Kadiatou Sidibe

ML OPS 765-01

Homework 1 –Documentation

1. Fork the lab2_factories repo

For this step, I forked the lab2_factories repository to my personal GitHub account to establish my development environment.

In AWS cloud9, I cloned the repository and configured the remote origin to point to my personal fork using my SSH authentication to ensure secure data transfer.

Verification:
I verified the remote configuration to ensure the local environment was correctly linked to my GitHub fork using the following command:

- git remote -v

2. Create an endpoint to dynamically add new topics and store in the topics file

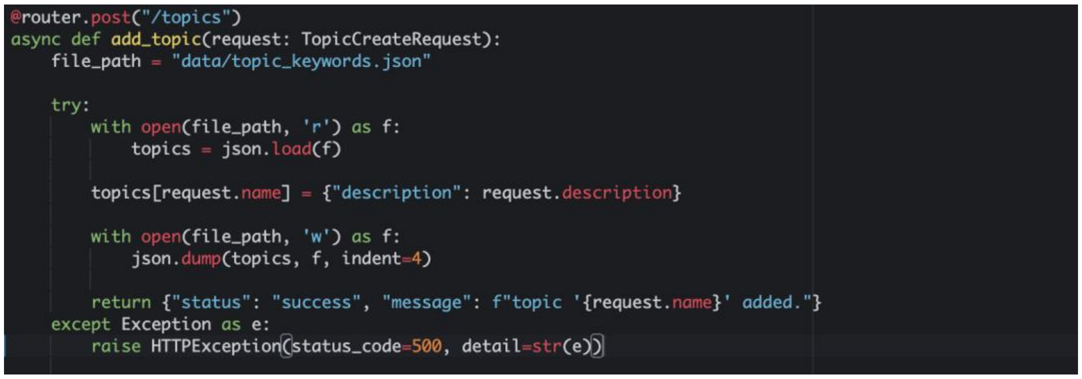
In this step, I implemented a new API endpoint, POST/topics to allow registration of new email categories and their descriptions in real-time without requiring manual code changes.

Technical phase:

- I defined a TopicCreateRequest schema in app/api/routes.py to ensure all incoming new topic includes a mandatory topic name and description.

```
class TopicCreateRequest(BaseModel):
    name: str
    description: str
```

- I updated the backend login in app/api/routes.py to handle incoming request by reading the existing definitions in data/topic_keywords.json file and appending the new topic data.


```python
@router.post("/topics")
async def add_topic(request: TopicCreateRequest):
    file_path = "data/topic_keywords.json"

    try:
        with open(file_path, 'r') as f:
            topics = json.load(f)

        topics[request.name] = {"description": request.description}

        with open(file_path, 'w') as f:
            json.dump(topics, f, indent=4)

        return {"status": "success", "message": f"topic '{request.name}' added."}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Verification:

I initialized the uvicorn server and utilized the Swagger UI to execute a "Try it out" test, successfully adding a new topic and verifying in within the JSON database. Detailed results of this verification can be found in the screenshots provided under **Step 5.**

3. **Create an endpoint to store emails. These should have an optional ground truth. This ground truth is useful for the similarity classifier**

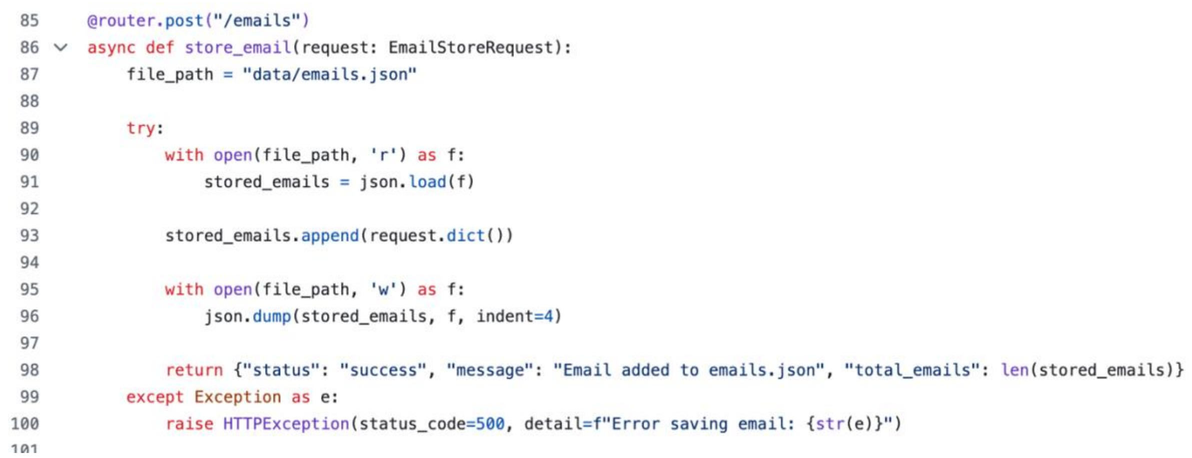
In this step, I implemented a new API endpoint, POST/emails, to store emails.

Technical phase:

- I implemented the EmailStoreRequest schema in app/api/routes.py to ensure that every stored email includes a subject and body, along with an optional ground_truth field.

