**Sistema Integrado Gestor de Eventos**

**(SIGE)**

**GLMR Desenvolvimentos**

Cruz das Almas

10/2020

Índice

[1. Introdução 3](#__RefHeading___Toc1221_2662037735)

[1.2. Proposito 3](#__RefHeading___Toc1223_2662037735)

[1.3. Escopo 3](#__RefHeading___Toc1225_2662037735)

[2. Arquitetura do Sistema 3](#__RefHeading___Toc1293_2662037735)

[2.1. Representação da Arquitetura 3](#__RefHeading___Toc1291_2662037735)

[2.2. Critérios da Avaliação Arquitetural 3](#__RefHeading___Toc1289_2662037735)

[2.3. Decisões Arquiteturais 4](#__RefHeading___Toc1277_2662037735)

[2.3.1. Modularização e Reutilização 4](#__RefHeading___Toc1275_2662037735)

[2.3.2. Tecnologias e Linguagens 4](#__RefHeading___Toc1273_2662037735)

[2.3.3. Segurança 5](#__RefHeading___Toc1271_2662037735)

[2.3. Visões de Caso de Uso 6](#__RefHeading___Toc1269_2662037735)

[2.4. Visão Lógica 8](#__RefHeading___Toc1259_2662037735)

[2.5. Visão da Implementação 9](#__RefHeading___Toc1255_2662037735)

[2.5.1. Modelagem de Classes 10](#__RefHeading___Toc1249_2662037735)

[2.6. Visão de Implantação 10](#__RefHeading___Toc1247_2662037735)

Cruz das Almas

10/2020

1. Introdução
   1. Proposito

Este documento apresenta a proposta de arquitetura do Sistema Integrado Gestor de Eventos trazendo visões que buscam cobrir os aspectos técnicos e lógicos relativo ao desenvolvimento da aplicação. O objetivo é evidenciar as principais decisões tomada durante sua elaboração e desenvolvimento.

* 1. Escopo

O escopo deste documento visa documentar o Sistema Integrado Gestor de Eventos do ponto de vista da arquitetura e suas divisões.

1. Arquitetura do Sistema

Esta seção detalha as características da arquitetura e as alternativas arquiteturais avaliadas e suas aplicações e impactos.

* 1. Representação da Arquitetura

Apresentamos uma série de visões: Casos de Uso, Lógica, Implementação e Implantação. Utilizamos modelos UML para melhor visualização.

* 1. Critérios da Avaliação Arquitetural
* Disponibilidade

Como o sistema é online para cadastro em eventos, sua disponibilidade prioritária é durante o período de eventos. Porém é desejável que esteja funcionando 24h por dia para qualquer que seja o uso.

* Escalabilidade

O software deve ser capaz de atingir 150mil usuários, podendo ser simultâneos ou não.

* Confiabilidade

Tomamos decisões quanto a segurança para evitar ataques e invasões ao sistema. Todo o código e documentos organizados para melhor manutenibilidade e assim evitar erros e travamentos Além de tomar medidas para que o sistema possa escalar de forma a suportar o maior número de inscrito simultaneamente. Buscamos oferecer um sistema que seja suportado em todos os navegadores e tamanhos de tela.

* Segurança

O software deve ser seguro para trânsito de informações e não comprometer a usabilidade e a sua eficiência.

* Eficiência

Utilizar interfaces minimalistas disponibilizando informações relevantes e melhor conexão com os servidores.

* 1. Decisões Arquiteturais
     1. Modificações Durante Implementação
* Troca de Banco de Dados

Originalmente foi escolhido MySQL para o banco de dados central da aplicação.

MySQL é um banco de dados relacional amplamente utilizado no mercado nos dias atuais. Possui uma licença dupla, parte liberada para uso não comercial de larga escala, parte proprietária. Porém sua procura foi reduzida com passar do tempo. Visando um projeto totalmente open source, MariaDB se mostrou uma melhor opção.

MariaDB é um banco de dados relacional, fork do MySQL, de licença GPL open source e pode ser amplamente utilizado e redistribuído. Além disso, outras vantagens se sobressaíram como facilidade de instalação, a não dependência de arquivos de terceiros, e possuir binários significativamente mais leves em relação ao MySQL.

