Project Symmetry

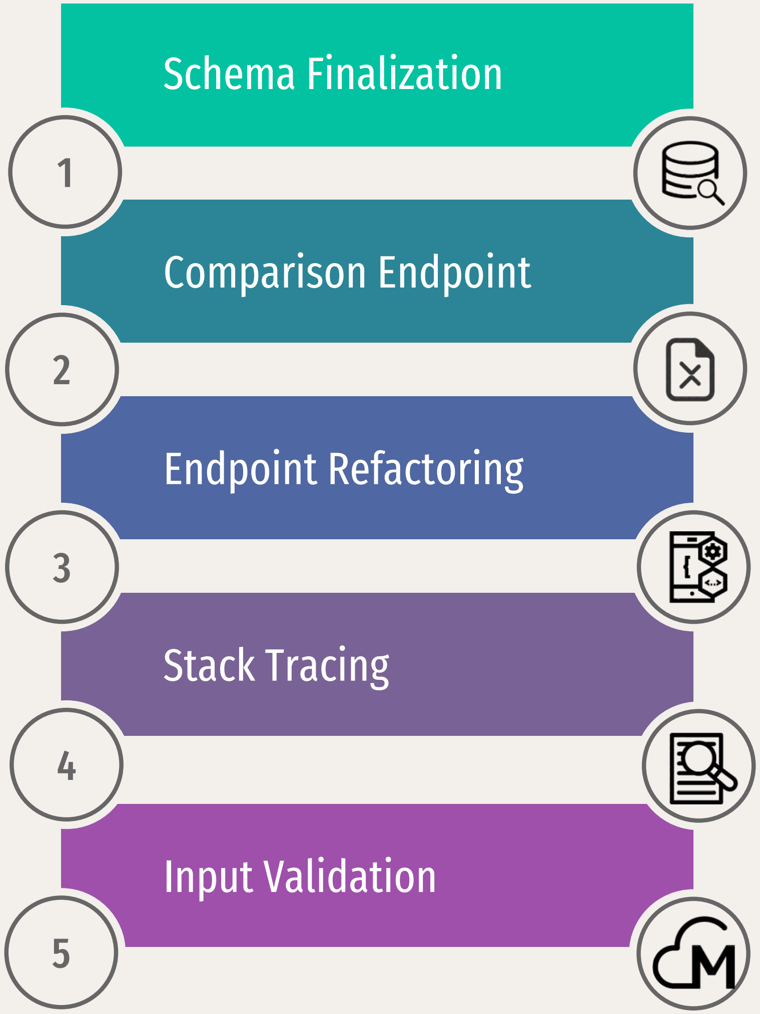
🥐Team Scrumptious🥐

API & Middleware

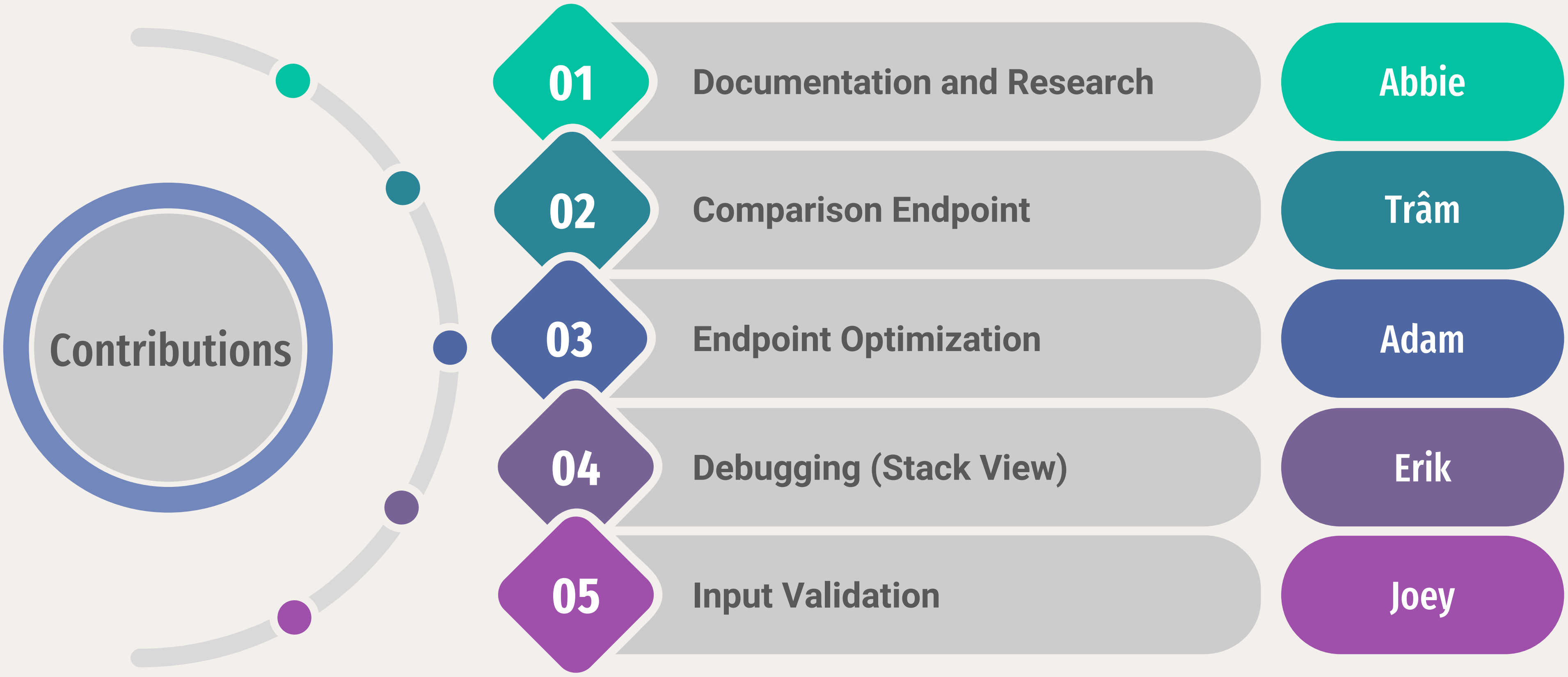
Accomplishments

Planned: 23/77

Completed: 25/77



1. Schema Finalization
2. Comparison Endpoint
3. Endpoint Refactoring
4. Stack Tracing
5. Input Validation



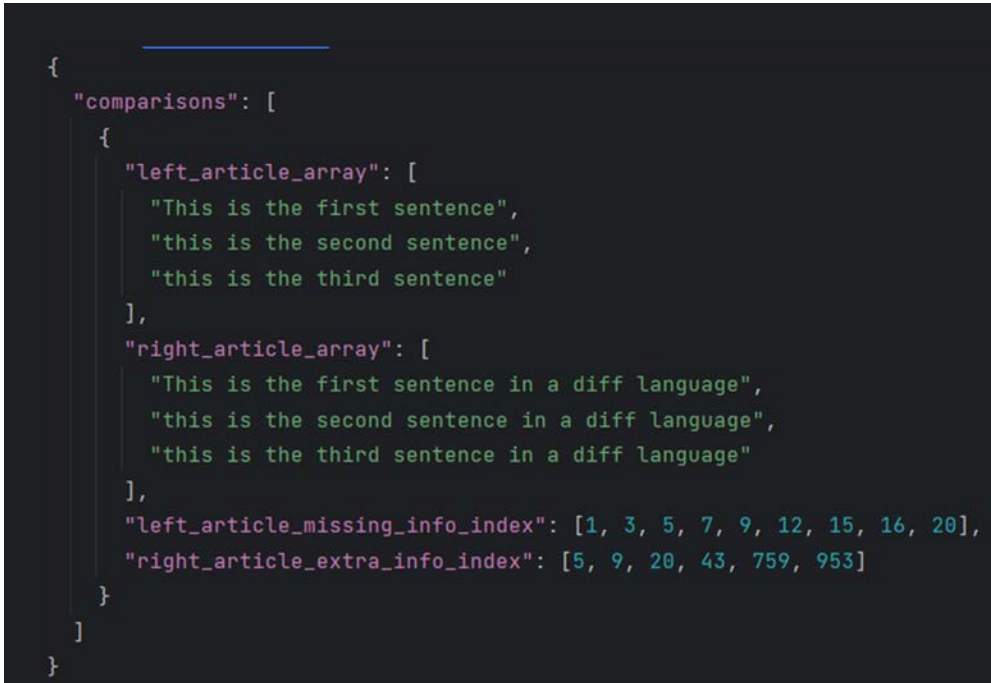
Contributions

01	Documentation and Research	Abbie
02	Comparison Endpoint	Trâm
03	Endpoint Optimization	Adam
04	Debugging (Stack View)	Erik
05	Input Validation	Joey

User Story 1

As a back-end developer, I want to ensure that APIs align with the front-end team’s requirements so that the data delivered is in a format that can be seamlessly integrated into the UI, ensuring a consistent and efficient user experience.

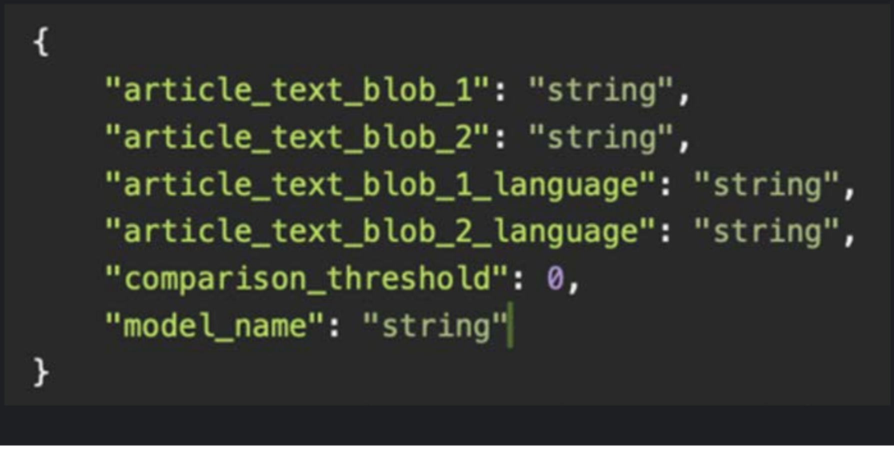
Size: 5



```
{
  "comparisons": [
    {
