Code Analysis Report

Analyzed Code:

```
import collections.abc
import json
from datetime import datetime, timezone
from os import path
from typing import Set, Tuple, Union
from uuid import UUID

import flask
import werkzeug
from db import db
from flask import Blueprint, abort, jsonify, request
from journalist_app import utils
from journalist_app.sessions import session
from models import (
InvalidUsernameException,
Journalist,
LoginThrottledException,
Reply,
SeenReply,
Source,
Submission,
WrongPasswordException,
)
from sqlalchemy import Column
from sqlalchemy.exc import IntegrityError
from store import NotEncrypted, Storage
from two_factor import OtpSecretInvalid, OtpTokenInvalid
from werkzeug.exceptions import default_exceptions

def get_or_404(model: db.Model, object_id: str, column: Column) -> db.Model:
result = model.query.filter(column == object_id).one_or_none()
if result is None:
abort(404)
return result
```

```
def make_blueprint() -> Blueprint:
api = Blueprint("api", __name__)

@api.route("/")
def get_endpoints() -> Tuple[flask.Response, int]:
endpoints = {
"sources_url": "/api/v1/sources",
"current_user_url": "/api/v1/user",
"all_users_url": "/api/v1/users",
"submissions_url": "/api/v1/submissions",
"replies_url": "/api/v1/replies",
"seen_url": "/api/v1/seen",
"auth_token_url": "/api/v1/token",
}
return jsonify(endpoints), 200

# Before every post, we validate the payload before processing the request
@api.before_request
def validate_data() -> None:
if request.method == "POST":
# flag, star, and logout can have empty payloads
if not request.data:
dataless_endpoints = [
"add_star",
"remove_star",
"flag",
"logout",
]
for endpoint in dataless_endpoints:
if request.endpoint == "api." + endpoint:
return
abort(400, "malformed request")
# other requests must have valid JSON payload
else:
try:
json.loads(request.data.decode("utf-8"))
except (ValueError):
abort(400, "malformed request")
```

```
@api.route("/token", methods=["POST"])
def get_token() -> Tuple[flask.Response, int]:
creds = json.loads(request.data.decode("utf-8"))

username = creds.get("username", None)
passphrase = creds.get("passphrase", None)
one_time_code = creds.get("one_time_code", None)

if username is None:
abort(400, "username field is missing")
if passphrase is None:
abort(400, "passphrase field is missing")
if one_time_code is None:
abort(400, "one_time_code field is missing")

try:
journalist = Journalist.login(username, passphrase, one_time_code)

response = jsonify(
{
"token": session.get_token(),
"expiration": session.get_lifetime(),
"journalist_uuid": journalist.uuid,
"journalist_first_name": journalist.first_name,
"journalist_last_name": journalist.last_name,
}
)

# Update access metadata
journalist.last_access = datetime.now(timezone.utc)
db.session.add(journalist)
db.session.commit()

session["uid"] = journalist.id

return response, 200
except (
LoginThrottledException,
InvalidUsernameException,
OtpSecretInvalid,
OtpTokenInvalid,
WrongPasswordException,
```

```
):
return abort(403, "Token authentication failed.")

@api.route("/sources", methods=["GET"])
def get_all_sources() -> Tuple[flask.Response, int]:
sources = Source.query.filter_by(pending=False, deleted_at=None).all()
return jsonify({"sources": [source.to_json() for source in sources]}), 200

@api.route("/sources/", methods=["GET", "DELETE"])
def single_source(source_uuid: str) -> Tuple[flask.Response, int]:
if request.method == "GET":
source = get_or_404(Source, source_uuid, column=Source.uuid)
return jsonify(source.to_json()), 200
elif request.method == "DELETE":
source = get_or_404(Source, source_uuid, column=Source.uuid)
utils.delete_collection(source.filesystem_id)
return jsonify({"message": "Source and submissions deleted"}), 200
else:
abort(405)

@api.route("/sources//add_star", methods=["POST"])
