Mesa Fácil

Documento de Arquitetura

Versão [1.2]

Autores:

Mat.	Nome	Função (responsabilidade)
211031468	Pedro Victor Salerno Martins	Desenvolvimento Back-end
211030620	Patrick Anderson Carvalho dos Santos	Scrum Master
170146243	João Lucas Costa Vale	Desenvolvimento Front-end
180100271	Emivalto da costa tavares junior	Gerente de Testes

Lembre-se que o alinhamento de documentos entre entregas Aprender3 e GIthub é da responsabilidade da equipe. Assim sendo, faz parte da avaliação da entrega.

[1] (*) – para cada integrante da equipe, considere sua participação tanto no Documento de Arquitetura, quanto nos demais documentos já entregues pela equipe (Visão do produto e do projeto; Declaração de escopo) e atribua um, porcentual. A soma dos percentuais de todos os integrantes deve fechar em 100%)

Sumário

1 Introdução

1.1 Propósito

Este documento de arquitetura do grupo VEGA tem como objetivo fornecer uma visão abrangente e detalhada da arquitetura do sistema de sistema de gestão integrado para restaurantes e estabelecimentos alimentícios. Ele serve como uma referência central para todas as partes interessadas, incluindo desenvolvedores, gerentes de projeto e stakeholders, garantindo que todos possuam uma compreensão clara e compartilhada da estrutura, dos componentes e das funcionalidades do sistema. Além disso, este documento orientará a implementação e evolução do sistema, garantindo que ele atenda aos requisitos definidos e se adapte às necessidades emergentes dos usuários.

1.2 Escopo

O escopo deste documento inclui a descrição dos componentes principais do sistema, suas interfaces, as tecnologias e padrões utilizados, bem como as diretrizes para a implementação e integração do sistema. Este documento abrange tanto os aspectos funcionais quanto os não funcionais da arquitetura, assegurando que o sistema atenda aos requisitos de desempenho, segurança e escalabilidade.

O projeto visa desenvolver uma plataforma robusta e escalável para a gestão integrada de processos de restaurantes. Este sistema será desenvolvido utilizando metodologias ágeis, especificamente Scrum e Extreme Programming (XP), para assegurar uma entrega contínua de valor e uma adaptação rápida às mudanças nos requisitos do cliente.

2 Representação Arquitetural

2.1 Definições

O sistema seguirá uma arquitetura baseada no padrão MVC (Model-View-Controller). Esta escolha foi feita pelo grupo considerando a necessidade de separar claramente a lógica de negócios (Modelo), a interface do usuário (Visão) e o fluxo de aplicação (Controlador). A arquitetura MVC facilita a manutenção, escalabilidade e permite que diferentes componentes do sistema sejam desenvolvidos e atualizados de forma independente.

2.2 Justifique sua escolha.

A escolha pela arquitetura MVC para o sistema VEGA foi fundamentada em diversos aspectos destacados nos documentos de Visão do Produto e Projeto e Declaração de Escopo do Produto. Seguem as principais justificativas:

1. **Separação de Responsabilidades**: Conforme descrito no documento de Visão do Produto e Projeto, o sistema VEGA necessita de uma abordagem que permita uma clara separação de responsabilidades entre a interface do usuário, a lógica de negócios e o controle do fluxo de aplicação. O padrão MVC atende a essa necessidade ao dividir a aplicação em três componentes distintos, facilitando a manutenção e a evolução do sistema.
2. **Facilidade de Manutenção e Escalabilidade**: A arquitetura MVC facilita a manutenção do sistema ao permitir que desenvolvedores trabalhem em diferentes partes da aplicação de forma independente. Por exemplo, alterações na interface do usuário (Visão) não afetam diretamente a lógica de negócios (Modelo) e vice-versa. Isso é crucial para um sistema como o VEGA, que deve se adaptar rapidamente às mudanças nas necessidades dos restaurantes, conforme mencionado na seção 1.3 dos documentos fornecidos.
3. **Reutilização de Componentes**: O uso do padrão MVC permite a reutilização de componentes, o que é particularmente importante para o VEGA, que precisa ser altamente customizável para diferentes tipos de restaurantes. A modularidade do MVC facilita a implementação de novas funcionalidades sem impactar negativamente o restante do sistema.
4. **Adaptação a Mudanças**: A metodologia ágil, destacada como parte central do desenvolvimento do VEGA, se beneficia da flexibilidade proporcionada pelo MVC. Alterações e novas