```python
class EmailStoreRequest(BaseModel):
    subject: str
    body: str
    ground_truth: str = None

```

- I designed the backend logic in app/api/routes.py to append these validated entries to the data/emails.json file.



```python
@router.post("/emails")
async def store_email(request: EmailStoreRequest):
    file_path = "data/emails.json"

    try:
        with open(file_path, 'r') as f:
            stored_emails = json.load(f)

        stored_emails.append(request.dict())

        with open(file_path, 'w') as f:
            json.dump(stored_emails, f, indent=4)

        return {"status": "success", "message": "Email added to emails.json", "total_emails": len(stored_emails)}
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error saving email: {str(e)}")
```

Verification:

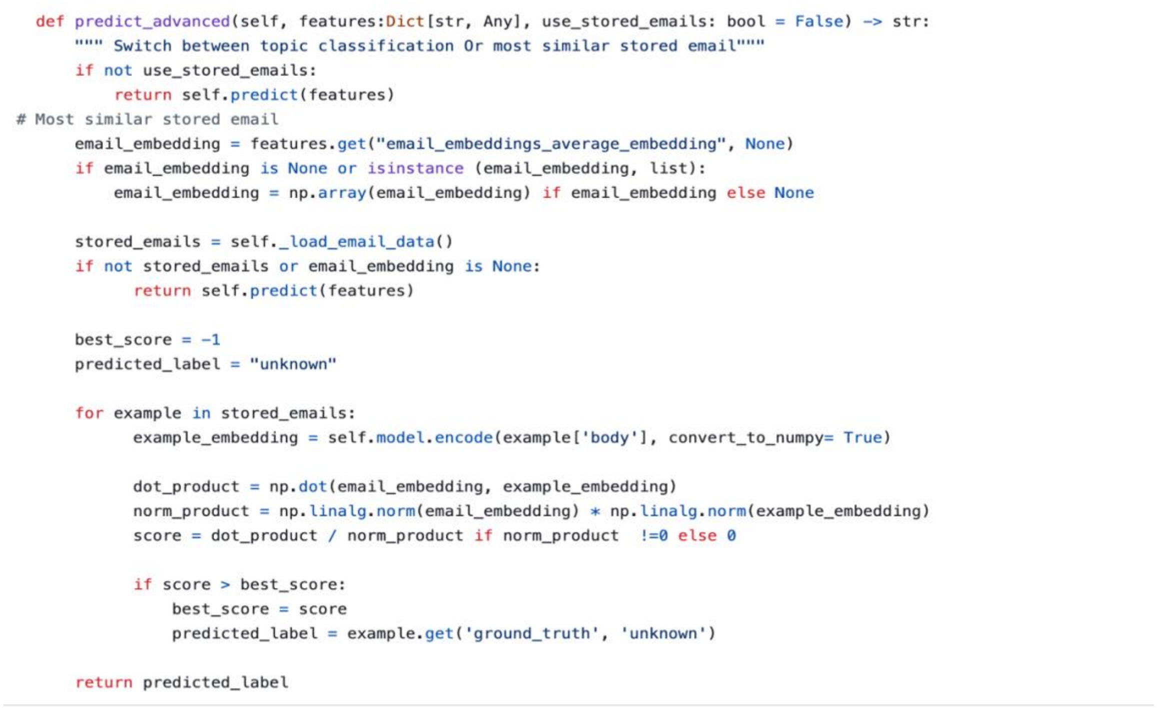
I initialized the uvicorn server and utilized the Swagger UI to execute a “Try it out” test, successfully adding a new email and verifying in the database. Detailed results of this verification can be found in the screenshots provided under **Step 7.**

4. **Update the classifier optionally use either the topic classification or to select the class of the most similar email from the stored emails**

In this step, I developed an advanced inference mode using Cosine Similarity. When the toggle is enabled, the system generates embeddings for incoming emails and compares them against the stored knowledge base in data/emails.json

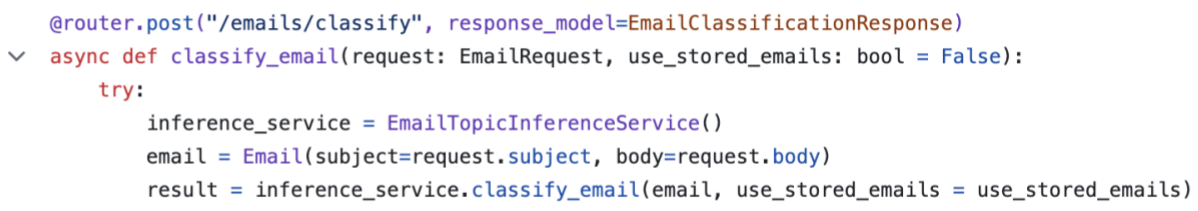
Technical phase:

- I updated the app/models/similary_model.py to include the predict advanced which utilizes **Cosine Similarity** to compare input embeddings against the stored knowledge base.



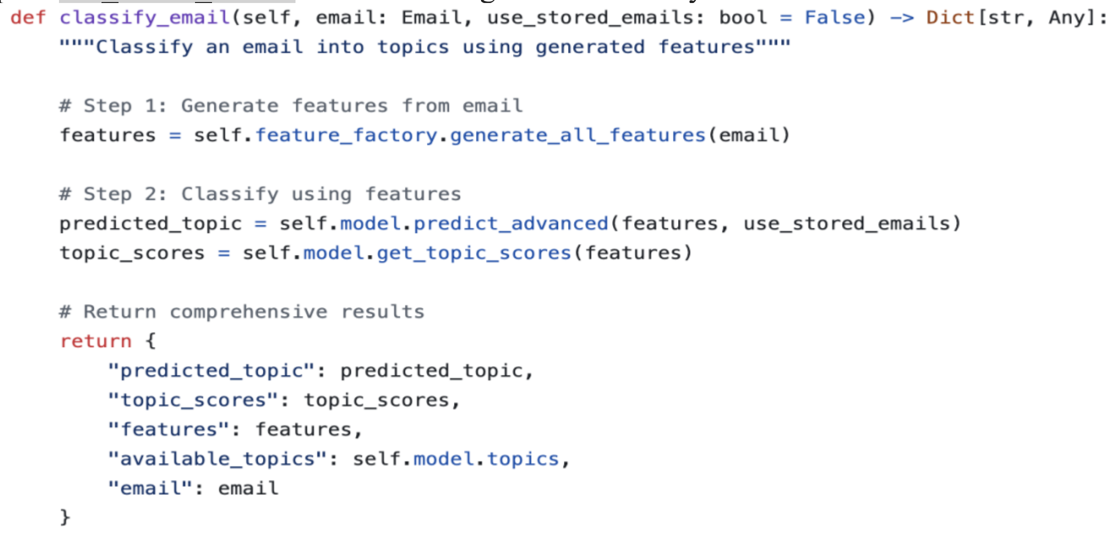
```
  def predict_advanced(self, features:Dict[str, Any], use_stored_emails: bool = False) -> str:
      """ Switch between topic classification Or most similar stored email"""
      if not use_stored_emails:
          return self.predict(features)
# Most similar stored email
      email_embedding = features.get("email_embeddings_average_embedding", None)
      if email_embedding is None or isinstance (email_embedding, list):
          email_embedding = np.array(email_embedding) if email_embedding else None

