**RELATÓRIO DO PROJETO FINAL - COUTURE HAVEN**

**Equipe:**

* Augusto Faria Soares - Desenvolvedor (1710558)
* Dionisio Rodrigues Pequeno Neves - Desenvolvedor (2010789)
* Guthierre Marques Teixeira - Desenvolvedor e analista de requisitos (2127307)
* Victor Kauan Lima de Oliveira - Desenvolvedor e analista de requisitos (2213002)

**Professor(a):** Samuel Lincoln Magalhães Barrocas

**Disciplina:** Programação Funcional

1. **Repositório:** <https://github.com/Dionisio-Rodrigues/Couture-Haven>
2. **Introdução**

Este documento tem como objetivo documentar as principais funcionalidades e requisitos da API do e-commerce Couture Haven, projeto desenvolvido para a disciplina de programação funcional.

O sistema foi desenvolvido utilizando as tecnologias Python (sobretudo, as bibliotecas Flask API, SQL Alchemy e PyJWT), SQLite (banco de dados) e Postman (cliente para o envio de requisições de HTTP e testes).

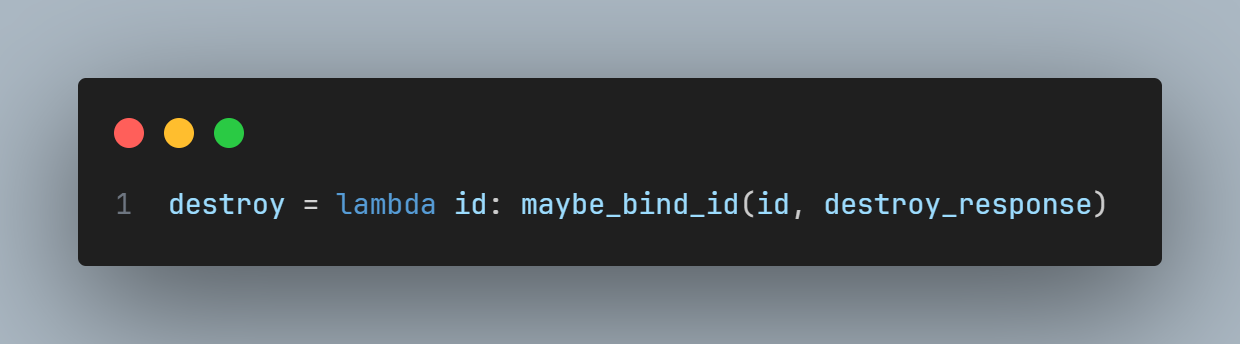
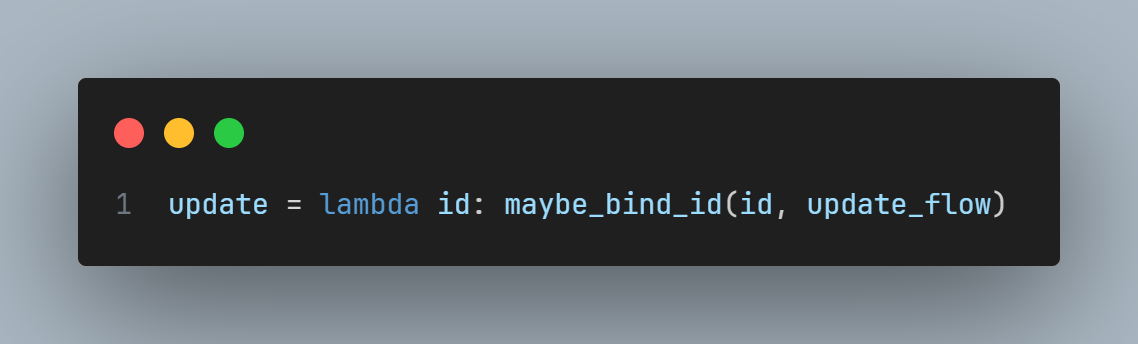
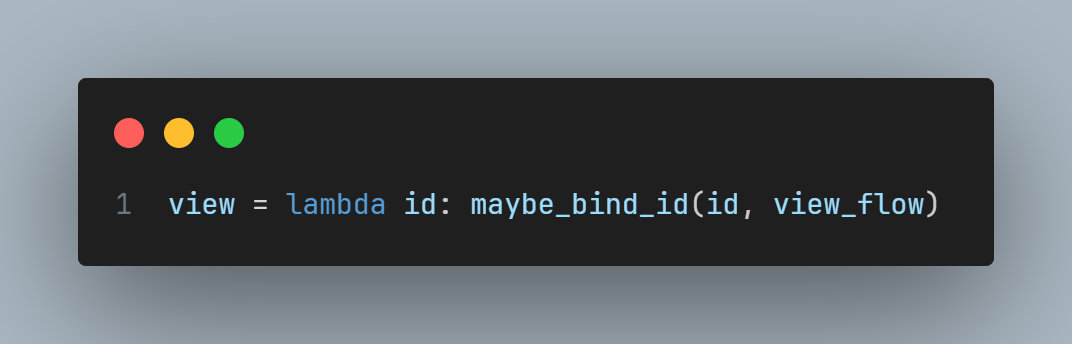
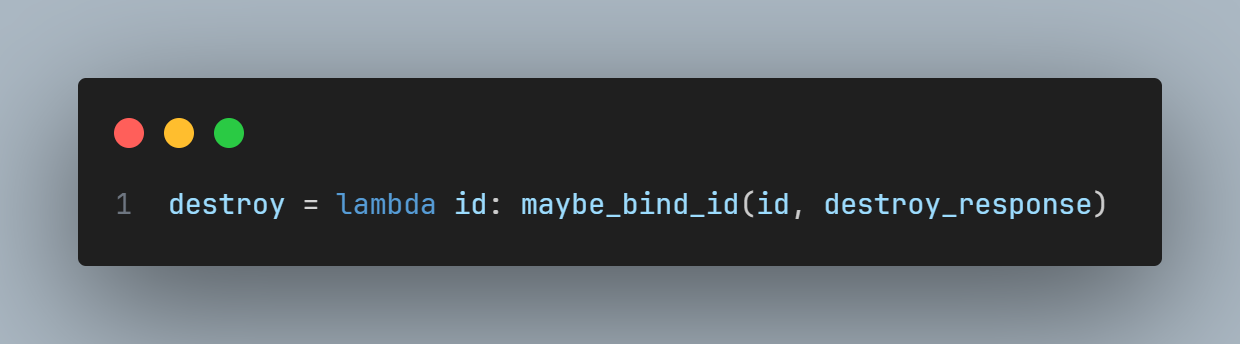
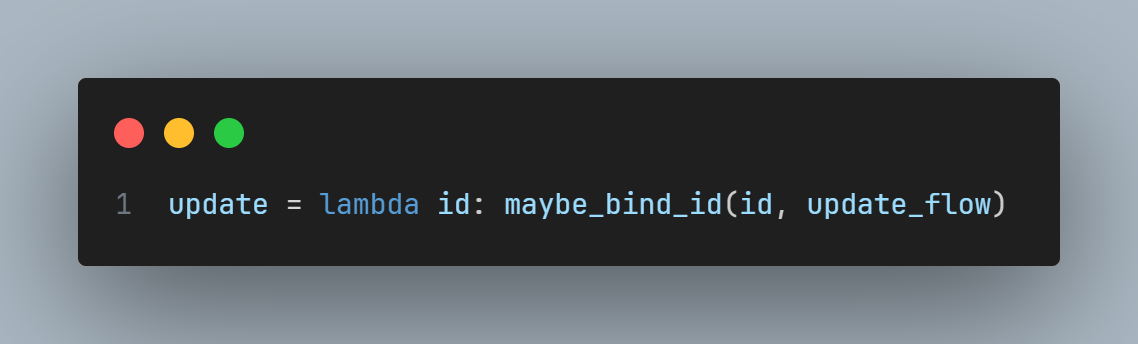
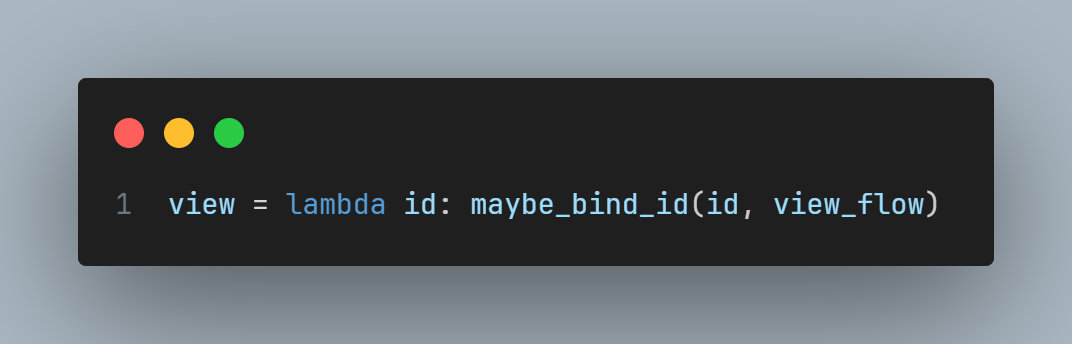
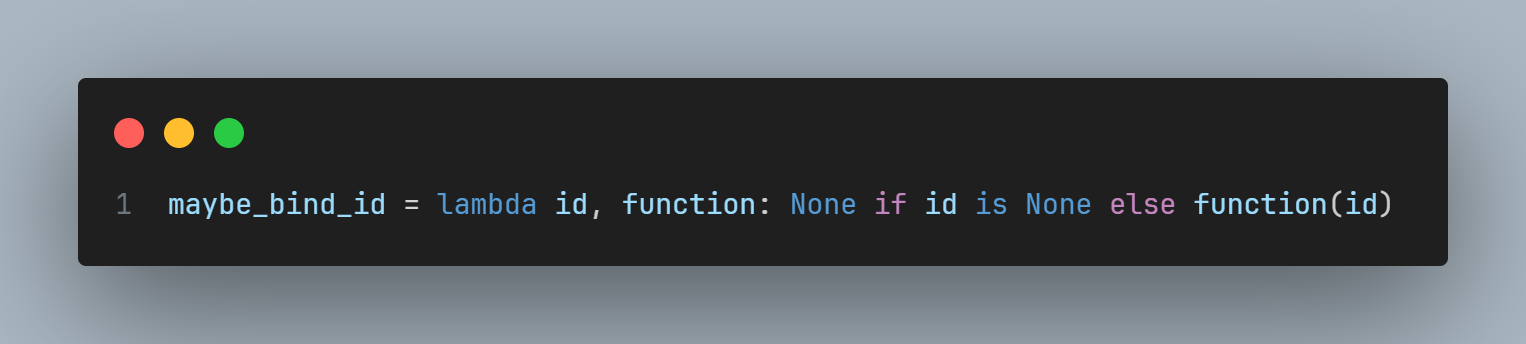
1. **Descrição Geral**

