SWE 573: Software Development Practice Fall 2024

Project Final Deliverables

Mert Ünlü – 2023719111

Student ID: 2023719111

Name-Surname: Mert Ünlü

Date: 20/12/2024

Project Name: Mysterium

Deployment URL: https://mysterium.onrender.com

Github Repository URL: https://github.com/MertUnlu-SWE/SWE-573-MertUnlu

Github Tag Version URL: https://github.com/MertUnlu-SWE/SWE-573-MertUnlu/releases/tag/v0.9

Demo Video: https://github.com/MertUnlu-SWE/SWE-573-MertUnlu/tree/main/ProjectMaterials/FinalDemo

HONOR CODE

Related to the submission of all the project deliverables for the Swe573 Fall 2024 semester project reported in this report, I Mert Ünlü declare that:

- I am a student in the Software Engineering MS program at Bogazici University and am registered for the Swe573 course during the Fall 2024 semester.

- All the material that I am submitting related to my project (including but not limited to the project repository, the final project report, and supplementary documents) have been exclusively prepared by myself.

- I have prepared this material individually without the assistance of anyone else with the exception of permitted peer assistance which I have explicitly disclosed in this report.

Mert Ünlü [illegible]

Table Of Contents

References

I followed the *Python Django - The Practical Guide* (https://www.udemy.com/course/python-django-the-practical-guide/) course on Udemy to build a solid foundation in Django and applied the knowledge gained according to the needs of my project. In addition, I utilized AWS RDS PostgreSQL and an S3 Bucket for media and static file storage, integrating them as per the requirements of the project. I also installed docs for pushing docker images to a docker hub repository which helped me in learning how to dockerize my application. During the project, I frequently consulted the official Django documentation to find solutions to problems or learn about the correct way to achieve specific tasks. Finally, for deployment I utilized Render (https://render.com/) to host the project for deploying my built docker image.

Overview

The Mysterium project is a web platform built to help people share, find, and talk about mysterious things with unknown purposes. It has been designed with features that allow surplus user engagement, semantic tagging, advanced search components and media rich posts. The project aims to ensure a conversational interface and strong architecture with reliable systems through the application of contemporary technologies.

Key Features:

Post Creation: Users can share objects by providing detailed descriptions, images, and multiple semantic tags.

Semantic Tagging: Tags are enriched with Wikidata integration, allowing users to connect posts with relevant entities for enhanced searchability.

Advanced Search: A robust search system supports filtering by attributes like size, color, material, and more.

User Interactions: Users can comment on posts, upvote or downvote content, and bookmark their comments for later access.

Post Editing: Post creators can edit their content and tags, ensuring the platform remains flexible and up to date.

Technological Stack:

Backend: Python Django framework with PostgreSQL database hosted on AWS RDS.

Frontend: HTML, CSS, and Bootstrap for responsive and intuitive designs.

Storage: AWS S3 for handling of media and static files.

Deployment: Dockerized application deployed on Render.

Development Journey:

The project was developed in an iterative way. From starting initial designs and mockups to a functional deployment, features were added and tested incrementally for focusing on maintaining a balance between functionality and user experience. Regular tests ensured the reliability of the system which achieved 80% unit test coverage.

Challenges and Achievements:

Connecting semantic tags with Wikidata and correct database interactions required extensive debugging due to added tags not being the intended tags with same title. It was later fixed by adding an option to search for tag and it lists found tags with q code and short description.

Implementing Docker, pushing image to Dockerhub and deploying the application on Render took more time than expected due to having some errors on render when deploying for the first time and it gave some exit codes that did not give any log information.

Serving static and media files was another challenge since Django in production mode does not serve files especially media files when running server, docker or deployment. Fortunately, AWS S3 Bucket enables users to upload and serve these files with some configuration with access keys and giving permission.

The project provided valuable experience in handling full-stack development, database integration, and deploying production-ready applications.

The Mysterium project is now live and ready for further enhancements.

Software Requirements Specification

This document outlines the software requirements for a mystery object platform, focusing on ownership and moderation, post management, user interaction, community features and general requirements to make sure user engagement.

