REQUIREMENTS DOCUMENT

PetSpeed

1 Project Overview

Integrated project with the discipline of Modeling and Object Oriented Programming.

1.1 Product Objective

The PetSpeed project aims to innovate in the market of pets, with a series of proposals until then nonexistent. According to a study published by LIDE magazine, the Brazilian pet market is the third largest in the world in terms of revenues, and is still against the crisis that plagues the country. Identifying opportunities to implement a platform that facilitates the processes of veterinarian care and pre-screening your pet.

1.2 Problems to solve

The conventional method of receiving veterinary care is the displacement of the pet to the clinic or pet shop which also offers the clinic service. However, it is possible to introduce a new way of operating in this segment.

Many animals feel uncomfortable on public transport and even in private cars, while others, in old age, complicate the procedure in question. In other cases, the problem is with the person in charge of the animal, who may not be able to go to the clinic or pet shop at the time. Still depending on other factors, distance and lack of knowledge are obstacles.

In extreme situations, such as emergencies and emergencies, due to the situation in question it becomes difficult to transport the animals in conventional vehicles, to an appropriate veterinary clinic.

1.3 Solutions

In view of the above, PetSpeed can solve all problems, even partially. The proposal is to solve all the problems in question, with an alternative mode of clinical-veterinary care.

The application proposes to take the service to the client, where the veterinarian is willing to go to the home of the animal that is in need of some clinical service. In this way, the impossibility of transportation due to difficulties of the animal responsible can be solved, since the service can be provided by the doctor who was called by the application.

For urgent and emergency issues, the client may request a pre-screening for his pet, also with a qualified doctor.

2 Project Scope

The work will be divided into 2 main parts, they are, scope definition and scope partitioning.

2.1 Scope definition

- Project Planning and Management
- Survey of Requirements
- Front-End Development
  Site Creation
  Development of the screens
- Back-End Development
  Functionalities Development
  Creating the Database
- Closed tests with developers
  Test Data Analysis
  Fixing Problems
- Open Tests with Users
  Test Data Analysis
  Fixing Problems
- Acquisition of the Developer License for Publication

2.2 Scope Partitioning

Initiation	• Project planning and management; • Requirements gathering.
Acquisitions	• Acquisition of the developer license for publication in the official Google store (Google Play).
Development	• Creating visual items; • Software development;
Tests	• Direct internal testing with developers to solve critical problems; • Direct external testing with users to solve specific problems; • Managing the data obtained; • Correction of identified problems.
Publication	• App publishing in the official Android store, Google Play.

3 Functional requirements (use cases)

3.1 Registration

Create user:

- Description of the use case: This use case allows the user to be created and stored as a new user in the system.

Change user information:

- Description of the use case: It allows the user to change their information made in the register, accessing the profile and their respective settings and having access to the change of their personal data.

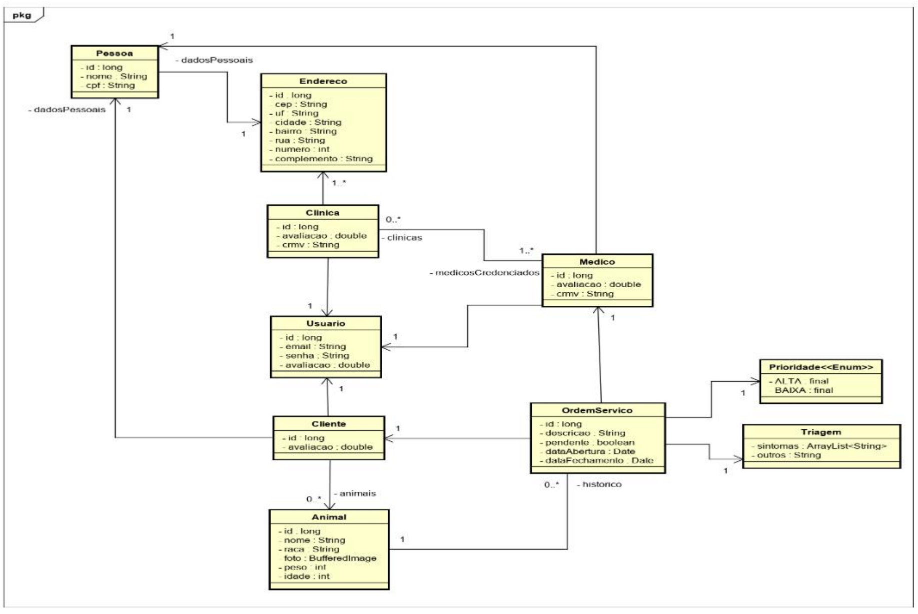
Get medical attention:

- Description of use case: This case allows access to the option to choose what kind of care the user will need.

