Project Symmetry

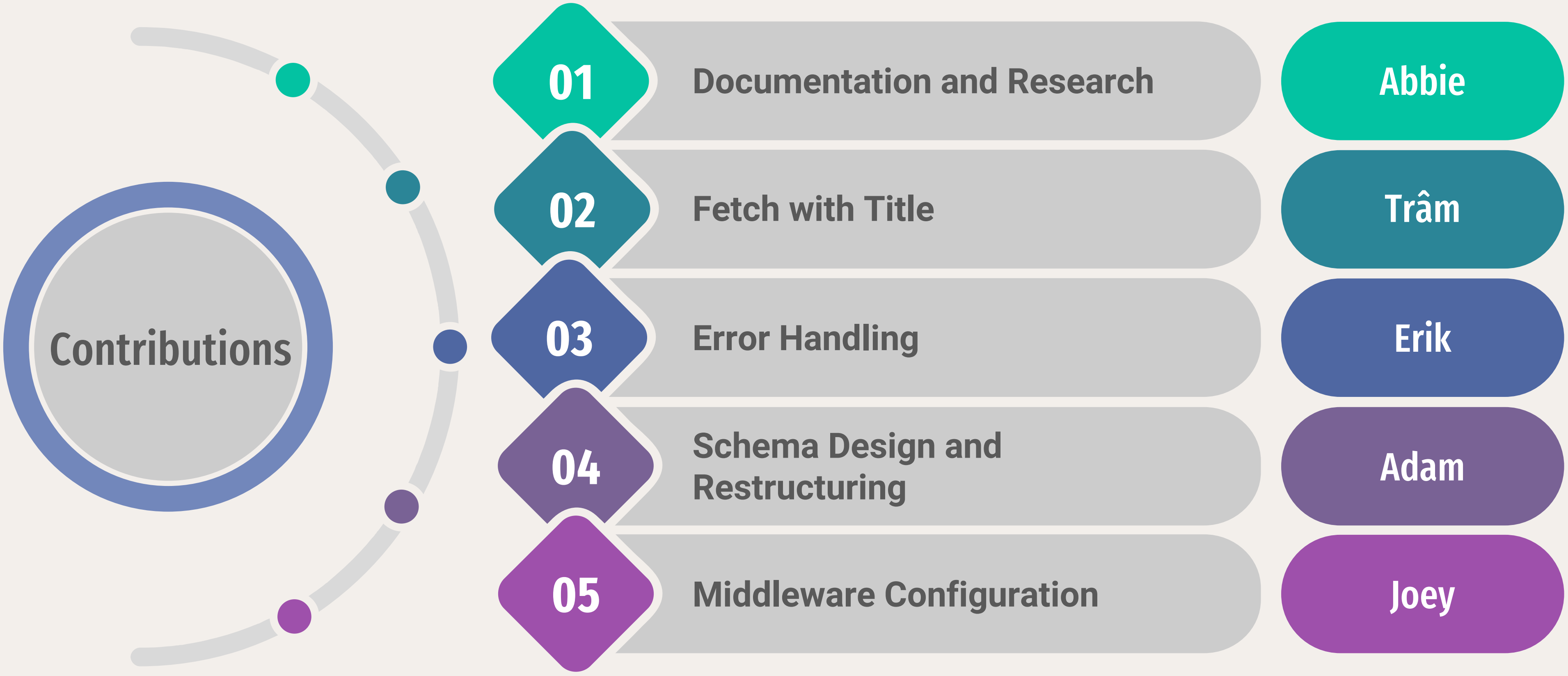
🥐Team Scrumptious🥐

API & Middleware

Accomplishments

Planned: 16/52
Completed: 21/52

1. Fetch with Title
2. Error Handling
3. Schema (Re)Design
4. Directory Restructuring
5. Security



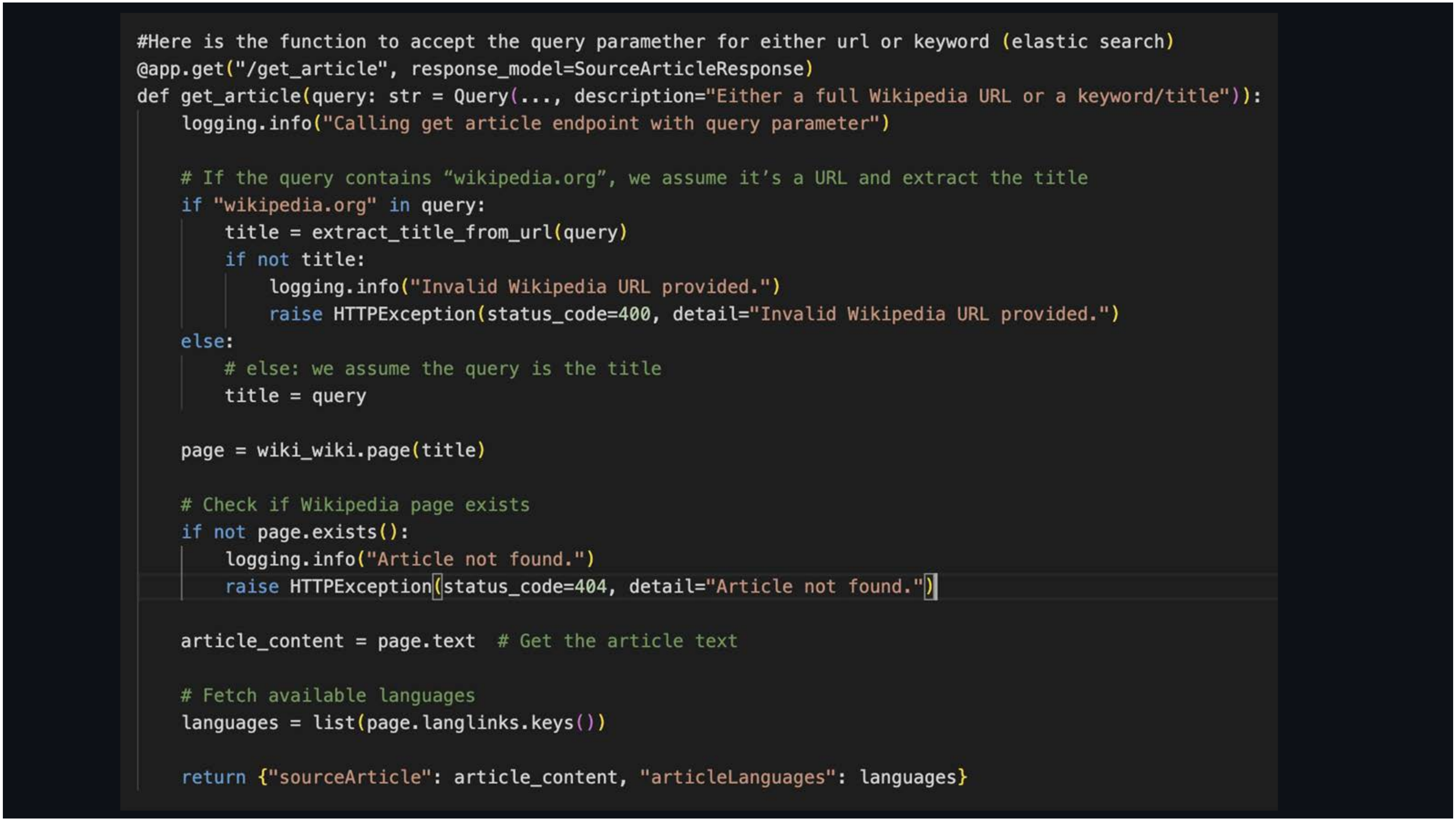
Contributions

01	Documentation and Research	Abbie
02	Fetch with Title	Trâm
03	Error Handling	Erik
04	Schema Design and Restructuring	Adam
05	Middleware Configuration	Joey

User Story 1

As a developer, I want my API to be able to fetch articles based on the URL or Title given so that users don't have to rely solely on the URL.