Requirements:

User Management

- 1.1. Users shall register and log in to Platform.
- 1.2. User profiles shall include bio, posts and bookmarked comments.

- 1.3. Admins can delete user accounts. However, posts with comments will remain on the platform.
- 1.4. Users cannot post anonymously.
- 1.5. Moderators shall have privileges to manage user-generated content and user accounts.
- 1.6. Users can delete their accounts.
- 1.7. Users can edit their accounts.

Content Creation and Management

2.1. **Post Creation:** Users can create posts on Platform, describing an object using various attributes such as:

- 2.1.1. Required fields such as title, description and image.
- 2.1.2. Optional fields such as material, size, color, weight, shape, texture, volume, taste, condition, and markings etc.
- 2.1.3. Adding semantic tags that must be connected to Wikidata entities and uploading images.

2.2. **Comments and Interaction:**

- 2.2.1. Users can comment on posts, ask questions, or share insights.
- 2.2.2. Users can upvote-downvote comments.
- 2.2.3. Users can bookmark their own comments.
- 2.2.4. Users can reply other user comments.

2.3. **Tagging and Semantic Search:**

- 2.3.1. Users can add multiple semantic tags, linking them to relevant Wikidata entries.
- 2.3.2. Advanced search functionality to filter posts by attributes and tags.

2.4. **Resolution System:**

- 2.4.1. Posts can be marked as "solved" when an object's function or identity is agreed upon.
- 2.4.2. Solved status can be updated by the post creator or moderators.
- 2.4.3. There can be only one "solved" comment per post.
- 2.4.4. Posts can be marked as “unsolved” if the information is not sufficient.

Content Visibility and Interaction

- 3.1. A feed page must show the latest or trending posts on Mysterium.
- 3.2. Advanced filtering options for posts based on object attributes, title, sorting or semantic tags.
- 3.3. Users can upvote or downvote posts and comments to indicate their relevance or helpfulness.

Moderation and Administration

- 4.1. Moderators have privileges to manage content.
- 4.2. Moderators have privileges to resolve disputes and oversee the tagging system.

Non-Functional Requirements:

Technology Stack

- 5.1. **Backend:** Python Django.
- 5.2. **Database:** PostgreSQL shall be hosted on AWS RDS.
- 5.3. **Frontend:** Bootstrap CSS for design and responsiveness and Glider JS for main page design.
- 5.4. **Deployment:** Render is required for deployment of built docker image.
- 5.5. The platform shall support computational operations, such as semantic search and advanced filtering.

User Privacy and Data Management

- 6.1. Users can choose to remain anonymous, but have to login for specific actions (upvote-downvote, comment, create post, bookmark comment, mark as solved).
- 6.2. User data such as posts with comments, shall persist even after account deletion.

Performance and Scalability

- 7.1. The platform shall support efficient search capabilities and semantic filtering.
- 7.2. The architecture must have smooth interactions, even for long discussions or large datasets.