funcionalidades podem ser integradas de forma mais eficiente e com menor risco de introduzir erros em outras partes do sistema.

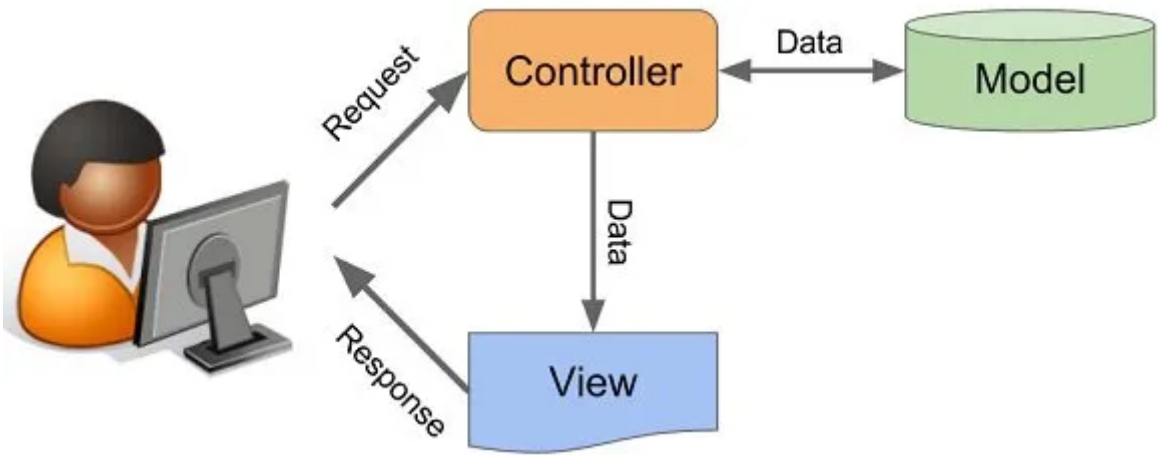
5. **Compatibilidade com Tecnologias Utilizadas**: O documento de Declaração de Escopo do Produto indica o uso de tecnologias como HTML, CSS, JavaScript, Python e frameworks como Django, que naturalmente suportam e são otimizados para a implementação do padrão MVC. Isso garante que a equipe de desenvolvimento possa utilizar as melhores práticas e ferramentas disponíveis para construir um sistema robusto e escalável.

Referências:

- Documento de Visão do Produto e Projeto, seção 1.3: "Objetivos do Produto"
- Documento de Declaração de Escopo do Produto, seção 1.4: "Tecnologias a Serem Utilizadas"

Esta arquitetura foi escolhida com o objetivo de atender de maneira eficiente às necessidades de flexibilidade, escalabilidade e manutenção do sistema VEGA, conforme delineado nos documentos de Visão do Produto e Declaração de Escopo.

2.3 Detalhamento



Neste esquema, o usuário faz uma requisição de um dado ou de uma função. Essa requisição vai para o controller, que executa a função e irá até o model, responsável pelo banco de dados, coletar as informações requisitadas. Uma vez coletados os dados, eles são processados pelo controller e enviados para a view para serem apresentados ao usuário. Ou seja:

Usuário -> **View**: Usuário faz uma requisição.

View -> **Controller**: View envia a requisição para o Controller.

Controller -> **Model**: Controller solicita dados ao Model.

Model -> **Banco de Dados**: Model acessa e coleta dados do banco de dados.

