Swe573 Software Development Practice, Fall 2024

Final Project Report

Yılmaz Ünal
Student ID: 2023719108
December 21, 2024

Deployment URL: https://swe573finder-849897479442.us-central1.run.app
GitHub Repository URL: https://github.com/ylmzunal/SWE573-Software-Development-Practice
Git tag version URL: https://github.com/ylmzunal/SWE573-Software-Development-Practice/releases/tag/v1.0.0

HONOR CODE
Related to the submission of all the project deliverables for the Swe573 Fall 2024 semester project reported in this report, I Yılmaz Ünal declare that:

- I am a student in the Software Engineering MS program at Bogazici University and am registered for 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.

Yılmaz Ünal

Contents

1 Project Overview

The Object Finder platform helps users identify and find information about mystery objects using their physical properties and other details. It focuses on man-made objects and provides tools for users to post queries, share insights, and contribute to solving mysteries collaboratively. The key features of the platform include:

1. **User Interaction:**
    - Users can create accounts, log in, and manage profiles with optional biographical details.
    - Anonymous posting is allowed but is traceable by administrators for moderation.
2. **Object Identification:**
    - Posts must include detailed object descriptions, images, and physical attributes such as material, size, color, and shape.
    - Tags and categories are generated from Wikidata for better organization and searchability.
3. **Collaborative Discussion:**
    - Users can comment on posts to ask questions or provide answers.
    - Posts and comments can be upvoted or downvoted to highlight valuable contributions.
    - A thread-based architecture ensures clarity in discussions.
4. **Advanced Search:**
    - The platform allows users to search objects based on keywords, physical attributes, or tags.
    - Results can be refined using filters like material, color, or weight.
5. **Reputation System:**
    - The platform allows users to search objects based on keywords, physical attributes, or tags.
    - Results can be refined using filters like material, color, or weight.
6. **Integration with Wikidata:**
    - Information is enhanced with data from Wikidata using SPARQL queries for object attributes like material and history.

2 Software Requirements Specification

2.1 Functional Requirements

2.1.1 User Registration and Login

2.1.1.1 Users must sign up or log in to create posts. Completed

2.1.1.2 When the user clicks ”Register”, the application should open the form of “Register”. Completed

2.1.1.3 If the user’s passwords do not match in the Register Form, the software shall give a ”passwords do not match” error. Completed

2.1.1.4 When the user clicks ”Login”, the application should open the form of “Login”. Completed

2.1.1.5 Passwords must follow security rules. If the password does not meet the rules, the system should show an error message. Completed

2.1.1.6 If the user enters an incorrect password, the software should give an ”incorrect credentials” error. Completed

2.1.1.7 Users can comment without logging in by choosing to post anonymously. However, administrators will still know their identity. Completed

2.1.1.8 Users can delete their accounts. However, posts involved in ongoing discussions cannot be removed. Not Completed

2.1.2 Post and Comment System

2.1.2.1 All posts can be seen by the public, but only logged-in users can create them. Completed

2.1.2.2 Every post must include an image, a title (header), and a description written by the user. Completed

2.1.2.3 Posts must clearly show if they are ”Solved” or ”Unsolved.” Completed

2.1.2.4 Users can manually add as many tags as they like from the list of tags provided by Wikidata. Not Completed

2.1.2.5 Posts should appear on the homepage in chronological order after being created. Completed

2.1.2.6 Users can add comments to posts to ask follow-up questions or provide additional answers. Completed

2.1.2.7 Comments can have tags to make them easier to find in searches. Not Completed

2.1.3 User Profiles

2.1.3.1 Each user has a profile page with biography, profile picture, total posts, comments, and upvotes/downvotes. Completed

2.1.3.2 Profiles should display contribution incentives like badges, ranks, and achievements. Completed

2.1.3.3 Users can view others' profiles, posts, and contributions. Not Completed

2.1.4 Contribution Incentives

2.1.4.1 Implement a reputation system based on upvotes, badges, and ranks to incentivize user contributions. Completed

2.1.4.2 To reward quality contributions, the system should offer badges like "Top Contributor" based on the relevance and quality of input. Completed

