# Diário de Decisões – Projeto TocaDaOnca

**Data:** 16 de Abril de 2025  
**Responsável:** José Lucas Aparecido Rocha

## 1. Introdução

Este documento tem o objetivo de registrar e justificar as principais decisões tomadas no desenvolvimento do projeto **TocaDaOnca**. A proposta abrange não só as escolhas arquiteturais e técnicas, mas também as definições de gestão do projeto e das tecnologias (stacks) utilizadas. O repositório base ([TocaDaOnca](https://github.com/joselucas0/TocaDaOnca/tree/main/TocaDaOnca)) serviu como referência para orientar as escolhas de implementação e estruturação da aplicação.

## 2. Decisões de Gerência de Projetos

### 2.1. Brainstorm e Alinhamento Inicial

* **Sessão de Brainstorm:**Realizamos uma sessão de brainstorming com toda a equipe para que **todos tivessem uma visão** abrangente do software, alinhando expectativas e contribuindo para a definição do escopo inicial. Essa etapa foi determinante para **identificar as funcionalidades principais – o MVP – composto pelo cadastro, login e a funcionalidade de reserva para o cliente**.
* **Definição do MVP:**A partir do brainstorming, priorizamos as funcionalidades essenciais para o lançamento inicial, criando sprints, possibilitando entregas rápidas e iterativas.

### 2.2. Planejamento Ágil e Acompanhamento

* **Mini Dailys:**Adotamos mini dailys para comunicação diária na equipe – cada membro compartilha o que foi feito, o que está sendo feito e quaisquer impedimentos. Essa prática, mesmo não formalmente documentada devido à limitação de tempo, foi **vital para manter a equipe sincronizada**.
* **Priorização Dinâmica:**Conforme o desenvolvimento avançava, as prioridades eram ajustadas para garantir que as próximas **tarefas estivessem alinhadas com o MVP**, possibilitando flexibilidade para reagir a atrasos ou adiantamentos.

### 2.3. Gestão de Tarefas via Trello

* **Organização Visual das Tarefas:** Utilizamos um quadro Trello estruturado da seguinte forma:  
  + **A Fazer:** Tarefas planejadas que aguardam início.
  + **Fazendo:** Atividades em andamento.
  + **Revisar:** Tarefas concluídas que necessitam de validação.
  + **Feito:** Tarefas finalizadas e aprovadas.
* **Justificativa:** Essa metodologia visível permite identificar rapidamente o status das atividades, revisar prioridades e **ajustar o fluxo de trabalho** para manter a entrega contínua e alinhada às expectativas.

### 2.4. Definição Visual e Funcional

* **Identidade Visual:**Desde o início, foi definida uma entidade visual para padronizar as telas do front-end, promovendo consistência e melhor experiência para o usuário.

## 3. Visão Geral da Arquitetura do Projeto

A análise do repositório demonstra a aplicação de boas práticas que garantem a manutenibilidade e a escalabilidade do sistema. Destacam-se:

### 3.1. Estrutura de Pastas e Camadas

* **Controllers:** Localizados na pasta Controllers, **tratam das requisições HTTP, direcionando os dados para os serviços correspondentes**.
* **Services:** A **centralização da lógica de negócio** na pasta Services contribui para a reutilização de código e facilita a manutenção.
* **Models:** **Definem as entidades que mapeiam as tabelas do banco de dados**, mantendo uma relação clara com a persistência dos dados via ORM.
* **AppDbContext & Migrations:** Esses componentes **gerenciam a comunicação com o banco de dados** e o versionamento do schema, garantindo atualizações controladas.
* **Views:** Separam a **camada de apresentação**, onde ficam as implementações das interfaces do front-end.
* **Program.cs e Arquivos de Configuração:** Concentram as **configurações gerais da aplicação**, desde a definição dos middlewares até a injeção de dependências.

### 3.2. Padrões de Projeto e Boas Práticas

* **Injeção de Dependências:** Utilizada para promover o **desacoplamento entre os componentes, facilitando testes e futuras evoluções**.
* **Separação de Camadas:** Isolar a lógica de apresentação, negócio e acesso a dados **permite uma organização clara e contribui para a escalabilidade da aplicação**.
* **Uso de Migrations:** As migrations garantem uma evolução controlada do banco de dados, minimizando riscos durante atualizações.

## 4. Justificativa do Uso de DTO's

### 4.1. Desacoplamento e Segurança

* **Encapsulamento das Entidades:**Os Data Transfer Objects (DTO's) foram adotados para separar a estrutura interna (models) dos dados que serão expostos via API, evitando a exposição **desnecessária e protegendo informações sensíveis**.

### 4.2. Suporte à Escalabilidade

* **Facilidade na Integração com Cache e Balanceamento:** DTO's enxutos tornam mais eficaz a implementação de estratégias de cache e balanceamento de carga, contribuindo para manter o desempenho mesmo com alta concorrência.