Model -> **Controller**: Model retorna os dados para o Controller.

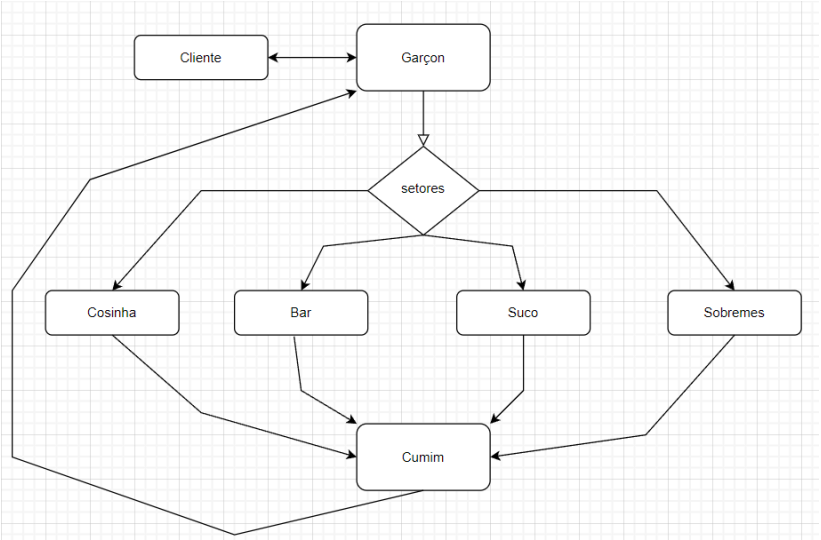
Controller -> **View**: Controller processa e envia os dados para a View.

View -> **Usuário**: View apresenta os dados ao usuário.

2.4 Metas e restrições arquiteturais

- Usabilidade, o sistema deve ser agil de ser utilizado
- Responsabilidade, a cardápio deverá ser responsivo para diferentes tipos de telas
- Segurança, o sistema deve ser seguro, uma vez que armazena dado confidenciais do funcionários e clientes
- Escalabilidade, o software de ser fácil processo evolutivo
- Manutenibilidade, o software deve ter fácil manutenção

2.5 Visão de Casos de uso (escopo do produto)



Nosso produto visa facilitar e ajudar a gerenciar esse fluxo de atendimento para o restaurante, e assim melhorando o atendimento do cliente do restaurante, tendo mais agilidade e também mais comodidade para o restaurante poder gerenciar.

Na primeira etapa o cliente faz o pedido para o garçom o garçom anota e comanda o pedido e assim o sistema vai identificar de onde é cada pedido e enviar para os setores corretos, após a confecção dos pedidos e o cumim leva o pedido para o garçom e assim o garçom serve a mesa com os pedido feitos pelos clientes.

2.6 Visão lógica

O sistema é subdividido nos seguintes módulos:

- Módulo de Gerenciamento de Funcionários
- Módulo de Gerenciamento de Mesas
- Módulo de Pedidos
- Módulo de Estoque de Cozinha
- Módulo de Banco de Dados

Razão Lógica de Cada Módulo:

- Módulo de Gerenciamento de Funcionários:
  - Propósito: Gerenciar informações e funções dos funcionários (gerentes, garçons, cozinheiros).
  - Funções: Cadastro de funcionários, atualização de informações, controle de acesso.
- Módulo de Gerenciamento de Mesas:
  - Propósito: Gerenciar a alocação e estado das mesas do restaurante.
  - Funções: Reserva de mesas, atualização de estado (disponível, ocupada, reservada).
- Módulo de Pedidos:
  - Propósito: Gerenciar os pedidos realizados pelos clientes.