      "left_article_array": [
        "This is the first sentence",
        "this is the second sentence",
        "this is the third sentence"
      ],
      "right_article_array": [
        "This is the first sentence in a diff language",
        "this is the second sentence in a diff language",
        "this is the third sentence in a diff language"
      ],
      "left_article_missing_info_index": [1, 3, 5, 7, 9, 12, 15, 16, 20],
      "right_article_extra_info_index": [5, 9, 20, 43, 759, 953]
    }
  ]
}
```

Backend → Frontend

Frontend → Backend

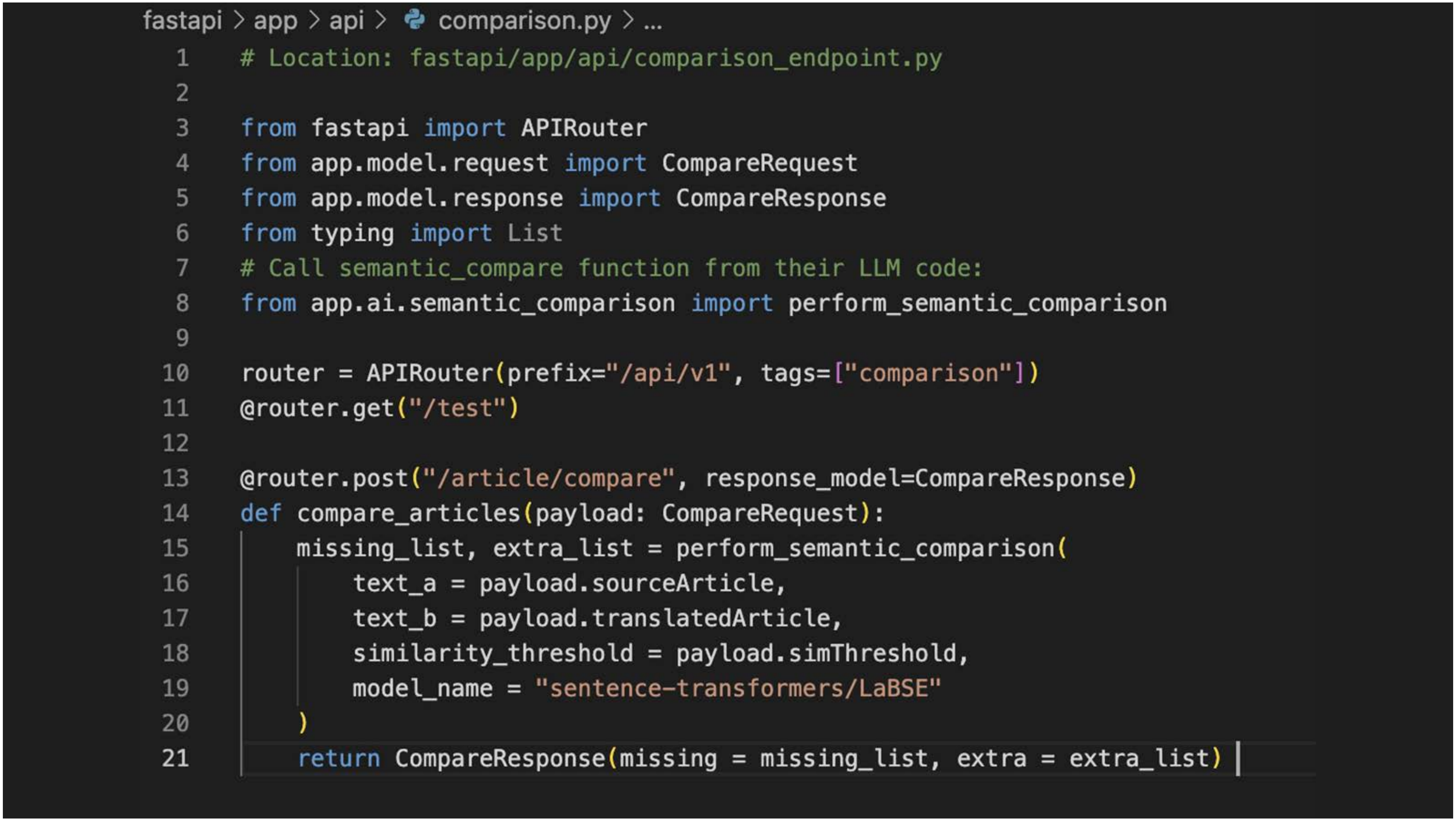


```
{
    "article_text_blob_1": "string",
    "article_text_blob_2": "string",
    "article_text_blob_1_language": "string",
    "article_text_blob_2_language": "string",
    "comparison_threshold": 0,
    "model_name": "string"
}
```