Size: 5

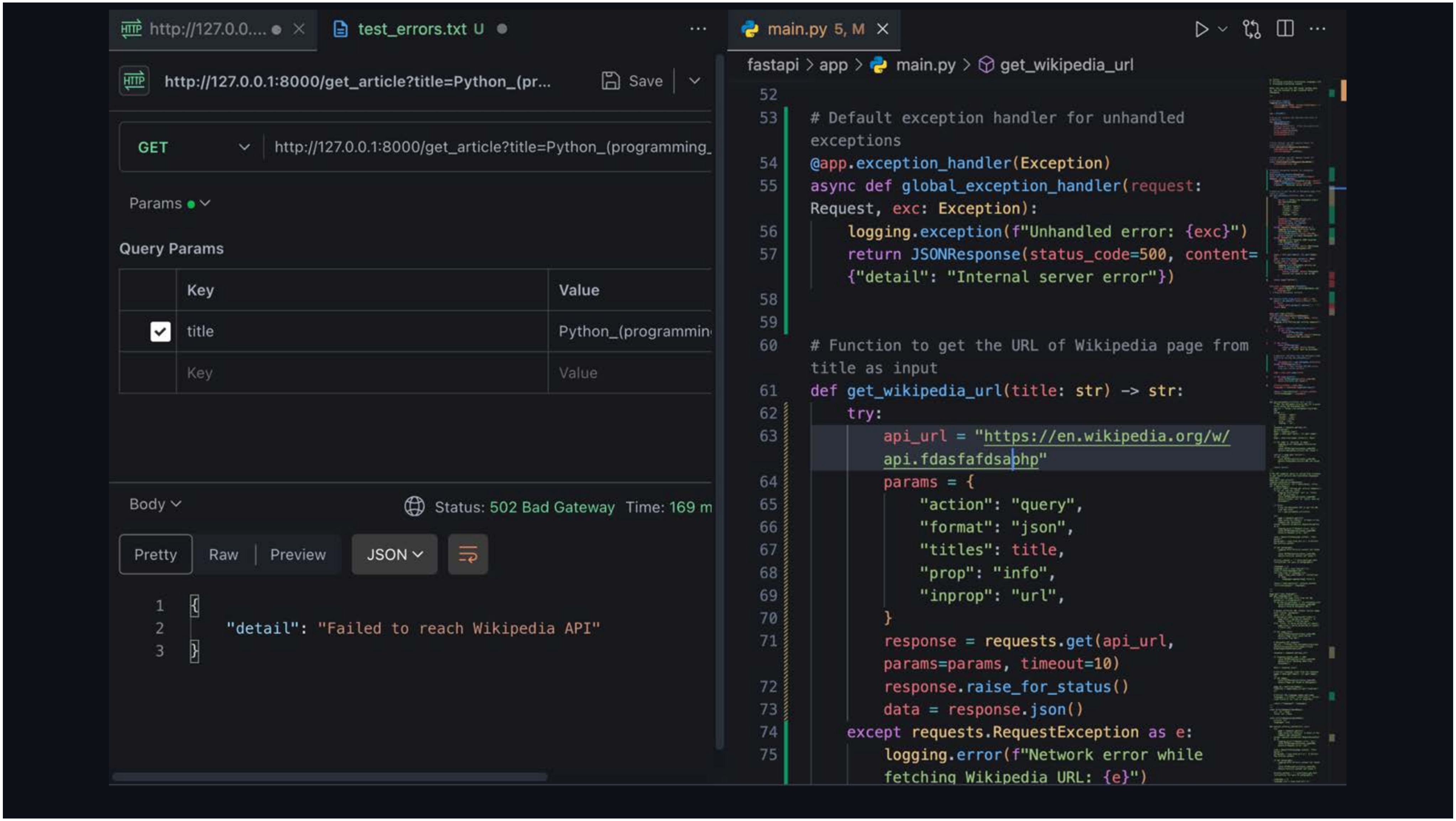


```python
#Here is the function to accept the query paramether for either url or keyword (elastic search)
@app.get("/get_article", response_model=SourceArticleResponse)
def get_article(query: str = Query(..., description="Either a full Wikipedia URL or a keyword/title")):
    logging.info("Calling get article endpoint with query parameter")

    # If the query contains “wikipedia.org”, we assume it’s a URL and extract the title
    if "wikipedia.org" in query:
        title = extract_title_from_url(query)
        if not title:
            logging.info("Invalid Wikipedia URL provided.")
            raise HTTPException(status_code=400, detail="Invalid Wikipedia URL provided.")
    else:
        # else: we assume the query is the title
        title = query

    page = wiki_wiki.page(title)

    # Check if Wikipedia page exists
    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}
```

As a back-end developer, I want error-handling mechanisms that return meaningful error responses so that consuming services can gracefully handle failures.

Size: 5

User Story 2



http://127.0.0.1:8000/get_article?title=Python_(pr...

