

|  |  |  |
| --- | --- | --- |
| **Sprint Documentation** | | |
| **Sprint #02 Desenvolver API Carrier** | **Start Date:**01/10/2024 | **Final Date:** 14/10/2024 | |
| **Team Members** | | | |
| Marco Antônio dos Santos Massom | | | |
| Kétlyn Christini Tonholi | | | |
|  | | | |
|  | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint Backlog** | | | |
| Task# | Description | Start Date | Final date |
| 035 | Sprint 02 - Desenvolver a API para Carrier | 01/10/2024 | 14/10/2024 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task Description** | | | | | |
| Task # | Description | Assigned To | Status | Estimated Hours | Logged Hours |
| 035 | Desenvolvido a API Carrier “Transportadora” com (Operações CRUD, Estruturação do Banco de Dados e Validação dos Dados.) | Kétlyn Tonholi e Marco Antônio Massom | Concluído | 40h | 10h |
|  |  |  |  |  |  |

|  |
| --- |
| **Sprint Results** |

Texto

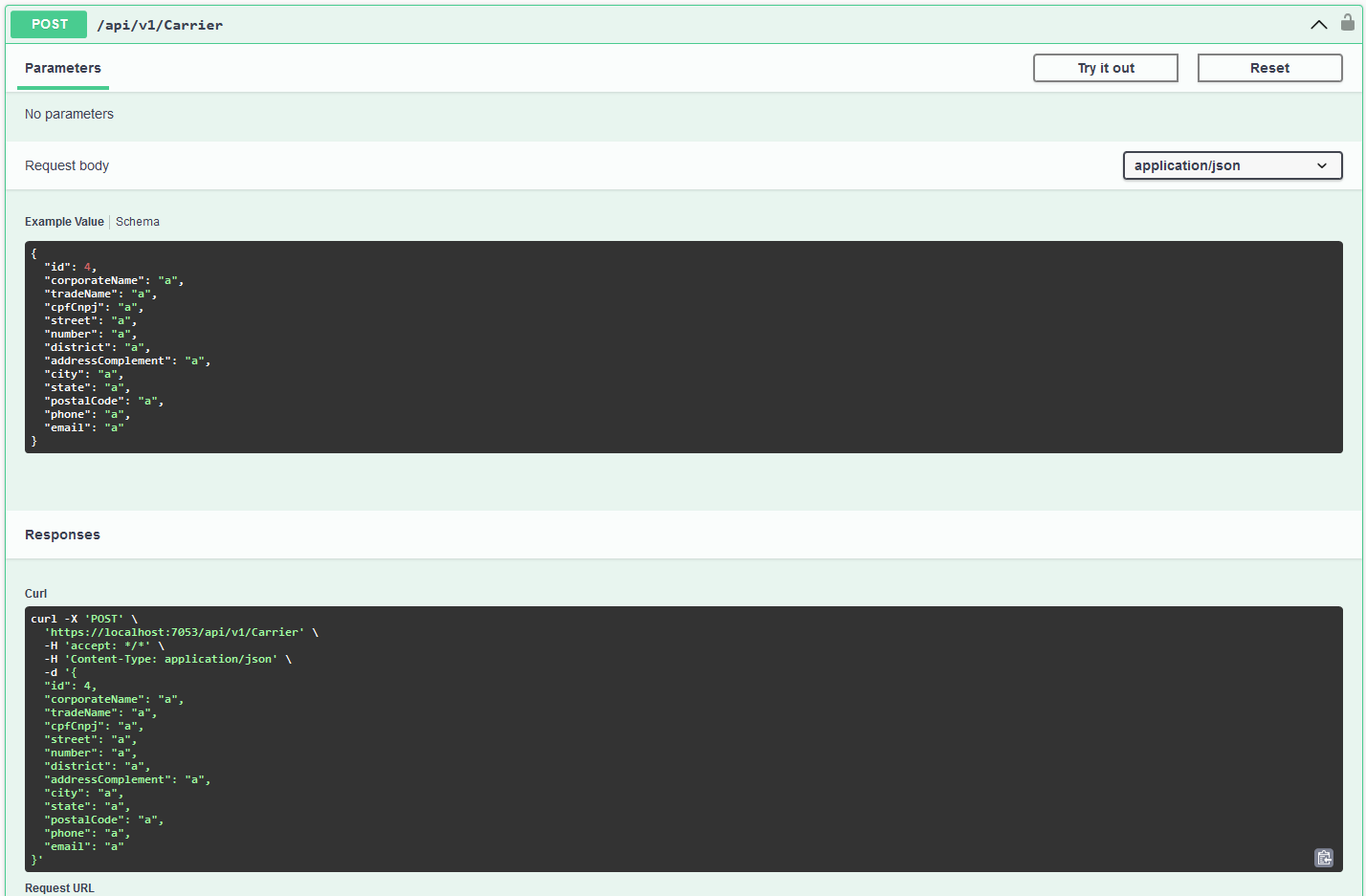
Descrição gerada automaticamente

Esse diagrama de classes foi fundamental para nos guiar durante o desenvolvimento da API. Ele nos deu uma visão clara de como a estrutura dos dados deveria ser organizada no banco de dados, o que facilitou muito a criação dos códigos para as operações de CRUD.

Com ele, conseguimos entender de forma mais fácil como cada informação da transportadora seria armazenada e recuperada. Por exemplo, campos como “CorporateName”, “TradeName” e ”CpfCNPJ” precisavam ser strings para suportar os dados textuais, enquanto o “Id”, sendo um número inteiro, funcionaria como a chave primária.

Além disso, o diagrama ajudou a manter o código organizado e coerente. A partir dessa estrutura visual, pudemos planejar cada parte da API de forma clara, o que nos ajudou a evitar confusões e a garantir que todas as funcionalidades estivessem bem implementadas. Também facilitou bastante a realização dos testes, já que sabíamos exatamente quais dados estavam sendo manipulados e como deveriam ser validados. No geral, o diagrama então foi um guia essencial para manter tudo alinhado entre o código e o banco de dados.