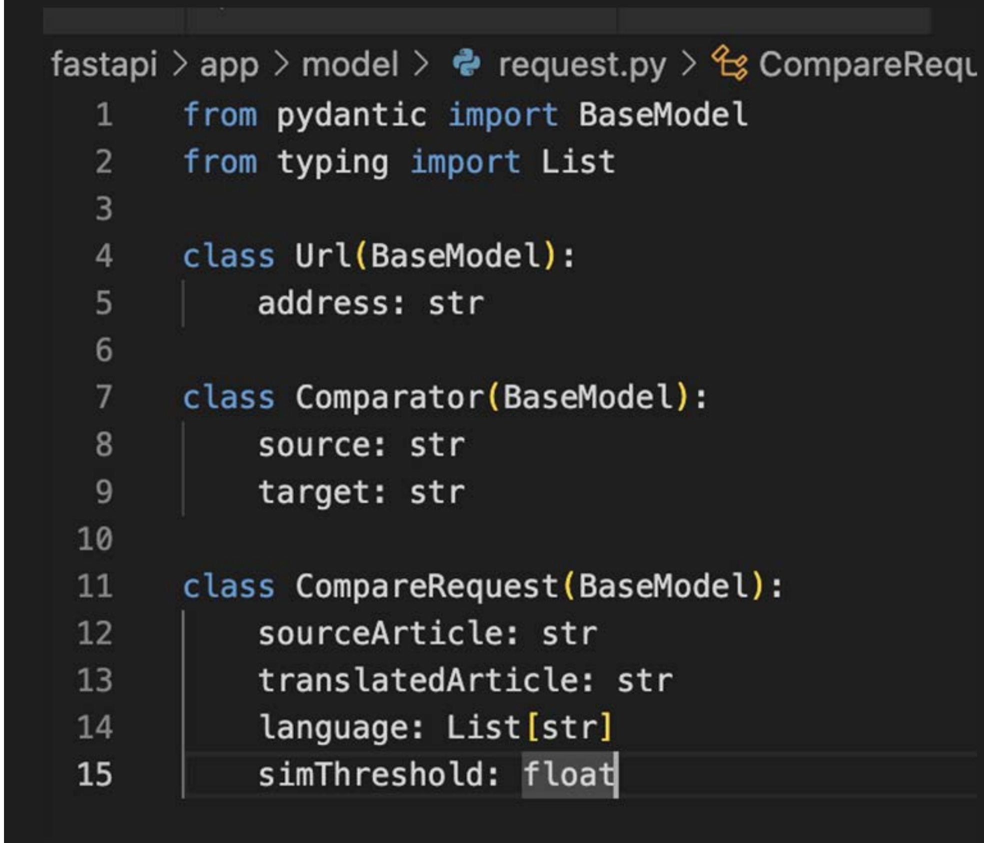
User Story 2

As a front-end developer, I want an API that retrieves both the original and translated versions of an article so that the front end can render and display this content.