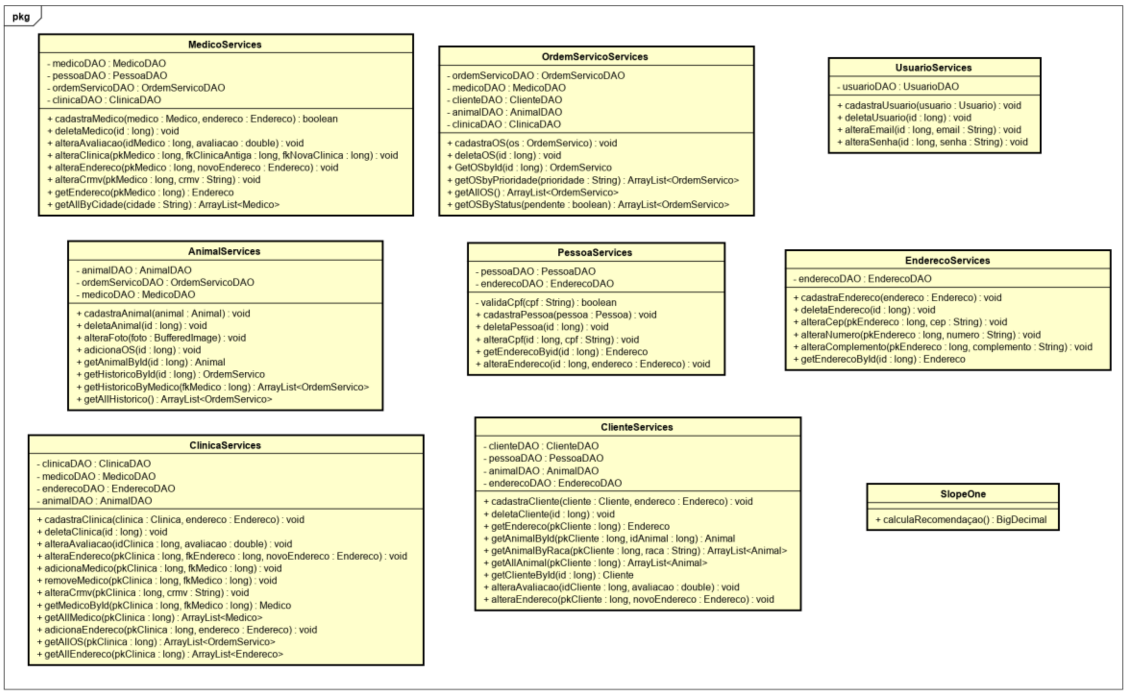
4 UML diagrams

Unified Modeling Language, establishes a common, semantically and syntactically rich visual for architecture, design and implementation of complex software systems. We will use the object-oriented model, the algorithms will be expressed through the definition of "objects". The diagrams presented will be:

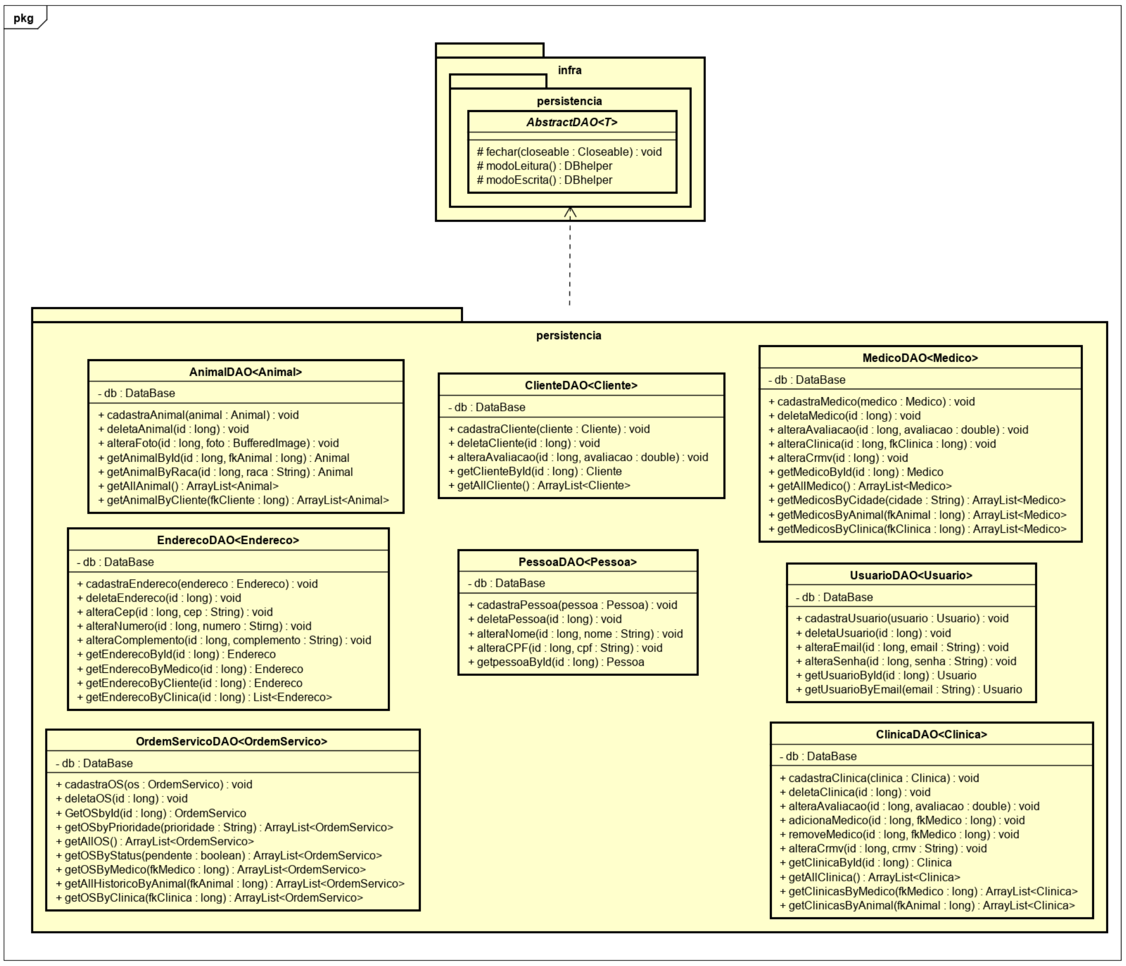
Domain Class Diagram: A model that incorporates behavior and data.



Business Class Diagram: Define the rules of your business.