Para o desenvolvimento do projeto foi utilizada a metodologia ágil SCRUM e a ferramenta Notion para a organização das sprints e atividades. O código do projeto está sendo armazenado em um repositório no GitHub.

Além disso, a utilização da programação funcional é um requisito obrigatório da disciplina, por isso, todos os scripts Python utilizam os conceitos de lambda, alta ordem, recursividade, list comprehension, currying, functores, monad, etc.

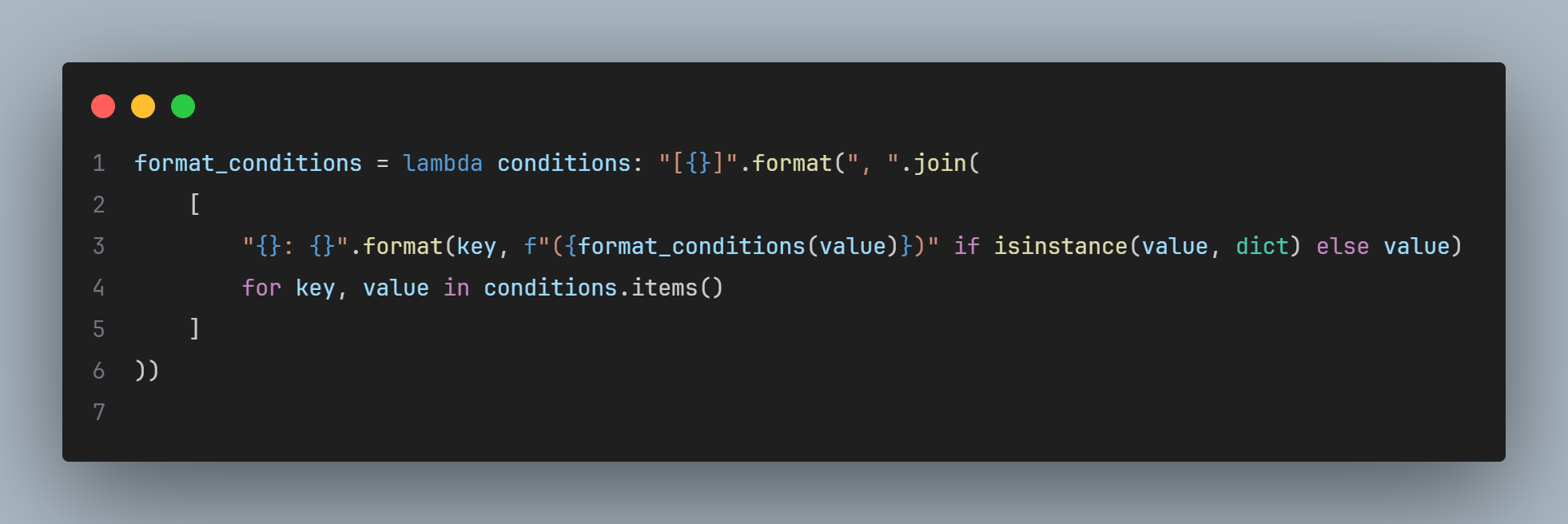
1. **Programação Funcional**
   1. **Função alta ordem**

O conceito de funções de alta ordem foi utilizado nos monads da aplicação, visto que os mesmos recebem uma função como parâmetro.