    - Funções: Criação de pedidos, atualização de status, associação com mesas.
- Módulo de Estoque de Cozinha:
    - Propósito: Gerenciar o estoque de ingredientes e materiais necessários para a cozinha.
    - Funções: Cadastro de itens, controle de quantidades, atualização de estoque.
- Módulo de Banco de Dados:
    - Propósito: Persistência e recuperação de dados para todos os outros módulos.
    - Funções: Operações de CRUD (Create, Read, Update, Delete) para funcionários, mesas, pedidos, e estoque.

Comunicação entre os Módulos – Interfaces:

- API de Funcionários: Interface para operações de gerenciamento de funcionários.
- API de Mesas: Interface para operações de gerenciamento de mesas.
- API de Pedidos: Interface para operações de gerenciamento de pedidos.
- API de Estoque: Interface para operações de gerenciamento de estoque.
- API de Banco de Dados: Interface para persistência e recuperação de dados.

Diagrama de Estados da Aplicação

O diagrama de estados mostra os diferentes estados pelos quais um pedido pode passar durante seu ciclo de vida no sistema. Este diagrama é essencial para entender as transições de estado e como os pedidos são processados. Exemplificando:

- Novo Pedido: Estado inicial quando um pedido é criado.
- Em Preparação: O pedido está sendo preparado pelo cozinheiro.
- Pronto para Servir: O pedido está pronto e aguardando ser servido.
- Aguardando Servir: O pedido está pronto e aguardando o garçom para ser levado à mesa.
- Servido: O pedido foi servido ao cliente.
- Concluído: O pedido foi consumido e o processo está encerrado.