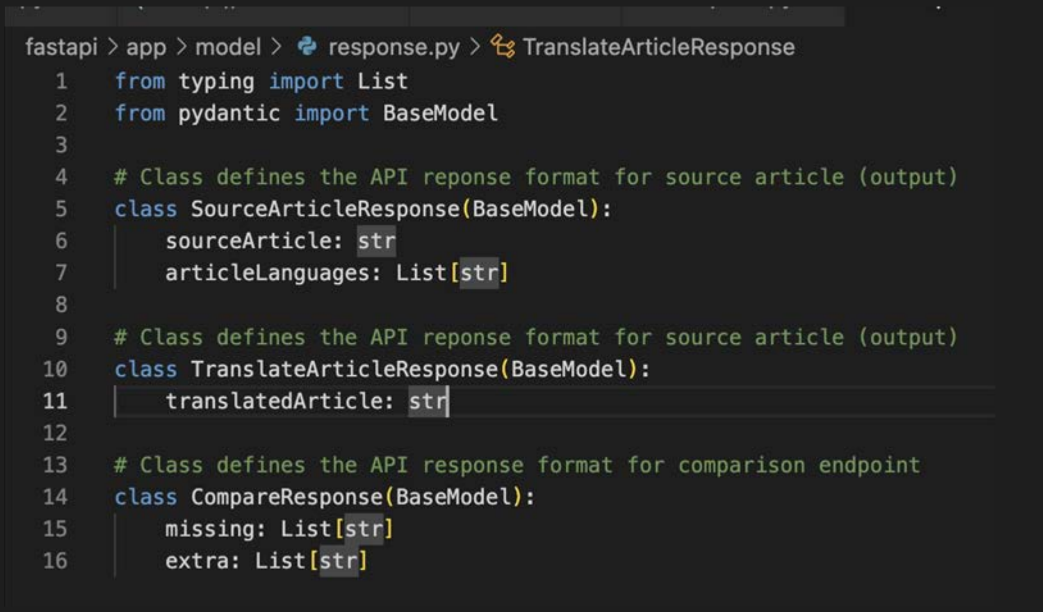
Size: 5



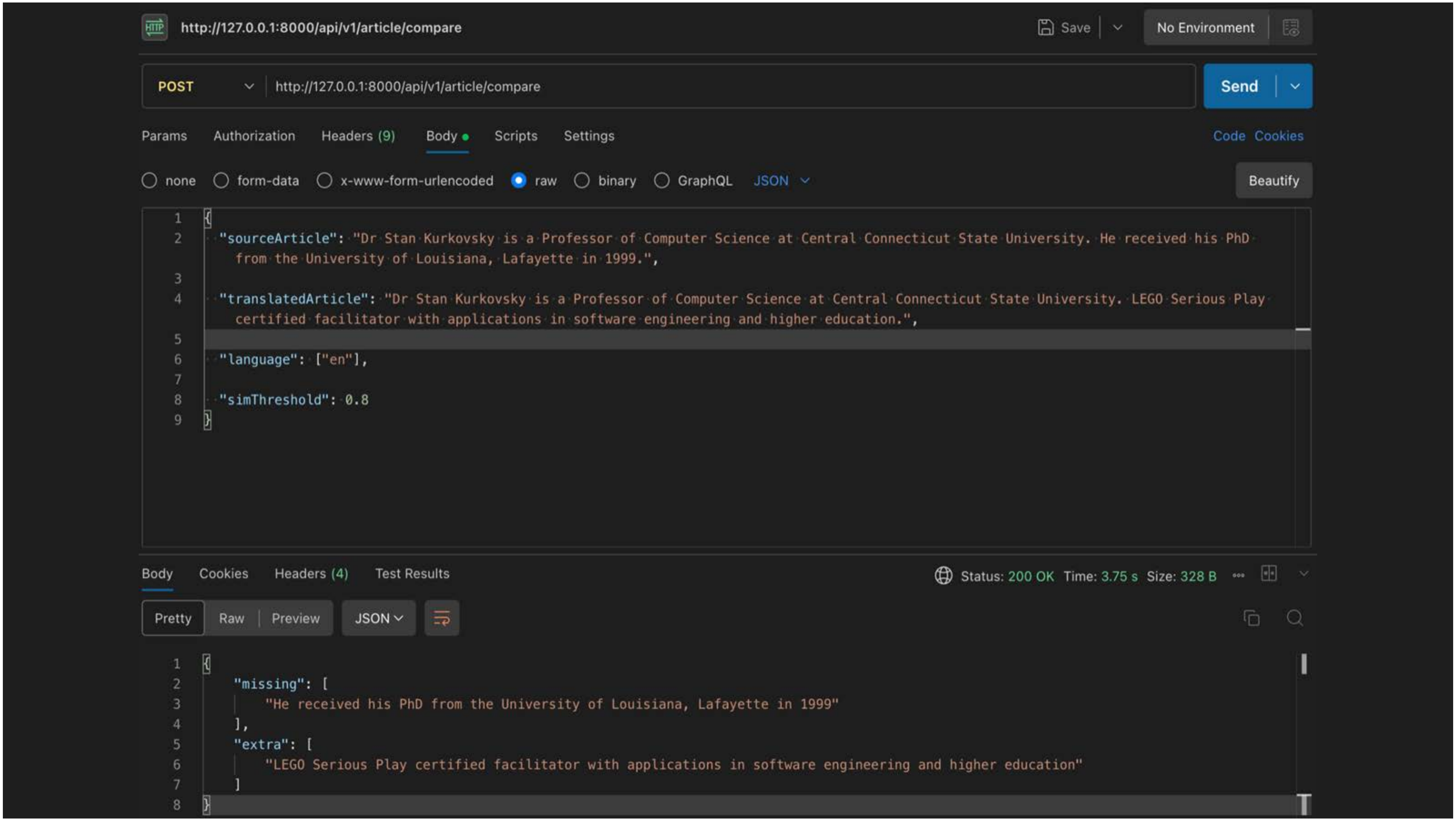
```python
# Location: fastapi/app/api/comparison_endpoint.py

from fastapi import APIRouter
from app.model.request import CompareRequest
from app.model.response import CompareResponse
from typing import List
# Call semantic_compare function from their LLM code:
from app.ai.semantic_comparison import perform_semantic_comparison

router = APIRouter(prefix="/api/v1", tags=["comparison"])
@router.get("/test")

@router.post("/article/compare", response_model=CompareResponse)
def compare_articles(payload: CompareRequest):
    missing_list, extra_list = perform_semantic_comparison(
        text_a = payload.sourceArticle,
        text_b = payload.translatedArticle,
        similarity_threshold = payload.simThreshold,
        model_name = "sentence-transformers/LaBSE"
    )
    return CompareResponse(missing = missing_list, extra = extra_list)
```



```python
from pydantic import BaseModel
from typing import List

class Url(BaseModel):
    address: str

class Comparator(BaseModel):
    source: str
    target: str

class CompareRequest(BaseModel):
    sourceArticle: str
    translatedArticle: str
    language: List[str]
    simThreshold: float
```


fastapi > app > model > response.py > TranslateArticleResponse

```python
from typing import List
from pydantic import BaseModel

# Class defines the API reponse format for source article (output)
class SourceArticleResponse(BaseModel):
    sourceArticle: str
    articleLanguages: List[str]

# Class defines the API reponse format for source article (output)
class TranslateArticleResponse(BaseModel):
    translatedArticle: str

# Class defines the API response format for comparison endpoint
class CompareResponse(BaseModel):
    missing: List[str]
    extra: List[str]
```



POST http://127.0.0.1:8000/api/v1/article/compare

```json
{
  "sourceArticle": "Dr Stan Kurkovsky is a Professor of Computer Science at Central Connecticut State University. He received his PhD from the University of Louisiana, Lafayette in 1999.",

  "translatedArticle": "Dr Stan Kurkovsky is a Professor of Computer Science at Central Connecticut State University. LEGO Serious Play certified facilitator with applications in software engineering and higher education.",

  "language": ["en"],

  "simThreshold": 0.8
}
```

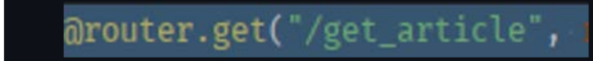
Status: 200 OK Time: 3.75 s Size: 328 B

```json
{
    "missing": [
        "He received his PhD from the University of Louisiana, Lafayette in 1999"
    ],
    "extra": [
        "LEGO Serious Play certified facilitator with applications in software engineering and higher education"
    ]
}
```

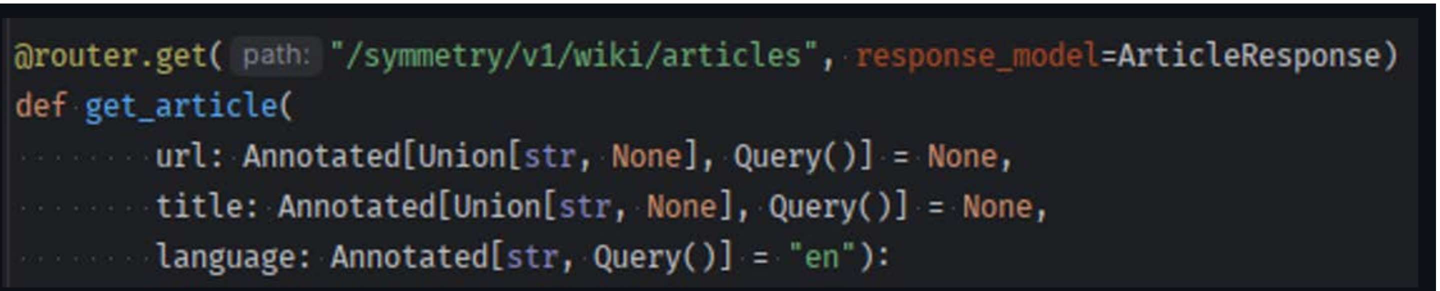
User Story 3

As a back-end developer, I want consistent response structures across all endpoints so that consuming services can reliably use the data.