GET http://127.0.0.1:8000/get_article?title=Python_(programming_

Query Params

	Key	Value
☑	title	Python_(programmin
	Key	Value

Status: 502 Bad Gateway Time: 169 m

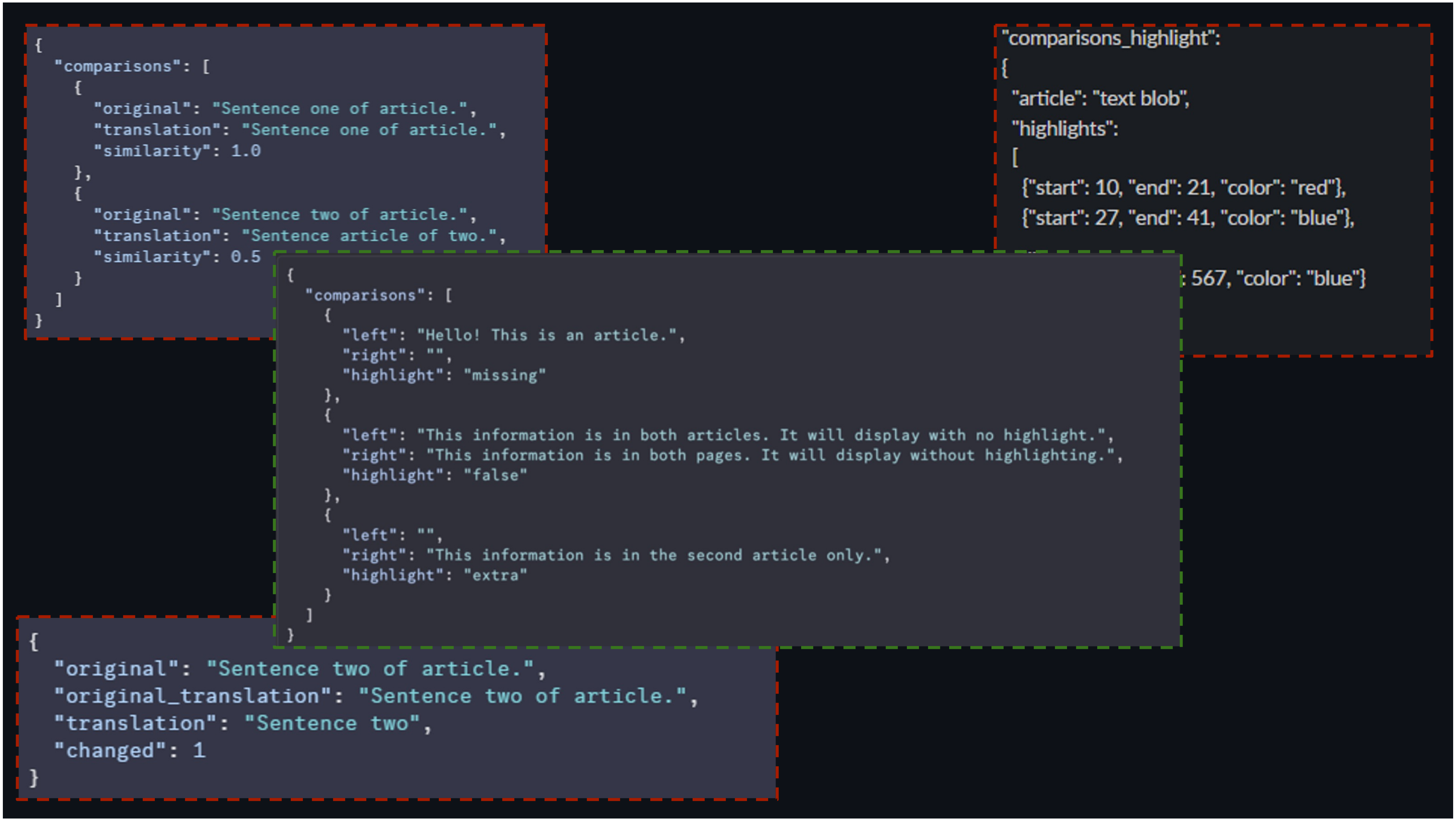
```json
{
    "detail": "Failed to reach Wikipedia API"
}
```

```python

# Default exception handler for unhandled exceptions
@app.exception_handler(Exception)
async def global_exception_handler(request: Request, exc: Exception):
    logging.exception(f"Unhandled error: {exc}")
    return JSONResponse(status_code=500, content={"detail": "Internal server error"})


# Function to get the URL of Wikipedia page from title as input
def get_wikipedia_url(title: str) -> str:
    try:
        api_url = "https://en.wikipedia.org/w/api.fdasfafdsaphp"
        params = {
            "action": "query",
            "format": "json",
            "titles": title,
            "prop": "info",
            "inprop": "url",
        }
        response = requests.get(api_url, params=params, timeout=10)
        response.raise_for_status()
        data = response.json()
    except requests.RequestException as e:
        logging.error(f"Network error while fetching Wikipedia URL: {e}")
```