      stored_emails = self._load_email_data()
      if not stored_emails or email_embedding is None:
            return self.predict(features)

      best_score = -1
      predicted_label = "unknown"

      for example in stored_emails:
            example_embedding = self.model.encode(example['body'], convert_to_numpy= True)

            dot_product = np.dot(email_embedding, example_embedding)
            norm_product = np.linalg.norm(email_embedding) * np.linalg.norm(example_embedding)
            score = dot_product / norm_product if norm_product  !=0 else 0

            if score > best_score:
                best_score = score
                predicted_label = example.get('ground_truth', 'unknown')

      return predicted_label
```

- I updated the POST/ emails/classify route to store a bool of use_stored_emails.



```
@router.post("/emails/classify", response_model=EmailClassificationResponse)
async def classify_email(request: EmailRequest, use_stored_emails: bool = False):
    try:
        inference_service = EmailTopicInferenceService()
        email = Email(subject=request.subject, body=request.body)
        result = inference_service.classify_email(email, use_stored_emails = use_stored_emails)
```

- I modified app/services/email_topic_inference.py, by updating the classify_email method to pass use_stored_emails boolean through the service layer.

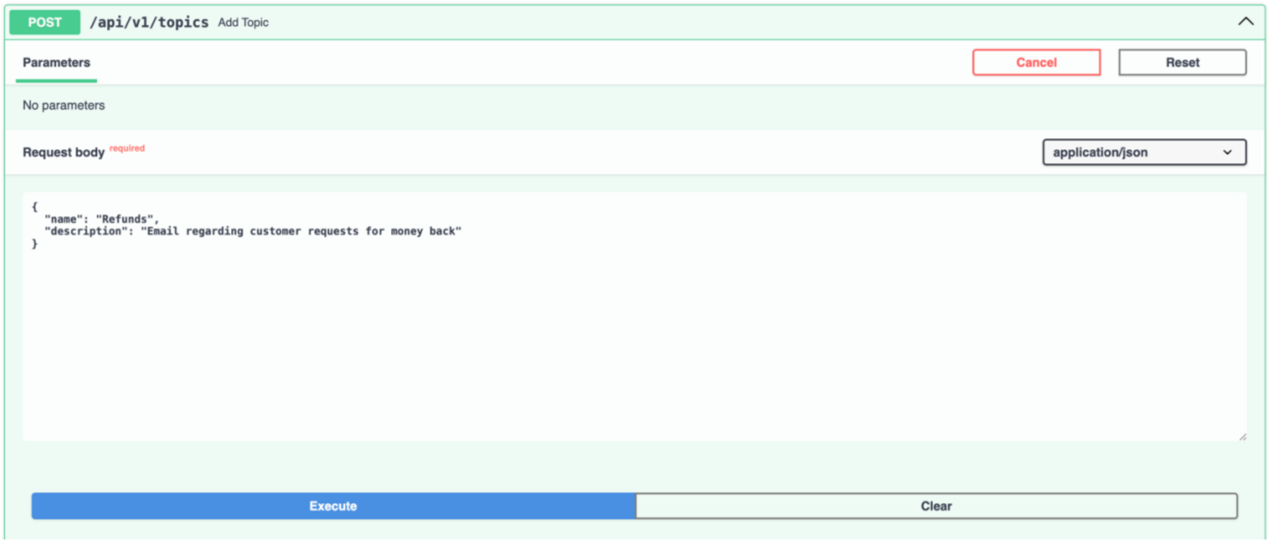