* 1. **Recursividade**

O conceito de recursividade é utilizado na função “format\_conditions”, uma função que formata um dicionário de condições e seus valores, caso sejam dicionários.



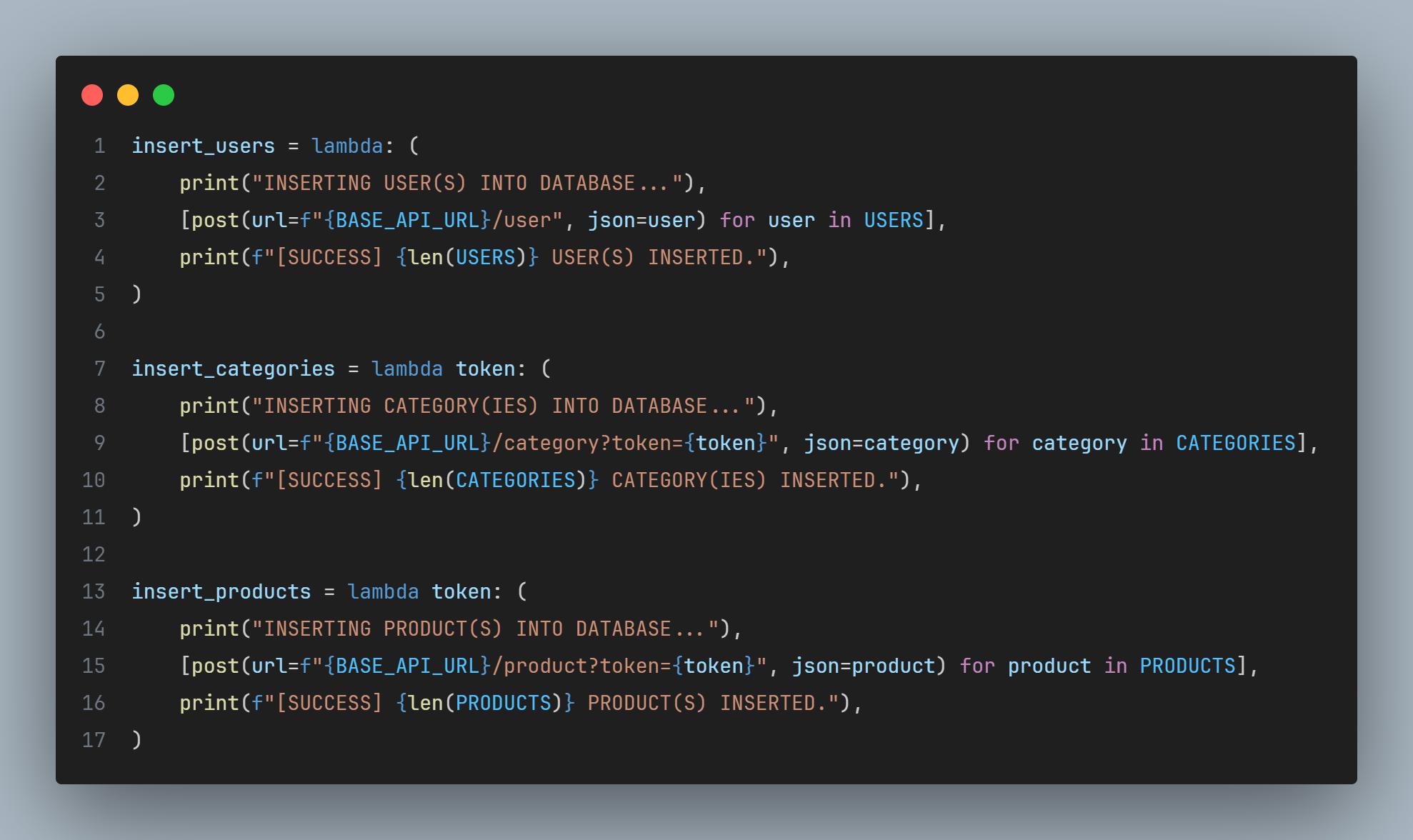
* 1. **Currying**

O conceito de currying (chamadas sucessivas) é utilizado na função “check\_password” para a autenticação do usuário do sistema.



