Documento de Arquitetura

Sumário

1 – Introdução

Este documento apresenta uma visão geral abrangente da arquitetura do sistema e utiliza uma série de visões arquiteturais diferentes para ilustrar os diversos aspectos do sistema. Sua intenção é capturar e transmitir as decisões significativas do ponto de vista da arquitetura que foram tomadas em relação ao sistema.

2 – Representação da arquitetura

Este documento apresenta a arquitetura como uma série de visões: visão de casos de uso, visão de processos, visão de implantação e visão de implementação. Essas visões utilizam a Linguagem Unificada de Modelagem (UML).

3 – Visão de casos de uso

Uma descrição da visão de casos de uso da arquitetura de software. A Visão de Casos de Uso é uma entrada importante para a seleção do conjunto de cenários e/ou casos de uso que são o foco de uma iteração. Ela descreve o conjunto de cenários e/ou os casos de uso que representam alguma funcionalidade central e significativa. Também descreve o conjunto de cenários e/ou casos de uso que possuem cobertura arquitetural substancial (que experimenta vários elementos de arquitetura) ou que enfatizam ou ilustram um determinado ponto complexo da arquitetura.

Os casos de uso listados a seguir são muito importantes para a arquitetura. Uma descrição desses casos de uso pode ser encontrada posteriormente nesta seção.

- UC – 02 Manter Roteiro
- UC – 05 Manter Destino
- UC – 11 Manter Programa
- UC – 10 Pesquisar no sistema

O diagrama a seguir representa os casos de uso do sistema:

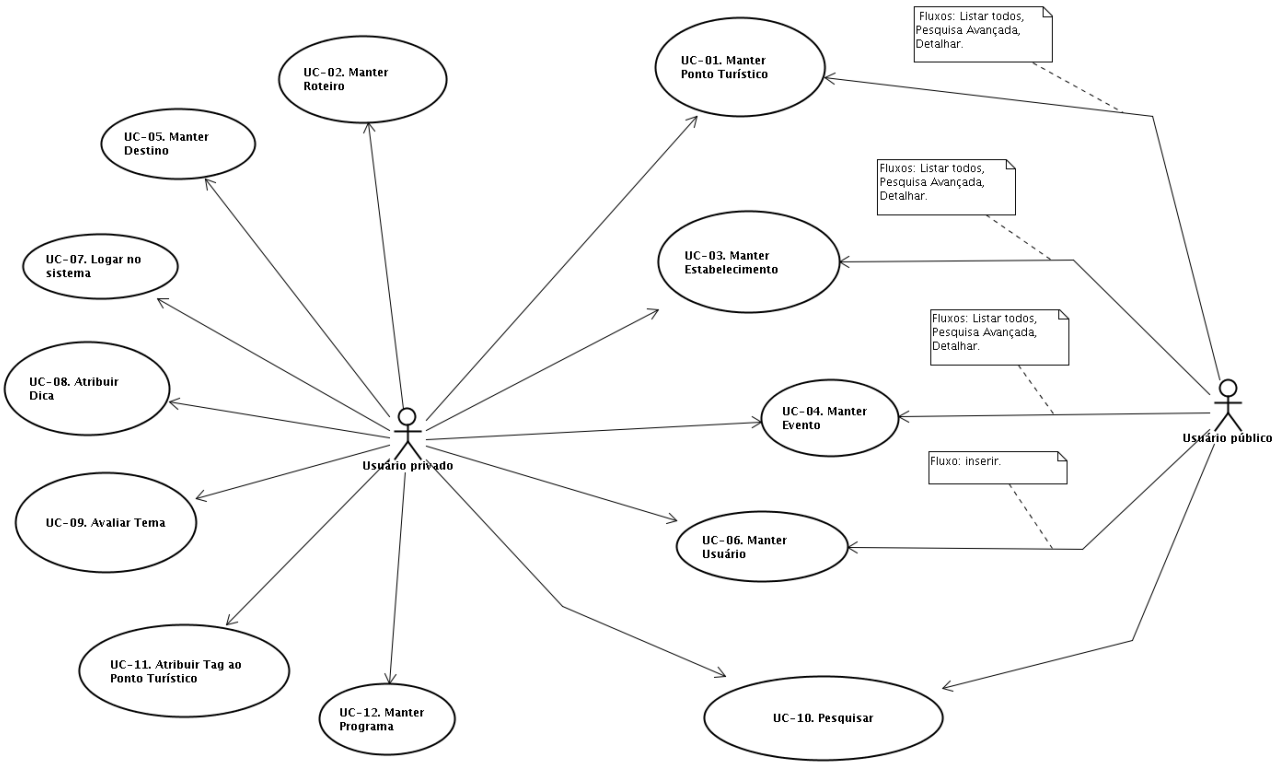

Figura 1: Modelo de casos de uso

Descrição dos casos de uso mais importantes:

UC – 02 Manter Roteiro:

Este Caso de Uso ocorre quando um viajante deseja planejar uma viajem. O primeiro passo para o planejamento de uma viajem é a criação de um roteiro, que irá localizar o ponto de partida e também irá armazenar uma descrição da viajem além de servir de agregador para os outros conteúdos do planejamento.

UC – 05 Manter Destino:

Este Caso de Uso ocorre após a criação de um roteiro de viajem, ele irá permitir ao viajante adicionar destinos ao roteiro criado, estabelecendo um custo planejado para cara destino adicionado. Este caso de uso também provê um cálculo do custo da viajem, baseado nos destinos adicionados, e o cálculo do período da viajem.

10 Pesquisar no sistema:

Este Caso de Uso ocorre quando um viajante deseja procurar alguma coisa no sistema. Ele realiza uma pesquisa em todas as entidades do sistema, recuperando para um mesmo termo informações sobre estabelecimentos, pontos turísticos, roteiros de outros usuários, eventos, etc.

UC – 11 Manter Programa:

Este Caso de Uso ocorre após à adição de um destino ao roteiro de viajem, ele irá permitir ao viajante adicionar programas ao destino escolhido, estabelecendo o custo e a data do programa. Este caso de uso também permite que o programa seja associado a um estabelecimento ou ponto turístico cadastrado no sistema e provê um cálculo do custo dos programas, além de um saldo daquele destino.

4 – Visão lógica

A descrição da visão lógica da arquitetura. Descreve as classes mais importantes, sua organização em pacotes e subsistemas de serviço, e a organização desses subsistemas em camadas. Descreve também as realizações de caso de uso mais importantes como, por exemplo, os aspectos dinâmicos da arquitetura. Os diagramas de classe podem ser incluídos para ilustrar os relacionamentos entre as classes, os subsistemas, os pacotes e as camadas arquiteturalmente significativas.

O sistema será desenvolvido utilizando o padrão de projetos arquitetural MVC (Model View Controller). A camada de Controller será responsável por coordenar as ações do usuário e realizar alterações no modelo. A camada de Model armazenará as classes que representam o domínio da aplicação que conterão as regras de negócio, além de abstrair a camada de persistência. A camada de View irá armazenar a interface gráfica do sistema.

Utilizaremos outra camada, chamada de Helper, que conterá operações utilitárias a todo o sistema ou a um determinado modelo.

A utilização dessas camadas define a forma como os componentes irão se comunicar, separando as responsabilidades de cada componente. A escolha deste padrão de projeto deve-se ao fato de sua eficacia estar comprovada e facilitar a manutenção do software.

A visão lógica é composta por 4 pacotes principais:

- **Controllers**
  - Contém classes para cada controlador da aplicação, funcionando com um centralizador e controlador das requisições do sistema.
- **Models**
  - Contém todos modelos de negócio da aplicação, o domínio.
- **Views**
  - Contém todas as interfaces utilizadas pelos atores para se comunicar com o sistema.
- **Helpers**

- Contém classes de suporte a geração da interface, conversão de dados, etc.