def add_star(source_uuid: str) -> Tuple[flask.Response, int]:
source = get_or_404(Source, source_uuid, column=Source.uuid)
utils.make_star_true(source.filesystem_id)
db.session.commit()
return jsonify({"message": "Star added"}), 201

@api.route("/sources//remove_star", methods=["DELETE"])
def remove_star(source_uuid: str) -> Tuple[flask.Response, int]:
source = get_or_404(Source, source_uuid, column=Source.uuid)
utils.make_star_false(source.filesystem_id)
db.session.commit()
return jsonify({"message": "Star removed"}), 200

@api.route("/sources//flag", methods=["POST"])
def flag(source_uuid: str) -> Tuple[flask.Response, int]:
return (
jsonify({"message": "Sources no longer need to be flagged for reply"}),
200,
)
```

```
@api.route("/sources//conversation", methods=["DELETE"])
def source_conversation(source_uuid: str) -> Tuple[flask.Response, int]:
if request.method == "DELETE":
source = get_or_404(Source, source_uuid, column=Source.uuid)
utils.delete_source_files(source.filesystem_id)
return jsonify({"message": "Source data deleted"}), 200
else:
abort(405)

@api.route("/sources//submissions", methods=["GET"])
def all_source_submissions(source_uuid: str) -> Tuple[flask.Response, int]:
source = get_or_404(Source, source_uuid, column=Source.uuid)
return (
jsonify({"submissions": [submission.to_json() for submission in
source.submissions]}),
200,
)

@api.route("/sources//submissions//download", methods=["GET"])
def download_submission(source_uuid: str, submission_uuid: str) ->
flask.Response:
get_or_404(Source, source_uuid, column=Source.uuid)
submission = get_or_404(Submission, submission_uuid, column=Submission.uuid)
return utils.serve_file_with_etag(submission)

@api.route("/sources//replies//download", methods=["GET"])
def download_reply(source_uuid: str, reply_uuid: str) -> flask.Response:
get_or_404(Source, source_uuid, column=Source.uuid)
reply = get_or_404(Reply, reply_uuid, column=Reply.uuid)

return utils.serve_file_with_etag(reply)

@api.route(
"/sources//submissions/",
methods=["GET", "DELETE"],
)
def single_submission(source_uuid: str, submission_uuid: str) ->
Tuple[flask.Response, int]:
if request.method == "GET":
get_or_404(Source, source_uuid, column=Source.uuid)
submission = get_or_404(Submission, submission_uuid, column=Submission.uuid)
return jsonify(submission.to_json()), 200
```

```
elif request.method == "DELETE":
get_or_404(Source, source_uuid, column=Source.uuid)
submission = get_or_404(Submission, submission_uuid, column=Submission.uuid)
utils.delete_file_object(submission)
return jsonify({"message": "Submission deleted"}), 200
else:
abort(405)

@api.route("/sources//replies", methods=["GET", "POST"])
def all_source_replies(source_uuid: str) -> Tuple[flask.Response, int]:
if request.method == "GET":
source = get_or_404(Source, source_uuid, column=Source.uuid)
return (
jsonify({"replies": [reply.to_json() for reply in source.replies]}),
200,
)
elif request.method == "POST":
source = get_or_404(Source, source_uuid, column=Source.uuid)
if request.json is None:
abort(400, "please send requests in valid JSON")

if "reply" not in request.json:
abort(400, "reply not found in request body")

data = request.json
if not data["reply"]:
abort(400, "reply should not be empty")

source.interaction_count += 1
try:
filename = Storage.get_default().save_pre_encrypted_reply(
source.filesystem_id,
source.interaction_count,
source.journalist_filename,
data["reply"],
)
except NotEncrypted:
return jsonify({"message": "You must encrypt replies client side"}), 400

# issue #3918
filename = path.basename(filename)
```

```
reply = Reply(session.get_user(), source, filename, Storage.get_default())

reply_uuid = data.get("uuid", None)
if reply_uuid is not None:
# check that is is parseable
try:
UUID(reply_uuid)
except ValueError:
abort(400, "'uuid' was not a valid UUID")
reply.uuid = reply_uuid

try:
db.session.add(reply)
seen_reply = SeenReply(reply=reply, journalist=session.get_user())
db.session.add(seen_reply)
db.session.add(source)
db.session.commit()
except IntegrityError as e:
db.session.rollback()
if "UNIQUE constraint failed: replies.uuid" in str(e):
abort(409, "That UUID is already in use.")
else:
raise e

return (
jsonify(
{
"message": "Your reply has been stored",
"uuid": reply.uuid,
"filename": reply.filename,
}
),
201,
)
else:
abort(405)

@api.route("/sources//replies/", methods=["GET", "DELETE"])
def single_reply(source_uuid: str, reply_uuid: str) -> Tuple[flask.Response,
int]:
```

```
get_or_404(Source, source_uuid, column=Source.uuid)
reply = get_or_404(Reply, reply_uuid, column=Reply.uuid)
if request.method == "GET":
return jsonify(reply.to_json()), 200
elif request.method == "DELETE":
utils.delete_file_object(reply)
return jsonify({"message": "Reply deleted"}), 200
else:
abort(405)

@api.route("/submissions", methods=["GET"])
def get_all_submissions() -> Tuple[flask.Response, int]:
submissions = Submission.query.all()
return (
jsonify(
{
"submissions": [
submission.to_json() for submission in submissions if submission.source
]
}
),
200,
)

@api.route("/replies", methods=["GET"])
def get_all_replies() -> Tuple[flask.Response, int]:
replies = Reply.query.all()
return (
jsonify({"replies": [reply.to_json() for reply in replies if
reply.source]}),
200,
)

@api.route("/seen", methods=["POST"])
def seen() -> Tuple[flask.Response, int]:
"""
Lists or marks the source conversation items that the journalist has seen.
"""

if request.method == "POST":
```

```
if request.json is None or not isinstance(request.json,
collections.abc.Mapping):
abort(400, "Please send requests in valid JSON.")

if not any(map(request.json.get, ["files", "messages", "replies"])):
abort(400, "Please specify the resources to mark seen.")

# gather everything to be marked seen. if any don't exist,
# reject the request.
targets: Set[Union[Submission, Reply]] = set()
for file_uuid in request.json.get("files", []):
f = Submission.query.filter(Submission.uuid == file_uuid).one_or_none()
if f is None or not f.is_file:
abort(404, f"file not found: {file_uuid}")
targets.add(f)

for message_uuid in request.json.get("messages", []):
m = Submission.query.filter(Submission.uuid == message_uuid).one_or_none()
if m is None or not m.is_message:
abort(404, f"message not found: {message_uuid}")
targets.add(m)

for reply_uuid in request.json.get("replies", []):
r = Reply.query.filter(Reply.uuid == reply_uuid).one_or_none()
if r is None:
abort(404, f"reply not found: {reply_uuid}")
targets.add(r)

# now mark everything seen.
utils.mark_seen(list(targets), session.get_user())

return jsonify({"message": "resources marked seen"}), 200

abort(405)

@api.route("/user", methods=["GET"])
def get_current_user() -> Tuple[flask.Response, int]:
return jsonify(session.get_user().to_json()), 200

@api.route("/users", methods=["GET"])
def get_all_users() -> Tuple[flask.Response, int]:
users = Journalist.query.all()
return jsonify({"users": [user.to_json(all_info=False) for user in users]}),
200
```

```
@api.route("/logout", methods=["POST"])
def logout() -> Tuple[flask.Response, int]:
session.destroy()
return jsonify({"message": "Your token has been revoked."}), 200

def _handle_api_http_exception(
error: werkzeug.exceptions.HTTPException,
) -> Tuple[flask.Response, int]:
# Workaround for no blueprint-level 404/5 error handlers, see:
# https://github.com/pallets/flask/issues/503#issuecomment-71383286
response = jsonify({"error": error.name, "message": error.description})

return response, error.code # type: ignore

for code in default_exceptions:
api.errorhandler(code)(_handle_api_http_exception)

return api
```