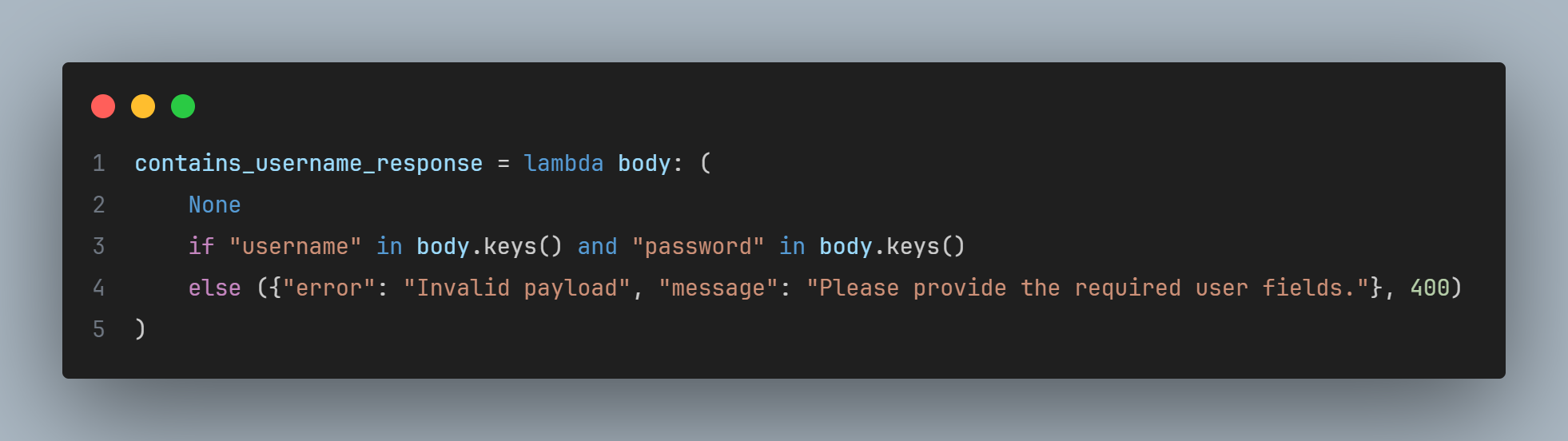
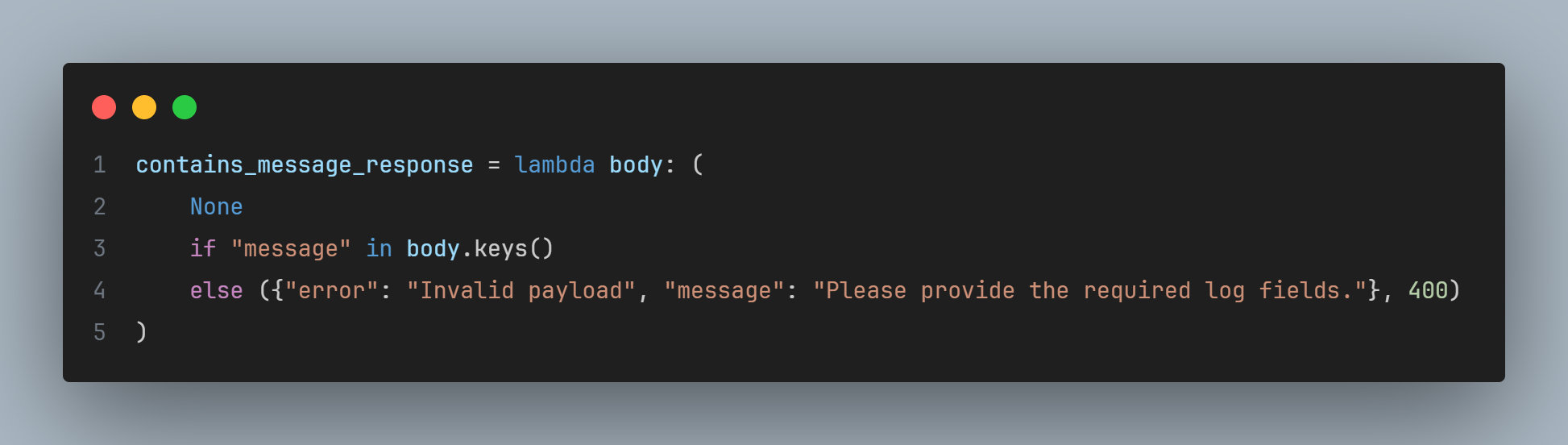
* 1. **List comprehension**

O conceito de list comprehension é utilizado para o populamento do banco de dados (seed).