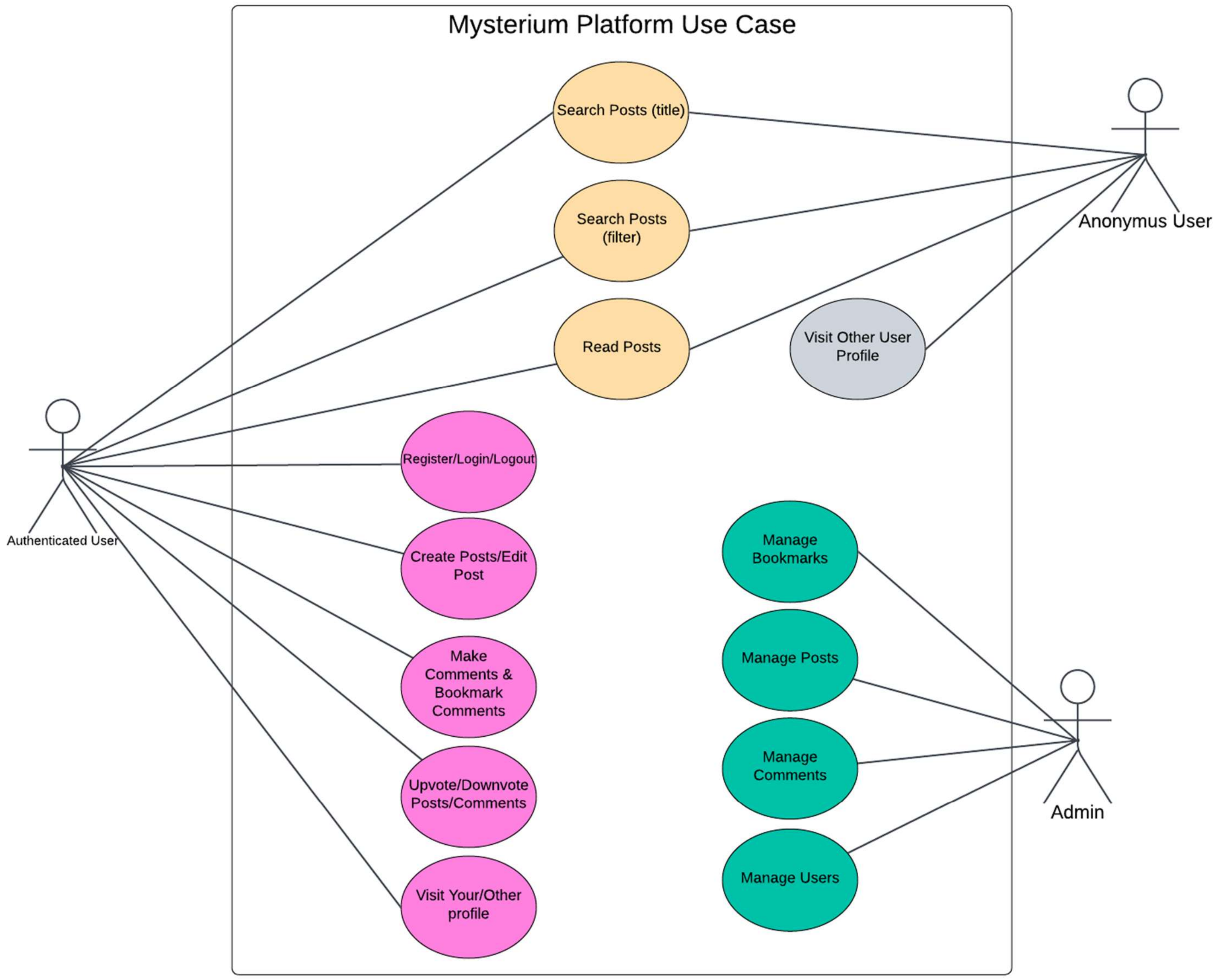
Usability and Accessibility

- 8.1. The user interface shall be intuitive and optimized for both desktop and mobile devices.

- 8.2. The platform must avoid a generic forum structure, focusing specifically on mysterious objects and their exploration.

Design

Use Case:



Mockup Scenario & Screens: You can visit the github page here, under the ProjectMaterials folder for screens and wiki page for scenario.

Status Of Project

Feature	Completed Requirements	Incomplete Requirements	Status
User Login/Logout/Register	Implemented and functional	None	Complete
User Create/Edit Post	Title, description, image, and descriptive fields with unit selection completed	Location map functionality in post creation/editing not implemented	Partial
User Comment on Post	Implemented and functional	None	Complete
User Upvote/Downvote on Post	Implemented and functional	None	Complete
User Upvote/Downvote on Comment	Implemented and functional	None	Complete
User Bookmark/Unbookmark Their Comment	Implemented and functional	None	Complete
User Profile Page	Implemented and functional	None	Complete
Bookmarked Comment List with Link to Post	Implemented and functional	None	Complete
Basic Search by Title	Implemented and functional	None	Complete
Filter Advanced Search	Implemented and functional	None	Complete
Showing Trending Posts According to Post Upvotes	Implemented and functional	None	Complete

