocorrencia.ID) MONTH)
                    WHEN 'YEAR' THEN DATE_ADD(evento.DATA_TERMINO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_ocorrencia.ID) YEAR)
                    ELSE evento.DATA_TERMINO -- Se for um evento único
                END
            )
        ELSE
            evento_excecao.DATA_TERMINO
        END AS ALIAS_DATA_TERMINO,

    evento_categoria.NOME AS ALIAS_EVENTO_CATEGORIA_NOME,
    evento_categoria.COR_FUNDO AS ALIAS_EVENTO_CATEGORIA_COR_FUNDO,
    evento_categoria.COR_BORDA AS ALIAS_EVENTO_CATEGORIA_COR_BORDA,
    evento_categoria.COR_TEXTO AS ALIAS_EVENTO_CATEGORIA_COR_TEXTO,

    evento_excecao.*,
    evento.*

    FROM evento
    LEFT JOIN evento_periodo ON (evento_periodo.ID = evento.EVENTO_PERIODO_ID)
    CROSS JOIN evento_ocorrencia
    LEFT JOIN evento_excecao ON (evento.ID = evento_excecao.EVENTO_ID AND evento_ocorrencia.ID = evento_excecao.EVENTO_OCORRENCIA_ID)
    LEFT JOIN evento_categoria ON (evento.EVENTO_CATEGORIA_ID = evento_categoria.ID)
```

Ainda na tela de agenda, é possível cadastrar e editar as categorias de eventos. Na Figura 42 tem-se a tela de edição de uma categoria, permitindo alterar o nome e a cor. O requisito funcional RF15 é atendido através da Figura 42.

Figura 42 - Tela de edição de uma categoria de evento



Existem duas maneiras de criar um evento no módulo de agenda, a primeira, que é mais simplificada, permite selecionar um horário no calendário conforme exemplificado na Figura 43; a segunda permite criar eventos de forma completa, com opções de recorrência do evento e também de configuração de lembretes. Na Figura 43 é exibida a forma simplificada de criação de eventos, já na Figura 44 é exibida a forma completa. Na Figura 45 são mostradas as opções de recorrência de um evento. Com base nas Figuras 43, 44 e 45 o requisito funcional R14 foi atendido.

Figura 43 - Exemplo de criação de evento a partir do calendário

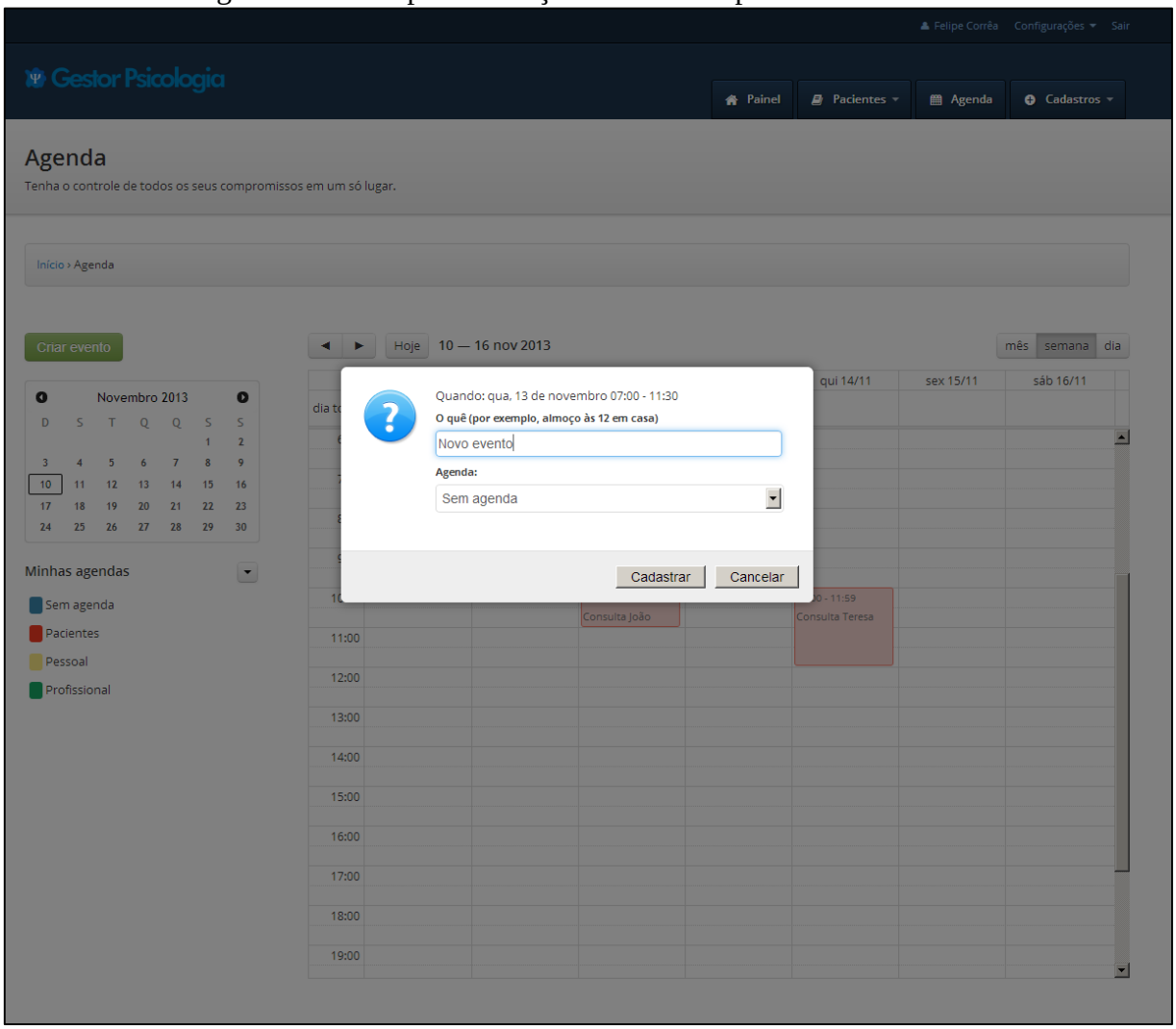


Figura 44 - Exemplo de criação de um evento de forma completa

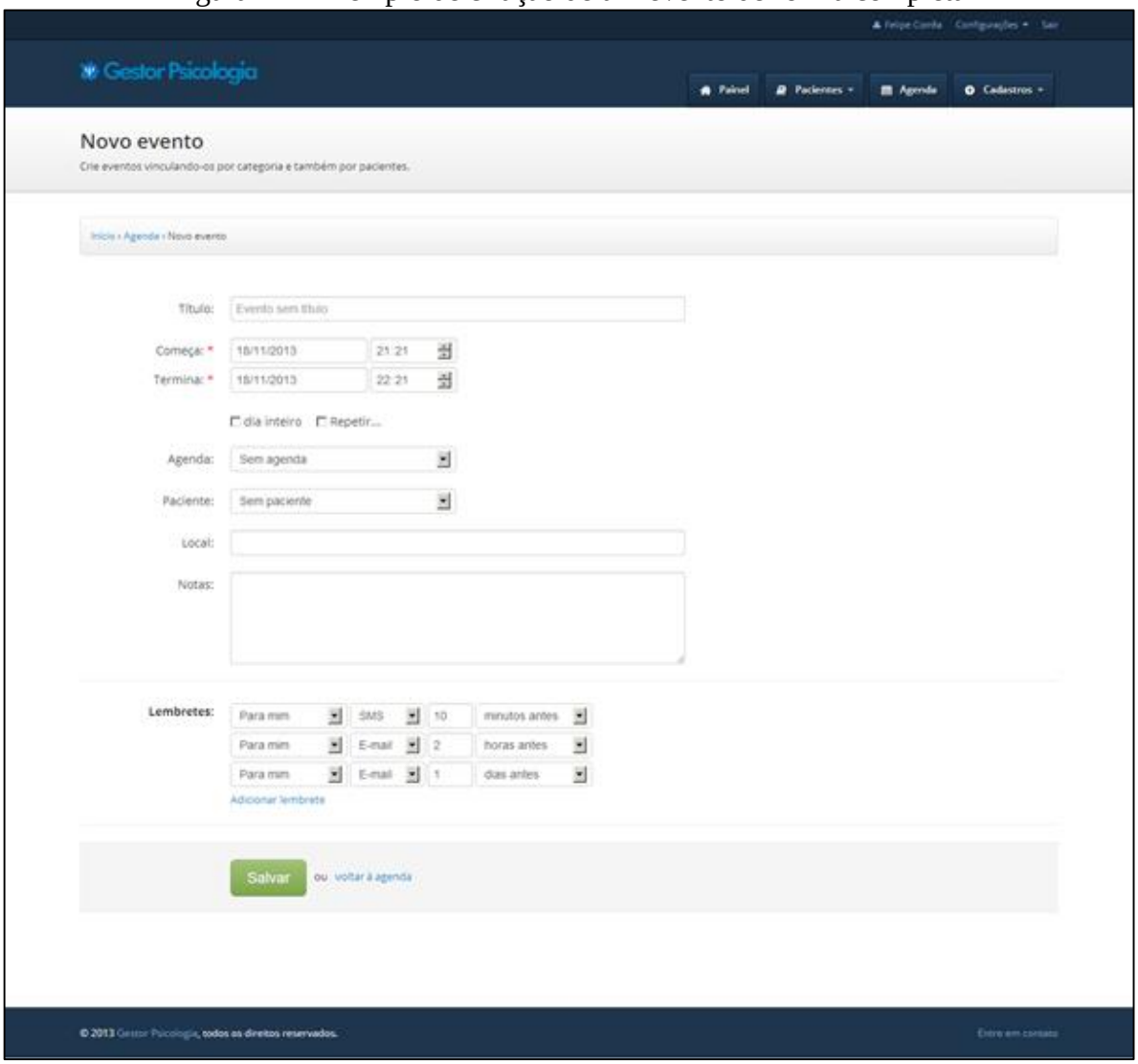
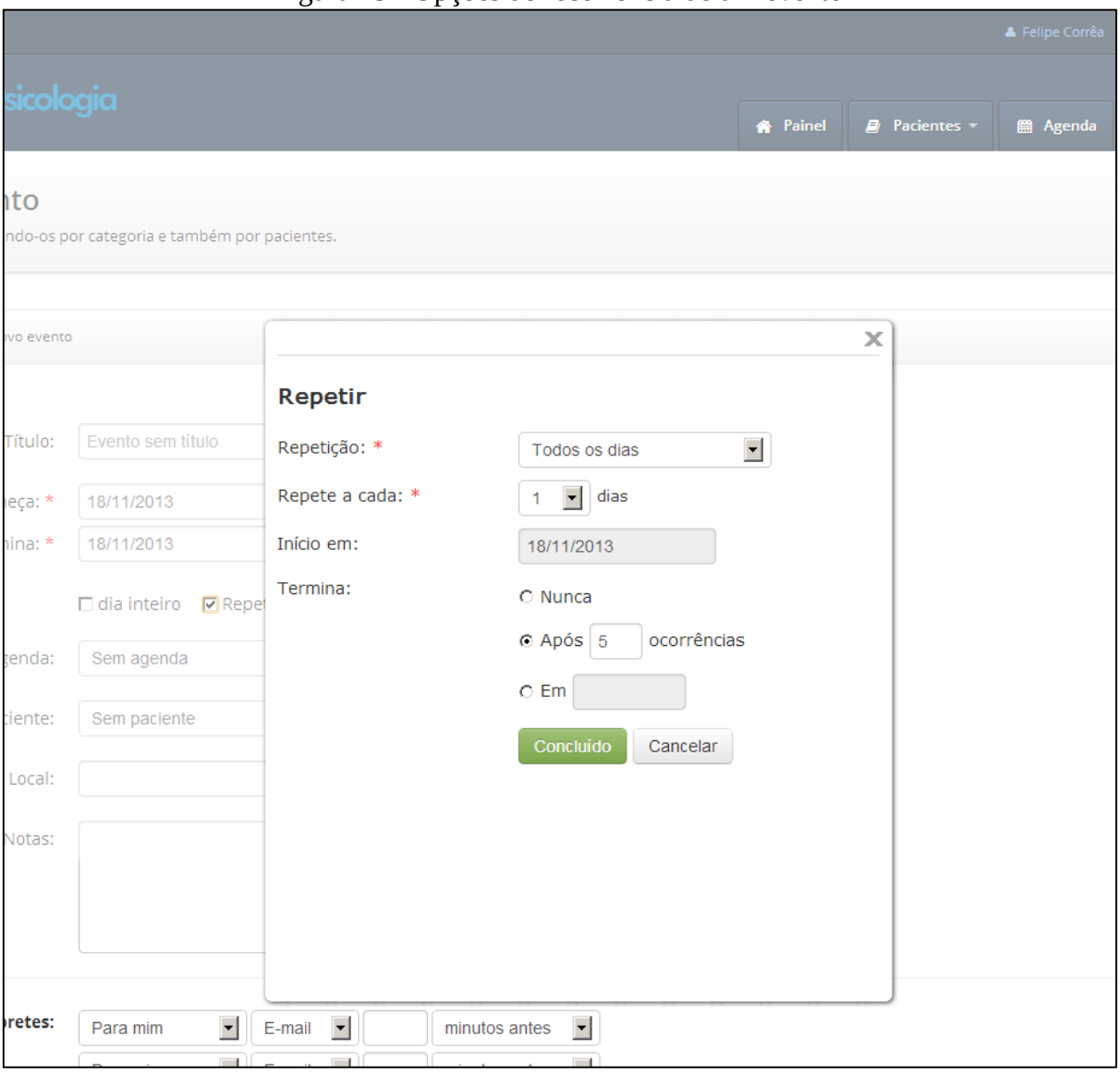


Figura 45 - Opções de recorrência de um evento



No trecho de código apresentado na Figura 46 são exibidas as ações executadas ao selecionar a repetição de um evento. Primeiro, os dados inseridos são validados com mensagens amigáveis ao usuário, depois disso, verifica-se o tipo de término selecionado para o evento. Caso seja nunca, então é inserido a informação *null* no campo recorrência contagem, caso o tipo seja um determinado número de ocorrências, então insere a quantidade de recorrências informada pelo usuário no campo recorrência contagem. Por último, caso a opção seja para encerrar a repetição em uma data, então é utilizado o método calcularOcorrenciasAtravesIntervaloData para através de uma data de término, calcular a quantidade de ocorrências que o evento terá e inserir no campo recorrência contagem.

Figura 46 - Treco de código da camada de controle da página de evento