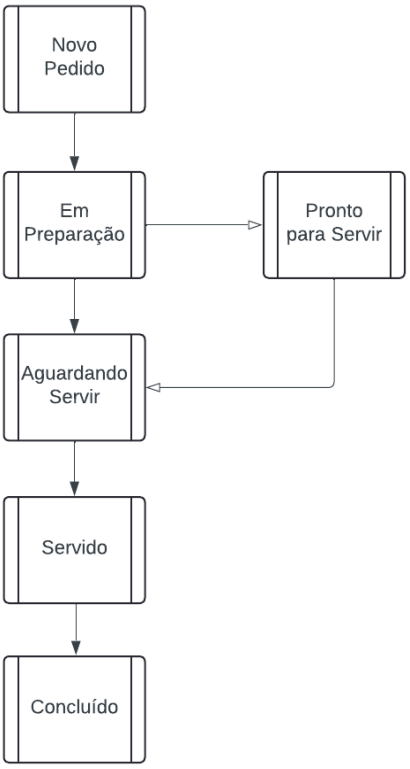
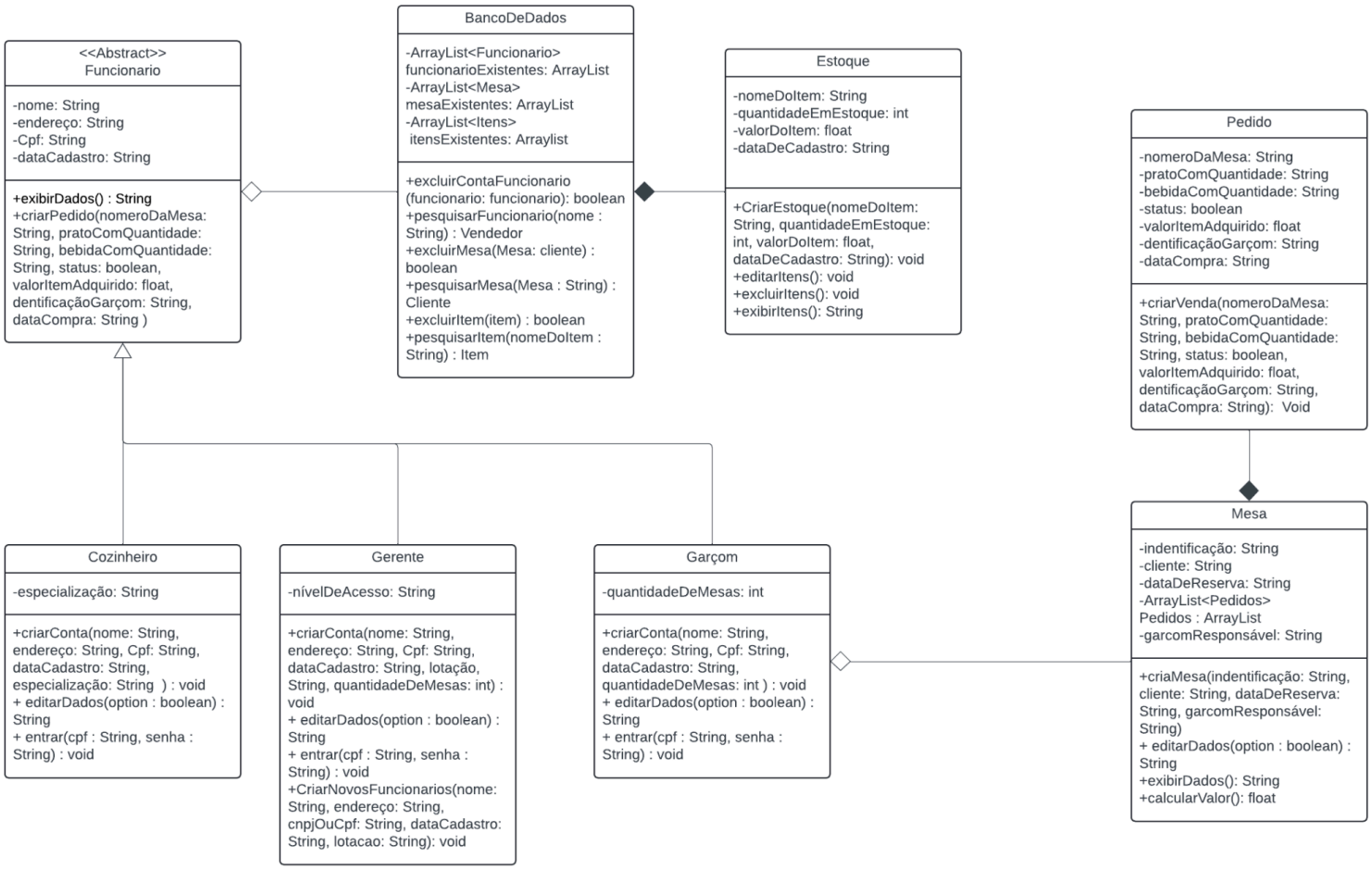
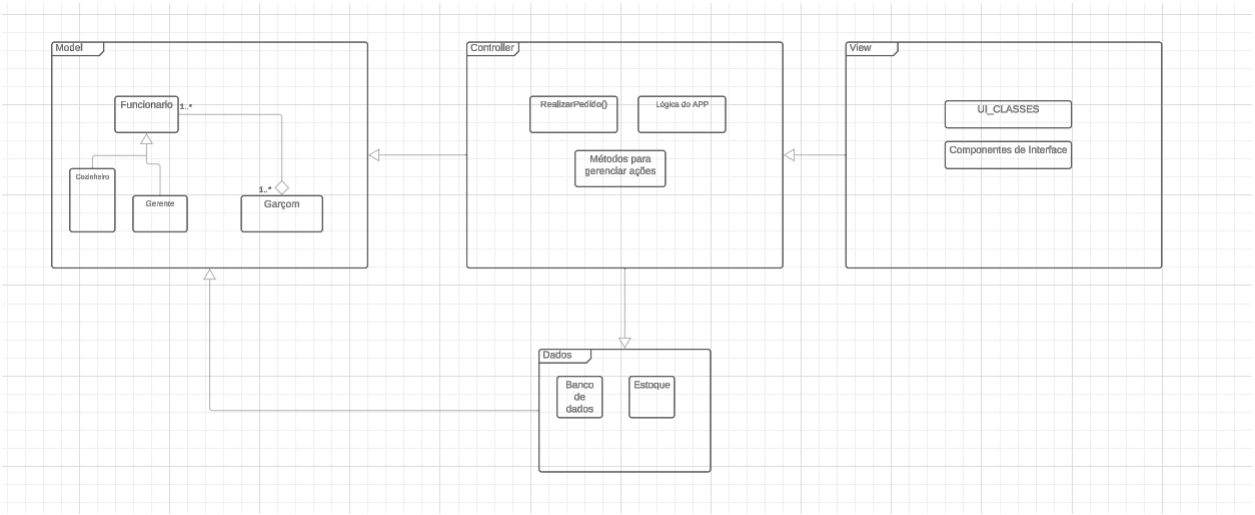