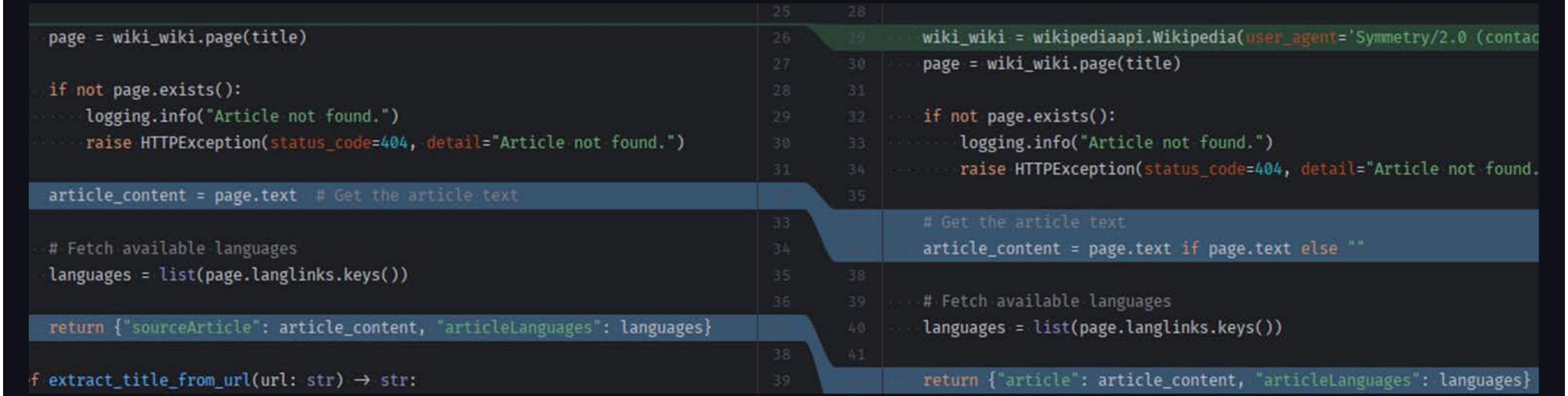
Size 5



```
@router.get("/get_article",
```

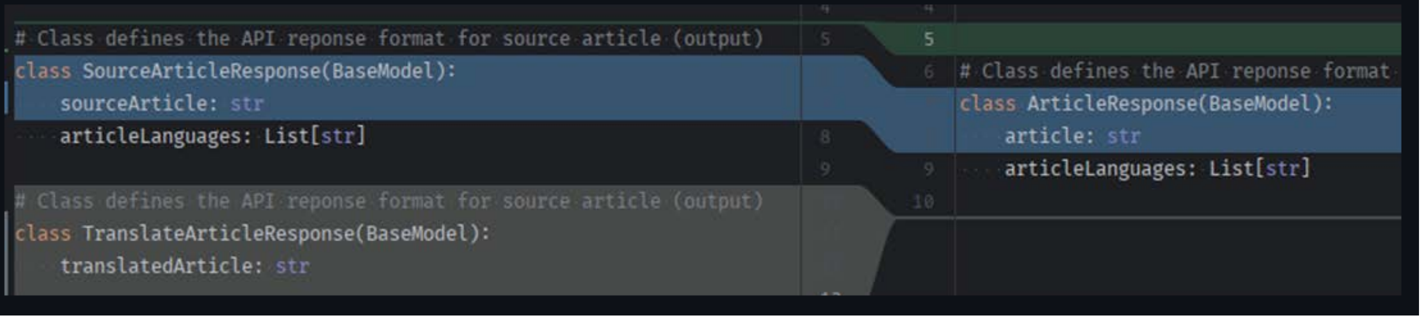


```
@router.get( path: "/symmetry/v1/wiki/articles", response_model=ArticleResponse)
def get_article(
        url: Annotated[Union[str, None], Query()] = None,
        title: Annotated[Union[str, None], Query()] = None,
        language: Annotated[str, Query()] = "en"):
```



```
    page = wiki_wiki.page(title)

    if not page.exists():
        logging.info("Article not found.")
        raise HTTPException(status_code=404, detail="Article not found.")

    article_content = page.text  # Get the article text

    # Fetch available languages
    languages = list(page.langlinks.keys())

    return {"sourceArticle": article_content, "articleLanguages": languages}

def extract_title_from_url(url: str) → str:
```

```
    wiki_wiki = wikipediaapi.Wikipedia(user_agent='Symmetry/2.0 (contac
    page = wiki_wiki.page(title)

    if not page.exists():
        logging.info("Article not found.")
        raise HTTPException(status_code=404, detail="Article not found.

    # Get the article text
    article_content = page.text if page.text else ""

    # Fetch available languages
    languages = list(page.langlinks.keys())

    return {"article": article_content, "articleLanguages": languages}
```



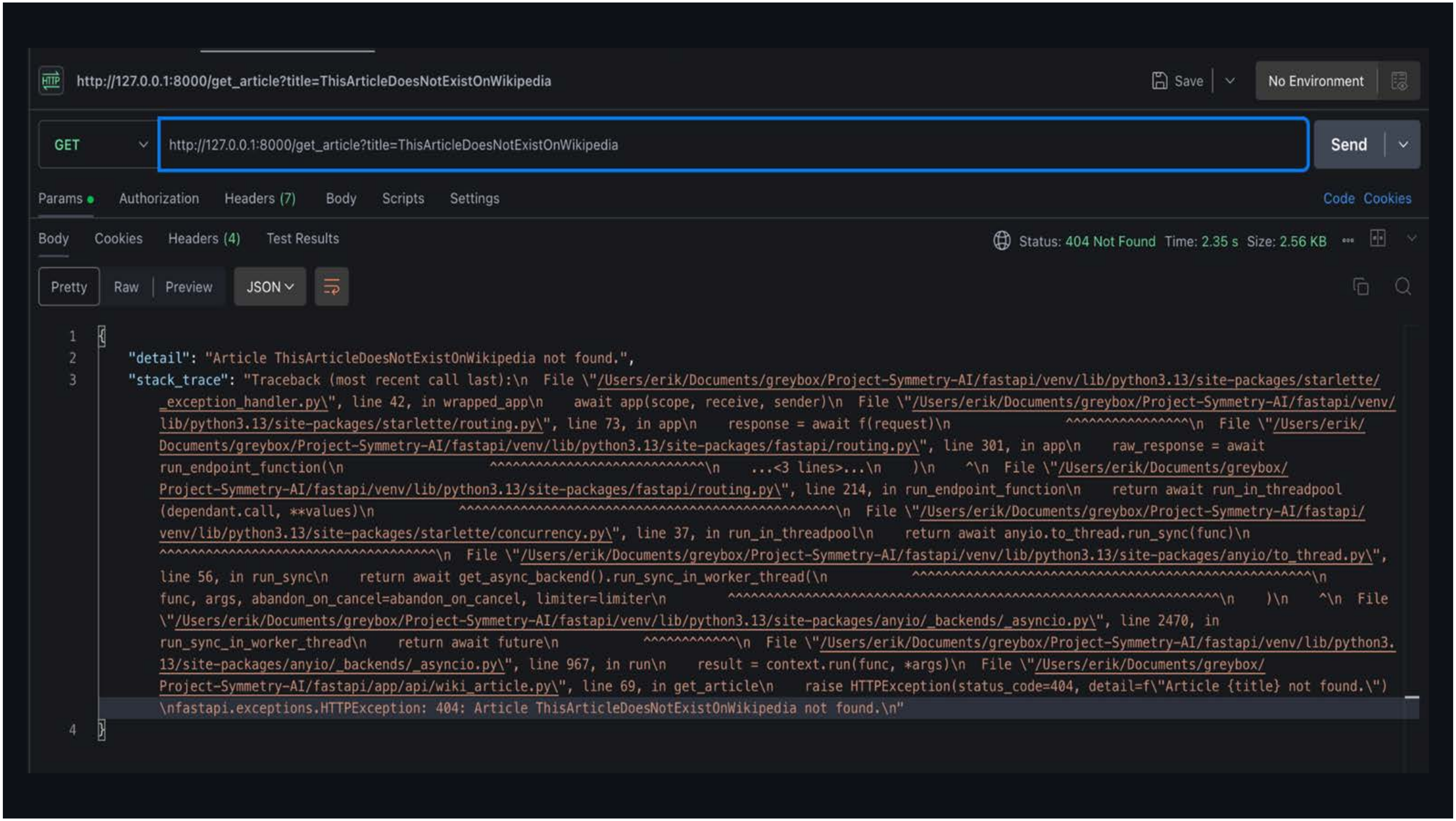
```
# Class defines the API reponse format for source article (output)
class SourceArticleResponse(BaseModel):
    sourceArticle: str
    articleLanguages: List[str]

# Class defines the API reponse format for source article (output)
class TranslateArticleResponse(BaseModel):
    translatedArticle: str
```

```
# Class defines the API reponse format
class ArticleResponse(BaseModel):
    article: str
    articleLanguages: List[str]
```