```
377                 if ($arrForm['Repetir'])
378                 {
379                     if (!$arrForm['EventoPeriodoId'])
380                         $erros[] = 'Repetir: O campo "Repetição" é de preenchimento obrigatório.';
381                     else
382                         $objNovoEvento->setEventoPeriodoId($arrForm['EventoPeriodoId']);
383
384                     if (!$arrForm['PeriodoFrequencia'])
385                         $erros[] = 'Repetir: O campo "Repete a cada" é de preenchimento obrigatório.';
386                     else
387                         $objNovoEvento->setPeriodoFrequencia($arrForm['PeriodoFrequencia']);
388
389                     if (!$arrForm['TerminaEm'])
390                         $erros[] = 'Repetir: É necessário escolher uma opção no campo "Termina".';
391                     else
392                     {
393                         switch ($arrForm['TerminaEm'])
394                         {
395                             case 'nunca':
396                                 $objNovoEvento->setRecorrenciaContagem(null);
397                                 break;
398
399                             case 'x_ocorrencias':
400
401                                 if (!$arrForm['TerminaEmXOcorrencias'])
402                                     $erros[] = 'Repetir: É necessário preencher o campo "Ocorrências" na opção "Termina".';
403                                 else
404                                     $objNovoEvento->setRecorrenciaContagem($arrForm['TerminaEmXOcorrencias']);
405                                 break;
406
407                             case 'data_especifica':
408                                 $objNovoEvento->setIsUsarDataTermino(true);
409
410                                 if (!$objNovoEvento->getEventoPeriodo())
411                                     $erros[] = 'O período escolhido é inválido';
412                                 elseif (!$arrForm['TerminaEmDataEspecifica'] || (!$arrForm['TerminaEmDataEspecifica'] instanceo
413                                     $erros[] = 'Repetir: É necessário preencher o campo "Data de término" na opção "Termina".';
414                                 elseif ($arrForm['DataInicio'] > $arrForm['TerminaEmDataEspecifica'])
415                                     $erros[] = 'Repetir: A Data de término da repetição precisa ser maior que a data de Começo
416                                 else {
417                                     $qtd_ocorrencias = EventoPeer::calcularOcorrenciasAtravesInvervaloData($objNovoEvento->getE
418                                     $objNovoEvento->setRecorrenciaContagem($qtd_ocorrencias);
419                                 }
420                                 break;
```

Ao configurar lembretes em um evento o sistema irá verificar todos os eventos que precisam ser enviados por *e-mail* e por SMS e irá disparar os respectivos avisos. Através da Figura 47 tem-se um evento configurado com dois lembretes 10 minutos antes, um por SMS e outro por *e-mail*. Nas Figuras 48 e 49 mostra-se o recebimento do lembrete por SMS e *por e-mail* exatamente no horário configurado. De acordo com as Figuras 47, 48 e 49 atende-se o requisito funcional RF16.

Figura 47 - Tela de evento com dois lembretes configurados

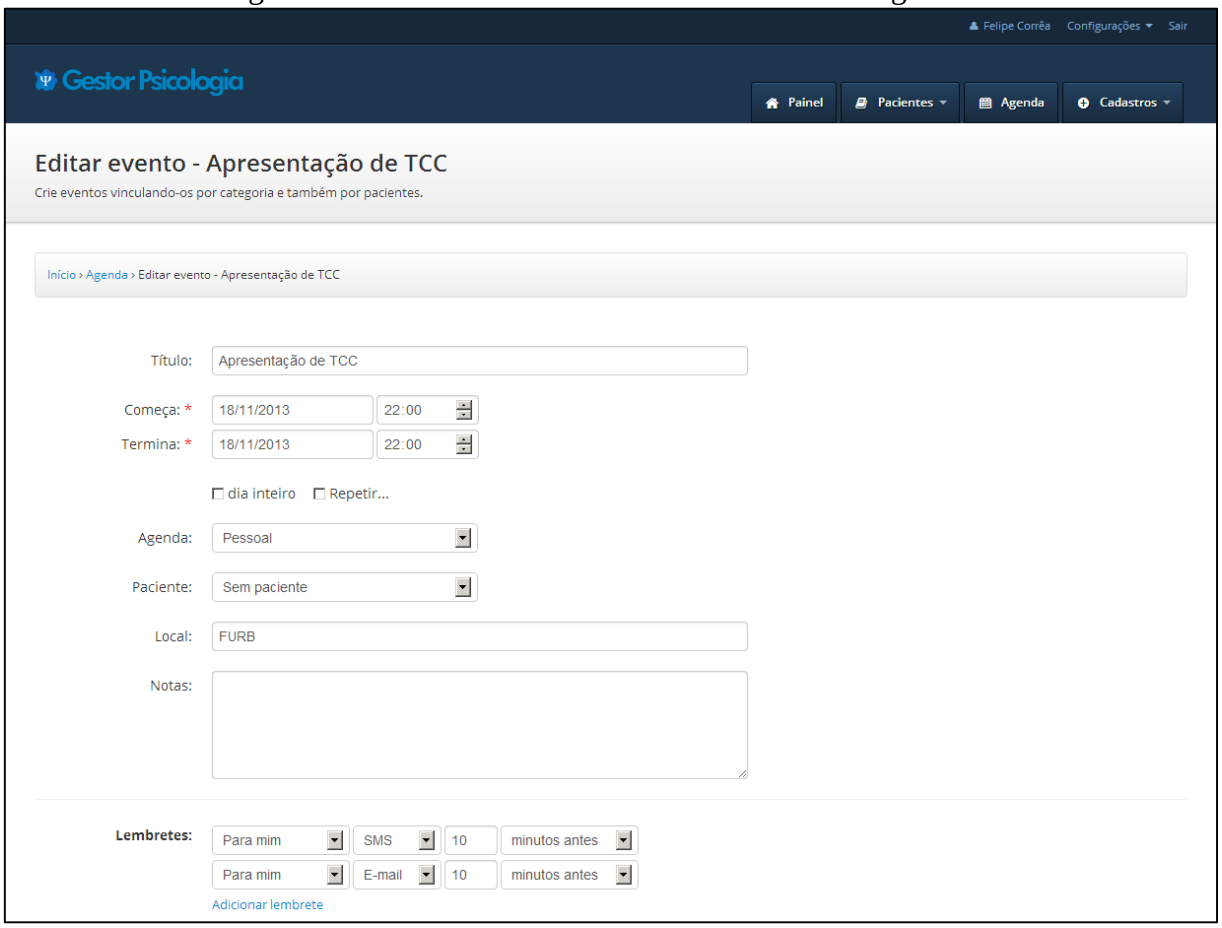
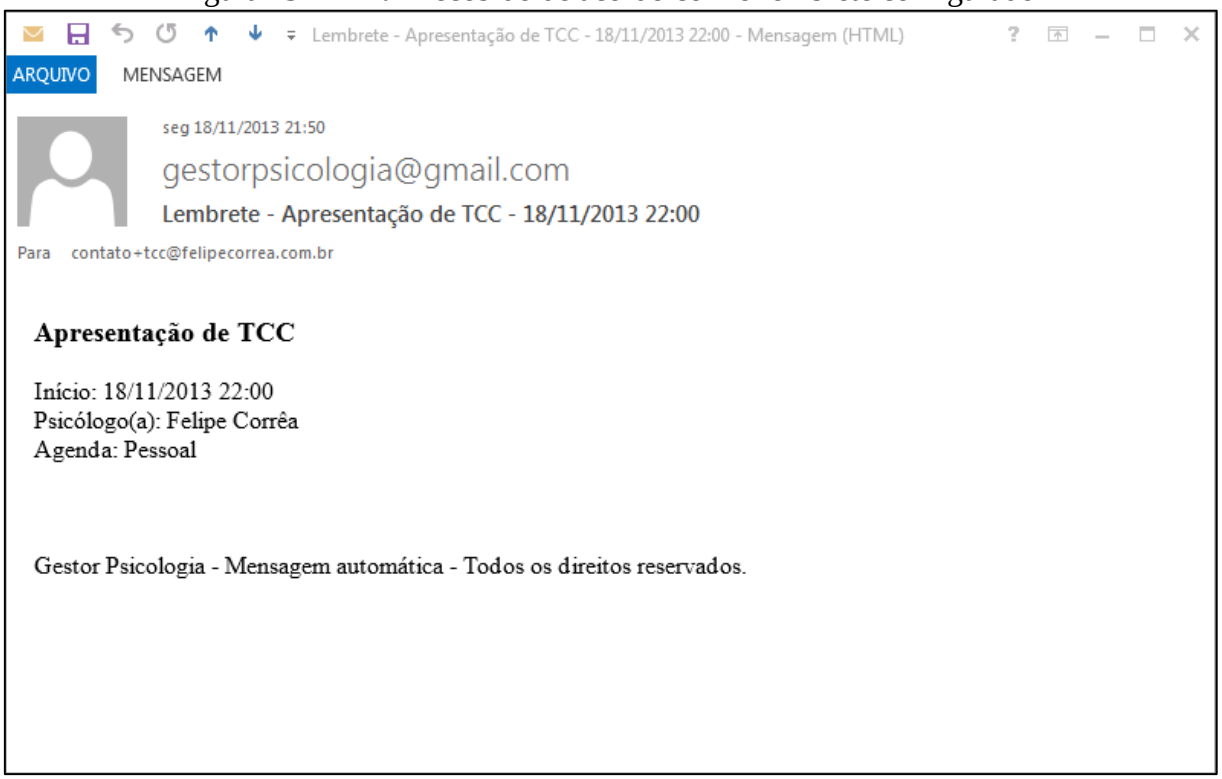


Figura 48 - SMS recebido de acordo com o lembrete configurado

Figura 49 - *E-mail* recebido de acordo com o lembrete configurado

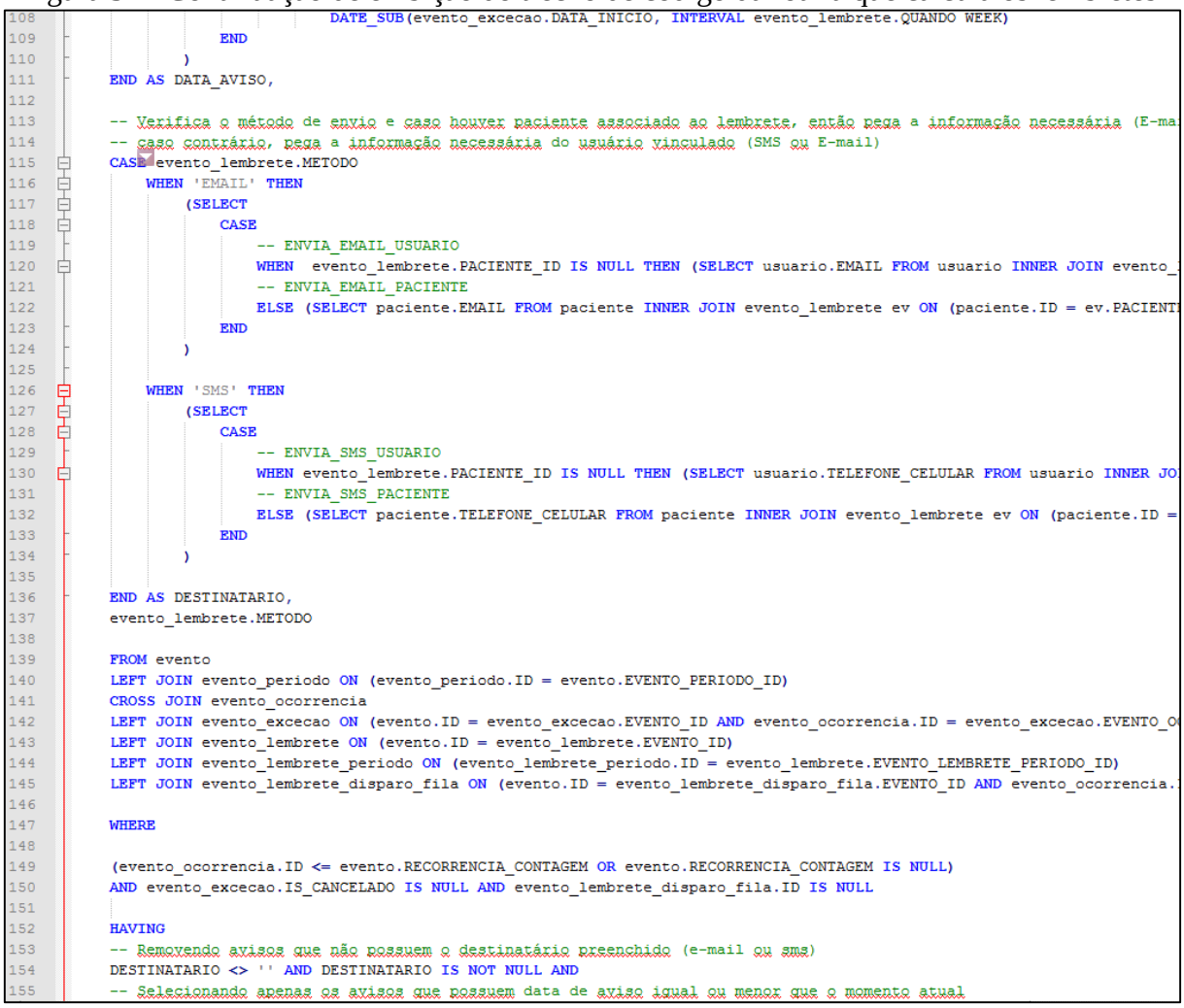


As Figuras 50 e 51 mostram trechos de código da *Store Procedure* responsável por calcular os lembretes de cada ocorrência de um evento e inserir na tabela de envio. O cálculo dos lembretes funciona da seguinte forma: calcula-se todos os eventos do sistema (calculando todas as ocorrências de um evento que possui recorrência, verificando inclusive as exceções existentes) e, depois, calcula-se a data e hora de disparo dos lembretes existentes para cada evento, sendo que é feito um filtro para inserir na tela de envio apenas os lembretes que já deveriam ter sido enviado, ou seja, a data do lembrete é igual ou já passou da data e hora atual.

Figura 50 - Trecho de código da rotina que calcula os lembretes e insere para a tabela de disparo fila