Diagrama de Atividades da Aplicação

Diagrama de Classes



Nesse diagrama de classes, é possível ver três classes: cozinheiro, gerente e garçom. Essas classes se relacionam através da herança com a classe abstrata pai chamada Funcionário. Essa classe abstrata se relaciona com a classe BancoDeDados, que, por sua vez, salva todas as informações dos funcionários e da classe Estoque. Além disso, a classe Garçom se relaciona com a classe Mesa, que é relacionada a um cliente e guarda uma lista de pedidos feitos. Por fim, essa classe se relaciona com a classe Pedidos.



2.7 Visão de Implantação

Para a implementação não será necessário um hardware muito potente já que o sistema irá funcionar basicamente no servidor, que será feito a requisição e voltará os dados necessários para que o usuário consiga utilizar a interface para o usuários, sendo ele o garçom, gerente ou o local que será feito o preparo dos pratos.
o sistema irá precisar basicamente de um terminal onde se possa ser feito a requisição e logo o poderá ser acessado com o mouse e teclado e ver tudo através do monitor, como banco de será utilizado o **MySQL** á que mais simples para fazer a integração de maneira mais viável, será utilizado também **Python** já possui diversas bibliotecas que podem ser implementadas e assim ajudando melhor no funcionamento do sistema no geral, isso no back-end, á para o front end será utilizado HTML, Flutter, CSS e JavaScript já que são linguagem já conhecidas pelos integrantes do grupo e também possui bibliotecas com diversas funcionalidades.

2.8 Restrições adicionais

Para garantir que o sistema funcione de maneira eficiente e atenda às necessidades do negócio e dos usuários, algumas restrições adicionais são necessárias. Estas restrições estão relacionadas a aspectos negociais e de qualidade de software.São elas:

- Autenticação de Usuário:
    - Descrição: Todos os usuários devem se autenticar utilizando um login e uma senha antes de acessar o sistema.
    - Justificativa: A autenticação garante que somente usuários autorizados, como gerentes, garçons e cozinheiros, possam acessar e modificar informações relevantes. Isso evita acessos não autorizados e protege os dados.
- Suporte para Múltiplos Usuários Concomitantemente:
    - Descrição: O sistema deve suportar até múltiplos usuários logados ao mesmo tempo.
    - Justificativa: Restaurantes podem ter vários funcionários trabalhando simultaneamente e vários clientes acessando o cardápio e realizando o pedido, e o sistema deve ser capaz de suportar essa carga para garantir a eficiência das operações
- Usabilidade
    - Descrição: O software deve ser intuitivo e fácil de usar para todos os funcionários e clientes, independentemente de seu nível de habilidade técnica.
    - Justificativa: Uma interface amigável e fácil de usar minimiza erros de operação e reduz o tempo de treinamento necessário para os novos funcionários.

3 Bibliografia

MEYER, Eric A. CSS: The Definitive Guide. 4th ed. O'Reilly Media, 2020.

FLANAGAN, David. JavaScript: The Definitive Guide. 7th ed. O'Reilly Media, 2020.

SCHWABER, Ken; SUTHERLAND, Jeff. The Scrum Guide. Scrum.org, 2020.