Static Code Metrics:

Metric	Value
Lines of Code (LOC)	325
Comment Density	0.03503184713375796
Cyclomatic Complexity	4
Maintainability Index	38.80917800649706
Halstead Metrics	
Unique Operators	8
Unique Operands	58
Total Operators	39
Total Operands	72
Volume	670.9277472487884
Effort	3331.503296683639
Depth of Inheritance Tree (DIT)	0
Coupling Between Object classes (CBO)	0
Lack of Cohesion of Methods (LCOM)	0
Fan-in	75
Fan-out	75

Number of Methods (NOM)	0

McCall's Intermediate Quality Metrics:

Metric	Value
Modifiability	0.49
Testability	0.65
Reliability	0.67
Understandability	-0.14
Self-Descriptiveness	0.18
Reusability	1.0
Portability	0.75
Efficiency	0

GPT Analysis Report:

Code Analysis and Suggestions for Improvement

Summary of Current State

The provided Python program serves as a Flask API for managing sources, journalist sessions, submissions, and replies. The metrics suggest that while the code is functional, it suffers from issues related to maintainability, understandability, and comment density.

Here are detailed feedback and suggestions regarding the structure, readability, and functionality:

1. Comment Density and Documentation

- **Current State**: The comment density is relatively low (approximately 3.5%).
- **Improvement Suggestions**: Increase the number of inline comments and docstrings. Each function should have a clear docstring explaining its purpose, parameters, return values, and potential exceptions raised. This will enhance understandability for other developers.

Example:

```
python
@api.route("/token", methods=["POST"])
def get_token() -> Tuple[flask.Response, int]:
"""
Handles token authentication for journalists.

Returns:
```

```
Tuple[flask.Response, int]: JSON response containing the token and status
code.
"""
```

2. Error Handling Improvements

- **Current State**: Error handling is performed with `abort()` calls but lacks a centralized mechanism for specific responses.

- **Improvement Suggestions**: Implement a custom error handler for specific exceptions and validation errors. This will eliminate some redundancy in error messages and improve the maintainability of the error handling approach.

Example:

```
python
@api.errorhandler(400)
def handle_bad_request(error):
return jsonify({"error": "Bad Request", "message": error.description}), 400
```

3. Single Responsibility Principle

- **Current State**: Functions like `get_token` are doing multiple things: fetching credentials, authenticating the user, managing the session, and handling errors.

- **Improvement Suggestions**: Break down large functions into smaller, more specific ones. For instance, a dedicated function for validating credentials can improve clarity and adherence to the single responsibility principle.

Example:

```
python
def validate_credentials(creds):
if 'username' not in creds:
raise ValueError("username field is missing")
# Additional checks...
```

4. Consistency of Endpoint Methods

- **Current State**: Some endpoints explicitly mention their allowed methods (e.g., "GET", "DELETE") while others do not.

- **Improvement Suggestions**: Ensure that all route definitions are consistent and clearly specify which HTTP methods are allowed. This helps reduce confusion and improves API documentation.

5. Use of Helper Functions

- **Current State**: Repeated patterns, such as fetching an object and handling potential `404` errors, are scattered across multiple endpoints.

- **Improvement Suggestions**: Use helper functions to encapsulate common operations. This would lead to DRY (Don't Repeat Yourself) principles and reduce code duplication.

Example:

```
python
def fetch_and_handle_404(model, object_id):
record = get_or_404(model, object_id, column=model.uuid)
return record
```

6. Unit Testing and Testability

- **Current State**: The metrics indicate a Testability index of 0.65, implying that parts of the code may be difficult to test.

- **Improvement Suggestions**: Isolation of business logic from HTTP handling can enhance testability. Consider separating service logic (business rules, database interactions) from controller logic (request handling). Writing unit tests for service functions improves coverage and confidence in code behavior.

7. Maintainability Index

- **Current State**: The Maintainability Index is below a desirable threshold (38.8).

- **Improvement Suggestions**: Refactor complex routes into smaller private functions that handle specific tasks (e.g., data validation, recording actions to the database). A focus on simpler, smaller functions can result in cleaner and more maintainable code.

8. Use of Type Hinting

- **Current State**: The code largely uses type hints, but there are instances where parameters can benefit from more precise type annotations.

- **Improvement Suggestions**: Ensure all function parameters and return types utilize Python's type hinting effectively. This aids in readability and understanding the expected types.

9. Potential Security Enhancements

- **Current State**: Authentication logic relies on fundamental checks.

- **Improvement Suggestions**: Implement rate limiting and account lockout mechanisms to enhance security, especially in the login route to mitigate brute force attacks.

Conclusion

By addressing these areas, the code's overall quality can be significantly improved, leading to enhanced readability, maintainability, and better adherence to best practices in software engineering. These changes will make it more robust against future modifications and assist new developers in understanding the code faster.