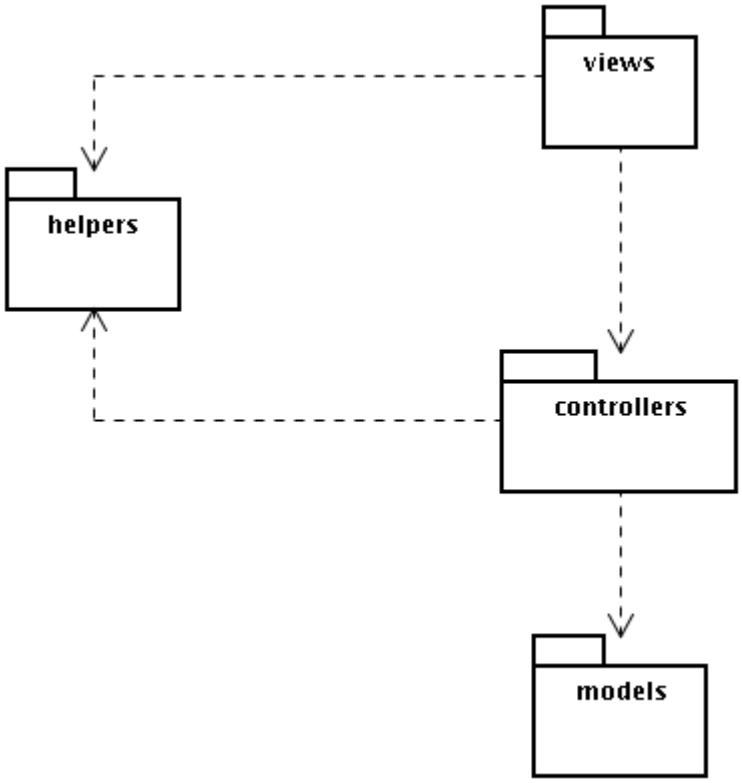

Figura 2: Diagrama de pacotes

5 – Visão de implantação

Esta seção descreve uma ou mais configurações (hardware) de rede física nas quais o software será implantado e executado. Para cada configuração, ela deve indicar no mínimo os nós físicos (computadores, CPUs) que executam o software e as respectivas interconexões (barramento, LAN, ponto a ponto e assim por diante.) Além disso, ela inclui um mapeamento dos processos da Visão de Processos nos nós físicos.

O servidor deverá ser uma plataforma UNIX, contendo o servidor web Apache com o módulo capistrano instalado, que é o módulo responsável por executar aplicações escritas na linguagem Ruby. A máquina cliente é qualquer dispositivo capaz de executar um navegador da Web e conectar-se ao servidor via Internet. O banco de dados deverá rodar em uma plataforma UNIX e possuir o SGBD PostgreSQL 8.3.