## 5. Decisões de Stacks e Padrões de Commits

### 5.1. Stack Tecnológica

* **Front-end:**
  + **Tecnologias:** HTML, CSS e JavaScript.
  + **Justificativa:** **Devido ao tempo disponível para o projeto, optou-se por não utilizar TypeScript.** As tecnologias escolhidas possibilitam a rápida implementação e têm ampla familiaridade na equipe, garantindo flexibilidade e agilidade na entrega das interfaces.
* **Back-end:**
  + **Tecnologias:** ASP.NET Core
  + **Justificativa:** ASP.NET Core foi escolhido por sua robustez, escalabilidade e suporte nativo à injeção de dependências, além de oferecer alta performance em aplicações web.
* **ORM e Banco de Dados:**
  + **Tecnologias:** Entity Framework Core e PostgreSQL
  + **Justificativa:**
    - **Entity Framework Core:** **Facilita o mapeamento** entre as entidades do sistema e as tabelas do banco de dados, permitindo uma evolução segura e controlada através de migrations.
    - **PostgreSQL:** Optamos pelo PostgreSQL devido à sua robustez, compatibilidade com as demandas do projeto e à familiaridade da equipe com este SGBD.
* **Uso de DTO's e Entities:**
  + **Justificativa:** A combinação de DTO's com entidades (Entities) permite que o sistema atenda às necessidades de desempenho e flexibilidade, **separando a lógica de apresentação dos dados armazenados e protegendo informações sensíveis.**

### 5.2. Padrões de Commit

Adotamos os **Conventional Commits** para **garantir rastreabilidade clara dos commits**. Embora não tenhamos definido uma única linguagem para as mensagens, os commits seguem um padrão consistente com uso de tags e escopos que facilitam a identificação do impacto das alterações:

| **Prefixo** | **Exemplo** | **Escopo Válido** |
| --- | --- | --- |
| feat: | feat(front): adiciona tela de login | front, api, db |
| fix: | fix(api): corrige status code 500 | front, api, db |
| refactor: | refactor(db): otimiza query de reservas | front, api, db |
| style: | style(front): ajusta padding do menu | front |
| docs: | docs: atualiza guia de instalação | (sem escopo) |

**Observações:**

* **Linguagem:** Apesar da ausência de definição única (com commits tanto em inglês quanto em português), a utilização correta das tags e escopos garante a rastreabilidade e organização dos commits.
* **Localização:** Para facilitar o entendimento, cada commit deve indicar seu escopo (por exemplo, *front* para alterações de interface, *api* para alterações na camada de serviços ou *db* para modificações no modelo ou no banco de dados).

### Modelo de Ramificação

Adotamos um fluxo de trabalho (branching model) que organiza o desenvolvimento em diferentes níveis, garantindo uma gestão eficiente das funcionalidades e uma integração controlada do código. Nosso modelo conta com as seguintes branches:

* **main:** Esta é a branch principal, onde se encontram as funções já integradas e estáveis, **prontas para a entrega**. Todo o código nesta branch deve passar por revisão e testes rigorosos antes de ser fundido, garantindo a confiabilidade da versão em produção.
* **development:** A branch *development* serve como ambiente de **integração para funções que já foram concluídas e estão em fase de testes ou refinamento**. Nela, são consolidadas as atualizações e correções provenientes das branches de feature, facilitando a preparação para a liberação final na *main*.
* **feature:** As branches *feature* são criadas para o **desenvolvimento de componentes ou novas funcionalidades específicas.** Cada tarefa ou componente a ser desenvolvido é realizada em sua própria *feature branch*, permitindo um ambiente isolado e colaborativo para a criação e experimentação. Ao final, essas branches são integradas à *development* após a conclusão e validação dos testes.

**Justificativa:** Esse modelo de ramificação **permite um fluxo organizado para o desenvolvimento, testes e integração contínua**. Ao segmentar as tarefas em *feature branches*, garantimos que o trabalho individual não comprometa a estabilidade da aplicação, enquanto a *development* consolida essas contribuições antes de serem promovidas à *main*. Assim, reduzimos riscos de conflitos e facilitamos a gestão das prioridades e entregas do projeto.

## 6. Outras Decisões Relevante

### 6.1. Controle de Versão e Colaboração

* **Git e GitHub:** O uso desses sistemas garantiu a rastreabilidade e colaboração efetiva por meio de branches, pull requests e mensagens de commit claras, alinhadas com os padrões adotados.

## 7. Conclusão

As decisões registradas neste diário refletem a evolução do projeto **TocaDaOnca** tanto do ponto de vista técnico quanto de gestão. O alinhamento realizado por meio de sessões de brainstorming, mini dailys, gestão visual das tarefas via Trello e definição de identidade visual permitiu que a equipe se mantivesse focada nos objetivos do MVP.

Paralelamente, a escolha de stacks tecnológicas – com HTML, CSS e JavaScript para o front, ASP.NET Core, Entity Framework Core e PostgreSQL para o back – aliada à utilização de DTO's, proporciona a agilidade, segurança e escalabilidade necessárias para o sucesso do projeto. A adoção dos Conventional Commits, mesmo com variação na linguagem, garante um histórico claro e organizado das alterações realizadas.

Nas próximas sprints, o foco será:

* Concluir a implementação e integração das funcionalidades reservadas ao gerente e funcionário.
* Finalizar as funcionalidades complementares, como o cadastro e a venda de produtos.
* Refinar as rotas de acesso, garantindo que as áreas restritas estejam devidamente segmentadas e seguras.