```
def classify_email(self, email: Email, use_stored_emails: bool = False) -> Dict[str, Any]:
    """Classify an email into topics using generated features"""

    # Step 1: Generate features from email
    features = self.feature_factory.generate_all_features(email)

    # Step 2: Classify using features
    predicted_topic = self.model.predict_advanced(features, use_stored_emails)
    topic_scores = self.model.get_topic_scores(features)

    # Return comprehensive results
    return {
        "predicted_topic": predicted_topic,
        "topic_scores": topic_scores,
        "features": features,
        "available_topics": self.model.topics,
        "email": email
    }
```

5. Demonstrate this creating new topics

I conducted the following three steps to confirm the API function work correctly.

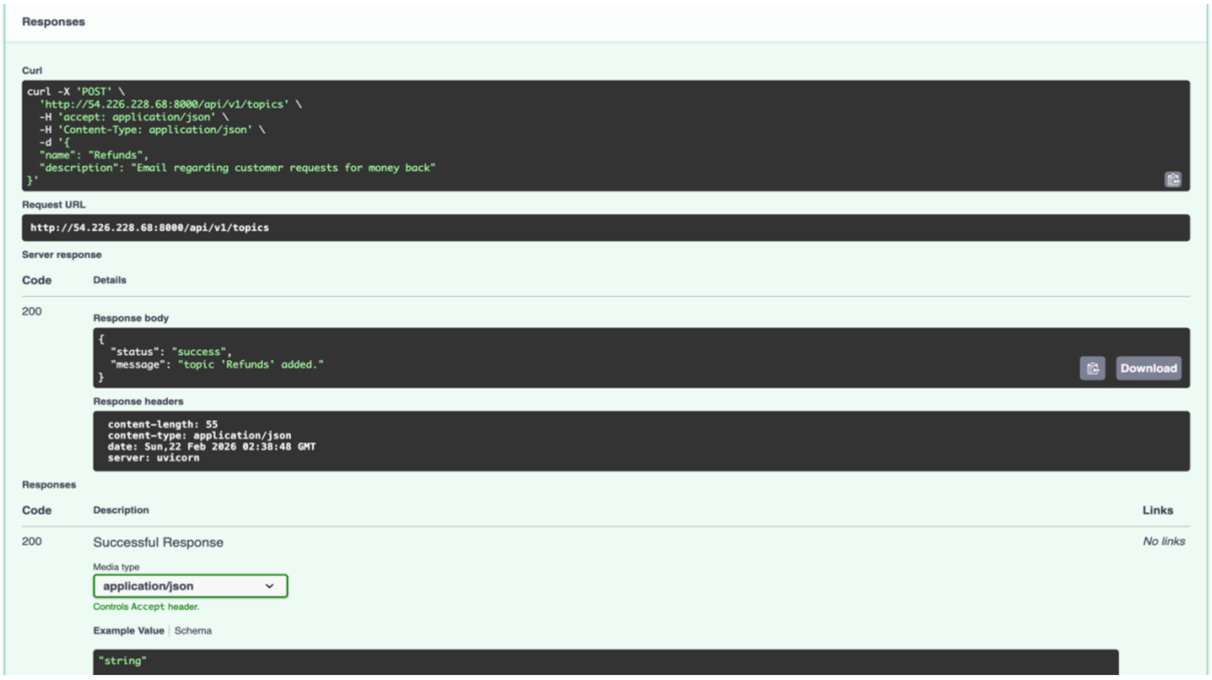
The first screenshot below shows a POST request being sent to the /api/v1/topics endpoint. The JSON request name and description.



```
{
  "name": "Refunds",
  "description": "Email regarding customer requests for money back"
}
```

After the execution, the second screenshot verifies that the server processed the request successfully.

- The status code is 200 (Success).
- The response message is “Topic ‘Refunds’ added.”



```
curl -X 'POST' \
  'http://54.226.228.68:8000/api/v1/topics' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Refunds",
  "description": "Email regarding customer requests for money back"
}'
```