Feature	Completed Requirements	Incomplete Requirements	Status
User Incentive	Not implemented due to time constraints	Feature skipped due to time constraints	Not Implemented
User Account Deletion	Not implemented due to time constraints	Feature skipped due to time constraints	Not Implemented
Moderator Login Page	Partially implemented; URL-based access only	Login page for moderators missing	Partial
Location with Map in Post Creation/Editing	Not implemented due to time constraints	Map integration skipped due to time constraints	Not Implemented

Status Of Deployment

The application is deployed on Render for production. I built a Docker image from the Dockerfile, tagged it as v1 and pushed it to my DockerHub repository. I also have my container running correctly. The PostgreSQL database is hosted on an AWS RDS instance, ensuring reliable and scalable data management. Moreover, AWS S3 Bucket is used for storing both media and static files because it provides efficient file handling and seamless integration with the application.

Deployment URL: https://mysterium.onrender.com

AWS S3 Bucket Media Files: https://mysterium.s3.eu-north-1.amazonaws.com/media/

AWS S3 Bucket Static Files: https://mysterium.s3.eu-north-1.amazonaws.com/static/

Installation Instructions

1. **Repository Setup**
    - Clone the project repository from GitHub and follow the instructions provided in the Project Wiki.
    - Before installing dependencies make sure to go to the correct project directory which is "cd mysteriumProject"
    - Make sure to run the command "pip install requirements.txt" for dependencies after cloning project
2. **Environment Configuration**
    - Update the necessary environment variables for AWS RDS PostgreSQL (database credentials) and AWS S3 Bucket (for storing media and static files) as described in the Wiki. **If you want access to the original ".env" file contact the project owner for details.**
3. **Docker & Docker Image**
    - Make sure to download Docker desktop for running and displaying the containerized project
    - Have an account on Dockerhub to be able to open a repository for the image that will be stored
    - The project uses a pre-built Docker image which is tagged and pushed to DockerHub. Refer to the Wiki for detailed steps on how to run and integrate it into your environment.
4. **Deployment on Render**
    - The application is deployed on Render. Ensure that the Docker image is accessible and the PostgreSQL database on AWS RDS is connected correctly.
    - Create a project at the dashboard.
    - After creation, create a web service that uses an existing docker image.
    - Copy your docker image from the Dockerhub repository including the tag.

Important Notes:

- Make sure your AWS credentials and permissions are correctly set for both RDS and S3 services.
- Refer to the Wiki for troubleshooting common issues.

User Manual

Users of the Mysterium platform can explore and share mysterious items using a straightforward and understandable interface. This manual guides on properly navigating and using the features of the platform.

Getting Started

To visit Mysterium users can enter the platform without logging in. The homepage serves as the central hub for browsing trending posts, searching for specific objects, and interacting with the community.

User Registration

Registration is mandatory for accessing the platform's full functionality. New users can register by clicking the "Login" then below the login, there is a “Register here” button and user can fill out the form with a username, first-last name, email address, and password. Once registered, users are automatically logged in and redirected to the homepage.

User Login

Existing users can log in by entering their registered email and password on the login page. Successful login grants access to personalized features like creating posts, commenting, upvoting content and bookmarking their own comments.

Anonymous User

Users who are not logged in can only navigate posts, do basic/advanced search, visit other users profile pages by clicking on their username.

Creating a Post

Users can share details about mysterious objects by creating a post. To do this, click the "Share Your Mystery" button at the main page, it will direct user to post creation page. Fill in mandatory fields like title and description and upload an image of the object. Optional fields allow users to add additional descriptive details such as width, length, height, material, size, color, weight and so on. Semantic tags can also be added to link the object to relevant Wikidata entries, enhancing searchability. User needs to write a tag then click search tag and it will display possible tags with q code and short description for user to select the desired one. Upon clicking the correct tag it will be added to the added tags section below the tag input area. User can also remove the added tag by clicking the “x” on the added tag.