```
delimiter //

CREATE PROCEDURE PROCURE_INSERIR_LEMBRETES_DISPARO_FILA ()
BEGIN

    -- Insere todos os lembretes que devem ser enviados ou já deveriam ter sido enviados para a tabela de disparo fila
    INSERT INTO evento_lembrete_disparo_fila
        (
        evento_lembrete_disparo_fila.EVENTO_ID,
        evento_lembrete_disparo_fila.EVENTO_OCORRENCIA_ID,
        evento_lembrete_disparo_fila.EVENTO_LEMBRETE_ID,
        evento_lembrete_disparo_fila.CONTA_ID,
        evento_lembrete_disparo_fila.USUARIO_ID,
        evento_lembrete_disparo_fila.PACIENTE_ID,
        evento_lembrete_disparo_fila.DATA_AGENDADO,
        evento_lembrete_disparo_fila.DESTINATARIO,
        evento_lembrete_disparo_fila.METODO
        )

    SELECT

        evento.ID EVENTO_ID,
        evento_ocorrencia.ID OCORRENCIA_ID,
        evento_lembrete.ID as EVENTO_LEMBRETE_ID,
        evento_lembrete.CONTA_ID,
        evento_lembrete.USUARIO_ID,
        evento_lembrete.PACIENTE_ID,

    -- Caso houver data de início da exceção então vai para a condição ELSE do CASE
    -- Se for uma ocorrência normal do evento então calcula a data do aviso com base na data de ocorrência do evento
    -- Se for uma exceção do evento, então calcula a data de aviso com base na data de exceção
    CASE COALESCE(evento_excecao.DATA_INICIO, 0)
        WHEN 0 THEN
            (SELECT
                CASE evento_lembrete_periodo.ALIAS
                        WHEN 'MINUTE' THEN
                            DATE_SUB(
                            (
                            SELECT
                                CASE evento_periodo.ALIAS
                                    WHEN 'DAY' THEN DATE_ADD(evento.DATA_INICIO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_oc
                                    WHEN 'WEEK' THEN DATE_ADD(evento.DATA_INICIO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_o
                                    WHEN 'MONTH' THEN DATE_ADD(evento.DATA_INICIO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_
                                    WHEN 'YEAR' THEN DATE_ADD(evento.DATA_INICIO, INTERVAL (evento.PERIODO_FREQUENCIA * evento_o
                                    ELSE evento.DATA_INICIO -- Se for um evento único
                                END
                            ),
                            INTERVAL evento_lembrete.QUANDO MINUTE)
```

Figura 51 - Continuação de exibição do trecho de código da rotina que calcula os lembretes



```
                                DATE_SUB(evento_excecao.DATA_INICIO, INTERVAL evento_lembrete.QUANDO WEEK)
                    END
                )
    END AS DATA_AVISO,

    -- Verifica o método de envio e caso houver paciente associado ao lembrete, então pega a informação necessária (E-ma
    -- caso contrário, pega a informação necessária do usuário vinculado (SMS ou E-mail)
    CASE evento_lembrete.METODO
        WHEN 'EMAIL' THEN
            (SELECT
                CASE
                    -- ENVIA_EMAIL_USUARIO
                    WHEN  evento_lembrete.PACIENTE_ID IS NULL THEN (SELECT usuario.EMAIL FROM usuario INNER JOIN evento_
                    -- ENVIA_EMAIL_PACIENTE
                    ELSE (SELECT paciente.EMAIL FROM paciente INNER JOIN evento_lembrete ev ON (paciente.ID = ev.PACIENT
                END
            )

        WHEN 'SMS' THEN
            (SELECT
                CASE
                    -- ENVIA_SMS_USUARIO
                    WHEN evento_lembrete.PACIENTE_ID IS NULL THEN (SELECT usuario.TELEFONE_CELULAR FROM usuario INNER JO
                    -- ENVIA_SMS_PACIENTE
                    ELSE (SELECT paciente.TELEFONE_CELULAR FROM paciente INNER JOIN evento_lembrete ev ON (paciente.ID =
                END
            )

    END AS DESTINATARIO,
    evento_lembrete.METODO

    FROM evento
    LEFT JOIN evento_periodo ON (evento_periodo.ID = evento.EVENTO_PERIODO_ID)
    CROSS JOIN evento_ocorrencia
    LEFT JOIN evento_excecao ON (evento.ID = evento_excecao.EVENTO_ID AND evento_ocorrencia.ID = evento_excecao.EVENTO_O
    LEFT JOIN evento_lembrete ON (evento.ID = evento_lembrete.EVENTO_ID)
    LEFT JOIN evento_lembrete_periodo ON (evento_lembrete_periodo.ID = evento_lembrete.EVENTO_LEMBRETE_PERIODO_ID)
    LEFT JOIN evento_lembrete_disparo_fila ON (evento.ID = evento_lembrete_disparo_fila.EVENTO_ID AND evento_ocorrencia.

    WHERE

    (evento_ocorrencia.ID <= evento.RECORRENCIA_CONTAGEM OR evento.RECORRENCIA_CONTAGEM IS NULL)
    AND evento_excecao.IS_CANCELADO IS NULL AND evento_lembrete_disparo_fila.ID IS NULL

    HAVING
    -- Removendo avisos que não possuem o destinatário preenchido (e-mail ou sms)
    DESTINATARIO <> '' AND DESTINATARIO IS NOT NULL AND
    -- Selecionando apenas os avisos que possuem data de aviso igual ou menor que o momento atual
```

A Figura 52 mostra o trecho de código responsável pela rotina de envio dos lembretes. A rotina busca os envios de lembrete que ainda não foram feitos, verifica se o método de envio é *e-mail* ou SMS e, com base nisso, envia para o destinatário (paciente ou usuário) a mensagem de lembrete.

Figura 52 - Trecho de código da rotina que dispara os lembretes por *e-mail* e por SMS

```
52                  EnvioEmail::enviar($assunto, $objDisparoFila->getDestinatario(), $mensagem);
53                  echo "- Envio para: {$objDisparoFila->getDestinatario()} concluído com sucesso<br>";
54
55                  $objDisparoFila->setDataEnvio(date('Y-m-d H:i:s'));
56                  $objDisparoFila->save();
57              }
58              catch (\Exception $e)
59              {
60                  echo 'Erro: '. $e->getMessage() .'<br>';
61              }
62
63          }
64          elseif ($objDisparoFila->getMetodo() == EventoLembreteDisparoFilaPeer::METODO_SMS)
65          {
66              echo '- Método: SMS';
67
68              try {
69
70                  $agenda = ($objEvento->getEventoCategoria()) ? '('. $objEvento->getEventoCategoria()->getNome()
71
72                  $mensagem = 'Lembrete: '. $objEvento->getTitulo() .' - '. $objDisparoFila->getDataInicioEvento('
73
74                  $envio = EnvioSMS::enviar($objDisparoFila->getDestinatario(), $mensagem);
75                  echo 'Retorno SMS: '. $envio .'<br>';
76
77                  $objDisparoFila->setDataEnvio(date('Y-m-d H:i:s'));
78                  $objDisparoFila->save();
79              }
80              catch (\Exception $e)
81              {
82                  echo 'Erro: '. $e->getMessage() .'<br>';
83              }
84          }
85      }
86
87      return '<br>fim da rotina';
88
89
90  })->bind('sistema_rotina_executar_disparo_fila');
```

A Figura 53 mostra o trecho de código da classe responsável pela conexão com o *gateway* de envio de SMS, sendo que há um tratamento do número de celular e o retorno com uma resposta do *gateway* após o envio.

Figura 53 - Código da classe responsável por enviar o SMS

```php
<?php

namespace GestorPsicologiaSistema\EnvioSMS;

class EnvioSMS
{
    public static function enviar($to, $message)
    {
        $to = formata_numero_celular($to);

        $url = 'https://gwsms.com.br/api/envio/';
        $data = array(
                    'token' => 'gestorpsicologia',
                    'celular' => $to,
                    'msg' => $message,
                    'data' => '',
                );

