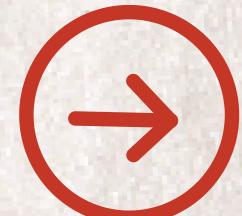


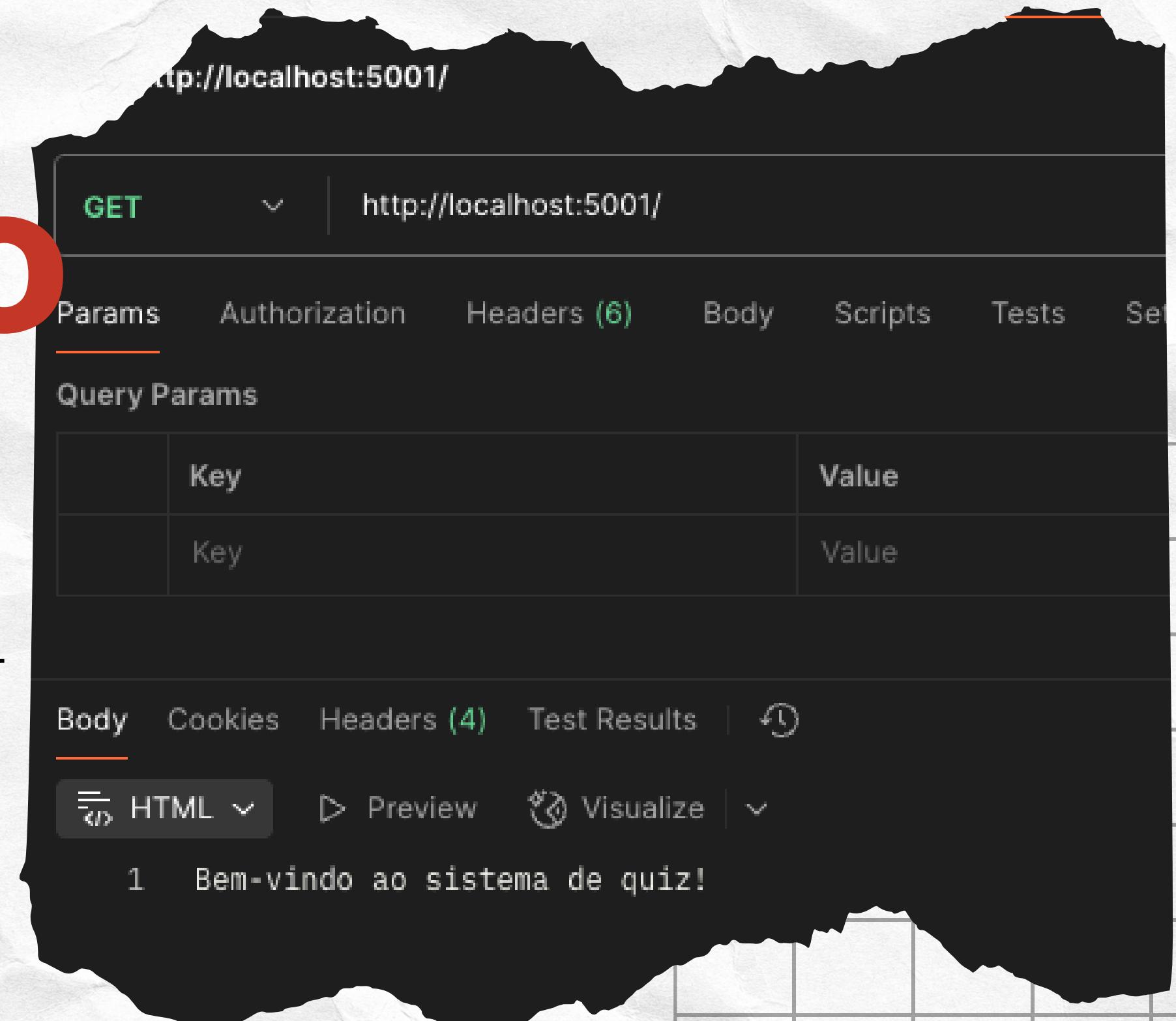
# InMemory DB Quiz em Redis



# Introdução

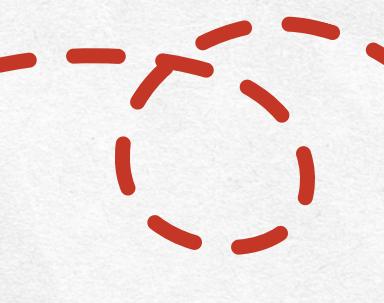
Este projeto implementa um sistema de quiz online usando **Flask** para o backend e **Redis** como **banco de dados**.

Professores podem criar quizzes, adicionar perguntas e alternativas, e os alunos podem responder as perguntas, com resultados sendo computados e exibidos em tempo real.