As a back-end developer, I want to be able to see the stack of function calls for each error message in order to make endpoint development more efficient

Size: 3

User Story 4



http://127.0.0.1:8000/get_article?title=ThisArticleDoesNotExistOnWikipedia

GET http://127.0.0.1:8000/get_article?title=ThisArticleDoesNotExistOnWikipedia

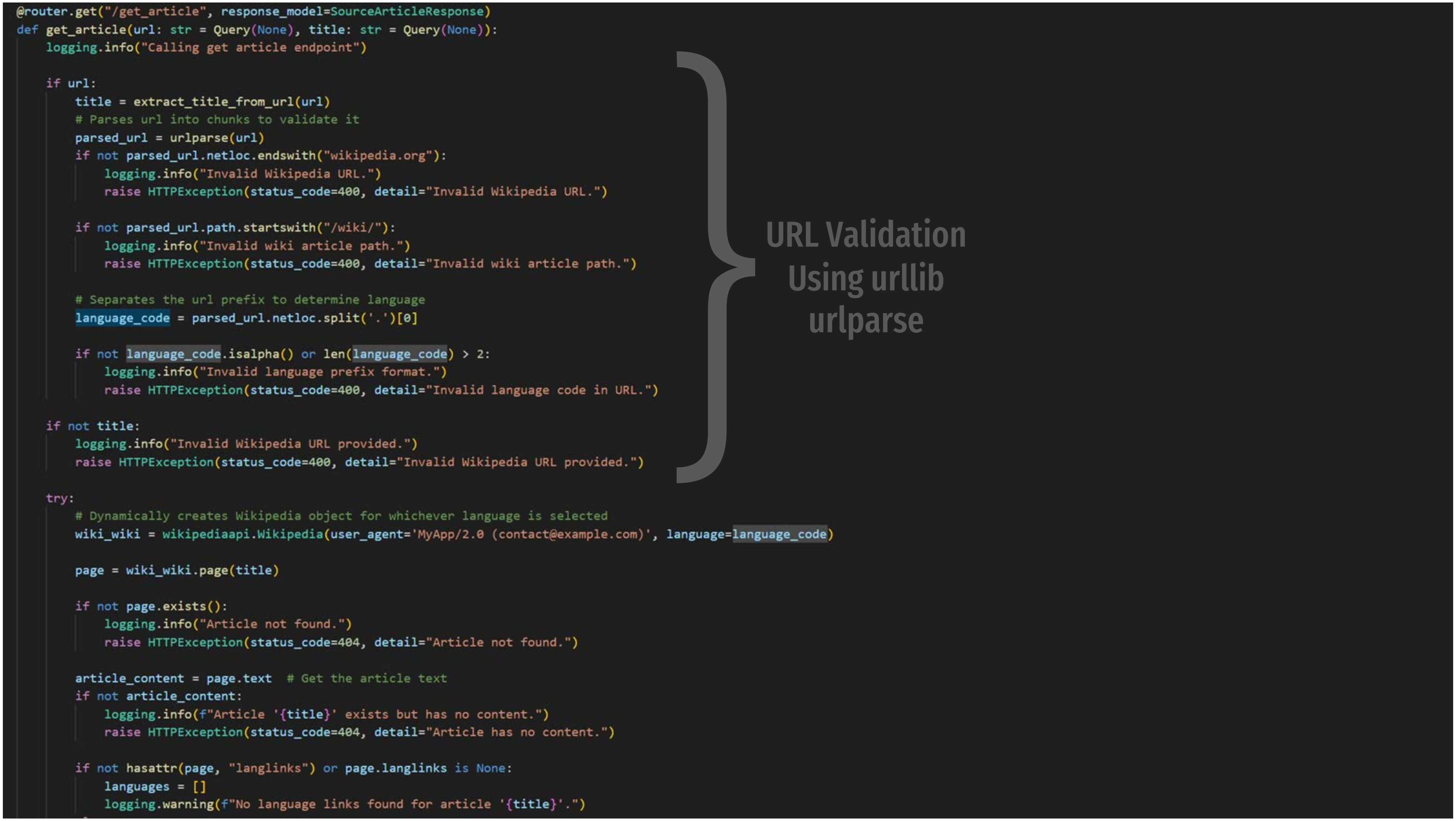
Status: 404 Not Found Time: 2.35 s Size: 2.56 KB

```
{
    "detail": "Article ThisArticleDoesNotExistOnWikipedia not found.",
    "stack_trace": "Traceback (most recent call last):\n  File \"/Users/erik/Documents/greybox/Project-Symmetry-AI/fastapi/venv/lib/python3.13/site-packages/starlette/
        _exception_handler.py\", line 42, in wrapped_app\n    await app(scope, receive, sender)\n  File \"/Users/erik/Documents/greybox/Project-Symmetry-AI/fastapi/venv/
        lib/python3.13/site-packages/starlette/routing.py\", line 73, in app\n    response = await f(request)\n               ^^^^^^^^^^^^^^^^\n  File \"/Users/erik/
        Documents/greybox/Project-Symmetry-AI/fastapi/venv/lib/python3.13/site-packages/fastapi/routing.py\", line 301, in app\n    raw_response = await
        run_endpoint_function(\n                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n    ...<3 lines>...\n    )\n    ^\n  File \"/Users/erik/Documents/greybox/
        Project-Symmetry-AI/fastapi/venv/lib/python3.13/site-packages/fastapi/routing.py\", line 214, in run_endpoint_function\n    return await run_in_threadpool
        (dependant.call, **values)\n           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n  File \"/Users/erik/Documents/greybox/Project-Symmetry-AI/fastapi/
        venv/lib/python3.13/site-packages/starlette/concurrency.py\", line 37, in run_in_threadpool\n    return await anyio.to_thread.run_sync(func)\n           
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n  File \"/Users/erik/Documents/greybox/Project-Symmetry-AI/fastapi/venv/lib/python3.13/site-packages/anyio/to_thread.py\",
        line 56, in run_sync\n    return await get_async_backend().run_sync_in_worker_thread(\n           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n        
        func, args, abandon_on_cancel=abandon_on_cancel, limiter=limiter\n        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n    )\n    ^\n  File
        \"/Users/erik/Documents/greybox/Project-Symmetry-AI/fastapi/venv/lib/python3.13/site-packages/anyio/_backends/_asyncio.py\", line 2470, in
        run_sync_in_worker_thread\n    return await future\n           ^^^^^^^^^^^^\n  File \"/Users/erik/Documents/greybox/Project-Symmetry-AI/fastapi/venv/lib/python3.
        13/site-packages/anyio/_backends/_asyncio.py\", line 967, in run\n    result = context.run(func, *args)\n  File \"/Users/erik/Documents/greybox/
        Project-Symmetry-AI/fastapi/app/api/wiki_article.py\", line 69, in get_article\n    raise HTTPException(status_code=404, detail=f\"Article {title} not found.\")
        \nfastapi.exceptions.HTTPException: 404: Article ThisArticleDoesNotExistOnWikipedia not found.\n"
}
```