User Story 3

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

Size 5



```
{
  "comparisons": [
    {
      "original": "Sentence one of article.",
      "translation": "Sentence one of article.",
      "similarity": 1.0
    },
    {
      "original": "Sentence two of article.",
      "translation": "Sentence article of two.",
      "similarity": 0.5
    }
  ]
}
```

```
"comparisons_highlight":
{
 "article": "text blob",
 "highlights":
 [
  {"start": 10, "end": 21, "color": "red"},
  {"start": 27, "end": 41, "color": "blue"},

                         : 567, "color": "blue"}
```

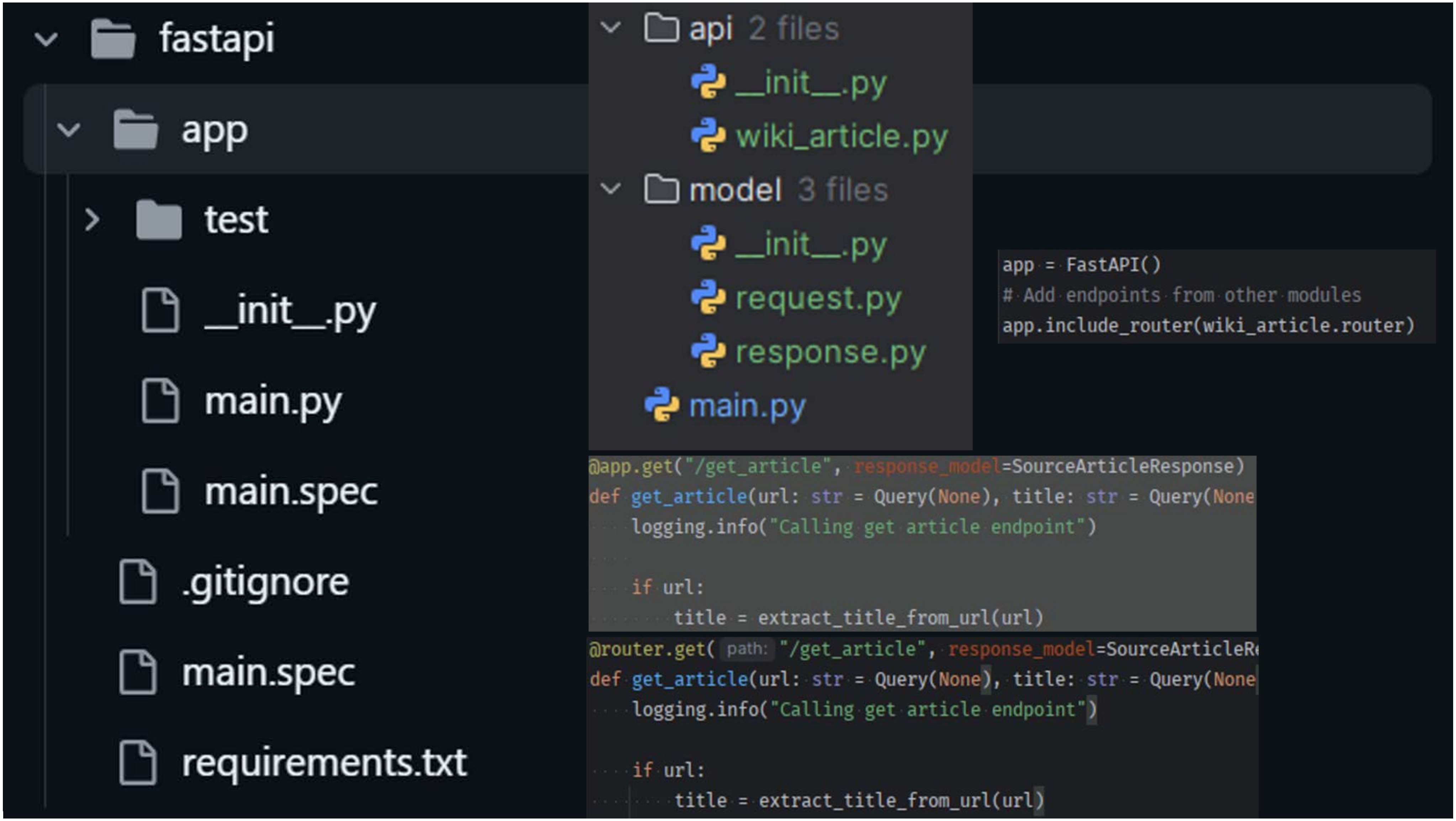
```
{
  "comparisons": [
    {
      "left": "Hello! This is an article.",
      "right": "",
      "highlight": "missing"
    },
    {
      "left": "This information is in both articles. It will display with no highlight.",
      "right": "This information is in both pages. It will display without highlighting.",
      "highlight": "false"
    },
    {
      "left": "",
      "right": "This information is in the second article only.",
      "highlight": "extra"
    }
  ]
}
```

```
{
  "original": "Sentence two of article.",
  "original_translation": "Sentence two of article.",
  "translation": "Sentence two",
  "changed": 1
}
```