```
http://54.226.228.68:8000/api/v1/topics
```

200

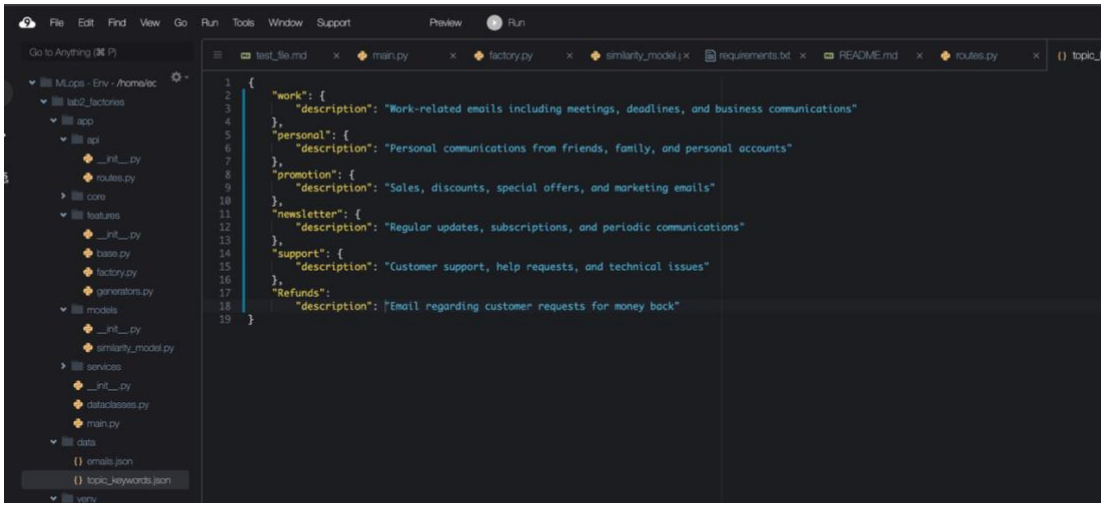
```
{
  "status": "success",
  "message": "topic 'Refunds' added."
}
```

```
 content-length: 55
 content-type: application/json
 date: Sun,22 Feb 2026 02:38:48 GMT
 server: uvicorn
```

200 Successful Response No links

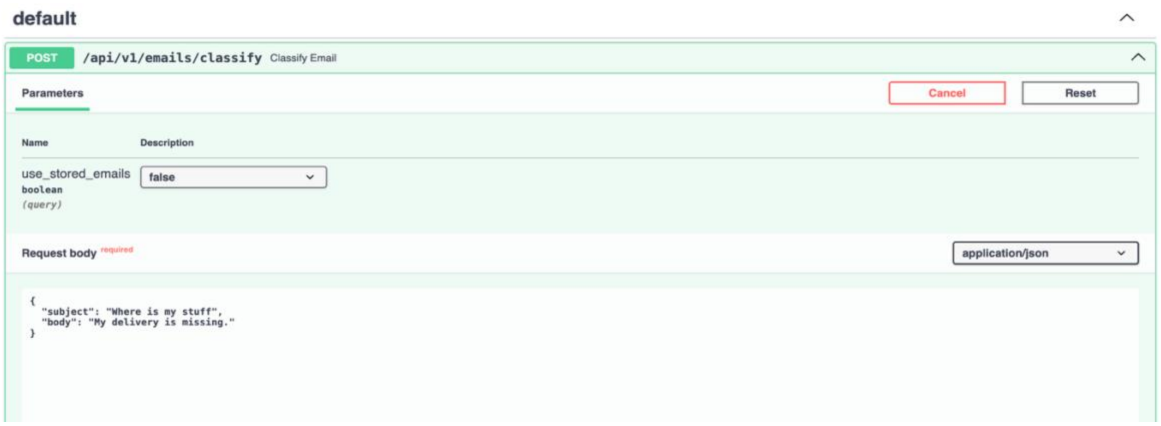
```
"string"
```

The third screenshot below shows the content of the data/topic_keywords.json file. It confirms that the system successfully stored the refunds topic and description.



```
{
    "work": {
        "description": "Work-related emails including meetings, deadlines, and business communications"
    },
    "personal": {
        "description": "Personal communications from friends, family, and personal accounts"
    },
    "promotion": {
        "description": "Sales, discounts, special offers, and marketing emails"
    },
    "newsletter": {
        "description": "Regular updates, subscriptions, and periodic communications"
    },
    "support": {
        "description": "Customer support, help requests, and technical issues"
    },
    "Refunds":
        "description": "Email regarding customer requests for money back"
}
```

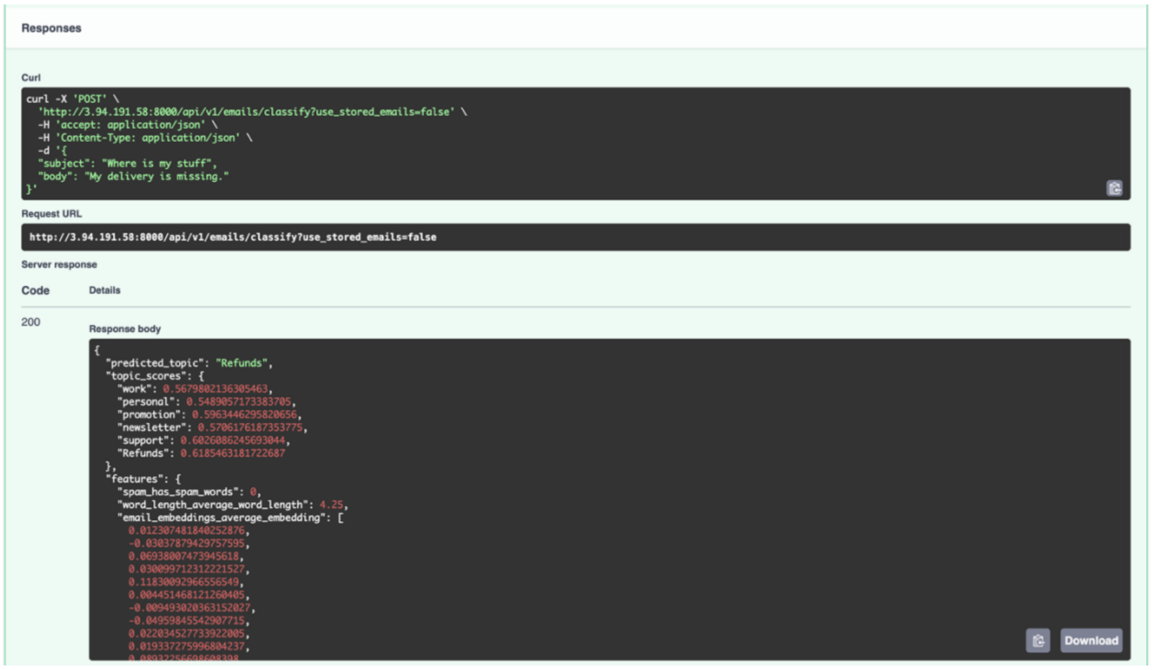
6. Demonstrating this performing inference on the new topics



use_stored_emails
boolean
(query)
false

application/json