        return post_to_url($url, $data);
    }

}
```

As Figuras 54 e 55 atendem aos requisitos funcionais RF04 e RF21, onde nas configurações do sistema o usuário poderá alterar seus dados cadastrais e também sua senha. Além disso, caso haja permissão, pode-se administrar os outros usuários. Na Figura 54 visualiza-se a listagem de todos os usuários cadastrados na conta e na Figura 55 tem-se o formulário de cadastro e edição de um usuário, destacando-se a área de permissões, na qual o acesso pode ser configurado de forma específica.

Figura 54 - Listagem dos usuários cadastrados na conta

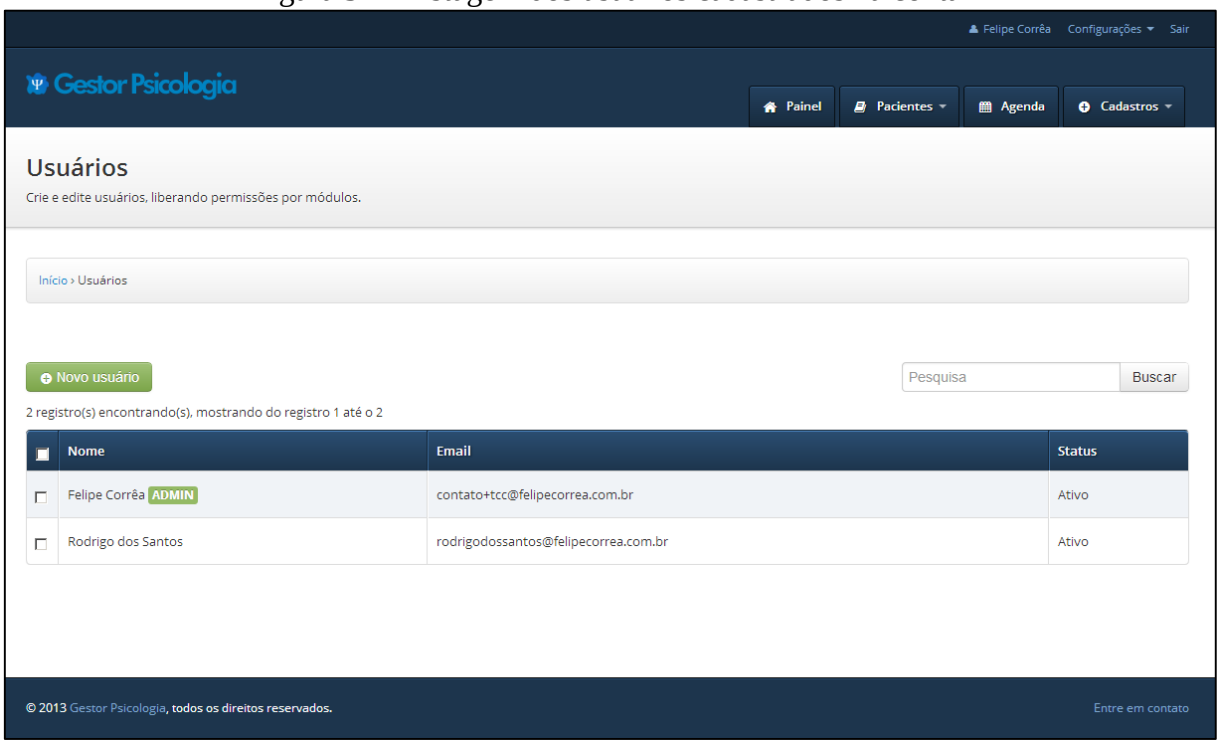
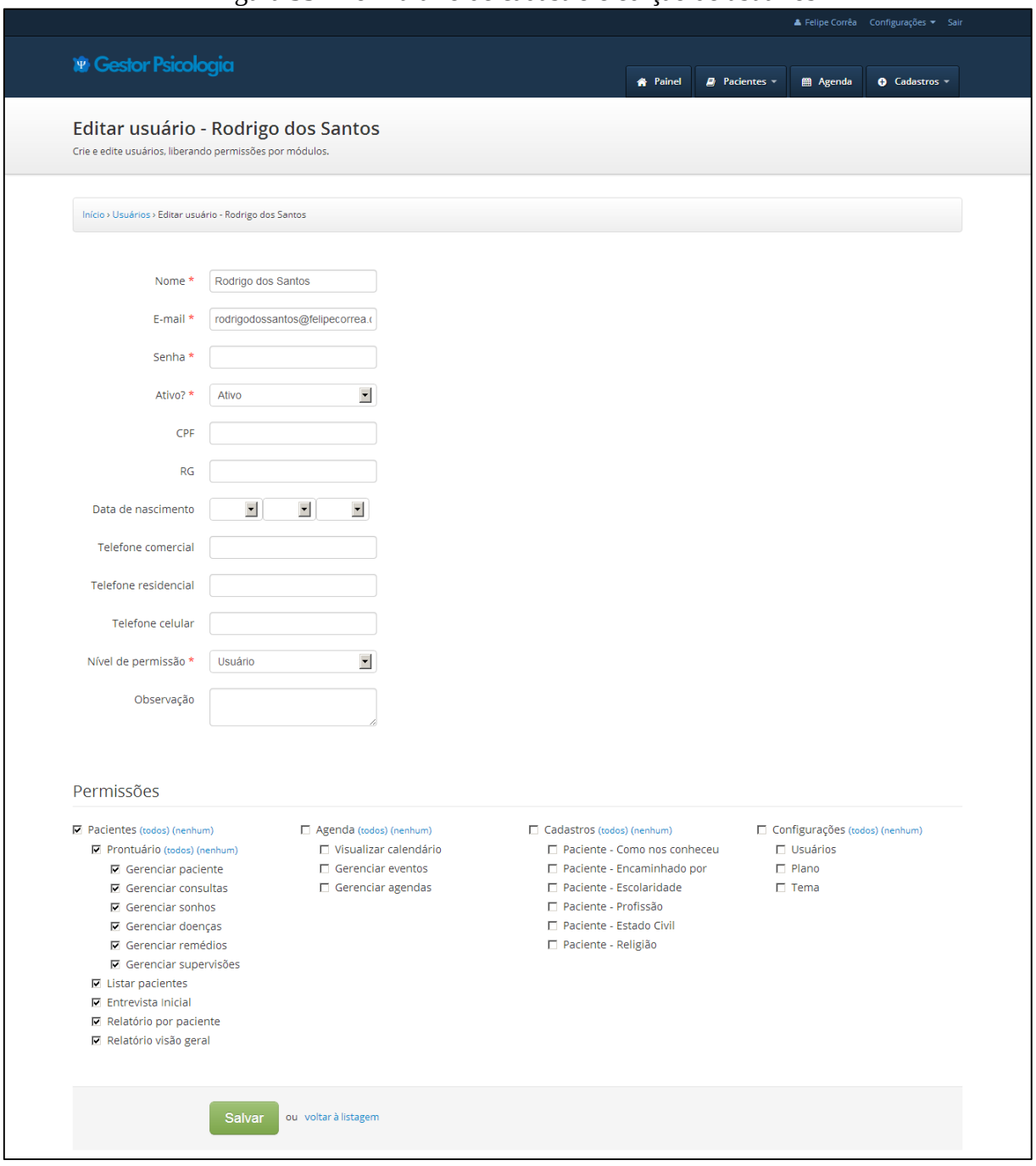


Figura 55 - Formulário de cadastro e edição de usuários



A Figura 56 mostra que o usuário pode também alterar o tema de sua aplicação, tornando as cores condizentes com o gosto de cada psicólogo. Na Figura 57 mostra-se a aplicação com o tema modificado. Desta forma, atende-se o requisito funcional R20 através das Figuras 56 e 57.

Figura 56 - Tela de mudança de tema

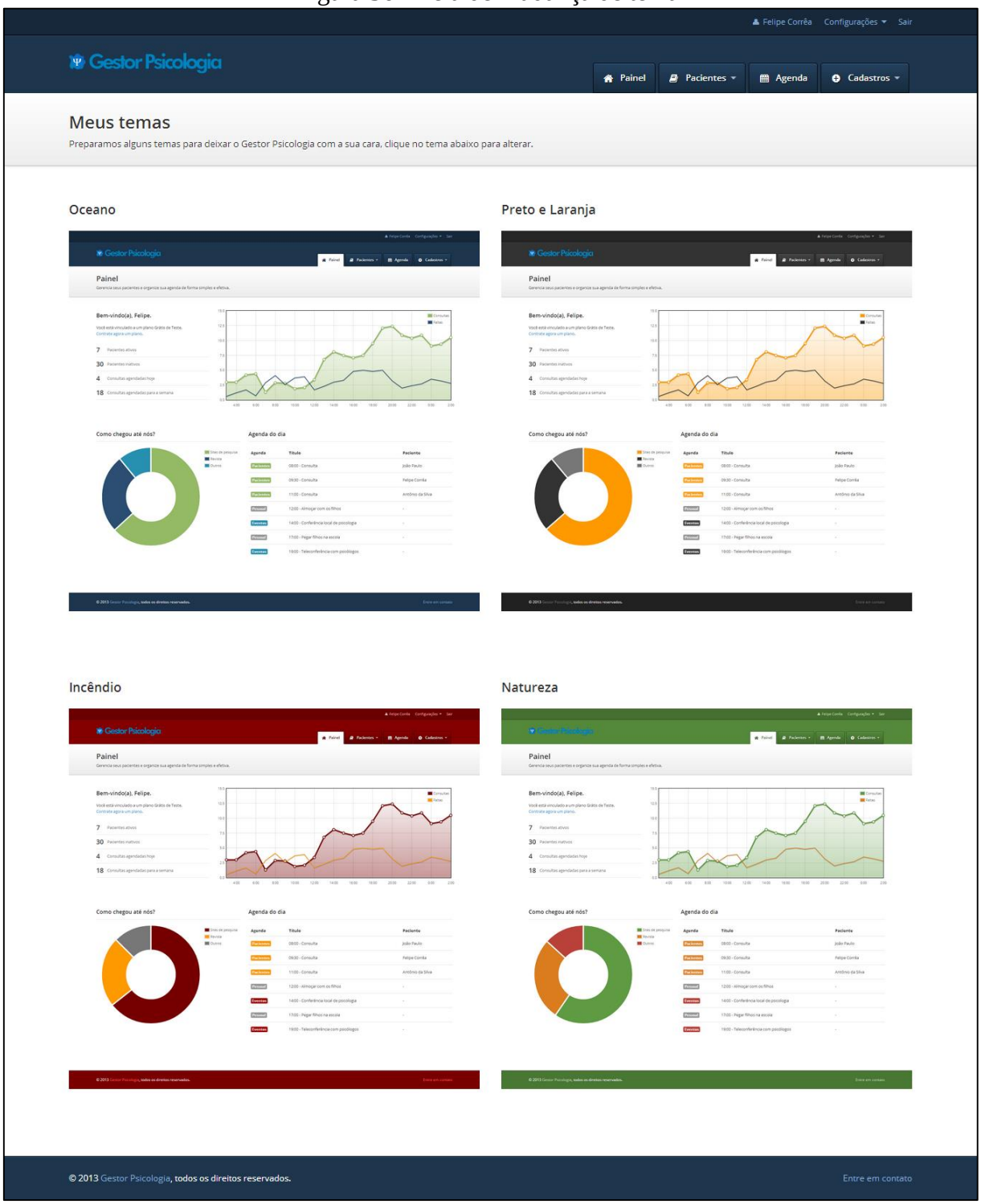
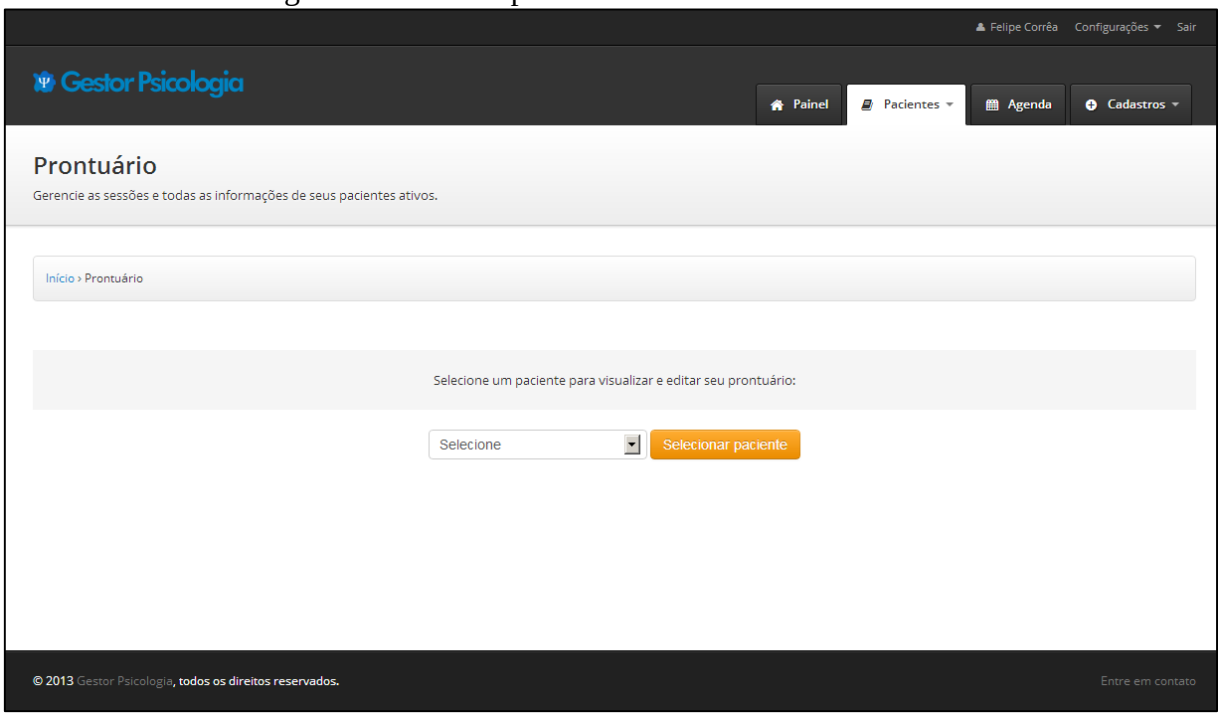


Figura 57 - Tela de prontuário com o tema modificado



Pode-se também contratar um novo plano alterando o plano atual ou modificando a forma de pagamento configurada. Assim sendo, na Figura 58 tem-se a tela de meu plano e na Figura 59, após o usuário clicar no plano bronze, o mesmo foi redirecionado para a tela de cadastro do intermediário de pagamento Moip, onde poderá informar seus dados cadastrais e escolher a forma de pagamento do novo plano contratado. Através das Figuras 58 e 59 o requisito funcional RF18 e o objetivo específico de descobrir o funcionamento e o uso de facilitador de pagamento foram atendidos.

Também na tela de meu plano demonstrada através da Figura 58, o usuário poderá cancelar sua conta a qualquer momento, sendo que é enviado um *e-mail* solicitando uma confirmação e caso o usuário aceitar, a conta será cancelada. Atende-se o RF19 conforme demonstração da Figura 58.

Figura 58 - Tela de meu plano

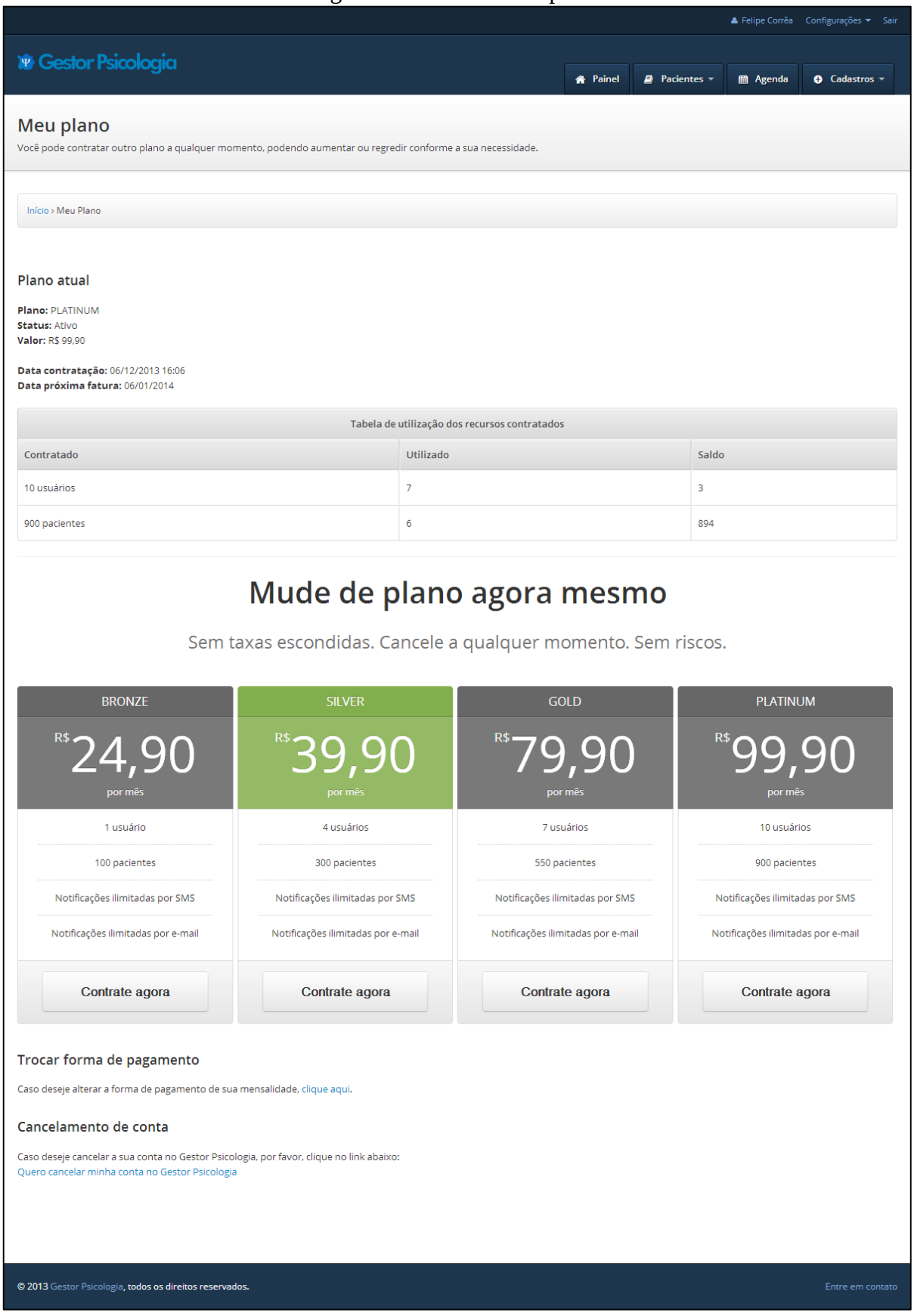
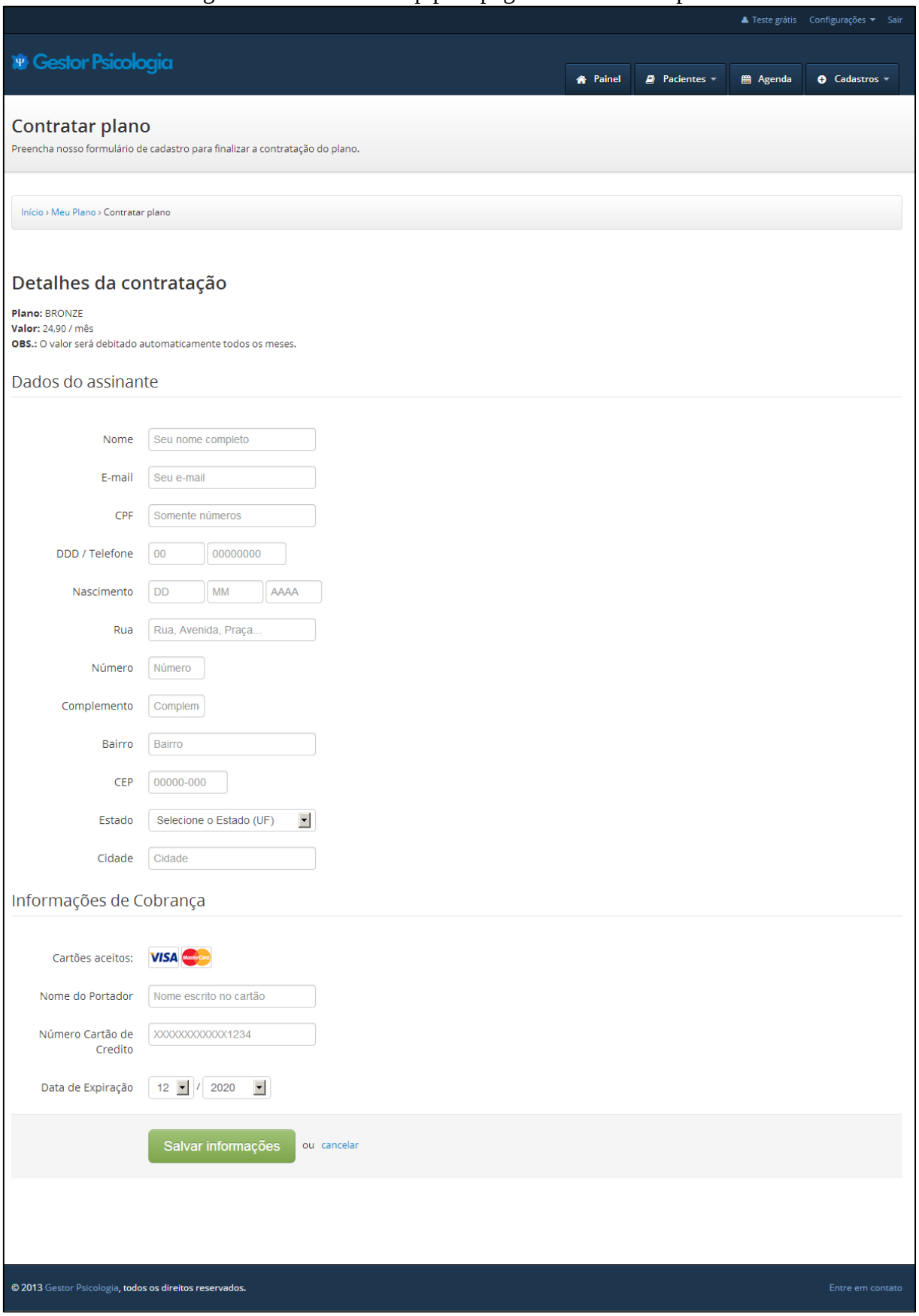


Figura 59 - Tela do Moip para pagamento do novo plano



3.3.3 Resultados e discussão

O sistema desenvolvido neste trabalho atendeu as expectativas propostas, possibilitando aos psicólogos independentes o gerenciamento de seus consultórios de psicologia, em uma plataforma *on-line* denominada Gestor Psicologia.

O Quadro 3 mostra um comparativo entre as principais funções dos sistemas correlatos com o programa desenvolvido.

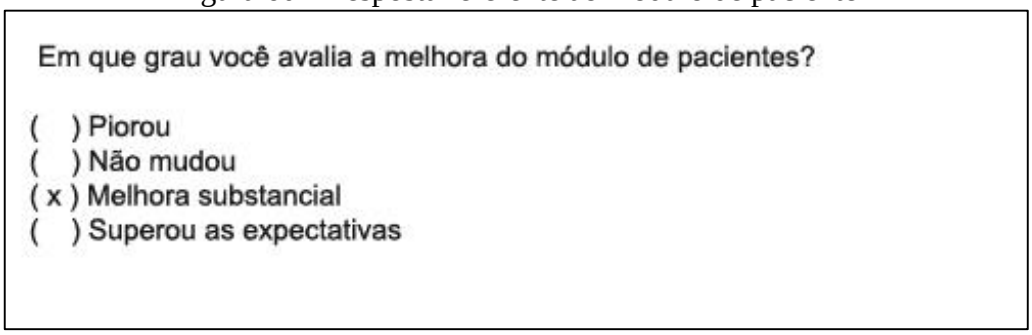
Quadro 3 - Comparativo das principais funções dos trabalhos correlatos

Funções	Gestor Psi	Insight	**Gestor Psicologia**
Administração de pacientes	X	X	X
Administração de prontuário	X	X	X
Administração de sonhos, doenças, remédios e supervisões			X
Estatísticas gerais dos pacientes			X
Módulo de agenda	X	X	X
Lembretes por *e-mail* ou SMS		X	X
Software distribuído no modelo SaaS	X		X
Pagamento on-line			X

Ao término do desenvolvimento deste trabalho foi realizado um questionário comparativo entre as funcionalidades da antiga versão, no caso o SAI, com o Gestor Psicologia, objetivando verificar a aceitação do novo sistema. Foi entrevistada uma única psicóloga, que utiliza o SAI há mais de três anos.

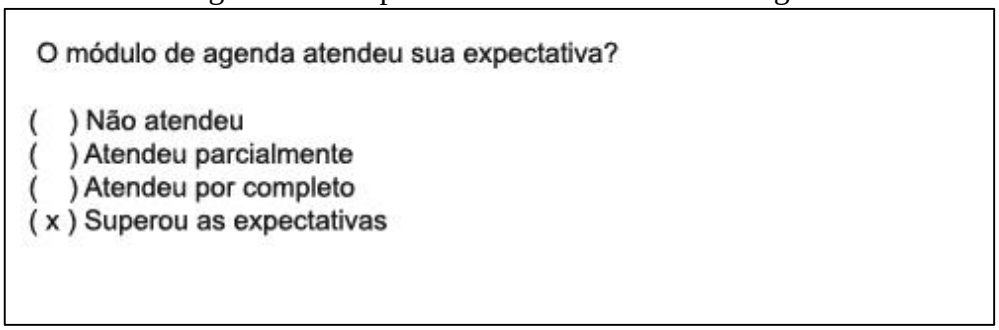
Na Figura 60 tem-se o resultado referente ao módulo de pacientes que foi reformulado. Nesta questão houve uma ótima aceitação por parte da psicóloga, que elogiou bastante a inclusão de informações padronizadas no formulário de entrevista inicial e o relatório de visão geral dos pacientes.

Figura 60 - Resposta referente ao módulo de paciente



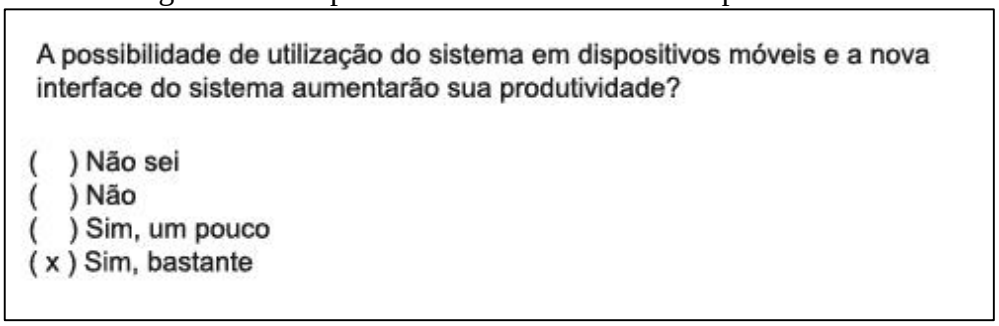
Verifica-se, na Figura 61, a expectativa com relação ao módulo de agenda, tendo em vista que este módulo não existia no SAI e que foi uma solicitação para o Gestor Psicologia.

Figura 61 - Resposta referente ao módulo de agenda



Questiona-se na Figura 62 o aumento de produtividade como consequência da nova interface do Gestor Psicologia e também da possibilidade de utilização em dispositivos móveis.

Figura 62 - Resposta referente ao aumento de produtividade



Em relação à implementação da aplicação, foi feito um comparativo entre as horas previstas no início do projeto de desenvolvimento e as horas que foram efetivamente

utilizadas. Sendo que o resultado foi uma margem de aproximadamente 51% de atraso. Mais detalhes sobre o relatório de horas podem ser vistos no Apêndice C.

4 CONCLUSÃO

A busca por uma boa saúde, física ou psicológica, tem sido cada vez mais procurada pelo indivíduo. Um dos profissionais que auxiliam na contemplação de um bem estar psicológico desejado, assim como em tratamentos dos mais diversos tipos, é o psicólogo.

Ao trabalhar com diferentes casos e diversas pessoas, que em sua maioria preferem a discrição quanto a frequentar um consultório psicológico, o profissional da área enfrenta problemas relacionados à falta de um funcionário para gerenciar sua agenda, pacientes e pagamentos, já que se compromete com a confidencialidade. Sendo assim, enfrenta algumas dificuldades de trabalho no seu dia-a-dia.

Paralelamente a este fato, a ideia do prontuário eletrônico está cada vez mais tomando forma em outros ramos da saúde. Com o aumento de pacientes, há uma dificuldade no gerenciamento dos mesmos e com a grande quantidade de papel, devido aos registros. Em um prontuário eletrônico, obtém-se facilidade de acesso e organização de informações, ou seja, usufruindo das novas tecnologias, os profissionais da saúde têm a oportunidade de agilizar e melhor administrar seus atendimentos. Contudo, ainda não há uma plataforma que supra todas as necessidades de uma clínica de psicólogos ou mesmo para um psicólogo independente.

Diante disso, o Gestor Psicologia fornece ao psicólogo a praticidade e todas as demais vantagens oferecidas por um serviço de prontuário eletrônico, entre outros benefícios. Permite uma melhor desenvoltura de serviço para ele, além de lhe poupar tempo e espaço, assim oferecendo um melhor atendimento ao paciente. O serviço também oportuniza a criação de análises estatísticas dos pacientes, permitindo saber, por exemplo, como o mesmo passou a conhecer o consultório (indicação, hospitais, internet, etc.); a quantidade total de consultas por mês e por semana; classificações em relação à religião, sexo e escolaridade; dentre diversas outras informações.

O referente trabalho alcançou seu objetivo ao desenvolver um sistema para psicólogos independentes, que permite o gerenciamento de consultórios de psicologia, em uma plataforma *on-line* denominada “Gestor Psicologia”. Isso só foi possível ao atingir os demais objetivos propostos: houve o entendimento do funcionamento de um consultório de psicologia, para assim criar um serviço que pudesse suprir as necessidades do cliente; o uso de facilitadores de pagamento foi compreendido e teve sua aplicação para gerenciar as cobranças e pagamentos do negócio; o software foi distribuído como serviço no modelo SaaS através da internet; e, por fim, aplicou-se o *layout* responsivo para permitir acesso de

dispositivos móveis, permitindo uma maior comodidade e mobilidade de consulta aos registros.

Logo, a criação de um sistema que centralize as informações de um psicólogo e que lhe proporcione segurança perante os dados armazenados, simboliza um avanço quando analisado as limitações que possuem atualmente e a falta de um sistema que permita o gerenciamento de seu negócio. O Gestor Psicologia apresenta este diferencial de trabalho, sendo um sistema único. Além disso, os próprios usuários poderão enviar sugestões e solicitações de novas implementações, que serão disponibilizadas para todos os demais clientes. Outra vantagem de um sistema unificado é a padronização de registros entre todos os psicólogos, o que permitirá os mais variados estudos voltados à área.

4.1 EXTENSÕES

Como sugestões de extensões para a continuidade do presente trabalho, tem-se:

a) implementar uma área administrativa para que os administradores do sistema possam acompanhar todas as contas que estão inadimplentes e também ter relatórios financeiros, como a previsão dos recebíveis com base nas contas ativas;