# Estrutura dos dados

1. **quiz**: Representa o objeto principal que contém todas as perguntas de um quiz.
  - Dentro de quiz, temos a chave questions que é uma lista de todas as perguntas no quiz.
2. **question**: Cada question possui uma lista de answers e também um campo para armazenar o response time.
  - answers: Uma lista de alternativas para cada pergunta.
  - response time: O tempo que o aluno levou para responder aquela pergunta.
3. **answer**: Cada answer possui um campo para votes e o tempo de resposta.
  - votes: Contabiliza quantos votos foram dados a cada alternativa (qual alternativa foi mais escolhida pelos alunos).
  - response time: O tempo em que a alternativa foi selecionada pelo aluno.
4. **votes**: Um campo para armazenar quantos alunos selecionaram uma determinada alternativa. Esse é um contador para cada alternativa de uma pergunta.



# Estrutura dos dados

```
quiz:<quiz_id>
  |
  +-- questions
    |
    +-- question_1
      |
      +-- answers
        |
        +-- answer_1
          |
          +-- votes: <count>
          |
          +-- response_time: <time_taken>
        |
        +-- answer_2
          |
          +-- votes: <count>
          |
          +-- response_time: <time_taken>
        |
        +-- answer_3
          |
          +-- votes: <count>
          |
          +-- response_time: <time_taken>
          |
          +-- response_time: <average_time_taken>
          |
          +-- (outros campos relacionados à questão)
          |
          +-- (outras questões)
    |
    +-- (outros campos relacionados ao quiz)
```

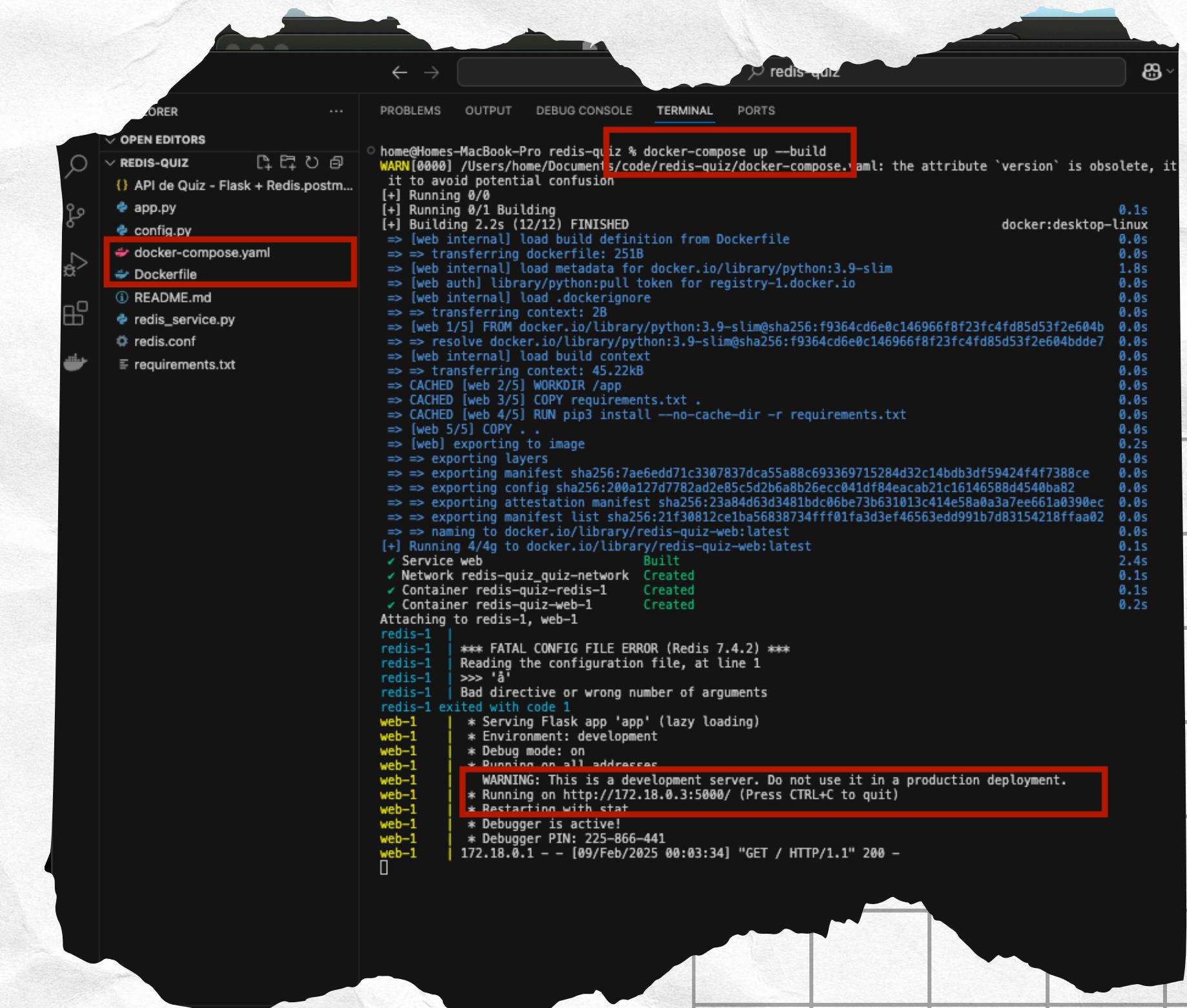
|  |  |  |
|--|--|--|
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Como rodar?