```
{
  "subject": "Where is my stuff",
  "body": "My delivery is missing."
}
```


Responses

Curl
```
curl -X 'POST' \
  'http://3.94.191.58:8000/api/v1/emails/classify?use_stored_emails=false' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "subject": "Where is my stuff",
  "body": "My delivery is missing."
}'
```
Request URL
```
http://3.94.191.58:8000/api/v1/emails/classify?use_stored_emails=false
```
Server response

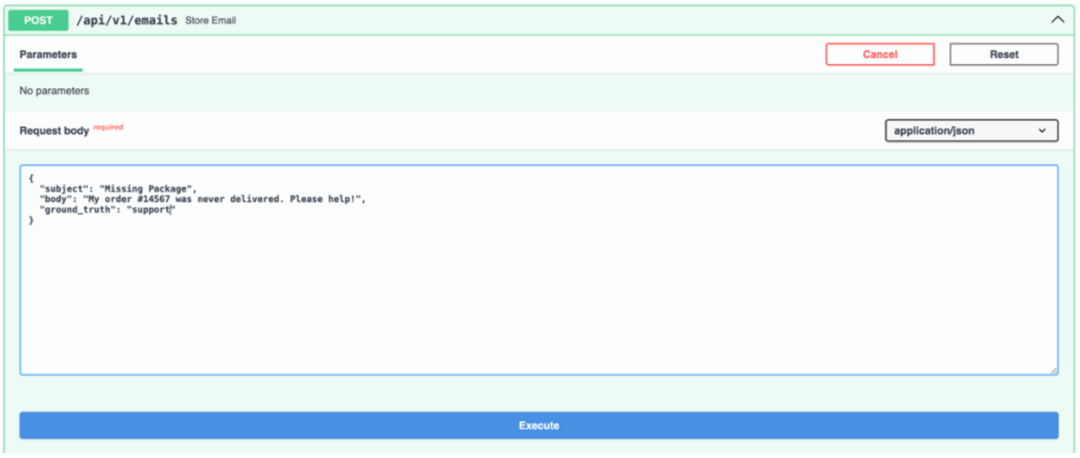
Code: 200

Response body
```
{
  "predicted_topic": "Refunds",
  "topic_scores": {
    "work": 0.5679802136305463,
    "personal": 0.5489057173383705,
    "promotion": 0.5963446295820656,
    "newsletter": 0.5706176187353775,
    "support": 0.6026086245693044,
    "Refunds": 0.6185463181722687
  },
  "features": {
    "spam_has_spam_words": 0,
    "word_length_average_word_length": 4.25,
    "email_embeddings_average_embedding": [
      0.012307481840252876,
      -0.03037879429757595,
      0.06938007473945618,
      0.03009971231222152?,
      0.11830092966556549,
      0.004451468121260405,
      -0.009493020363152027,
      -0.04959845542907715,
      0.022034527733922005,
      0.019337275996804237,
```

7. Demonstrate this adding new emails

To confirm the API is functioning correctly and data is being persisted, the following three parts processed was performed.

The screenshot below shows a POST request being sent to the /api/v1//emails endpoint. The JSON request body includes the subject, body, and the ground truth.

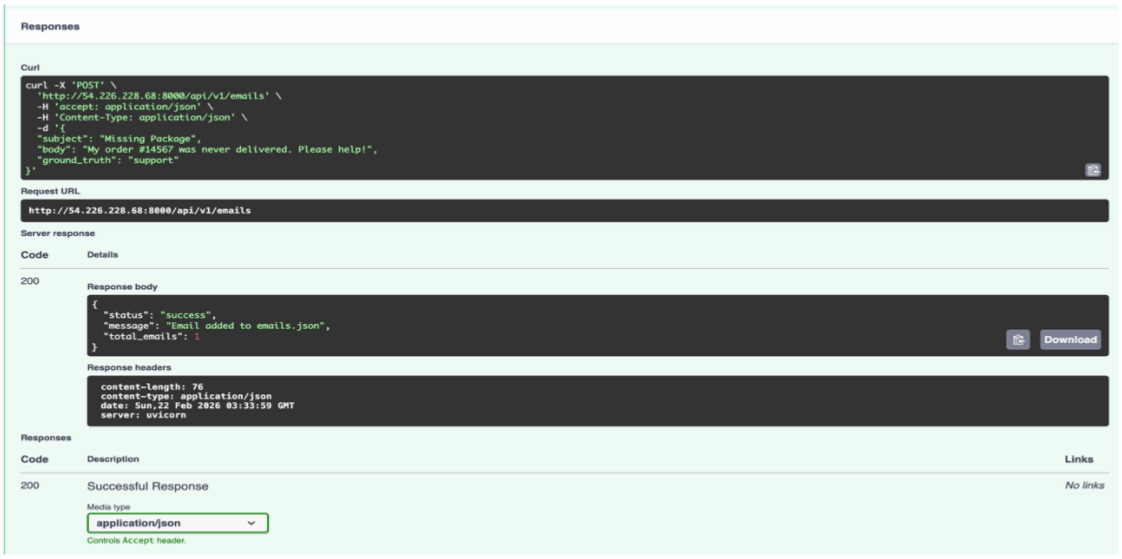