b) implementar um módulo financeiro para os psicólogos, que será integrado ao prontuário e também à agenda, permitindo que com base nas consultas pré-agendadas de um paciente possa-se ter uma previsão de recebíveis dos pacientes. Além disso, possibilitará a gestão de contas a pagar, contas a receber, fluxo de caixa, centros de custo, dentre outros benefícios;
c) implementar um módulo de *backup* para os psicólogos, de modo que os próprios usuários possam guardar suas informações;
d) melhorar a forma como as informações dos prontuários são armazenadas no banco de dados, de modo a utilizar criptografia e dificultar o roubo de dados no caso possíveis ataques.

REFERÊNCIAS

ALMEIDA, Fabrício Fernandes; CANTAL, Clara; COSTA JUNIOR, Áderson Luiz. Prontuário psicológico orientado para o problema: um modelo em construção. **Psicol. cienc. prof.**, Brasília, v. 28, n. 2, jun. 2008. Disponível em: <http://pepsic.bvsalud.org/scielo.php?script=sci_arttext&pid=S1414-98932008000200016&lng=pt&nrm=iso>. Acesso em: 20 ago. 2013.

BASTOS, Antônio Virgílio Bittencourt; GOMIDE, Paula Inez Cunha. O psicólogo brasileiro: sua atuação e formação profissional. **Psicol. cienc. prof.**, Brasília, v. 9, n. 1, 1989, p. 6-15. Disponível em <http://pepsic.bvsalud.org/scielo.php?pid=S1414-98931989000100003&script=sci_arttext>. Acesso em: 14 abr. 2013.

CAMBIUCCI, Waldemir. Uma introdução ao Software + Serviços, SaaS e SOA. **Microsoft Developer Network.** [S.l.], mai. 2009. Disponível em: <http://msdn.microsoft.com/pt-br/library/dd875466.aspx#bm_3_2>. Acesso em: 17 abr. 2013.

CAPITÃO, Cláudio Garcia; SCORTEGAGNA, Silvana Alba; BAPTISTA, Makilim Nunes. A importância da avaliação psicológica na saúde. **Aval. psicol.**, Porto Alegre, v. 4, n. 1, jun. 2005. Disponível em: <http://pepsic.bvsalud.org/scielo.php?script=sci_arttext&pid=S1677-04712005000100009&lng=pt&nrm=iso>. Acesso em: 20 ago. 2013.

CAPRIO, Griffin. **Design Patterns**: Dependency Injection. [S.l.], set. 2005. Disponível em: <http://msdn.microsoft.com/en-us/magazine/cc163739.aspx>. Acesso em: 10 out. 2013.

CARR, Richard. **Multiton Design Pattern**. [S.l.], mar. 2012. Disponível em: <http://www.blackwasp.co.uk/Multiton.aspx>. Acesso em: 11 out. 2013.

CERON, João M. et. al. Vulnerabilidades em Aplicações Web: uma Análise Baseada nos Dados Coletados em Honeypots. In: SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 8., 2008, Gramado. **Anais...** Gramado: UFRGS. 2008. p. 277-278. Disponível em: <http://sbseg2008.inf.ufrgs.br/anais/data/pdf/st06_02_resumo.pdf>. Acesso em: 16 out. 2013.

CONSELHO FEDERAL DE MEDICINA. **Resolução nº 1.638**, de 9 de agosto de 2012. Brasília, DF, 2012. Disponível em: <http://www.portalmedico.org.br/resolucoes/cfm/2002/1638_2002.htm>. Acesso em: 19 ago. 2013.

CONSELHO FEDERAL DE MEDICINA. **Resolução nº 1.821**, de 23 de nov. de 2007. Brasília, DF, 2007. Disponível em: <http://www.portalmedico.org.br/resolucoes/cfm/2007/1821_2007.htm>. Acesso em: 19 ago. 2013.

CONSELHO FEDERAL DE MEDICINA; SOCIEDADE BRASILEIRA DE INFORMÁTICA EM SAÚDE. **Cartilha sobre Prontuário Eletrônico**: A Certificação de Sistemas de Registro Eletrônico de Saúde. [S.l.], fev. 2012. Disponível em:

<http://portal.cfm.org.br/crmdigital/Cartilha_SBIS_CFM_Prontuario_Eletronico_fev_2012.pdf>. Acesso em: 19 ago. 2013.

DUARTE, Maria Eduarda. A Avaliação Psicológica na Intervenção Vocacional: Princípios, Técnicas e Instrumentos. In: SILVA, José Tomás; TAVEIRA, Maria do Céu (Coords.). **Psicologia Vocacional**: Perspectivas para a intervenção. Portugal: Coimbra, 2008, p. 139-158.

FOWLER, Martin. **Padrões de arquitetura de aplicações corporativas**. Tradução: Acauan Fernandes. Porto Alegre: Artmed, 2006. 493 p, il.

FREEMAN, Eric; FREEMAN, Elisabeth. **Use a cabeça!**: Padrões de projetos. Rio de Janeiro: Alta Books, 2005. 494 p, il.

GAMMA, Erich et. al. **Design patterns**: elements of reusable object-oriented software. Reading, Massachusetts: Addison-Wesley, 1994. xv, 395p.

GESTOR PSI. **Portal - Sistema de prontuários eletrônicos e gestão de serviços em psicologia.** São Carlos, SP, 2004. Disponível em: <http://portal.gestorpsi.com.br/vantagens/generalidade/>. Acesso em: 01 jun. 2013.

INSIGHT. **Insight Gestión de psicólogos.** [S.l.], 2006. Disponível em: <http://www.grupoinsight.com/quees.htm>. Acesso em: 14 abr. 2013.

JANSSEN, Cory. **Object-Relational Mapping (ORM)**. [S.l.], 2013a. Disponível em: <http://www.techopedia.com/definition/24200/object-relational-mapping--orm>. Acesso em: 15 set. 2013.

JANSSEN, Cory. **Session Hijacking**. [S.l.], 2013b. Disponível em: <http://www.techopedia.com/definition/4101/session-hijacking>. Acesso em: 15 set. 2013.

LIPNER, Steve; HOWARD, Michael. **O ciclo de vida do desenvolvimento da segurança de computação confiável**. [S.l.], mar. 2005. Disponível em: <http://msdn.microsoft.com/pt-br/library/ms995349.aspx>. Acesso em: 16 out. 2013.

MACIEL, Flávio. **Intermediários de Pagamentos**: o que são e como escolher. Porto Alegre, RS, 2012. Disponível em: <http://www.ipagare.com.br/>. Acesso em: 10 out. 2013.

MACORATTI, José Carlos. **Modelando sistemas em UML – Casos de uso**. São Paulo, nov. 2004. Disponível em: <http://imasters.com.br/artigo/2753/uml/modelando_sistemas_em_uml_-_casos_de_uso/>. Acesso em: 17 nov. 2013.

MINETTO, Elton Luís. **Frameworks para desenvolvimento em PHP**. São Paulo: Novatec, 2007. 188 p, il.

MOIP. **Checkout Transparente**. [S.l.], 2013. Disponível em: <https://site.moip.com.br/checkout-transparente/#.UhVMOj_TL7c>. Acesso em: 11 out. 2013.

MSDN. **Lazy Initialization**. [S.l.], 2013. Disponível em: <http://msdn.microsoft.com/en-us/library/dd997286.aspx>. Acesso em: 10 out. 2013.

PONTES, Penha. **A importância da Psicologia na sociedade.** Guarabira, PB, 2005. Disponível em: <http://www.brejo.com/colunistas/wmview.php?ArtID=155> . Acesso em: 13 abr. 2013.

PROPEL. **Documentation**. [S.l.], 2013. Disponível em: <http://propelorm.org/>. Acesso em: 12 set. 2013.

REDE SEGURA. **Série Ataques**: Saiba mais sobre o Cross-site Scripting (XSS). São Paulo, SP, jan. 2012a. Disponível em: <http://www.redesegura.com.br/2012/01/saiba-mais-sobre-o-cross-site-scripting-xss/>. Acesso em: 11 out. 2013.

REDE SEGURA. **Série Ataques**: Saiba mais sobre os ataques de Injeção de SQL. São Paulo, SP, fev. 2012b. Disponível em: <http://www.redesegura.com.br/2012/02/serie-ataques-saiba-mais-sobre-os-ataques-de-injecao-de-sql/>. Acesso em: 11 out. 2013.

SEIXAS, Pablo de Sousa; COELHO-LIMA, Fellipe; COSTA, Ana Ludmila Freire Costa. Caracterização de dissertações/Teses que versam sobre a profissão de Psicólogo no Brasil. In: YAMAMOTO, Oswaldo H.; COSTA, Ana Ludmila F (orgs.). **Escritos sobre a profissão de psicólogo no Brasil.** Natal: EDUFRN, 2010. p. 61-97.

SILEX. **Introduction**. [S.l.], 2013. Disponível em: <http://silex.sensiolabs.org/>. Acesso em: 16 out. 2013.

SYMFONY. **Security**. [S.l.], 2013, il. color. Disponível em: <symfony.com/doc/current/book/security.html>. Acesso em: 15 nov. 2013.

TAVARES, Marcelo. Considerações Preliminares à Condução de uma Avaliação Psicológica. **Avaliação Psicológica**, Brasília, v.11, n. 3, dez. 2012, p. 321-334. Disponível em: <http://pepsic.bvsalud.org/scielo.php?pid=S1677-04712012000300002&script=sci_arttext&tlng=en >. Acesso em: 06 jun. 2013

TELES, Maria Luiza Silveira. **O que é psicologia.** 9. ed. São Paulo: Brasiliense, 1995. 70 p. (Primeiros passos, 222).

APÊNDICE A – Detalhamento dos Casos de Uso

Nos casos de uso a seguir, tem-se o detalhamento dos principais casos de uso contemplados nos diagramas apresentados na seção 3.2.2. O Quadro 4 apresenta o caso de uso Escolher plano.

Quadro 4 – Descrição do caso de uso “UC01 – Escolher plano”

UC01 Escolher plano
Permite ao visitante escolher um dentre os vários planos apresentados. **Ator:** Visitante **Constraints** *Pré-condição.* O visitante deverá acessar a página de planos. *Pós-condição.* O visitante escolheu um plano. **Cenários** **Escolher plano {Principal}.** 1. O *site* apresenta os planos disponíveis para utilização; 2. O visitante escolhe um plano; 3. O sistema redireciona o usuário para a página de criação de conta.

No Quadro 5 pode-se visualizar o detalhamento do caso de uso Criar conta.