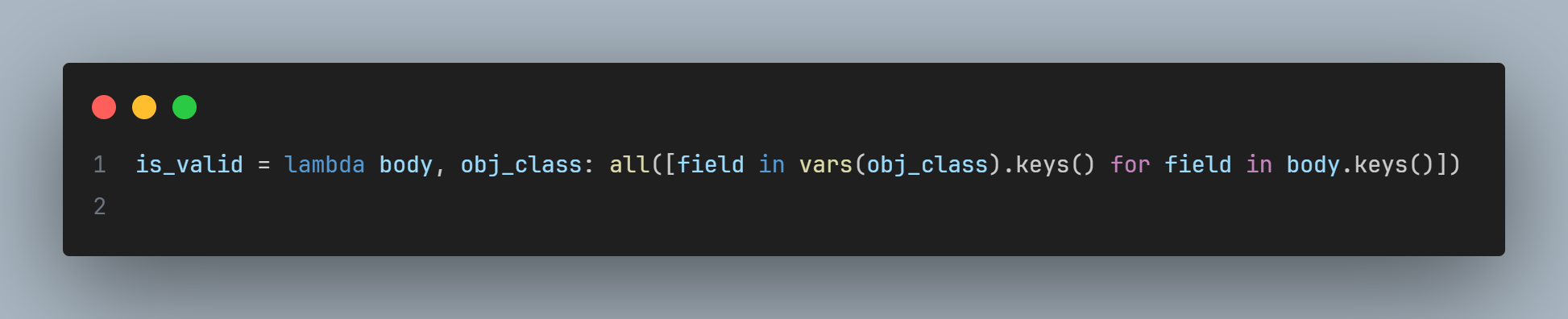
* 1. **Dicionário**

Dicionários são utilizados no escopo de algumas funções do sistema, como a “to\_dict” responsável por representar modelos em formato de dicionário e “contains\_name\_response”, “contains\_message\_response” ou “contains\_username\_response” que verifica se um dicionário possui determinadas chaves.