User Story 4

As a developer, I want the repository to be organized into easily-understandable and maintainable packages.

Size: 3



```
fastapi
  app
    test
    __init__.py
    main.py
    main.spec
  .gitignore
  main.spec
  requirements.txt
```

```
api 2 files
  __init__.py
  wiki_article.py
model 3 files
  __init__.py
  request.py
  response.py
main.py
```

```
app = FastAPI()
# Add endpoints from other modules
app.include_router(wiki_article.router)
```

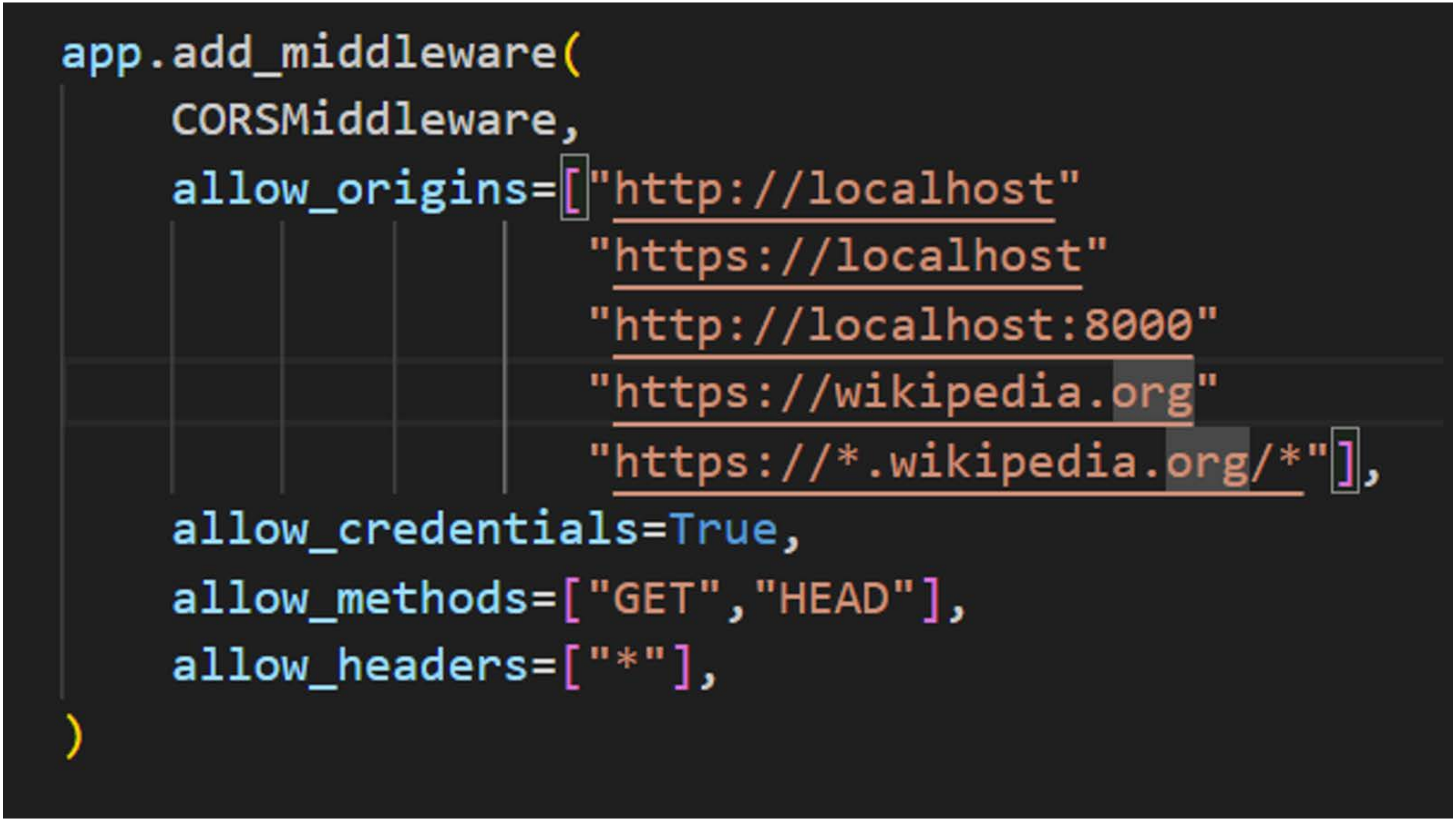
```
@app.get("/get_article", response_model=SourceArticleResponse)
def get_article(url: str = Query(None), title: str = Query(None
    logging.info("Calling get article endpoint")

    if url:
        title = extract_title_from_url(url)
```

```
@router.get( path: "/get_article", response_model=SourceArticleRe
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)
```