Quadro 5 - Descrição do caso de uso “UC02 – Criar conta”

UC02 Criar conta
Permite ao visitante preencher um formulário de cadastro para criar uma conta no sistema e poder acessá-lo. **Ator:** Visitante **Constraints** *Pré-condição.* O visitante deverá ter escolhido um plano. *Pós-condição.* O visitante deverá ter uma conta criada no sistema.

Pós-condição. O visitante deverá ter um usuário de acesso ao sistema.

Cenários

Criar conta {Principal}.

1. O usuário preenche os campos Nome, Empresa, Telefone, *E-mail*, Senha e Aceitação dos termos de uso;
2. Sistema valida os dados do usuário;
3. Sistema cria uma conta para o usuário;
4. Sistema apresenta mensagem de que a conta foi criada com sucesso.

Criar conta – E-mail já existente {Exceção}

Caso no passo 2 do cenário principal o sistema valide que já existe um usuário cadastrado utilizando o *e-mail* informado:

1. O Sistema exibirá a mensagem ao Usuário: "Verificamos que já existe um usuário cadastrado com o *e-mail* informado, por favor, utilize outro.".

Criar conta – Validar dados {Exceção}

Caso no passo 2 do cenário principal o sistema valide que algum dos campos obrigatórios não está preenchido:

1. O Sistema exibirá a mensagem ao Usuário: "Preencha os campos obrigatórios.".

No Quadro 6 pode-se visualizar o detalhamento do caso de uso Administrar pacientes.

Quadro 6 - Descrição do caso de uso "UC05 – Administrar pacientes"

UC05 Administrar pacientes

Permite ao usuário administrar os pacientes, cadastrando-os, editando-os ou excluindo-os.

Ator: Usuário

Constraints

Pré-condição. O usuário deve estar logado no sistema.

Pré-condição. O usuário deve ter permissão para acessar.

Pós-condição. Um novo paciente cadastrado no sistema.

Pós-condição. Um paciente editado no sistema.

Pós-condição. Um paciente excluído no sistema.

Cenários

Cadastrar paciente {Principal}.

1. O sistema apresenta a tela de listagem de pacientes;

2. O usuário clica no botão “Entrevista inicial”;

3. O sistema apresenta a tela de entrevista inicial do paciente, com os campos:

Ativo, Nome do paciente, Referência, Data de nascimento, Sexo, Identidade sexual, Estado Civil, Cônjuge, Filhos observação, Nome do responsável, Telefone celular, Telefone casa, Telefone trabalho, *E-mail*, Religião, Profissão, Escolaridade, Escola, Reside com, Como chegou até nós, Encaminhado por, Data primeira consulta e Queixa principal.

4. O usuário preenche os campos;

5. O sistema valida o formulário;

6. O sistema redireciona o usuário para a tela de listagem de pacientes com a mensagem “Parabéns, um novo paciente foi cadastrado com sucesso.”.

Deletar paciente {Alternativo}.

No passo 1 do cenário principal ao acessar a listagem, na coluna "Ações" ao clicar no botão "Deletar", deve-se seguir as ações abaixo:

1. Exibir uma mensagem informando:

"Você realmente deseja deletar o paciente "Nome do paciente" com todas as informações vinculadas (consultas, sonhos, etc.)?";

2. O usuário clica no botão "Ok";

3. O sistema irá verificar se existe alguma inconsistência;

4. O sistema irá deletar logicamente o paciente;

5. O sistema irá apresentar a mensagem "Paciente deletado com sucesso.".

Editar paciente {Alternativo}.

Caso no passo 1 do cenário principal, na coluna "Ações" ao clicar no botão "Editar", deve-se seguir os passos abaixo:

1. O Sistema exibe a mesma tela do passo 3 do cenário principal, porém, com as informações preenchidas e podendo ser editadas;

2. O usuário altera os campos;

3. O sistema valida o formulário;
4. O sistema redireciona o usuário para a tela de listagem de pacientes com a mensagem "Parabéns, o paciente "Nome do paciente" foi editado com sucesso.".

Buscar contato {Alternativo}.
No passo 1 do cenário principal ao acessar a listagem de pacientes, é possível buscar um paciente com base nos procedimentos abaixo:
1. O usuário preenche o campo de busca com o nome do paciente que se deseja encontrar;
2. O usuário clica no botão "Buscar";
3. O sistema irá exibir a mesma tela de listagem de pacientes filtrando pelo termo informado pelo usuário.

Cadastrar paciente - Preenchimento de dados obrigatórios {Exceção}
Caso no passo 5 do cenário principal ou no passo 3 do cenário alternativo "Editar usuário", algum dos campos obrigatórios não estiver preenchido:
1. O Sistema permanecerá na mesma tela e mostrará todos os campos existentes já preenchidos com os valores anteriormente colocados, porém, alertará o usuário sobre os campos obrigatórios que devem ser preenchidos.

No Quadro 7 pode-se visualizar o detalhamento do caso de uso Administrar eventos da agenda.

Quadro 7 - Descrição do caso de uso "UC20 – Administrar eventos da agenda"

UC20 Administrar eventos da agenda
Permite ao usuário administrar os seus eventos da agenda podendo criar, editar e deletar eventos. Pode-se também habilitar eventos que se repetirão e criar lembretes por *e-mail* e SMS.

Ator: Usuário

Constraints
Pré-condição. O usuário deve estar logado no sistema.
Pré-condição. O usuário deve ter permissão para acessar.
Pós-condição. Um novo evento cadastrado no sistema.

Pós-condição. Um evento editado no sistema.

Pós-condição. Exclusão de um evento do sistema.

Cenários

Cadastrar evento {Principal}.

1. O sistema apresenta uma tela exibindo os eventos do usuário;
2. O usuário clica no botão “Criar evento”;
3. O sistema apresenta a tela de cadastro de evento contendo os campos:
Título, Data de começo, Data de término, Repetir, Agenda, Paciente, Local e Notas;
4. O usuário preenche os campos desejados;
5. O sistema valida as informações, inclusive verificando se existem lembretes e repetições;
6. O sistema cria um evento;
7. O sistema apresenta a mensagem de sucesso: “O evento foi cadastrado com sucesso.”.

Deletar evento {Alternativo}.

No passo 3 do cenário principal ao clicar no botão “Excluir”, deve-se seguir os passos abaixo:

1. Exibir uma mensagem informando:
"Você realmente deseja deletar o evento "Título do evento"?";
2. O usuário clica no botão "Ok";
3. O sistema deleta o evento;
4. O sistema apresenta a mensagem "Evento deletado com sucesso.".

Editar evento {Alternativo}.

Caso no passo 1 do cenário principal, ao clicar em um evento já cadastrado, deve-se seguir os passos abaixo:

1. O sistema exibe a mesma tela do passo 3 do cenário principal, porém, com as informações preenchidas e podendo ser editadas;
2. O usuário altera os campos;
3. O sistema valida o formulário;
4. O sistema salva as alterações do evento;
5. O sistema redireciona o usuário para a tela de exibição dos eventos do usuário com a

mensagem “O evento “Título do evento” foi editado com sucesso.”.

Adicionar lembrete ao evento {Alternativo}.

No passo 1 do cenário principal ao clicar o botão de "Adicionar lembrete", deve-se seguir as ações abaixo:

1. O sistema apresenta um formulário a parte contendo os campos:

Destinatário (Para mim ou para o paciente), Tipo de envio (*E-mail* ou SMS), quando o alerta será enviado;

2. O usuário preenche as informações.

Remover lembrete do evento {Alternativo}.

No passo 1 do cenário alternativo “Adicionar lembretes ao evento” ao clicar no botão “Remover lembrete", deve-se seguir as ações abaixo:

1. O sistema remove o lembrete que estava vinculado ao evento.

Adicionar repetição ao evento {Alternativo}.

No passo 3 do cenário principal ao selecionar a opção "Repetir", deve-se seguir as ações abaixo:

1. O sistema apresenta um formulário em um janela contendo os campos:

Repetição, Repete a cada, Início em, Termina em;

2. O usuário preenche as informações;

3. O usuário clica no botão “Concluído”;

4. O sistema fecha a janela de repetição.

Remover repetição ao evento {Alternativo}.

No passo 1 do cenário principal ao remover a seleção da opção "Repetir", deve-se seguir as ações abaixo:

1. O sistema removerá a repetição do evento.

Cadastrar evento - Preenchimento de dados obrigatórios {Exceção}.

Caso no passo 5 do cenário principal e no passo 3 do cenário alternativo “Editar evento”, algum dos campos obrigatórios não estiver preenchido.

1. O sistema permanecerá na mesma tela e mostrará todos os campos existentes já

preenchidos com os valores anteriormente colocados, porém, alertará o usuário sobre os campos obrigatórios que devem ser preenchidos.

Cadastrar evento – Validar dados {Exceção}.