MariaDB possui um suporte ativo e é totalmente compatível com as tecnologias escolhidas para a implementação. Em termos de códigos de configuração, não há diferença entre as configurações de conexão entre MariaDB e MySQL. Ambas são iguais.

* + 1. Modularização e Reutilização

Visando uma reutilização de código para possíveis futuros projetos, e uma manutenibilidade facilitada, a melhor escolha para o padrão do projeto é a estrutura modularizada em Model-View-Controller (MVC).

MVC é um padrão de arquitetura de software que separa a aplicação em três camadas.

A camada de interação com usuário é chamada de View. A camada de manipulação de dados é a Model, e a camada de controle é a Controller. Além disso ainda temos o roteamento e a comunicação com o banco de dados.

* + 1. Tecnologias e Linguagens

Buscando a melhor performance, escalabilidade e produtividade da equipe, a escolha da tecnologia adotada é baseada em alguns pontos de importância: Tecnologias atualizadas e com suporte ativo; Comunidade ativa e participativa; Curva de aprendizagem baixa e Conhecimento próximo ao já adquirido pela equipe.

Tecnologias:

Lado do Servidor: Plataforma NodeJS com Framework Back-End Express

Banco de Dados: MySQL ou MariaDB

Lado do Cliente: React e React-Native

Servidor Web: Nginx

* + 1. Segurança

Com base nas tecnologias atuais, descrevemos algumas medidas de segurança a serem tomadas no sistema para prevenção de ataques. Explicamos abaixo.

* Autenticação por token

Ao efetuar um login no sistema é necessário um token para validar o usuário e reconhecer a sessão aberta, apenas com esse token o usuário pode acessar outras partes do sistema. Caso esse código token seja algo padrão como nome do usuário ou Id do mesmo no banco, pode ser facilmente alterado para simular outro usuário, facilitando a entrada em contas privada.

* Criptografia de senhas

Sempre ao salvar uma senha em algum ponto de cadastro, deve-se salvar uma hash da senha e nunca a senha em si. Muitas vezes usuários cadastram a mesma senha em múltiplos sistemas e o vazamento desses dados sensíveis expõem o cliente em suas outras contas. Uma forma de prevenção é nunca guardar senhas, apenas uma criptografia dela para quando um login for feito, a hash salva ser comparada com a hash da senha digitada no campo.

* Prevenção contra man-in-the-middle

Toda troca de requisições entre cliente e servidor é feita via protocolos, para impedir o ataque de MITM deve-se usar protocolos criptografados para mesmo que o dado seja interceptado, ele não seja compreendido. Pra isso poderemos usar um certificado ssl, assim usando https aumenta segurança em toda interação.

* Prevenção contra Cross-site Scripting

Essa é uma técnica onde foca campos de submissão como inputs, textarea e outras tags que permitem setar atributos, adicionando um código javascript que executa um script no site e quando passado para o servidor abre margem para um usuário controlar ações de outros, gerar links maliciosos adicionando o código js na url com o domínio seguro do site, conseguindo roubar token de acesso e navegar na conta da vítima. A forma mais usada de prevenção é verificar se códigos foram inseridos em todos os campos de entradas, como nos campos via form, passagem por parâmetro post, na url via get e etc… Tudo deve ser cuidadosamente validado.

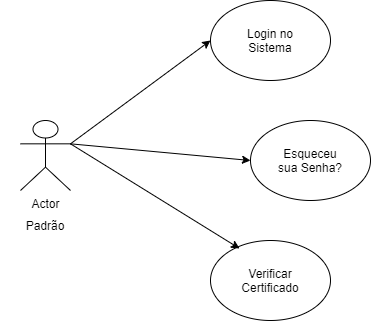
* Prevenção contra SQL Injectio

SQL Injection é uma das formas mais perigosas de ataque, ele expõe o banco de dados diretamente caso não seja filtrado todos os pontos de vulnerabilidade. Um invasor pode entender as consultas realizadas ao banco e forçar uma query diretamente do backend via frontend, podendo assim clonar o banco, modificar ou excluir. Para evitar esse tipo de falha deve-se interceptar fechamentos de tags forçada nos campos de entrada. Cuidadosamente todos os campos devem ser avaliados para impedir que código malicioso para o servidor passe.