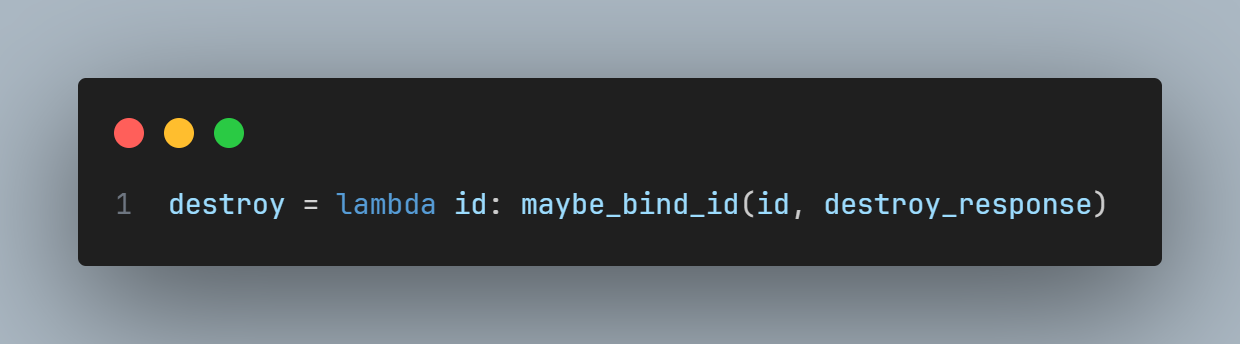
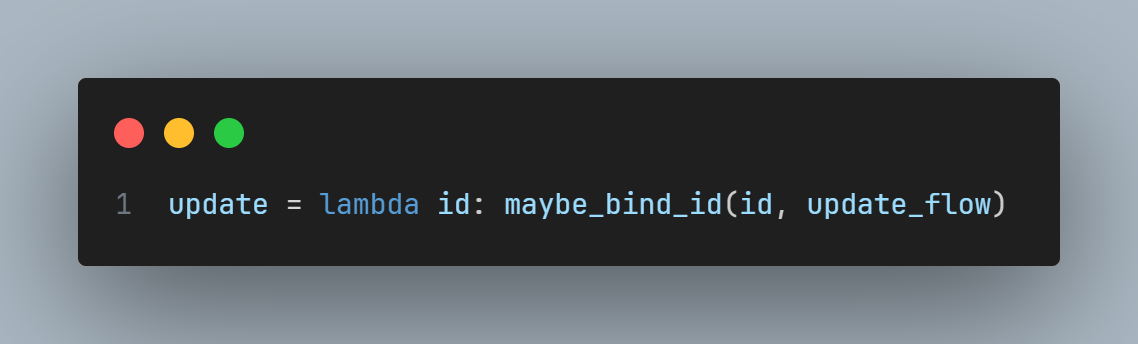
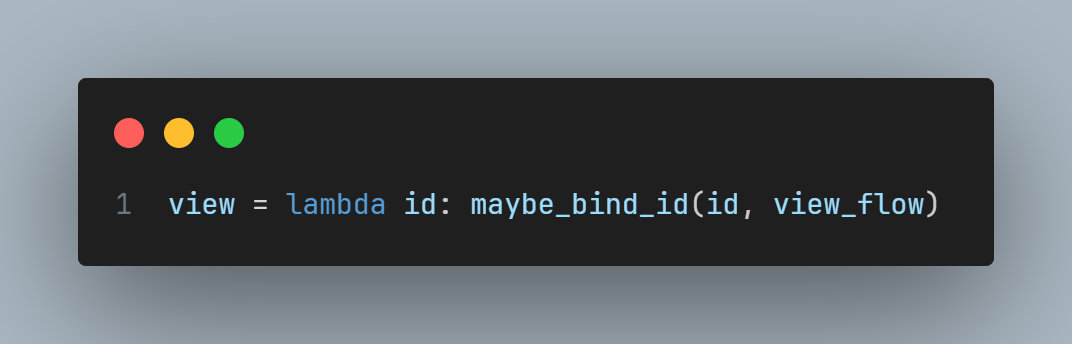
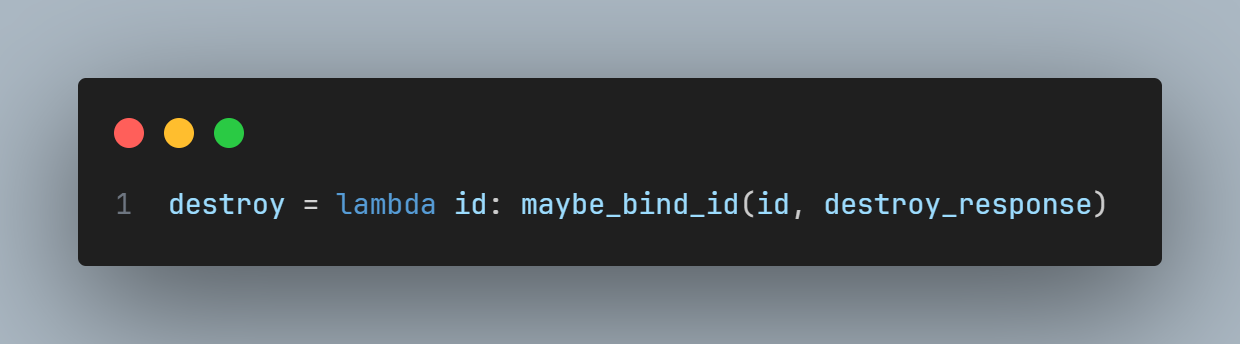
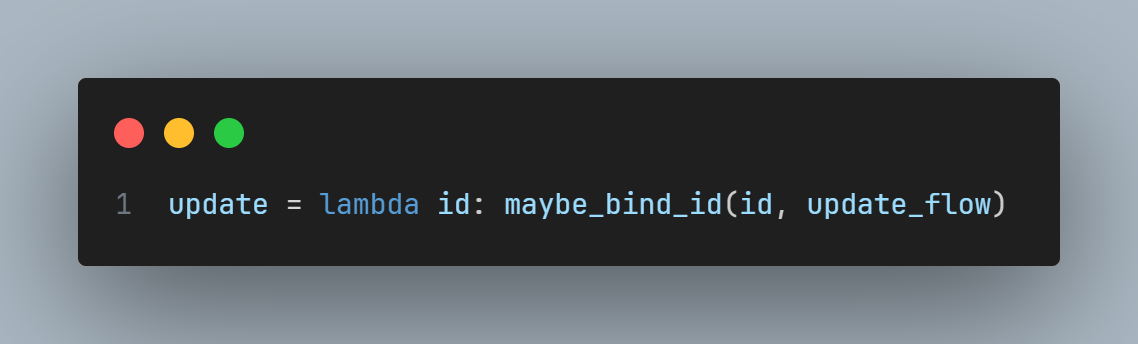
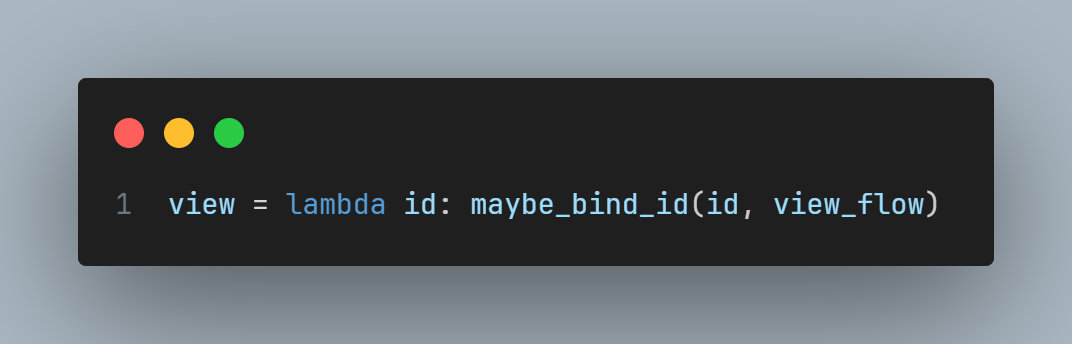
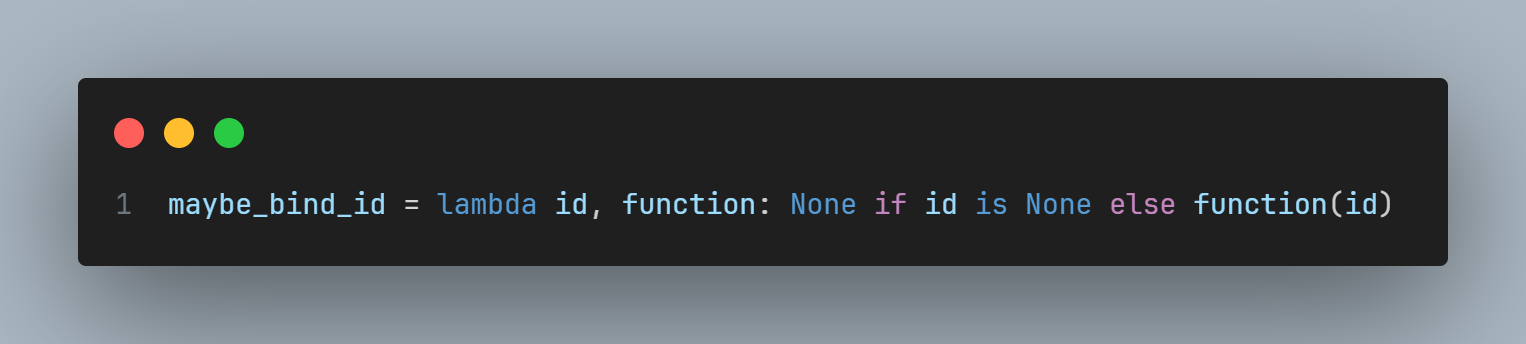