```
{
  "subject": "Missing Package",
  "body": "My order #14567 was never delivered. Please help!",
  "ground_truth": "support"
}
```

After the tryout is executed, the second screenshot verifies that the server processed the request successfully.

- The status code is 200 (Success).
- The response message is “Email added to emails.json”

- It shows the total_emails count updated to 1, confirming the backend logic is working.



Responses

Curl

```
curl -X 'POST' \
  'http://54.226.228.68:8000/api/v1/emails' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "subject": "Missing Package",
  "body": "My order #14567 was never delivered. Please help!",
  "ground_truth": "support"
}'
```

Request URL

```
http://54.226.228.68:8000/api/v1/emails
```

Server response

Code 200 — Response body

```
{
  "status": "success",
  "message": "Email added to emails.json",
  "total_emails": 1
}
```

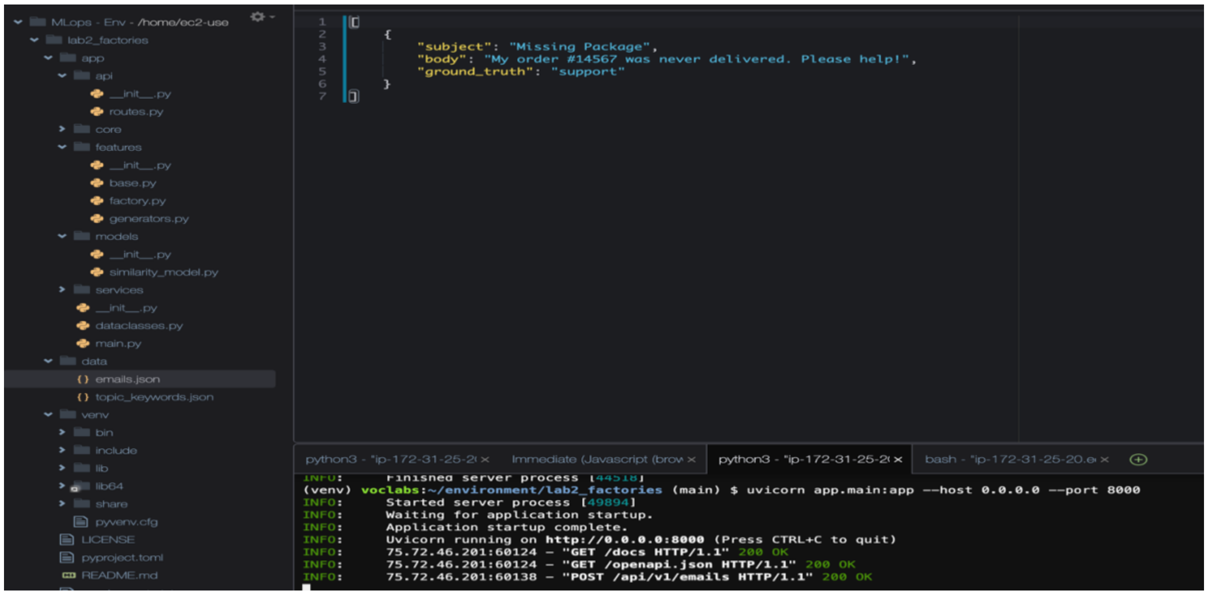
Response headers

```
 content-length: 76
 content-type: application/json
 date: Sun,22 Feb 2026 03:33:59 GMT
 server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls Accept header.	No links