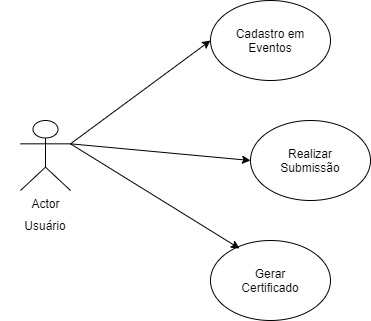
* Utilizar Cross-Origin Resource Sharing

Um papel crucial do sistema é impedir que consultas a ele sejam feitas de locais não confiáveis, com o cross-origin resource sharing podemos setar de onde consultas podem ser feitas e restringir apenas a domínios conhecidos por nós, impedindo que qualquer outra página externa a nosso sistema interaja diretamente com o servidor.

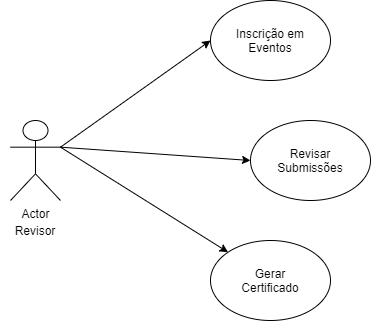
* 1. Visões de Caso de Uso
* Padrão para todos os Atores



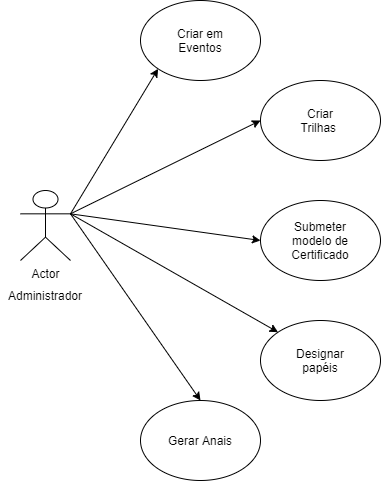
* Ator Usuário



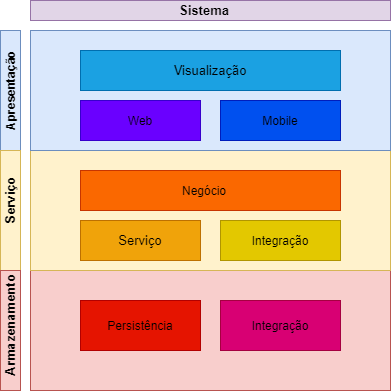
* Ator Revisor



* Ator Administrador



* 1. Visão Lógica
* Camadas do Sistema



* Camada de Apresentação

Esta camada conterá as interfaces visuais, as quais interagirão diretamente com o usuário em ambas as plataformas.

* Camada de Serviço

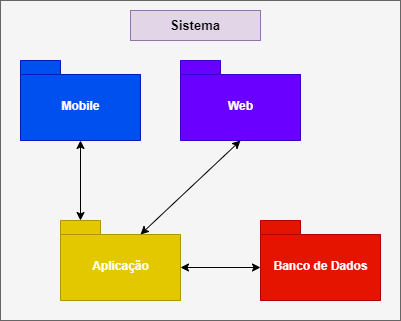
Esta camada conterá todas as informações de serviço, servidores, e regras de negócio, que possibilitarão a integração entre as plataformas de apresentação e a comunicação com o servidor web.

* Camada de Armazenamento

Esta camada conterá todas as informações necessárias de persistência e integração entre banco de dados e os objetos do serviço. O objetivo é separar a camada de negócios das linguagens e controles dos bancos de dados.

* Interação entre Camadas

A comunicação entre as camadas se dará pela interação entre as plataformas de visualização, seja web ou mobile, e a aplicação, que tratará os negócios e serviços e fará a persistência em um banco de dados, podendo enviar respostas a camada de visualização caso seja necessário.