pkg
MedicoServices
- medicoDAO : MedicoDAO
- pessoaDAO : PessoaDAO
- ordemServicoDAO : OrdemServicoDAO
- clinicaDAO : ClinicaDAO
+ cadastraMedico(medico : Medico, endereco : Endereco) : boolean
+ deletaMedico(id : long) : void
+ alteraAvaliacao(idMedico : long, avaliacao : double) : void
+ alteraClinica(pkMedico : long, fkClinicaAntiga : long, fkNovaClinica : long) : void
+ alteraEndereco(pkMedico : long, novoEndereco : Endereco) : void
+ alteraCrmv(pkMedico : long, crmv : String) : void
+ getEndereco(pkMedico : long) : Endereco
+ getAllByCidade(cidade : String) : ArrayList<Medico>
OrdemServicoServices
- ordemServicoDAO : OrdemServicoDAO
- medicoDAO : MedicoDAO
- clienteDAO : ClienteDAO
- animalDAO : AnimalDAO
- clinicaDAO : ClinicaDAO
+ cadastraOS(os : OrdemServico) : void
+ deletaOS(id : long) : void
+ GetOSbyId(id : long) : OrdemServico
+ getOSbyPrioridade(prioridade : String) : ArrayList<OrdemServico>
+ getAllOS() : ArrayList<OrdemServico>
+ getOSByStatus(pendente : boolean) : ArrayList<OrdemServico>
UsuarioServices
- usuarioDAO : UsuarioDAO
+ cadastraUsuario(usuario : Usuario) : void
+ deletaUsuario(id : long) : void
+ alteraEmail(id : long, email : String) : void
+ alteraSenha(id : long, senha : String) : void
AnimalServices
- animalDAO : AnimalDAO
- ordemServicoDAO : OrdemServicoDAO
- medicoDAO : MedicoDAO
+ cadastraAnimal(animal : Animal) : void
+ deletaAnimal(id : long) : void
+ alteraFoto(foto : BufferedImage) : void
+ adicionaOS(id : long) : void
+ getAnimalById(id : long) : Animal
+ getHistoricoById(id : long) : OrdemServico
+ getHistoricoByMedico(fkMedico : long) : ArrayList<OrdemServico>
+ getAllHistorico() : ArrayList<OrdemServico>
PessoaServices
- pessoaDAO : PessoaDAO
- enderecoDAO : EnderecoDAO
- validaCpf(cpf : String) : boolean
+ cadastraPessoa(pessoa : Pessoa) : void
+ deletaPessoa(id : long) : void
+ alteraCpf(id : long, cpf : String) : void
+ getEnderecoByid(id : long) : Endereco
+ alteraEndereco(id : long, endereco : Endereco) : void
EnderecoServices
- enderecoDAO : EnderecoDAO
+ cadastraEndereco(endereco : Endereco) : void
+ deletaEndereco(id : long) : void
+ alteraCep(pkEndereco : long, cep : String) : void
+ alteraNumero(pkEndereco : long, numero : String) : void
+ alteraComplemento(pkEndereco : long, complemento : String) : void
+ getEnderecoById(id : long) : Endereco
ClinicaServices
- clinicaDAO : ClinicaDAO
- medicoDAO : MedicoDAO
- enderecoDAO : EnderecoDAO
- animalDAO : AnimalDAO
+ cadastraClinica(clinica : Clinica, endereco : Endereco) : void
+ deletaClinica(id : long) : void
+ alteraAvaliacao(idClinica : long, avaliacao : double) : void
+ alteraEndereco(pkClinica : long, fkEndereco : long, novoEndereco : Endereco) : void
+ adicionaMedico(pkClinica : long, fkMedico : long) : void
+ removeMedico(pkClinica : long, fkMedico : long) : void
+ alteraCrmv(pkClinica : long, crmv : String) : void
+ getMedicoById(pkClinica : long, fkMedico : long) : Medico
+ getAllMedico(pkClinica : long) : ArrayList<Medico>
+ adicionaEndereco(pkClinica : long, endereco : Endereco) : void
+ getAllOS(pkClinica : long) : ArrayList<OrdemServico>
+ getAllEndereco(pkClinica : long) : ArrayList<Endereco>
ClienteServices
- clienteDAO : ClienteDAO
- pessoaDAO : PessoaDAO
- animalDAO : AnimalDAO
- enderecoDAO : EnderecoDAO
+ cadastraCliente(cliente : Cliente, endereco : Endereco) : void
+ deletaCliente(id : long) : void
+ getEndereco(pkCliente : long) : Endereco
+ getAnimalById(pkCliente : long, idAnimal : long) : Animal
+ getAnimalByRaca(pkCliente : long, raca : String) : ArrayList<Animal>
+ getAllAnimal(pkCliente : long) : ArrayList<Animal>
+ getClienteById(id : long) : Cliente
+ alteraAvaliacao(idCliente : long, avaliacao : double) : void
+ alteraEndereco(pkCliente : long, novoEndereco : Endereco) : void
SlopeOne
+ calculaRecomendaçao() : BigDecimal

Persistence Class Diagram: Permanently preserves objects.