Editing a Post

Post creators can edit their content by visiting the post's detail page. Clicking "Edit Post" allows users to modify details or add new tags. However, user needs to re-enter the previous tags again due to a bug that erases previous added tags after clicking edit post button. Except this bug every other previous field will be displayed correctly and does not have any issues.

Commenting on Posts

Engage with the community by commenting on posts. Users can share insights, ask questions, or provide additional context. All comments are displayed near the post detail with timestamps and user details.

Upvoting and Downvoting

To express agreement or disagreement with a post or comment, users can upvote or downvote. These votes affect the visibility and ranking of posts on the platform.

Bookmarking Comments

Users can bookmark their own comments for easy access later. Bookmarked comments are stored in the user's profile, complete with links to the original posts.

Profile Management

The profile page displays user information, created posts, and a list of bookmarked comments. Users can manage their personal details and track their activity history from this section. Users can also visit other users profile page by clicking on their username if they posted an object or have commented on a post.

Search Features

Mysterium offers both basic and advanced search functionalities. Users can search for posts by title or apply filters such as material, color, tags and sorting option for more specific results. Advanced search ensures that users can find objects matching their interests.

Trending Posts

The homepage highlights trending posts based on upvotes. This feature allows users to discover the most popular content quickly.

Logging Out

Users can log out by clicking the "Logout" button in the navigation bar. This ensures account security, especially on shared devices.

Additional Features

- **Semantic Tagging**: Add semantic tags linked to Wikidata to enhance the post's context and discoverability.
- **Privacy and Security**: Only logged-in users can interact with posts to maintain a secure and authentic community experience.

Test Results

User Tests

1. User Registration

- Navigate to the application's URL.
- Click the "Login" button.
- If don't have an account, click the registration button below.
- Fill out the registration form with a username, first-last name, email, and password.
- Submit the form and verify redirection to the homepage.
- Confirm that the user is logged in, and the username is displayed in the navigation bar.

2. User Login

- Access the login page via the application's URL or redirection after logout.
- Enter a valid email and password.
- Submit the form and verify successful login with a welcome message.

3. Post Creation

- Log in with valid credentials.
- Navigate to the post creation page.
- Fill in the required fields (e.g., title, description, tags) and upload an image.
- Submit the form and confirm that the new post appears on the homepage or in the relevant category.

4. Post Editing

- Log in as the creator of a post.
- Navigate to the post detail page and click "Edit Post."
- Modify fields like title, description, or tags.
- Save changes and confirm updates are reflected on the post detail page.

5. Post Deletion

- Log in as the creator of a post.
- Navigate to the post detail page and click "Delete Post."
- Confirm the deletion action.
- Verify that the post no longer appears in the listings or search results.

6. Commenting on a Post

- Log in and view a post detail page.
- Enter a comment in the comment box and submit.
- Confirm the comment is displayed under the post with the correct timestamp and user information.

7. Bookmarking a Comment

- Navigate to a post detail page and locate your comment.
- Click the "Bookmark" button.
- Verify that the comment is listed in the user's profile under the "Bookmarks" section.

8. Upvoting/Downvoting Posts and Comments

- Log in and navigate to a post or comment.
- Click the "Upvote" or "Downvote" button.
- Confirm that the vote count updates appropriately.
- Test toggling and removing votes.

9. Advanced Search

- Navigate to the search page.
- Use filters like tags, price range, or material to refine the search.
- Submit the query and verify that the results match the applied filters.

10. Semantic Tagging

- During post creation, enter tags and search them that are connected to Wikidata.
- Save the post and verify that tags appear with linked metadata.

11. Profile Management

- Log in and access the user profile page.
- Verify the display of personal information, created posts and bookmarked comments.

12. Logout

- Log out from the application.
- Confirm redirection to the homepage with no user-specific actions available.

Unit Tests

The application's functionality was extensively tested through a series of unit tests designed to validate individual components and their integration into the system. Below is a detailed description of each test category and its purpose:

ViewTests

1. test_register_view

This confirms that a person can sign up as a user by making sure the site creates a user entry for him. And it also checks that the user is correctly redirected.