* 1. **Functor (map, filter, reduce…)**

Os functores são utilizados na função de validação “is\_valid” e de conversão de modelos para dicionário “models\_to\_dict”.



* 1. **Monad**



1. **Requisitos**
   1. **Funcionais**

| **ID** | **Funcionalidade** |
| --- | --- |
| **RF01** | O sistema deve possuir um endpoint de criação de produtos. |
| **RF02** | O sistema deve possuir um endpoint de atualização parcial de produtos. |
| **RF03** | O sistema deve possuir um endpoint de atualização integral de produtos. |
| **RF04** | O sistema deve possuir um endpoint de listagem de produtos. |
| **RF05** | O sistema deve possuir um endpoint de listagem individual de produtos. |
| **RF06** | O sistema deve possuir um endpoint de exclusão de produtos. |
| **RF07** | O sistema deve possuir um endpoint de criação de categorias. |
| **RF08** | O sistema deve possuir um endpoint de atualização parcial de categorias. |
| **RF09** | O sistema deve possuir um endpoint de atualização integral de categorias. |
| **RF10** | O sistema deve possuir um endpoint de listagem de categorias. |
| **RF11** | O sistema deve possuir um endpoint de listagem individual de categorias. |
| **RF12** | O sistema deve possuir um endpoint de exclusão de categorias. |
| **RF13** | O sistema deve possuir um endpoint de criação de usuários. |
| **RF14** | O sistema deve possuir um endpoint de autenticação de usuários. |

O arquivo “product.py”, em “src/routes”, contém a aplicação dos primeiros 6 requisitos funcionais (01 - 06), visto que possui todos os endpoints relacionados à entidade produto, realizando operações no banco de dados através de seus serviços.

O arquivo “category.py”, em “src/routes”, contém a aplicação dos próximos 6 requisitos funcionais (07 - 12), nele estão os endpoints relacionados à entidade categoria.

Por último, os requisitos funcionais 13 e 14 estão nos arquivos “user.py” e “auth.py”, em “src/routes”, respectivamente.

* 1. **Não-funcionais**

| **ID** | **Funcionalidade** |
| --- | --- |
| **RNF01** | O sistema deve utilizar a linguagem de programação Python e as bibliotecas Flask API e SQL Alchemy para o servidor web. |
| **RNF02** | O sistema deve utilizar o banco de dados SQLite. |
| **RNF03** | O sistema deve utilizar os conceitos da programação funcional (lambda, currying, recursividade, monad…) nos seus endpoints e serviços. |
| **RNF04** | O sistema deve utilizar JWT e criptografía de dados para o sistema de autenticação. |

O sistema utiliza a linguagem de programação Python e as bibliotecas Flask API (para o servidor web) e SQL Alchemy (para conexão com o banco de dados SQLite).

Todos os endpoints e serviços utilizam o conceito de programação funcional (lambda, currying, recursividade, monad, etc).

Além disso, o sistema de autenticação do sistema utiliza a biblioteca PyJWT para criação de JWT e criptografia de dados da biblioteca Werkzeug.