User Story 5

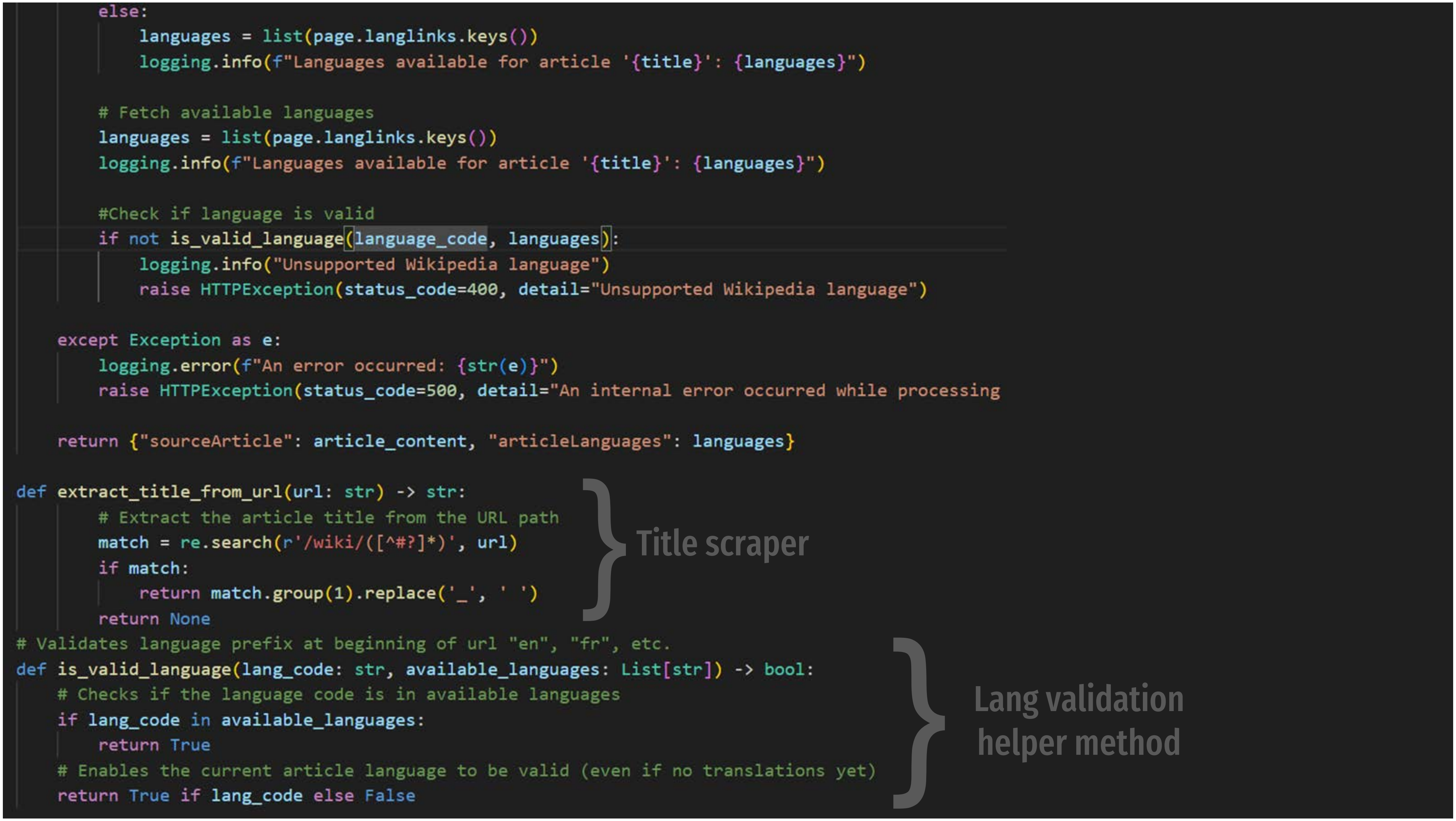
As a back-end developer, I want to parse and validate incoming URLs so that I can reduce invalid requests and decrease server load.

Size 5



```python
@router.get("/get_article", response_model=SourceArticleResponse)
def get_article(url: str = Query(None), title: str = Query(None)):
    logging.info("Calling get article endpoint")

    if url:
        title = extract_title_from_url(url)
        # Parses url into chunks to validate it
        parsed_url = urlparse(url)
        if not parsed_url.netloc.endswith("wikipedia.org"):
            logging.info("Invalid Wikipedia URL.")
            raise HTTPException(status_code=400, detail="Invalid Wikipedia URL.")

        if not parsed_url.path.startswith("/wiki/"):
            logging.info("Invalid wiki article path.")
            raise HTTPException(status_code=400, detail="Invalid wiki article path.")

        # Separates the url prefix to determine language
        language_code = parsed_url.netloc.split('.')[0]

        if not language_code.isalpha() or len(language_code) > 2:
            logging.info("Invalid language prefix format.")
            raise HTTPException(status_code=400, detail="Invalid language code in URL.")

    if not title:
        logging.info("Invalid Wikipedia URL provided.")
        raise HTTPException(status_code=400, detail="Invalid Wikipedia URL provided.")

    try:
        # Dynamically creates Wikipedia object for whichever language is selected
        wiki_wiki = wikipediaapi.Wikipedia(user_agent='MyApp/2.0 (contact@example.com)', language=language_code)

        page = wiki_wiki.page(title)

        if not page.exists():
            logging.info("Article not found.")
            raise HTTPException(status_code=404, detail="Article not found.")

        article_content = page.text  # Get the article text
        if not article_content:
            logging.info(f"Article '{title}' exists but has no content.")
            raise HTTPException(status_code=404, detail="Article has no content.")

        if not hasattr(page, "langlinks") or page.langlinks is None:
            languages = []
            logging.warning(f"No language links found for article '{title}'.")
```