Caso no passo 5 do cenário principal e no passo 3 do cenário alternativo "Editar evento", o sistema valide que existe alguma inconsistência nos dados preenchidos:
1. O sistema exibirá uma mensagem informando ao usuário o que deverá ser ajustado.

No Quadro 8 pode-se visualizar o detalhamento do caso de uso Administrar usuários da conta.

Quadro 8 - Descrição do caso de uso "UC23 – Administrar usuários da conta"

UC23 Administrar usuários da conta
Permite gerenciar os usuários disponíveis em uma conta.

Ator: Usuário

Constraints
Pré-condição. O usuário deve estar logado no sistema.
Pré-condição. O usuário deve ter permissão para acessar.
Pós-condição. Um novo usuário cadastrado no sistema.
Pós-condição. Um usuário editado no sistema.
Pós-condição. Um usuário excluído do sistema.

Cenários
Cadastrar usuário {Principal}.
1. O sistema apresenta a tela de listagem de usuários;
2. O usuário clica no botão "Novo usuário";
3. O sistema apresenta a tela de cadastro de usuário com os campos:
Nome e *E-mail* do novo usuário;
4. O usuário preenche os campos;
5. O sistema valida o formulário;
6. O sistema redireciona o usuário para a tela de listagem de usuários com a mensagem "Parabéns, um convite para o novo usuário foi enviado com sucesso.".

Deletar usuário {Alternativo}.

No passo 1 do cenário principal ao acessar a listagem, na coluna "Ações" ao clicar no botão "Deletar", deve-se seguir as ações abaixo:

1. Exibir uma mensagem informando:

"Você realmente deseja deletar o usuário "Nome do usuário"?";

2. O usuário clica no botão "Ok";

3. O sistema verifica se existe alguma inconsistência;

4. O sistema deleta o usuário;

5. O sistema irá apresentar a mensagem "Usuário deletado com sucesso.".

Editar usuário {Alternativo}.

Caso no passo 1 do cenário principal, na coluna "Ações" ao clicar no botão "Editar", deve-se seguir os passos abaixo:

1. O sistema exibe a mesma tela do passo 3 do cenário principal, porém, com as informações preenchidas e podendo ser editadas;

2. O usuário altera os campos;

3. O sistema valida o formulário;

4. O sistema redireciona o usuário para a tela de listagem de usuários com a mensagem “Parabéns, o usuário “Nome do usuário” foi editado com sucesso.”.

Reenviar convite {Alternativo}.

Caso no passo 1 do cenário principal, na coluna "Ações" ao clicar no botão "Reenviar convite", deve-se seguir os passos abaixo:

1. Exibir uma mensagem informando:

"Você realmente deseja reenviar o convite ao usuário "Nome do usuário"?";

2. O sistema reenvia o convite por *e-mail*;

3. O sistema redireciona o usuário para a tela de listagem de usuários com a mensagem “Parabéns, o convite para o usuário “Nome do usuário” foi reenviado com sucesso.”.

Cadastrar usuário - Preenchimento de dados obrigatórios {Exceção}

Caso no passo 5 do cenário principal e no passo 3 do cenário alternativo “Editar usuário”, algum dos campos obrigatórios não estiver preenchido.

1. O Sistema permanecerá na mesma tela e mostrará todos os campos existentes já preenchidos com os valores anteriormente colocados, porém, alertará o usuário sobre os campos obrigatórios que devem ser preenchidos.

Deletar usuário – Restrições {Exceção}

Caso no passo 3 do cenário alternativo "Deletar usuário", o usuário esteja tentando deletar seu próprio usuário:

1. O sistema exibirá a seguinte mensagem ao usuário: "Você não pode excluir seu próprio usuário.";
2. O sistema não irá deletar este usuário.

No Quadro 9 pode-se visualizar o detalhamento do caso de uso Cancelar conta.

Quadro 9 - Descrição do caso de uso "UC25 – Cancelar conta"

UC25 Cancelar conta

Permite cancelar uma conta.

Ator: Usuário

Constraints

Pré-condição. O usuário deve estar logado no sistema.

Pré-condição. O usuário deve ter permissão para acessar.

Pós-condição. Uma conta cancelada.

Cenários

Cancelar conta {Principal}.

1. O sistema apresenta de Mudar de plano;
2. O usuário clica no botão "Cancelar minha conta no GestorPsicologia";
3. O sistema atualiza a tela e exibe a mensagem "Enviamos um *e-mail* com as instruções de cancelamento da sua conta.".

***E-mail* de cancelar conta {Alternativo}.**

No passo 3 do cenário principal ao abrir o *e-mail* de cancelamento da conta, o usuário irá

clicar no link “Cancelar minha conta” existente no *e-mail* e então será redirecionado para a tela de cancelamento no sistema:

1. O sistema apresenta uma tela com as informações de cancelamento;
2. O usuário clica no botão “Aceito os termos de cancelamento”;
3. O sistema faz a exclusão de todas as informações da conta do usuário;
4. O sistema faz o *logoff* do usuário;
5. O sistema redireciona o usuário para uma tela de agradecimento pela utilização do serviço.

APÊNDICE B – Dicionário de Dados

Este apêndice apresenta o dicionário de dados das principais tabelas utilizadas no sistema, com informações de atributos, tipos, tamanhos, campos obrigatórios, descrições, chaves primárias e estrangeiras, ou *Primary Keys* (PK) e *Foreign Keys* (FK). Nos Quadros de 10 a 16 estão o dicionário de dados das principais tabelas do sistema. Foram utilizados os seguintes tipos de atributos:

a) *int*: para valores numéricos;
b) *varchar*: para valores de texto, podendo também possuir números;
c) *text*: para valores de texto longo, podendo também possuir números;
d) *date*: para valores do tipo data;
e) *datetime*: para valores do tipo data e hora;
f) *boolean*: para valores de verdadeiro ou falso;
g) *double*: para valores monetários;
h) *enum*: para opções pré-definidas em um campo.

Quadro 10 - Dicionário de dados da tabela "usuario"

USUARIO			
Tabela responsável por armazenar os usuários de todas as contas do sistema.			
Nome do atributo	Tipo	Tamanho	Descrição
id (PK)	*int*	10	Identificador único
nome	*varchar*	150	Nome do usuário
email	*varchar*	150	*E-mail* do usuário (utilizado para enviar notificação por *e-mail*)
senha	*varchar*	100	*Hash* da senha do usuário
is_ativo	*boolean*	1	Informa se o usuário está ativo
conta_id (FK)	*int*	10	Identificador da conta no sistema
cpf	*varchar*	20	CPF do usuário
rg	*varchar*	20	RG do usuário
data_nascimento	*date*	-	Data de nascimento do usuário
is_dono_conta	*boolean*	1	Informa se é o dono da conta
telefone_comercial	*varchar*	20	Telefone comercial do usuário
telefone_residencial	*varchar*	20	Telefone residencial do usuário

telefone_celular	*varchar*	20	Telefone celular do usuário (utilizado para enviar notificação por SMS)
observacao	*varchar*	400	Campo de observação sobre o usuário
nivel_permissao	*enum*	-	Nível de permissão no sistema, pode ser Usuário ou Administrador
data_criacao	*datetime*	-	Data de criação do usuário
data_atualizacao	*datetime*	-	Data de atualização do usuário
data_exclusao	*datetime*	-	Data de exclusão do usuário
unique_key	*varchar*	30	*Hash* aleatório criado para em conjunto com outro *hash* estático gerar a senha, desta forma, mesmo que dois usuários possuam a mesma senha no sistema, o *hash* da senha será diferente

Quadro 11 - Dicionário de dados da tabela "plano_contratado"

PLANO_CONTRATADO			
Tabela responsável por armazenar todos os planos contratados por uma conta, sendo que uma conta poderá ter apenas um plano ativo por vez.			
Nome do atributo	Tipo	Tamanho	Descrição
id (PK)	*int*	10	Identificador único
conta_id (FK)	*int*	10	Identificador da conta no sistema
plano_id (FK)	*int*	10	Identificador do plano no sistema
is_ativo	*boolean*	1	Informa se o usuário está ativo
data_contratacao	*datetime*	-	Data da contratação do plano
valor_plano	*double*	-	Valor mensal do plano contratado
porcentagem_desconto	*int*	2	Porcentagem de desconto
recorrencia	*int*	2	Recorrência do pagamento em meses (1 para mensal, 3 para trimestral, 12 para anual)
dia_vencimento	*int*	2	Dia de vencimento da fatura
tipo	*enum*	-	Tipo do plano, pode ser Grátis ou Pago
data_teste_expiracao	*date*	-	Data de explicação do plano grátis

Quadro 12 - Dicionário de dados da tabela "log_evento"

LOG_EVENTO			
Tabela responsável por armazenar os *logs* de todas as ações executadas pelos usuários do sistema.			
Nome do atributo	Tipo	Tamanho	Descrição
id (PK)	*int*	10	Identificador único
usuario_id (FK)	*int*	10	Identificador do usuário no sistema
modulo_id (FK)	*int*	10	Identificador do módulo no sistema
acao_id (FK)	*int*	10	Identificador da ação no sistema