Nosso projeto é todo configurado em docker, então desde que você tenha o docker instalado em sua máquina, é possível rodar o projeto sem nenhuma configuração adicional, e só rodar:

- **docker-compose up --build**

na raiz do projeto que o docker já faz tudinho por você

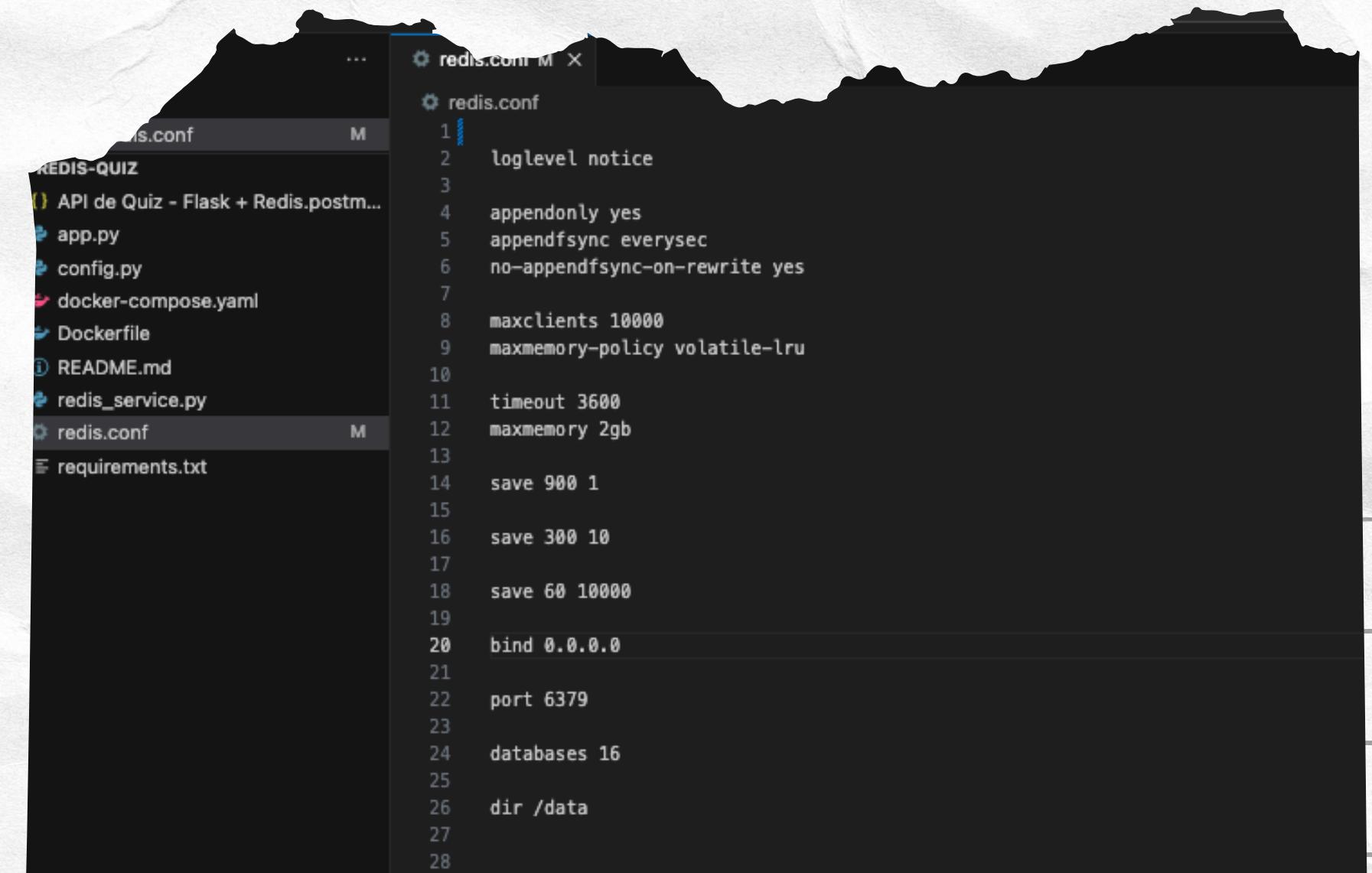


The screenshot shows a terminal window within a dark-themed IDE interface. The terminal tab is selected at the top. The command entered is `docker-compose up --build`. The output shows the build process for a service named `redis-quiz`. It starts with a warning about the `version` attribute being obsolete. The build process involves several steps: running Dockerfiles, transferring files, loading metadata, pulling tokens, and building contexts. It then creates a network (`redis-quiz_quiz-network`), two Redis containers (`redis-1` and `redis-2`), and two web containers (`web-1` and `web-2`). The Redis containers fail with fatal configuration errors due to a bad directive. The web containers start successfully, serving the Flask application on port 5000. A warning message indicates that this is a development server and should not be used in production.

```
home@Homes-MacBook-Pro redis-quiz % docker-compose up --build
WARN[000] /Users/home/Documents/code/redis-quiz/docker-compose.yaml: the attribute `version` is obsolete, it is to avoid potential confusion
[+] Running 0/0
[+] Running 0/1 Building
[+] Building 2.2s (12/12) FINISHED
--> [web internal] load build definition from Dockerfile
--> => transferring dockerfile: 251B
--> [web internal] load metadata for docker.io/library/python:3.9-slim
--> [web auth] library/python:pull token for registry-1.docker.io
--> [web internal] load .dockerignore
--> => transferring context: 2B
--> [web 1/5] FROM docker.io/library/python:3.9-slim@sha256:f9364cd6e0c146966f8f23fc4fd85d53f2e604b
--> => resolve docker.io/library/python:3.9-slim@sha256:f9364cd6e0c146966f8f23fc4fd85d53f2e604bdde7
--> [web internal] load build context
--> => transferring context: 45.22kB
--> CACHED [web 2/5] WORKDIR /app
--> CACHED [web 3/5] COPY requirements.txt .
--> CACHED [web 4/5] RUN pip3 install --no-cache-dir -r requirements.txt
--> [web 5/5] COPY .
--> [web] exporting to image
--> => exporting layers
--> => exporting manifest sha256:7ae6edd71c3307837dca55a88c693369715284d32c14bdb3df59424f4f7388ce
--> => exporting config sha256:200a127d7782ad2e85c5d2b6a8b26ecc041df84eacab21c16146588d4540ba82
--> => exporting attestation manifest sha256:23a84d63d3481bcd06be73b631013c414e58a0a3a7ee661a0390ec