URL Validation
Using urllib
urlparse



```python
        else:
            languages = list(page.langlinks.keys())
            logging.info(f"Languages available for article '{title}': {languages}")

        # Fetch available languages
        languages = list(page.langlinks.keys())
        logging.info(f"Languages available for article '{title}': {languages}")

        #Check if language is valid
        if not is_valid_language(language_code, languages):
            logging.info("Unsupported Wikipedia language")
            raise HTTPException(status_code=400, detail="Unsupported Wikipedia language")

    except Exception as e:
        logging.error(f"An error occurred: {str(e)}")
        raise HTTPException(status_code=500, detail="An internal error occurred while processing

    return {"sourceArticle": article_content, "articleLanguages": languages}

def extract_title_from_url(url: str) -> str:
        # Extract the article title from the URL path
        match = re.search(r'/wiki/([^#?]*)', url)
        if match:
            return match.group(1).replace('_', ' ')
        return None
# Validates language prefix at beginning of url "en", "fr", etc.
def is_valid_language(lang_code: str, available_languages: List[str]) -> bool:
    # Checks if the language code is in available languages
    if lang_code in available_languages:
        return True
    # Enables the current article language to be valid (even if no translations yet)
    return True if lang_code else False
```

Title scraper

Lang validation helper method

What Worked

01

Redefining

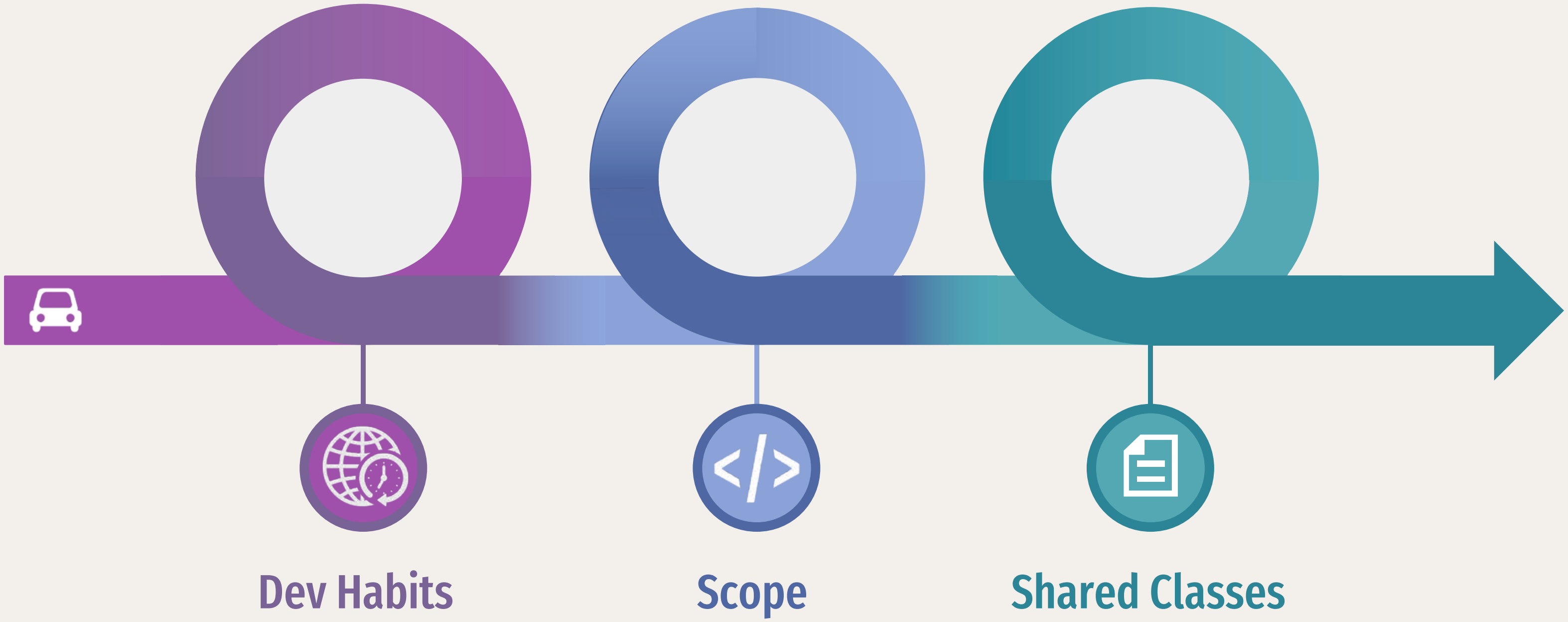
02

Paired Engineering

03

Confirmation

Roadblocks



Lessons Learned

Version Control

Merges and Branches and Conflicts, oh my!

Accountability

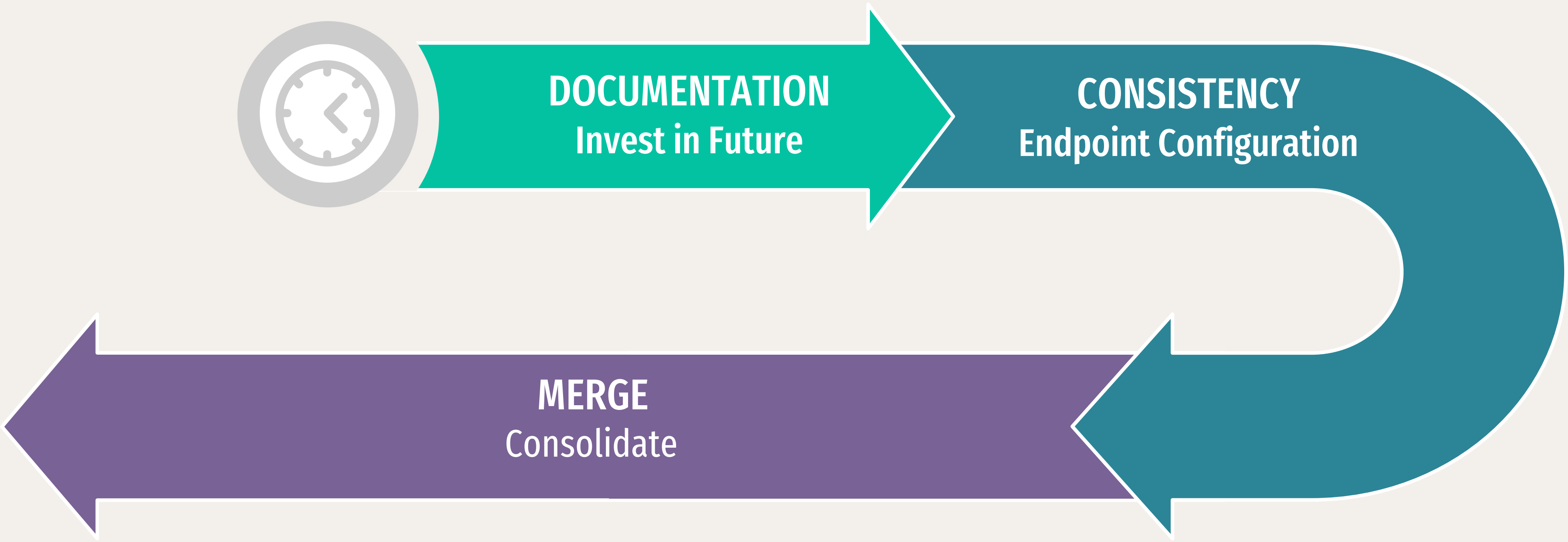
Last minute work

Compromise

Finding what works

Documentation

Finding solutions



DOCUMENTATION
Invest in Future

CONSISTENCY
Endpoint Configuration

MERGE
Consolidate

Thank you

By any chance is anyone familiar with this particular airplane pressure control component?

- anonymous