log_tipo_evento_id (FK)	*int*	10	Identificador do tipo de evento no sistema
informacao_cliente	*varchar*	3000	Informações da sessão usuário no momento da gravação do log (navegador, *URL*, parâmetros enviados, etc.)
data	*datetime*	-	Data do log
mensagem	*varchar*	400	Detalhes adicionais do evento

Quadro 13 - Dicionário de dados da tabela "paciente"

PACIENTE			
Tabela responsável por armazenar todos os pacientes de um usuário.			
Nome do atributo	Tipo	Tamanho	Descrição
id (PK)	*int*	10	Identificador único
nome	*varchar*	150	Nome do paciente
referencia	*varchar*	45	Código de referência da ficha interna do paciente
sexo	*enum*	-	Pode ser M ou F
identidade_sexual	*varchar*	100	Identidade sexual do paciente
filhos_observacao	*varchar*	350	Informações sobre os filhos do paciente
nome_responsavel	*varchar*	150	Nome do responsável, caso o paciente não tenha a maior idade legal
conjuge	*varchar*	150	Nome do cônjuge do paciente
telefone_comercial	*varchar*	20	Telefone comercial do paciente
telefone_residencial	*varchar*	20	Telefone residencial do paciente
telefone_celular	*varchar*	20	Telefone celular do paciente (utilizado para

			enviar notificação por SMS)
email	*varchar*	150	*E-mail* do paciente (utilizado para enviar notificação por *e-mail*)
escola	*varchar*	150	Escola do paciente
residecom	*varchar*	150	Pessoas que residem com o paciente
data_primeira_consulta	*date*	-	Data da primeira consulta do paciente
queixa_principal	*text*	-	Queixa principal do paciente
is_ativo	*boolean*	1	Indica se o paciente está ativo
paciente_estado_civil_id (FK)	*int*	10	Identificador de estado civil do paciente
paciente_profissao_id (FK)	*int*	10	Identificador de profissão do paciente
paciente_religiao_id (FK)	*int*	10	Identificador de religião do paciente
paciente_escolaridade_id (FK)	*int*	10	Identificador de escolaridade do paciente
paciente_encaminhado_por_id (FK)	*int*	10	Identificador de quem encaminhou o paciente
paciente_como_conheceu_id (FK)	*int*	10	Identificador de como o paciente chegou até o psicólogo
data_criacao	*datetime*	-	Data de criação do paciente
data_atualizacao	*datetime*	-	Data de atualização do paciente
data_exclusao	*datetime*	-	Data de exclusão do paciente
conta_id (FK)	*int*	10	Identificador da conta no sistema
empresa_id (FK)	*int*	10	Identificador da empresa no sistema
usuario_id (FK)	*int*	10	Identificador do usuário no sistema

Quadro 14 - Dicionário de dados da tabela "evento"

EVENTO			
Tabela responsável por armazenar todos os eventos do módulo agenda.			
Nome do atributo	Tipo	Tamanho	Descrição
id (PK)	*int*	10	Identificador único
data_inicio	*datetime*	-	Data de início do evento

data_termino	*datetime*	-	Data de término no evento
dia_inteiro	*boolean*	1	Indica se o evento possui duração de dia inteiro
titulo	*varchar*	100	Título do evento
local	*varchar*	100	Informações sobre o local do evento
descricao	*varchar*	400	Notas do evento
recorrencia_contagem	*int*	11	Indica a quantidade de repetições que um evento terá
periodo_frequencia	*int*	11	Indica a frequência em que o evento será repetido
evento_periodo_id (FK)	*int*	10	Indica o tipo de recorrência do evento (diário, semanal, mensal, anual)
is_usar_data_termino	*boolean*	1	Indica se o usuário selecionou uma data de término na escolha da recorrência
is_evento_sistema	*boolean*	1	Indica se é um evento criado pelo sistema
evento_categoria_id (FK)	*int*	10	Categoria (agenda) ao qual o evento está vinculado
paciente_id (FK)	*int*	10	Paciente ao qual o evento está vinculado
conta_id (FK)	*int*	10	Identificador da conta no sistema
usuario_id (FK)	*int*	10	Identificador do usuário no sistema
data_criacao	*datetime*	-	Data de criação do usuário
data_atualizacao	*datetime*	-	Data de atualização do usuário

Quadro 15 - Dicionário de dados da tabela "evento_excecao"

EVENTO_EXCECAO			
Tabela responsável por armazenar todas as exceções de determinada ocorrência de um evento, desta forma, se um evento for recorrente e tiver na ocorrência 5 o título alterado, será nesta tabela que a exceção da ocorrência 5 será gravada.			
Nome do atributo	Tipo	Tamanho	Descrição
evento_id (PK)	*int*	10	Identificador de um evento
evento_ocorrencia_id (PK)	*int*	10	Identificador de uma ocorrência específica do evento
is_especializado	*boolean*	1	Indica se o evento é especializado em relação ao evento principal

is_cancelado	*boolean*	1	Indica se a ocorrência do evento está cancelada
data_inicio	*datetime*	-	Data de início da exceção do evento
data_termino	*datetime*	-	Data de término da exceção do evento
dia_inteiro	*boolean*	1	Indica se a exceção do evento possui duração de dia inteiro
titulo	*varchar*	100	Título da exceção evento
local	*varchar*	100	Informações sobre o local da exceção evento
descricao	*varchar*	400	Notas da exceção do evento
usuario_id (FK)	*int*	10	Identificador do usuário no sistema
conta_id (FK)	*int*	10	Identificador da conta no sistema
data_criacao	*datetime*	-	Data de criação da exceção do evento
data_atualizacao	*datetime*	-	Data de atualização da exceção do evento

Quadro 16 - Dicionário de dados da tabela "evento_lembrete_disparo_fila"

EVENTO_LEMBRETE_DISPARO_FILA			
Tabela responsável por armazenar todos os disparos de lembretes que terão que ser feitos pelo sistema.			
Nome do atributo	Tipo	Tamanho	Descrição
id (PK)	*int*	10	Identificador único
data_agendado	*datetime*	-	Data em que o disparo está agendado para ser enviado
data_inicio_evento	*datetime*	-	Data em que o evento irá iniciar (utilizado na mensagem disparada)
data_envio	*datetime*	-	Data em que o disparo foi efetivamente enviado
metodo	*enum*	-	Método que será utilizado para o disparo, pode ser SMS ou EMAIL
destinatario	*varchar*	150	Destinatário que receberá o disparo, pode ser um número de celular ou *e-mail*
data_criacao	*datetime*	-	Data de criação do disparo
evento_lembrete_id	*int*	10	Lembrete ao qual este disparo está

(FK)			vinculado
evento_ocorrencia_id (FK)	*int*	10	Ocorrência do evento ao qual o disparo pertence
evento_id (FK)	*int*	10	Evento ao qual o disparo pertence
paciente_id (FK)	*int*	10	Paciente ao qual o disparo está vinculado
usuario_id (FK)	*int*	10	Identificador do usuário no sistema
conta_id (FK)	*int*	10	Identificador da conta no sistema

APÊNDICE C – Comparação de horas previstas e realizadas

Na Figura 63 é apresentado o comparativo de horas previstas no início do projeto de desenvolvimento e o tempo que foi efetivamente trabalhado em cada atividade. O total de atraso foi de 51%.

Figura 63 - Comparação de horas previstas e realizadas

		EDT	Nome da tarefa	Duração prevista	Duração real	% concluída
1	✓	**1**	**Gestor Psicologia**	**481 hrs**	**727 hrs**	**100%**
2	✓	**1.1**	**Gerência do projeto**	**44 hrs**	**68 hrs**	**100%**
3	✓	**1.1.1**	**Planejamento**	**16 hrs**	**40 hrs**	**100%**
4	✓	1.1.1.1	Elaborar plano de projeto	16 hrs	40 hrs	100%
5	✓	**1.1.2**	**Reunião**	**28 hrs**	**28 hrs**	**100%**
6	✓	1.1.2.1	Reunião 1	4 hrs	4 hrs	100%
7	✓	1.1.2.2	Reunião 2	4 hrs	4 hrs	100%
8	✓	1.1.2.3	Reunião 3	4 hrs	4 hrs	100%
9	✓	1.1.2.4	Reunião 4	4 hrs	4 hrs	100%
10	✓	1.1.2.5	Reunião 5	4 hrs	4 hrs	100%
11	✓	1.1.2.6	Reunião 6	4 hrs	4 hrs	100%
12	✓	1.1.2.7	Reunião 7	4 hrs	4 hrs	100%
13	✓	**1.2**	**Levantamento de requisitos**	**6 hrs**	**9 hrs**	**100%**
14	✓	1.2.1	Entrevista de entendimento	2 hrs	6 hrs	100%
15	✓	1.2.2	Documento de requisitos	4 hrs	3 hrs	100%
16	✓	**1.3**	**Análise e projeto**	**134 hrs**	**168 hrs**	**100%**
17	✓	1.3.1	Prototipação das telas	60 hrs	80 hrs	100%
18	✓	1.3.2	Definição de arquitetura	8 hrs	16 hrs	100%
19	✓	1.3.3	Criação dos diagramas de casos de uso	16 hrs	16 hrs	100%
20	✓	1.3.4	Descrição dos casos de uso	30 hrs	32 hrs	100%
21	✓	1.3.5	Modelo de banco de dados	20 hrs	24 hrs	100%
22	✓	**1.4**	**Desenvolvimento**	**281 hrs**	**393 hrs**	**100%**
23	✓	1.4.1	Site (Recursos básicos)	40 hrs	42 hrs	100%
24	✓	1.4.2	Site (Planos / Usuários / Login)	60 hrs	64 hrs	100%
25	✓	1.4.3	Módulo de pacientes	50 hrs	85 hrs	100%
26	✓	1.4.4	Módulo de agenda	86 hrs	141 hrs	100%
27	✓	1.4.5	Módulo de configurações / Planos	45 hrs	61 hrs	100%
28	✓	**1.5**	**Implantação**	**16 hrs**	**21 hrs**	**100%**
29	✓	1.5.1	Registro do domínio	1 hrs	2 hrs	100%
30	✓	1.5.2	Servidor de aplicação / web	8 hrs	8 hrs	100%
31	✓	1.5.3	Banco de dados	4 hrs	8 hrs	100%
32	✓	1.5.4	Instalação e configuração no servidor	3 hrs	3 hrs	100%