2. test_login_view

This is used both to check whether there is a working login function and to confirm that if the user enters login information, he is indeed directed to the relevant page of the interface.

3. test_post_creation_view

This helps in simulating the creation of a post. It checks that a post is created and stored in the system's database. It additionally checks that the post details are accurately displayed.

4. test_post_detail_view

This makes sure the post view is able to show the title and description which the post maker has input with tags and an image. It also tests for posts that have more than one tag.

5. test_edit_post

This assures that changing fields of a post, like the title, description, tag and so forth have been done successfully and those changes are displayed.

6. test_fetch_wikidata

Tests the ability to integrate with a Wikidata API by fetching tag information. The tag fetch test expects a typical response where tags are provided and an error response where no tags are given.

7. test_vote_post

This confirms that the functionality for upvoting a post works.

The test confirms that the upvote management is always increased when a vote is made, and the next actions are performed properly.

8. test_mark_as_solved

Verifies that if a user is not a post creator, then such functionality is not accessible to them. Attempting to do so will be denied, while someone with such privileges will be able to do so with success.

9. test_unmark_as_solved

Confirms that the creator of a particular post is the only individual who can unlock a post which has been marked as solved. All other attempts to do so will be disallowed.

10. test_bookmark_comment

Validates that an action of bookmarking any comment is available for the user and that the bookmark will be located in the database.

11. test_unbookmark_comment

Questions the processes of the system when attempts to remove the bookmark have been made. The comment that was previously bookmarked should be able to be unbookmarked without problems.

12. test_view_bookmarked_comments

Verifies and confirms that any and all comments which have been bookmarked are located within the profile of the individual user.

13. test_duplicate_bookmark

Guarantees that any duplicates of a bookmark which have previously already been created onto the same comment will not be accepted. It examines this concerning the appropriate error message provided for the occasion.

14. test_unbookmark_nonexistent

Analyzes the systems function when comments which have not been targeted with bookmarks intended for unmarking are attempted to be unbookmarked. The system should never encounter any problems doing this in summary for all the other commands.

15. test_basic_search

Validates the working of the basic search feature for the user. The test ensures that all the posts which satisfy the query have been returned and that the results have been filtered correctly.

16. test_profile_view

The page comprising of profiles verifies if the users, posts and bookmarks shown correspond to the correct details. Furthermore, it makes certain that only self-owned private data is accessible to the users.

17. test_logout_view

Verifies the logout feature by ensuring users are taken to the home page and a prompt is shown to the user on successful logout.

EmailBackendTests

18. test_authenticate_success

This tests if a person provides a pre registered email and password is able to log in without any problems.

19. test_authenticate_failure

In this case, all the details are checked to ensure that if these are not the valid details, then logging in is impossible.

20. test_get_user

Here, putting a user's identification to information and storing it into a database is cross checked.

PostFormTests

21. test_post_form_valid

Making sure that the data sent through a form is properly attached with appropriate details, the feature is cross tested.

22. test_post_form_invalid

It guarantees that a form submission that is lacking compulsory elements is blocked.

CommentFormTests

23. test_comment_form_valid

Also a properly filled out comment form is checked, ensuring it is submitted properly.

24. test_comment_form_invalid

Determines that if a comment post that does not have all the elements is submitted, it is declared as Euro valid.

PostModelTests

25. test_post_creation

Also to make sure that when a post is done all up votes and solved parameters have not been changed from the allowance's synod.

CommentModelTests

26. test_comment_creation

In this case the sample tests if the comment has been done and then correctly combines the comment to the relevant post.

WikidataUtilsTests

27. test_fetch_wikidata_info_success

Also makes mocks and tests to capture the successful attempts made to fetch the tag details from an API of Wikidata.

28. test_fetch_wikidata_tags_success

Checks if the retrieval of multiple tags from the Wikidata API is done and that the output is a valid one.

Current Status:

- Total Tests Implemented: **28**
- Coverage: **80%** of unit tests written.
- The missing tests were not written due to not having enough time but all implemented main functionalities have been written and tested.
- Results: All implemented tests are **passing** but there are missing tests.