User Story 5

As a back-end developer, I want our configured middleware to be exceptionally secure and have selective origin parameters.

Size 3



```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost"
                   "https://localhost"
                   "http://localhost:8000"
                   "https://wikipedia.org"
                   "https://*.wikipedia.org/*"],
    allow_credentials=True,
    allow_methods=["GET","HEAD"],
    allow_headers=["*"],
)
```

What Worked

01

Compromise

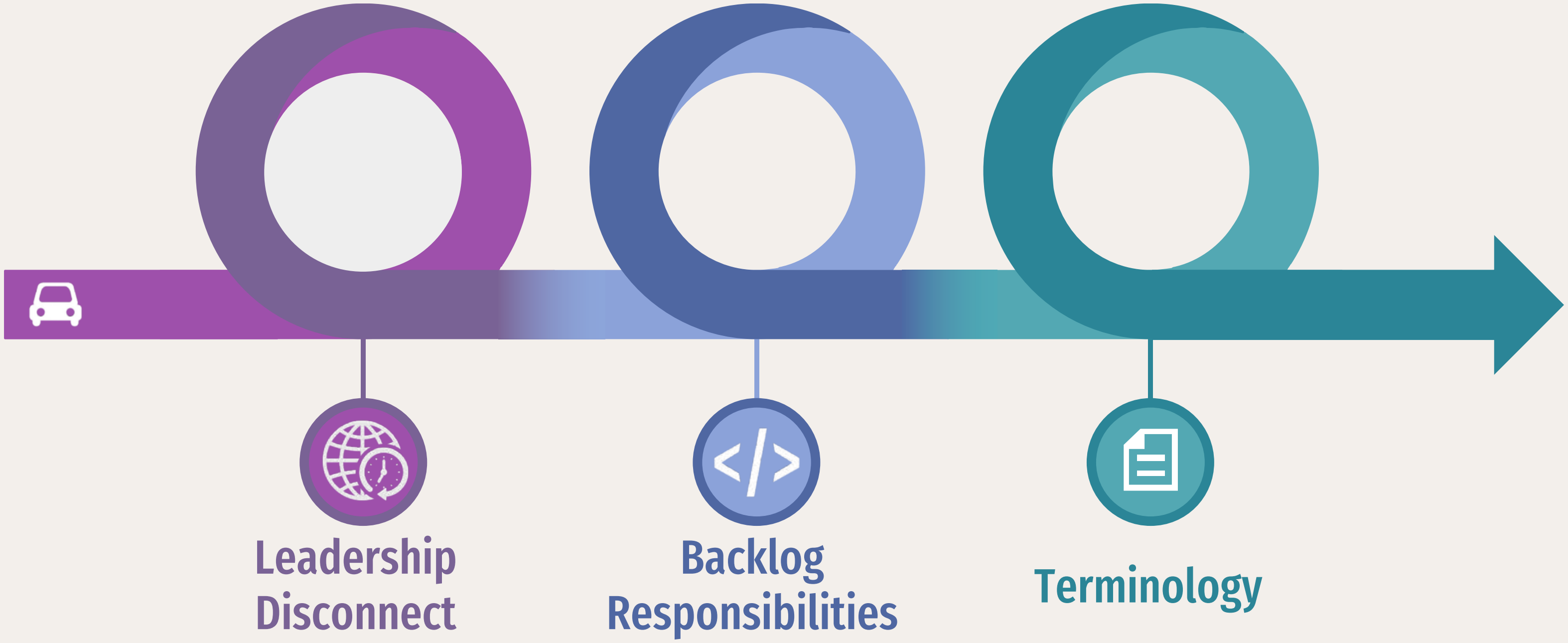