--> => exporting manifest list sha256:21f30812ce1ba56838734fff01fa3d3ef46563edd991b7d83154218ffaa02
--> => naming to docker.io/library/redis-quiz-web:latest
[+] Running 4/4g to docker.io/library/redis-quiz-web:latest
  Service web      Built
  Network redis-quiz_quiz-network Created
  Container redis-quiz-redis-1   Created
  Container redis-quiz-web-1     Created
Attaching to redis-1, web-1
redis-1 | *** FATAL CONFIG FILE ERROR (Redis 7.4.2) ***
redis-1 | Reading the configuration file, at line 1
redis-1 | >>> 'â'
redis-1 | Bad directive or wrong number of arguments
redis-1 exited with code 1
web-1 | * Serving Flask app 'app' (lazy loading)
web-1 | * Environment: development
web-1 | * Debug mode: on
web-1 | * Running on all addresses
web-1 | WARNING: This is a development server. Do not use it in a production deployment.
web-1 | * Running on http://172.18.0.3:5000/ (Press CTRL+C to quit)
web-1 | * Restarting with stat
web-1 | * Debugger is active!
web-1 | * Debugger PIN: 225-866-441
web-1 | 172.18.0.1 - - [09/Feb/2025 00:03:34] "GET / HTTP/1.1" 200 -
```

# Requisitos

Para atender os requisitos de performance e usabilidade, o **arquivo redis.conf** contém diversas configurações que garantem o desempenho e performance da aplicação



The screenshot shows a code editor with the file `redis.conf` open. The file contains the following configuration options:

```
1 loglevel notice
2 appendonly yes
3 appendfsync everysec
4 no-appendfsync-on-rewrite yes
5 maxclients 10000
6 maxmemory-policy volatile-lru
7 timeout 3600
8 maxmemory 2gb
9 save 900 1
10 save 300 10
11 save 60 10000
12 bind 0.0.0.0
13 port 6379
14 databases 16
15 dir /data
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

home@Homes-MacBook-Pro redis-quiz % docker-compose up --build  
WARN[0000] /Users/home/Documents/code/redis-quiz/docker-compose.yaml: the attribute `ver  
move it to avoid potential confusion  
[+] Running 0/0  
[+] Running 0/1 Building  
Building 2.2s (12/12) FINISHED  
internal

# Requisitos

Já o arquivo **`redis_service.py`** é onde estão os métodos que utilizamos para manipular as cargas e consultas no redis, além de ser onde definimos o **TTL padrão de 30** dias para armazenamento de todas as chaves

The screenshot shows a code editor interface with a dark theme. The left sidebar lists project files: `redis.conf`, `redis_service.py`, `app.py`, `config.py`, `docker-compose.yaml`, `Dockerfile`, `README.md`, and `requirements.txt`. The right pane displays the content of `redis_service.py`:

```
1 import json
2 from config import get_redis_client
3 # TTL padrão em segundos (30 dias)
4 TTL_PADRAO = 30 * 24 * 60 * 60 # 30 dias em segundos
5
6 def create_quiz(quiz_id, title):
7     redis_client = get_redis_client()
8
9     redis_client.hset(f"quiz:{quiz_id}", "title", title)
10    redis_client.expire(f"quiz:{quiz_id}", TTL_PADRAO)
11    return {"status": "Quiz created successfully!"}
12
13 def add_question(quiz_id, question_id, question_data):
14     redis_client = get_redis_client()
15
16     redis_client.set(f"quiz:{quiz_id}:question:{question_id}", json.dumps(question_data))
17     redis_client.expire(f"quiz:{quiz_id}:question:{question_id}", TTL_PADRAO)
18
19     return {"status": "Question added successfully!"}
20
21 def get_quiz(quiz_id):
22     redis_client = get_redis_client()
23
24     title = redis_client.hget(f"quiz:{quiz_id}", "title")
25
26     return {"quiz_id": quiz_id, "title": title}
27
28 def get_question(quiz_id, question_id):
29     redis_client = get_redis_client()
30
31     question_data = redis_client.get(f"quiz:{quiz_id}:question:{question_id}")
32
33     if question_data:
```

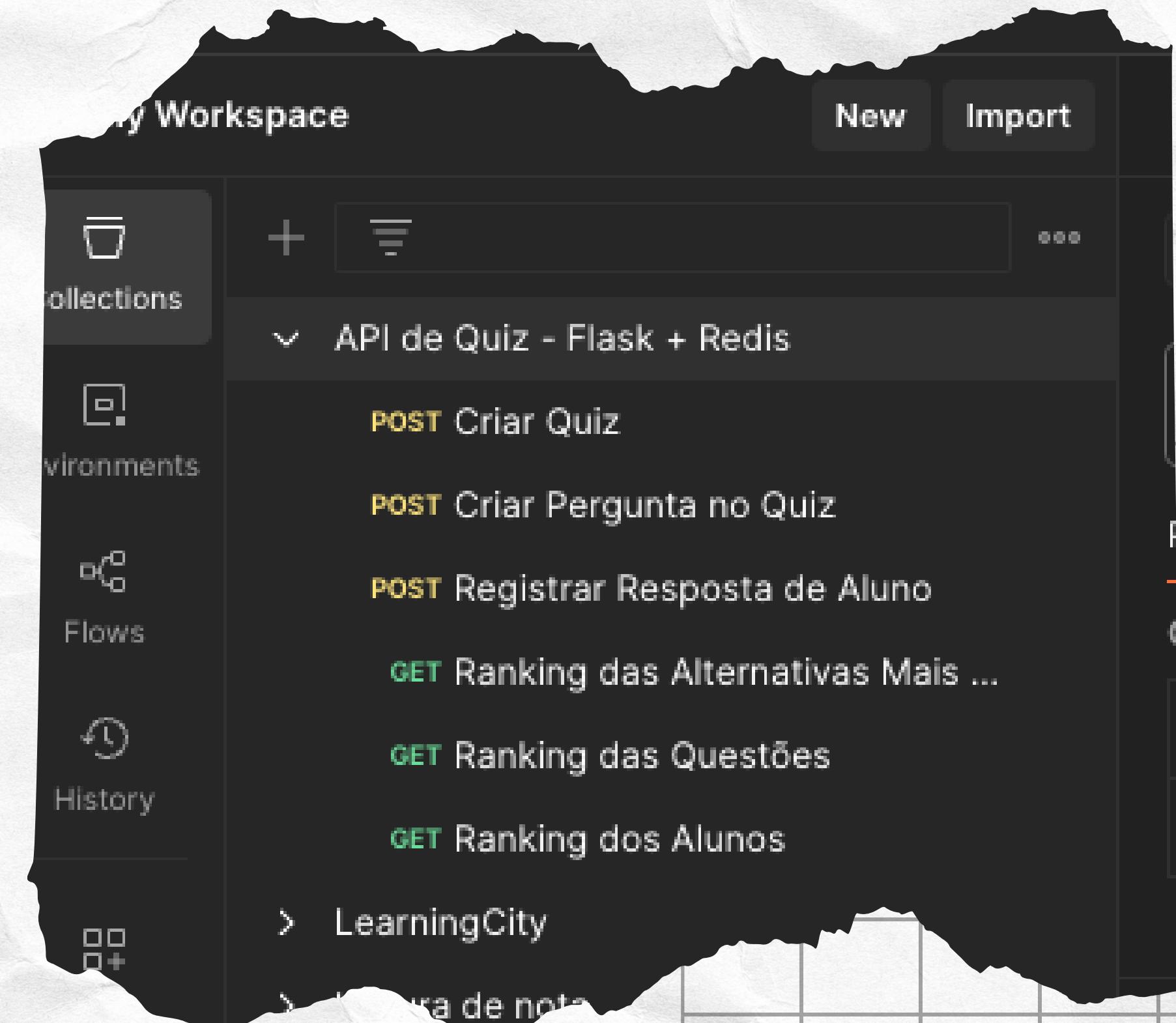
Below the code editor is a terminal window showing the output of a `docker-compose up --build` command:

```
home@Home-MacBook-Pro redis-quiz % docker-compose up --build
WARN[0000] /Users/home/Documents/code/redis-quiz/docker-compose.yaml: the attribute `version` is obsolete, it
move it to avoid potential confusion
[+] Running 0/0
[+] Running 0/1 Building
[+] Building 2.2s (12/12) FINISHED
=> [web internal] load build definition from Dockerfile
=> => transferring dockerfile: 251B
=> [web internal] load metadata for docker.io/library/python:3.8-slim
=> [web auth] library/python:pull token for docker.io/library/python:3.8-slim
=> [web internal] load .dockerignore
=> => transferring context: 2B
=> => 1/5 FROM docker.io/library/python:3.8-slim
=> =>   resolve docker.io/library/python:3.8-slim
=> =>   0.1s
=> =>   0.0s
=> =>   0.0s
=> =>   1.8s
=> =>   0.0s
=> =>   0.0s
=> =>   0.0s
```

# Requisitos

E no arquivo **API de Quiz - Flask + Redis.postman\_collection.json** é onde você encontra a nossa collection pra rodar a API.

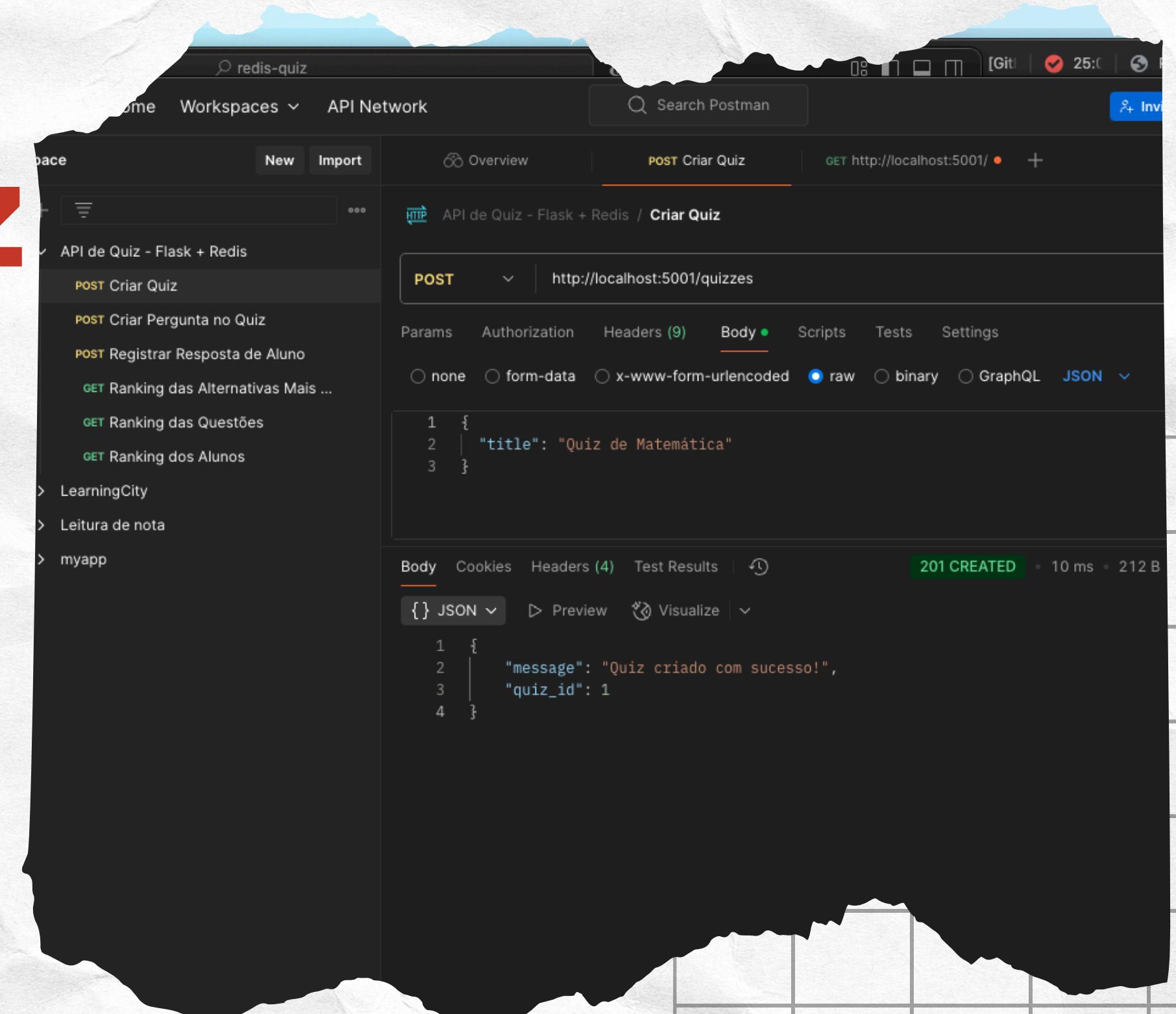
É só importar no postman e está tudo pronto pra começar. Vamos?



# Criando o Quiz

A primeira API da nossa collection, é onde os professores vão cadastrar os quizzes.

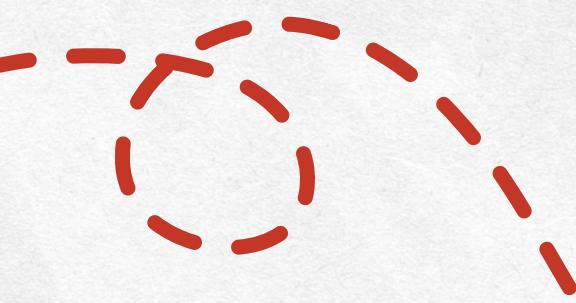
Quando um quiz é criado há uma mensagem de sucesso no retorno, assim como o ID do quiz que acaba de ser criado no banco



# Perguntas

Para adicionar perguntas também é bem fácil, basta executar a proxima API da collection, passando o ID retornado anteriormente, com os dados da pergunta que quer cadastrar.

Em seu retorno também há uma mensagem de sucesso com o ID cadastrado



The screenshot shows the Postman application interface. The left sidebar lists workspaces, and the main area shows a collection named "API de Quiz - Flask + Redis" with several endpoints listed under it. A specific POST request titled "Criar Pergunta no Quiz" is selected. The "Body" tab is active, showing a JSON payload:

```
1 {  
2   "question": "Quanto é 2 + 2?",  
3   "alternatives": ["A. 3", "B. 4", "C. 5", "D. 6"]  
4 }
```

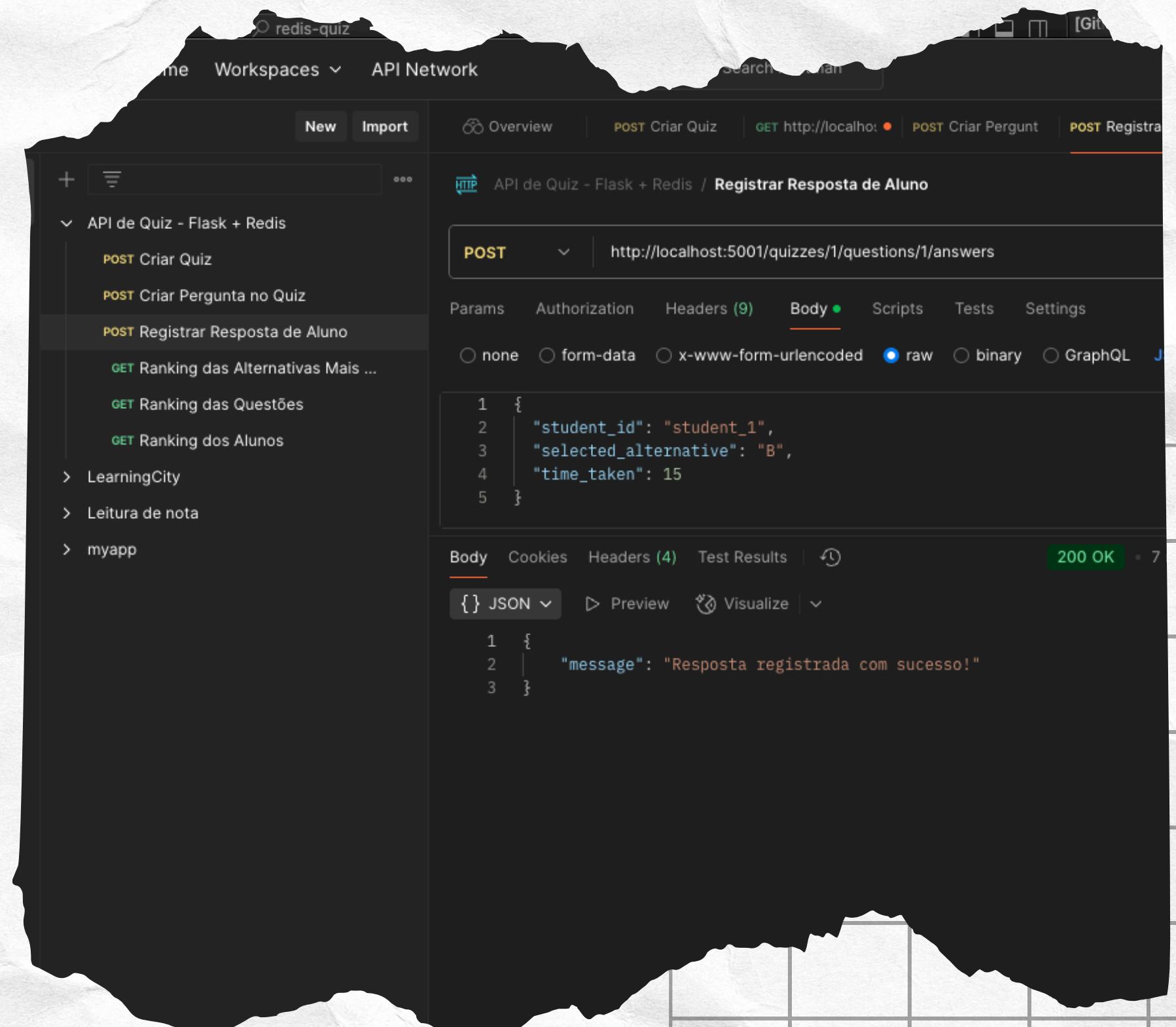
Below the body, the response is shown with a status of "201 CREATED". The response JSON is:

```
1 {  
2   "message": "Pergunta criada com sucesso!",  
3   "question_id": 2  
4 }
```

# Respostas

Para registrar as respostas também é bem fácil, basta executar a próxima API da collection, passando o ID retornado anteriormente, com os dados da resposta do aluno.

Em seu retorno também há uma mensagem de sucesso.



# Rankings

Nós temos na proxima API o ranking das alternativas mais votadas, onde passamos o id do quizz e o id da pergunta, pra ver qual alternativa os alunos mais votaram

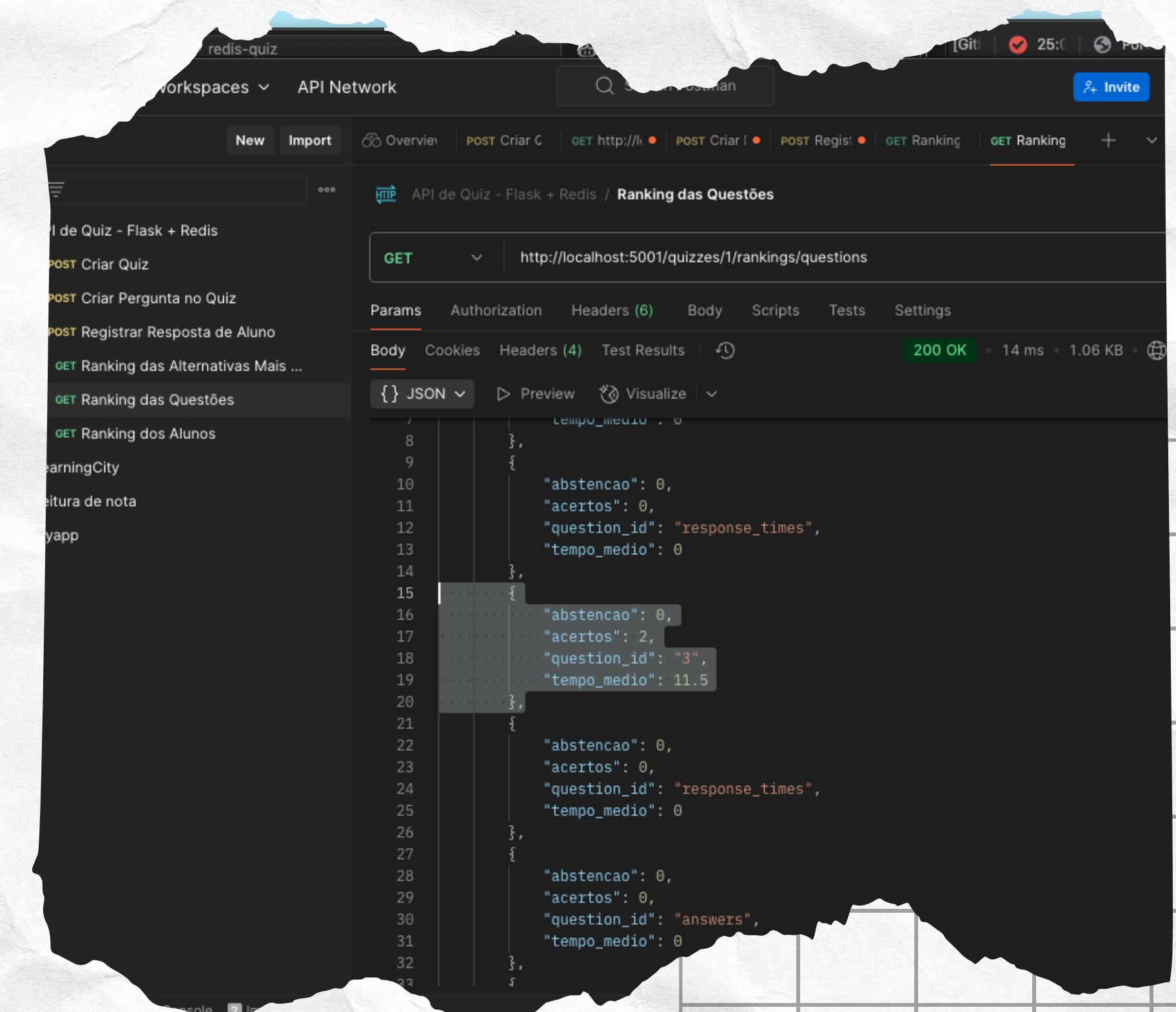
The screenshot shows the Postman application interface. The workspace is named "API de Quiz - Flask + Redis". A GET request is selected with the URL `http://localhost:5001/quizzes/1/questions/1/rankings/alternatives`. The "Body" tab displays a JSON response:

```
1 {  
2   "ranking": {  
3     "B": 1  
4   }  
5 }
```

# INMEMORY DB - QUIZ EM REDIS

# Rankings

Nós temos na proxima API o ranking das questões, onde podemos ver o número de abstenções, tempo médio de resposta e a quantidade de acertos



# Rankings

E na próxima, e ultima temos o ranking de alunos, com numero de acertos e tempo médio entre as respostas.

The screenshot shows the Postman application interface. The left sidebar lists workspaces, and the main area shows an API endpoint for 'API de Quiz - Flask + Redis'. The endpoint is 'Ranking dos Alunos' (GET http://localhost:5001/quizzes/1/rankings/students). The 'Body' tab displays a JSON response:

```
1 {  
2   "ranking": [  
3     {  
4       "acertos": 1,  
5       "student_id": "student_1",  
6       "tempo_medio": 3.0  
7     },  
8     {  
9       "acertos": 1,  
10      "student_id": "student_4",  
11      "tempo_medio": 20.0  
12    }  
13  ]  
14 }
```

Com estas APIs é possível construir um aplicativo dinâmico de quizzes com um desempenho incrível e configuração prática.

Seu código está disponível gratuitamente no repositório  
**<https://github.com/daniela-csantos/rediz-quiz>**

Esperamos que você se divirta muito!



# Obrigada