**Testes dos endpoints**

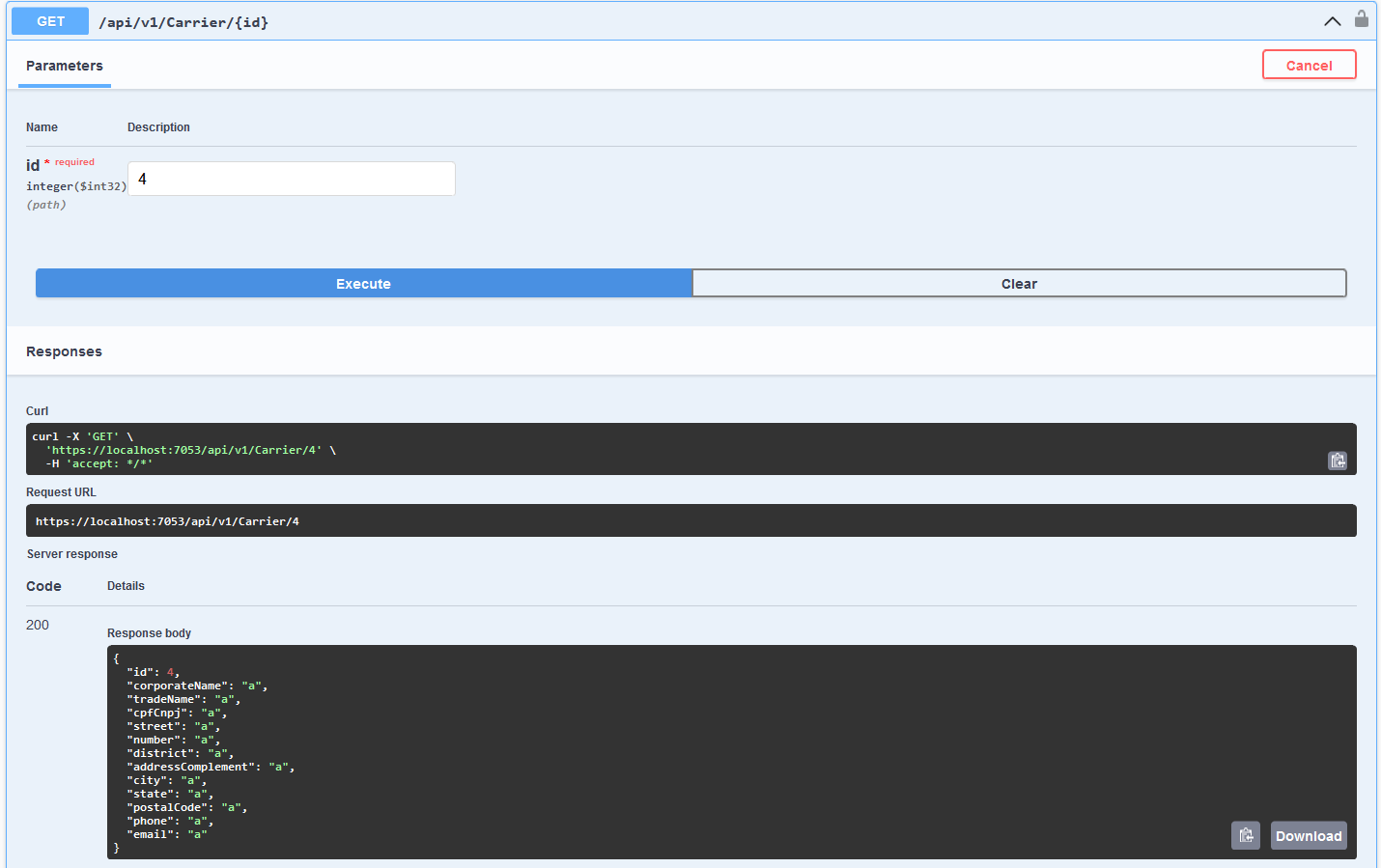
****

**Padrão do plano de fundo

Descrição gerada automaticamente**

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

****

**Padrão do plano de fundo

Descrição gerada automaticamente**

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

**Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente**

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

**Padrão do plano de fundo

Descrição gerada automaticamente com confiança baixa**

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente com confiança média**

**O que implementamos na API**

Durante essa sprint, trabalhamos juntos no desenvolvimento de uma API para gerenciar o cadastro de transportadoras (Carrier). Nosso objetivo principal foi implementar funcionalidades para cadastrar, atualizar, excluir e listar transportadoras, utilizando os métodos HTTP (GET, GET by ID, POST, PUT, DELETE).

A API foi desenvolvida com foco em garantir que o sistema de cadastro de transportadoras seja eficiente, com validação adequada dos dados e com uma estrutura flexível para futuras modificações.

**1. GET /carriers**

Esse endpoint retorna todas as transportadoras cadastradas no sistema. Nós configuramos a resposta para que todos os dados da transportadora sejam exibidos de forma clara, facilitando a consulta.

**2. GET /carriers/{id}**

Aqui, implementamos uma busca específica por ID, que retorna os dados de uma transportadora específica. Isso é importante quando se precisa de informações detalhadas sobre uma empresa sem ter que percorrer toda a lista.

**3. POST /carriers**

Esse endpoint foi feito para adicionar uma nova transportadora no sistema. Validamos todos os campos antes de permitir que o cadastro seja concluído, garantindo que nenhum dado obrigatório fique de fora.

**4. PUT /carriers/{id}**

Nesse endpoint, desenvolvemos a lógica para atualizar as informações de uma transportadora. Qualquer dado da empresa pode ser atualizado, e o sistema retorna uma mensagem confirmando que as mudanças foram realizadas com sucesso.

**5. DELETE /carriers/{id}**

Por fim, criamos a funcionalidade de excluir uma transportadora do sistema com base no ID. Isso permite que as empresas que não estão mais em operação sejam removidas.