* 1. Visão da Implementação

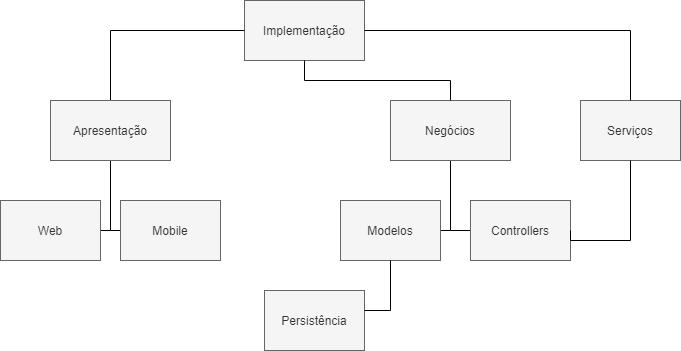
Todo software do servidor estará em uma única camada, e a versão do cliente dividida em suas duas categorias, web e mobile.

* Desempenho

O sistema será construído para suportar uma média de 150 mil usuário, e poderá contar padrões de projeto e boas práticas para uma escalabilidade adequada ao número de usuários acessando simultaneamente.

* Qualidade

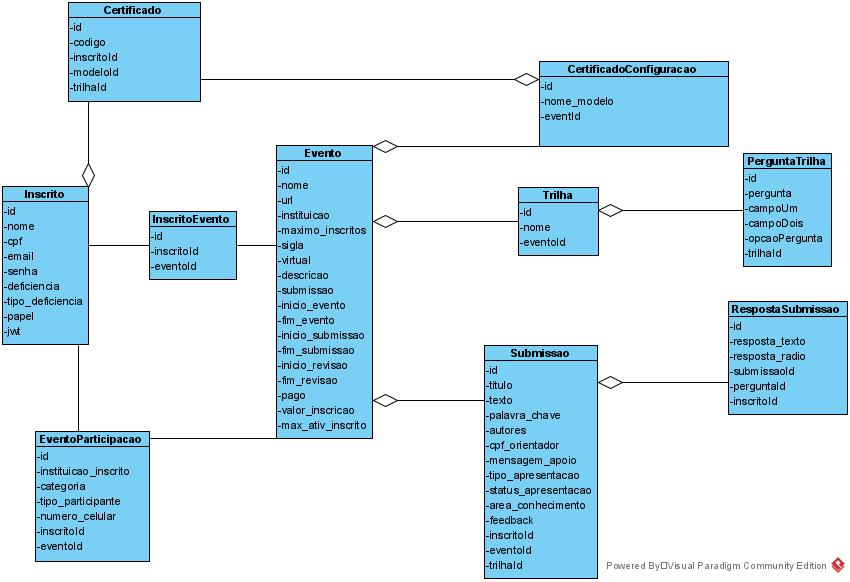
O desenvolvimento do sistema conta com o uso de tecnologias atualizadas e com uma taxa de aceitação elevada em relação aos equipamentos utilizados pelos clientes.



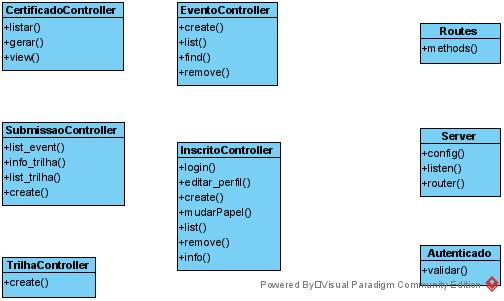
* + 1. Modelagem de Classes

As classes foram modeladas a partir do padrão MVC. Desta forma temos modelos, controladores e view que está sendo tratado como um sistema a parte. Além das classes de configurações e sistema.

* Modelos



* Sistema e Controllers



* 1. Visão de Implantação

A implantação do sistema deve ser feita em um servidor físico ou serviço de nuvem. O servidor da aplicação ficará responsável pelo sistema, pela aplicação web e pelo serviço de banco de dados. A tecnologia escolhida foi o servidor Nginx que é open source e possui uma fácil taxa de aprendizagem para utilização. Sua configuração é simples e facilitada e tem uma grande compatibilidade com aplicativos comumente utilizados na web. Possui um bom suporte a balanceamento de carga e conexões simultâneas.

A versão mobile será implantada na loja de sua respectiva plataforma. A versão do cliente, através da internet e servidores web, se comunicam com o sistema.