02

Paired Engineering

03

Meeting Planning

Roadblocks



Leadership Disconnect

Backlog Responsibilities

Terminology

Lessons Learned

Intergroup Development

Coding with other teams

Accountability

Last minute work

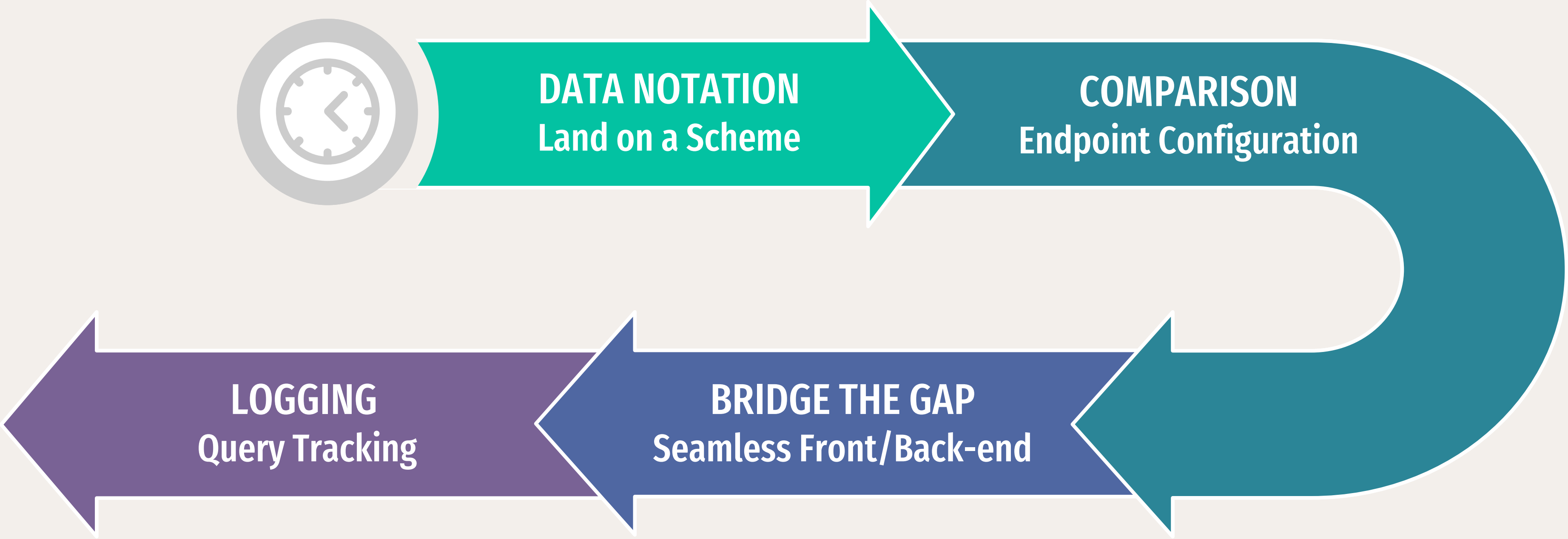
Regional Time Clarity

Meeting confirmation

Documentation

Finding solutions

Future



DATA NOTATION
Land on a Scheme

COMPARISON
Endpoint Configuration

BRIDGE THE GAP
Seamless Front/Back-end

LOGGING
Query Tracking

Thank you

With all my undying love for Wikipedia...

- anonymous