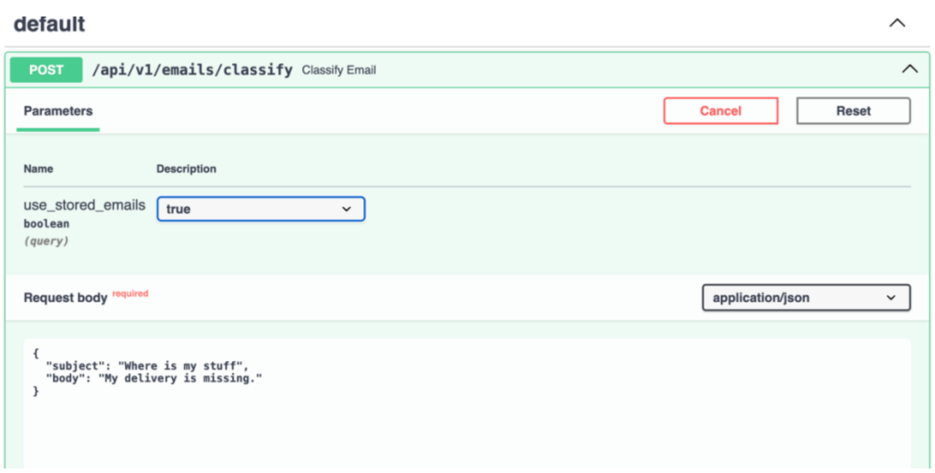
The third screenshot below shows the content of the data/emails.json file. It confirms that the system successfully stored the "Missing Package" email.



```
[
  {
    "subject": "Missing Package",
    "body": "My order #14567 was never delivered. Please help!",
    "ground_truth": "support"
  }
]
```

```
INFO:     Finished server process [44518]
(venv) voclabs:~/environment/lab2_factories (main) $ uvicorn app.main:app --host 0.0.0.0 --port 8000
INFO:     Started server process [49894]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     75.72.46.201:60124 - "GET /docs HTTP/1.1" 200 OK
INFO:     75.72.46.201:60124 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:     75.72.46.201:60138 - "POST /api/v1/emails HTTP/1.1" 200 OK
```

8. Demonstrate this performing inference from the email data



default

POST /api/v1/emails/classify Classify Email

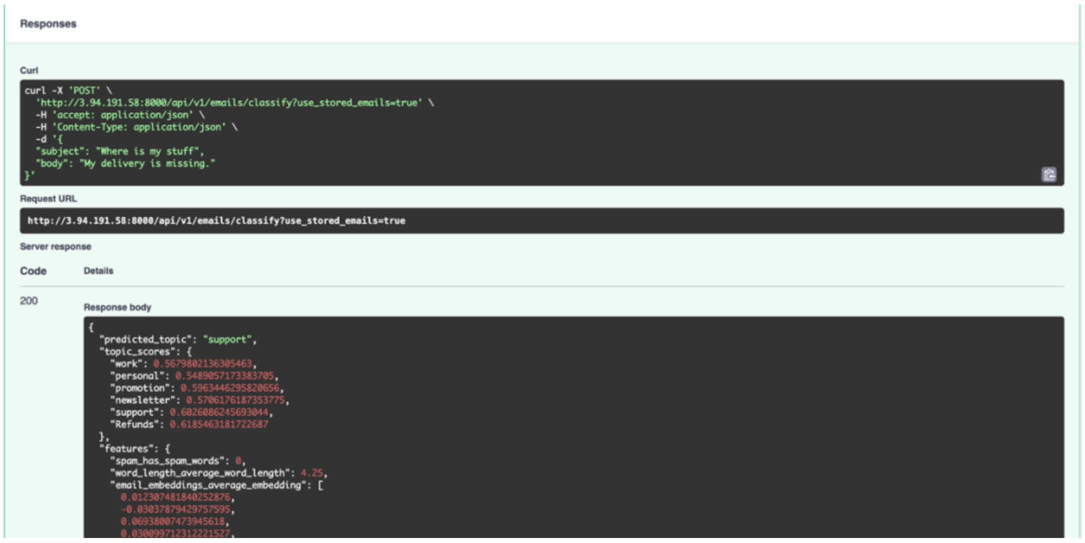
Parameters

Cancel Reset

Name	Description
use_stored_emails boolean *(query)*	true

Request body required — application/json

```
{
  "subject": "Where is my stuff",
  "body": "My delivery is missing."
}
```



Responses

Curl

```
curl -X 'POST' \
  'http://3.94.191.58:8000/api/v1/emails/classify?use_stored_emails=true' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "subject": "Where is my stuff",
  "body": "My delivery is missing."
}'
```

Request URL

```
http://3.94.191.58:8000/api/v1/emails/classify?use_stored_emails=true
```

Server response

Code	Details
200	Response body

```
{
  "predicted_topic": "support",
  "topic_scores": {
    "work": 0.5679802136305463,
    "personal": 0.5489057173383705,
    "promotion": 0.5963446295820656,
    "newsletter": 0.5706176187353775,
    "support": 0.6026086245693044,
    "Refunds": 0.6185463181722687
  },
  "features": {
    "spam_has_spam_words": 0,
    "word_length_average_word_length": 4.25,
    "email_embeddings_average_embedding": [
      0.012307481840252876,
      -0.03037879429757595,
      0.06938007473945618,
      0.030099712312221527,
```