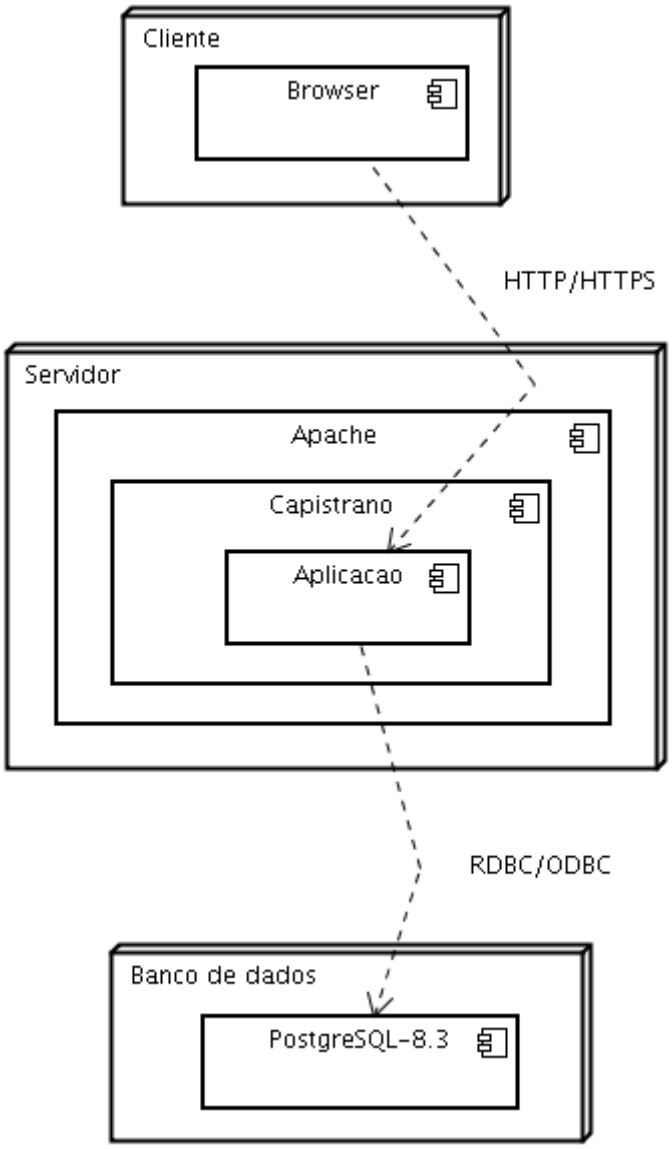

Figura 3: Diagrama de implantação

6 – Visão de implementação

O projeto utilizará o framework MVC Rails 2.3+, que é open source e foi desenvolvido em linguagem de programação Ruby. O Rails auxilia na utilização do MVC fornecendo uma estrutura de fácil entendimento e evolução, não permitindo furos nas camadas determinadas.

A escolha do framework foi baseada nas seguintes premissas:

Produtividade: Ele possui mecanismos para prototipação rápida, gerando controladores e interfaces gráficas a partir de um modelo; como foi desenvolvido sobre uma linguagem de script, não compilada, não é necessário parar o servidor de aplicação em nenhum momento; possui mecanismos nativos para criação de testes automáticos; possui um motor de templates nativo para auxiliar na criação das interfaces gráficas.

Facilidade de evolução: O Rails utiliza o conceito de convenção sobre configuração, o que reduz consideravelmente os passos necessários para adicionar novos recursos.

Conhecimento da equipe: Dos membros da equipe, apenas duas pessoas possuem experiência com desenvolvimento de software e a curva de aprendizado do Ruby e do Rails é consideravelmente menor que a dos outros frameworks e linguagens.

Material disponível: O framework foi criado em 2003 e rapidamente se popularizou, por isso possui uma grande quantidade de livros e artigos a seu respeito.

O Rails é um meta-framework, um agregado de quatro outros frameworks, que são: ActionPack; ActionSupport; ActionRecord; ActionMailer. Dentre os frameworks analisados este foi o melhor na categoria dos meta-frameworks, possuindo baixa curva de aprendizado e simplicidade em sua utilização.

O ActiveRecord é o framework de persistência, mapeamento objeto relacional, que irá abstrair o banco de dados, mapeando as tabelas para o modelo de criado, permitindo a manipulação dos dados sem o uso explicito de SQL. O ActionPack realizará a separação e organização estrutural do projeto, coordenando o uso do MVC e gerenciando o uso da interface gráfica. Ele também possui um motor de templates, para auxiliar na criação das interfaces gráficas. O ActionMailer fornecerá recursos para envio e recebimento de emails. O ActionSupport provê diversos recursos utilitários para todo o sistema.

Os casos de teste serão feitos de forma automatizada, através do framework de testes RTest. Um teste automático é um programa que verifica se um outro programa está funcionando conforme o esperado. O RTest provê recursos para criação de testes de unidade, integração e funcionais, realizando as verificações programadas automaticamente. Os testes automáticos são importantes pois com eles conseguimos sempre verificar o funcionamento de uma determinada funcionalidade, realizando revisões constantes e mantendo todo o sistema verificado.

Um recurso bastante utilizado será o de inclusão de módulos do Ruby, também chamado de “mistura”. Uma mistura, ou *mixin*, assemelha-se a herança múltipla presente em linguagens como C++. Quando um módulo é incluso quaisquer objetos criados a partir da classe poderão usar os métodos de instância do módulo incluso, como se tivessem sido definidos na própria classe, permitindo inclusive a sobrescrita desses métodos.

A documentação interna será feita através de comentários de bloco logo acima da declaração dos métodos e classes, explicitando os parâmetros quando necessário.

7 – Visão de dados

Será utilizado o SGBD objeto relacional open source PostgreSQL 8.3+. O PostgreSQL é um dos SGBDs open source mais avançados da atualidade.

8 – Tamanho e Desempenho

O software conforme projetado suportará 100 usuários simultâneos. A escala além deste nível pode ser obtida através da adição de outros níveis do servidor web e do servidor de dados.

9 – Qualidade

O software conforme projetado provê extensibilidade, confiabilidade, portabilidade, manutenibilidade, testabilidade, reusabilidade e interoperabilidade.