pkg
infra
persistencia
AbstractDAO<T>
# fechar(closeable : Closeable) : void
# modoLeitura() : DBHelper
# modoEscrita() : DBHelper
persistencia
AnimalDAO<Animal>
- db : DataBase
+ cadastraAnimal(animal : Animal) : void
+ deletaAnimal(id : long) : void
+ alteraFoto(id : long, foto : BufferedImage) : void
+ getAnimalById(id : long, fkAnimal : long) : Animal
+ getAnimalByRaca(id : long, raca : String) : Animal
+ getAllAnimal() : ArrayList<Animal>
+ getAnimalByCliente(fkCliente : long) : ArrayList<Animal>
ClienteDAO<Cliente>
- db : DataBase
+ cadastraCliente(cliente : Cliente) : void
+ deletaCliente(id : long) : void
+ alteraAvaliacao(id : long, avaliacao : double) : void
+ getClienteById(id : long) : Cliente
+ getAllCliente() : ArrayList<Cliente>
MedicoDAO<Medico>
- db : DataBase
+ cadastraMedico(medico : Medico) : void
+ deletaMedico(id : long) : void
+ alteraAvaliacao(id : long, avaliacao : double) : void
+ alteraClinica(id : long, fkClinica : long) : void
+ alteraCrmv(id : long) : void
+ getMedicoById(id : long) : Medico
+ getAllMedico() : ArrayList<Medico>
+ getMedicosByCidade(cidade : String) : ArrayList<Medico>
+ getMedicosByAnimal(fkAnimal : long) : ArrayList<Medico>
+ getMedicosByClinica(fkClinica : long) : ArrayList<Medico>
EnderecoDAO<Endereco>
- db : DataBase
+ cadastraEndereco(endereco : Endereco) : void
+ deletaEndereco(id : long) : void
+ alteraCep(id : long, cep : String) : void
+ alteraNumero(id : long, numero : Stirng) : void
+ alteraComplemento(id : long, complemento : String) : void
+ getEnderecoById(id : long) : Endereco
+ getEnderecoByMedico(id : long) : Endereco
+ getEnderecoByCliente(id : long) : Endereco
+ getEnderecoByClinica(id : long) : List<Endereco>
PessoaDAO<Pessoa>
- db : DataBase
+ cadastraPessoa(pessoa : Pessoa) : void
+ deletaPessoa(id : long) : void
+ alteraNome(id : long, nome : String) : void
+ alteraCPF(id : long, cpf : String) : void
+ getpessoaById(id : long) : Pessoa
UsuarioDAO<Usuario>
- db : DataBase
+ cadastraUsuario(usuario : Usuario) : void
+ deletaUsuario(id : long) : void
+ alteraEmail(id : long, email : String) : void
+ alteraSenha(id : long, senha : String) : void
+ getUsuarioById(id : long) : Usuario
+ getUsuarioByEmail(email : String) : Usuario
OrdemServicoDAO<OrdemServico>
- db : DataBase
+ cadastraOS(os : OrdemServico) : void
+ deletaOS(id : long) : void
+ GetOSbyId(id : long) : OrdemServico
+ getOSbyPrioridade(prioridade : String) : ArrayList<OrdemServico>
+ getAllOS() : ArrayList<OrdemServico>
+ getOSByStatus(pendente : boolean) : ArrayList<OrdemServico>
+ getOSByMedico(fkMedico : long) : ArrayList<OrdemServico>
+ getAllHistoricoByAnimal(fkAnimal : long) : ArrayList<OrdemServico>
+ getOSByClinica(fkClinica : long) : ArrayList<OrdemServico>
ClinicaDAO<Clinica>
- db : DataBase
+ cadastraClinica(clinica : Clinica) : void
+ deletaClinica(id : long) : void
+ alteraAvaliacao(id : long, avaliacao : double) : void
+ adicionaMedico(id : long, fkMedico : long) : void
+ removeMedico(id : long, fkMedico : long) : void
+ alteraCrmv(id : long, crmv : String) : void
+ getClinicaById(id : long) : Clinica
+ getAllClinica() : ArrayList<Clinica>
+ getClinicasByMedico(fkMedico : long) : ArrayList<Clinica>
+ getClinicasByAnimal(fkAnimal : long) : ArrayList<Clinica>

5 Software Development Plan

Day 30/04/2019	Requeriments Document;
Day 21/05/2019	Requirements Documents with UML diagrams;
Day 28/05/2019	Architecture Document;
Day 04/05/2019	Start requirements interface document;

6 “R-E-A-L FU-CK-IN-G laws”

Code is law, is there any universal laws for software engineering? What was proposed in the video is that yes, there are universal laws that are applied in software engineering and can not be questioned.

Fundamental laws for all systems:

FU: Any system, be it large or small, must have provisions that allow it to be corrected and updated in or out of the system.

CK: Every system, large or small, must have provisions that allow the system to acquire and make use of the contextual knowledge and decide whether it is working properly or not.

The fuck property: Any fu-ck-compliant system can check the conditions of its operating environment and decide whether or not its actions are consistent with the current context of that environment. If he decides that they are not coherent, they can self-repair, can call for outside help, and can be repaired by someone outside the intervention.

IN: You have to have provisions that allow you to be independent of the network as possible, it should work independently of the internet, can not hang at once, in the absence of the network must have certain degradation.

G: In general, for all types of systems, security and integrity should be treated with a much greater importance than functionality, which in turn has to be balanced with usability.

And in the real world:

A: reused in different contexts and different systems;

E: extended to accommodate new behaviors;

A: analytics measure what is happening and how it is evolving;

L: reduce both independence and dependencies with other systems that they potentially interact with.

6.1 Relation to the stated properties

The proposed system meets the fuck property, the moment it identifies an error in its context it may request an external correction. In addition, it obeys the law "IN", if the internet stops working, the system will not lock and will lose a certain function gradually. In the laws of the "R-E-A-L" world, the system obeys primarily "A" law, for it is always measuring what happens and constantly evolves.