**Estrutura do Cadastro de Transportadoras**

Cada transportadora deve ser registrada com as seguintes informações no banco de dados, com todos os tipos e validações necessários:

|  |  |  |
| --- | --- | --- |
| Campo | Tipo | Descrição |
| **ID** | **Int** | Identifica a chave primária. |
| **CorporateName** | **String** | Razão social da empresa. (Varchar(100)) |
| **TradeName** | **String** | Nome comercial. (Varchar(100)) |
| **CpfCnpj** | **String** | Número de inscrição na Receita Federal (CPF/CNPJ). (Varchar(14)) |
| **Street** | **String** | Nome da rua. (Varchar(100)) |
| **Number** | **String** | Número do endereço. (Varchar(5)) |
| **District** | **String** | Bairro. (Varchar(50)) |
| **AddressComplement** | **String** | Complemento do endereço. (Varchar(100)) |
| **City** | **String** | Cidade. (Varchar(100)) |
| **State** | **String** | Estado. (Varchar(2)) |
| **PostalCode** | **String** | CEP. (Varchar(8)) |
| **Phone** | **String** | Número de telefone. (Varchar(11)) |
| **Email** | **String** | Endereço de e-mail. (Varchar(50)) |

**Funcionalidades Não Implementadas**

**Autenticação e Autorização de Usuários**

* **Justificativa:**  
  A razão principal é que o foco dessa sprint era a criação da estrutura básica da API e das operações CRUD (Create, Read, Update, Delete). Como a gestão de autenticação envolve uma camada extra de complexidade, como controle de permissões, decidimos que seria mais produtivo concluir primeiro o básico da API.

**Paginação nas Listagens de Transportadoras**

* **Justificativa:**  
  Como estamos numa fase inicial da API e ainda não prevemos um grande volume de dados nas primeiras etapas, deixamos a implementação da paginação para uma sprint posterior. Isso também permite que tenhamos uma API mais simples nesta fase inicial, focando em garantir que as operações básicas funcionem corretamente.

**Validação Avançada de Dados (CPF/CNPJ, E-mail, Telefone)**

* **Justificativa:**  
  Embora tenhamos feito uma validação básica dos dados fornecidos (como os tamanhos de varchar), a validação mais complexa para formatos de CPF, CNPJ, e-mail e telefone não foi totalmente implementada. Isso exigiria a inclusão de expressões regulares e regras mais detalhadas de validação, o que demandaria mais tempo e testes. Decidimos que, para esta sprint, uma validação básica seria suficiente.

**Integração com Serviços Externos (Exemplo: Correios para validação de CEP)**

* **Justificativa:**  
  Outro recurso que consideramos foi a integração com APIs externas, como a API dos Correios, para validação automática de CEPs. Isso ajudaria a garantir que os endereços inseridos pelas transportadoras estejam corretos. No entanto, decidimos não implementar essa integração agora porque ela depende de configurações externas e pode complicar o desenvolvimento. Além disso, como estamos focados em uma versão básica e funcional da API, essa integração seria um complemento que pode ser adicionado mais tarde.

**Aprendizados**

Conseguimos concluir a API dentro do prazo e com todas as funcionalidades previstas. Além da implementação dos endpoints, também fizemos a validação dos campos no banco de dados, garantindo que tudo esteja funcionando corretamente. Todos os testes foram realizados com sucesso, e a API está pronta para ser integrada com outras partes do sistema.

Durante essa sprint, nós aprendemos várias lições importantes, como definir prioridades nos ajudou a focar nas funcionalidades essenciais, garantindo que entregássemos uma API funcional no prazo, a colaboração e comunicação constantes facilitaram a resolução rápida de problemas e nos mantiveram alinhados. Testes unitários mostraram o quanto é importante validarmos cada funcionalidade, reforçando que precisamos investir mais em testes automáticos no futuro. Manter o código limpo e bem documentado também facilitou nosso trabalho em equipe e garantiu que tudo estivesse claro para as futuras manutenções. Esses aprendizados vão nos ajudar a melhorar em sprints futuras e entregar ainda mais qualidade no projeto.