2.1.5 Search and Filtering Requirements

2.1.5.1 The website must have a search bar for users to find content.Completed

2.1.5.2 Users should be able to filter search results by tags, attributes, dates, and other criteria. Completed

2.1.5.3 Post titles, attributes, and tags should all be searchable to make content easy to find. Completed

2.1.6 User Interface Requirements

2.1.6.1 The homepage must show recent posts in chronological order. Completed

2.1.6.2 After creating a post, the user should be automatically redirected to the post page. Completed

2.1.6.3 A "Categories" page must organize posts based on tags and attributes. Not Completed

2.1.7 Moderation and Admin System

2.1.7.1 Admins and moderators can change post statuses (e.g., resolved or unresolved). Completed

2.1.7.2 A moderation system allows users to report inappropriate content, and admins can enforce bans. Completed

2.1.7.3 Admins must review anonymous posts and enforce moderation policies. Completed

2.1.8 Object Search Functionality

2.1.8.1 Users can search for objects using images, voice, keywords, or filters. Not Completed

2.1.8.2 Results can be sorted by relevance and categories. Not Completed

2.1.8.3 Advanced search supports detailed attributes like material, size, weight, and functionality. Completed

2.1.8.4 The search system should support complex queries, e.g., "metal objects under 1kg and 10cm long." Not Completed

2.1.9 Displaying Results

2.1.9.1 Search results should include object images, names, and descriptions. Completed

2.1.9.2 Additional details such as object price and ratings must be displayed. Not Completed

2.2 Non-Functional Requirements

2.2.1 Performance

2.2.1.1 The system should efficiently handle simultaneous user interactions.

2.2.1.2 Search functionality must return results within 2-3 seconds for complex queries.

2.2.1.3 Optimized Wikidata queries to reduce response times

2.2.2 Scalability

2.2.2.1 The platform should scale to accommodate a growing number of users and interactions.

2.2.2.2 Use efficient database architectures like PostgreSQL or MySQL.

2.2.3 Security

2.2.3.1 Encrypt all user data, especially sensitive information like passwords.

2.2.3.2 Use HTTPS for secure communications.

2.2.3.3 Ensure security measures for image uploads to prevent malicious content.

2.2.4 Usability

2.2.4.1 Provide an intuitive UI for entering object attributes and searching content.

2.2.4.2 Ensure responsiveness for both desktop and mobile devices.

2.2.5 Reliability

2.2.5.1 Ensure 99.9% uptime with database and server redundancy.

2.2.5.2 Perform regular backups to prevent data loss.

2.2.6 Maintainability

2.2.6.1 Maintain modular and clean code for easy updates.

2.2.6.2 Keep developer documentation updated.

2.2.6.3 Implement logging and error monitoring for issue detection.

2.2.7 Interoperability

2.2.7.1 Provide interfaces for integrating additional APIs, especially for features like image search.

2.2.8 Extensibility

2.2.8.1 Design the platform to accommodate new features and attributes with minimal architecture changes.

2.2.9 Target Response Time

2.2.9.1 The search system must return results within 5 seconds.

2.2.10 UI Design

2.2.10.1 Ensure the UI is user-friendly with clear navigation.

2.2.10.2 Make core functionalities easily accessible.

3 Design Documents

3.1 System Overview

The Object Finder platform is designed to help users identify and find information about human-made objects by inputting their attributes. It uses a combination of Django for the backend, PostgreSQL for the database, and Google Cloud Storage for handling media files.

3.2 System Architecture

The system is designed using the Django web framework and follows a modular architecture. The key components of the architecture include:

- **Frontend**: Responsible for user interaction and visual presentation. Built using Django templates, HTML, CSS, and JavaScript for dynamic and responsive design.
- **Backend**: Implements the core logic and manages interactions with the database and external APIs. Built on Django, which follows the Model-View-Template (MVT) architecture.

- **Database**: PostgreSQL hosted on AWS Relational Database Service (RDS) is used for reliable and scalable data storage. It handles user information, posts, comments, tags, and other metadata.
- **Media Handling**: Google Cloud Storage is used to store and serve images and media files uploaded by users.
- **External API Integration**: Integration with Wikidata via SPARQL queries provides enriched information for object identification.
- **Hosting and Deployment**: The system is deployed on Google Cloud Run with support for static file serving and secure connections.

3.3 System Components

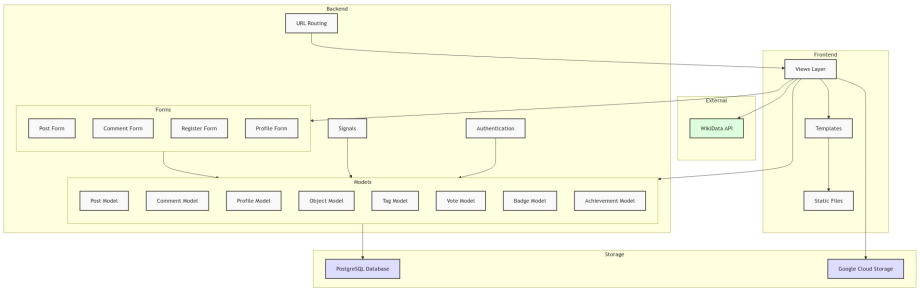

Figure 1: Structure Diagram

3.3.1 User Management

- Users can register, log in, and manage profiles.
- Profile details include a bio, profile picture, and statistics on posts and contributions.
- Admins can track anonymous user actions for moderation purposes.

3.3.2 Posts and Comments

- Users can create posts with a title, description, image, and object attributes such as material, color, shape, size, and weight.
- Posts can include tags derived from Wikidata for better categorization.
- Users can comment on posts, upvote or downvote comments.

3.3.3 Search Functionality

- Users can search for posts by keywords, tags, or object attributes.
- Advanced filtering options allow users to refine searches by specific attributes like material or size.
- Integration with Wikidata enhances search results with detailed object information.

3.3.4 Reputation System

- Users earn badges and ranks based on their contributions, such as creating posts, commenting, or receiving upvotes.
- Examples of badges include "Active Poster" for frequent posts or "Helpful Commenter" for well-received comments.

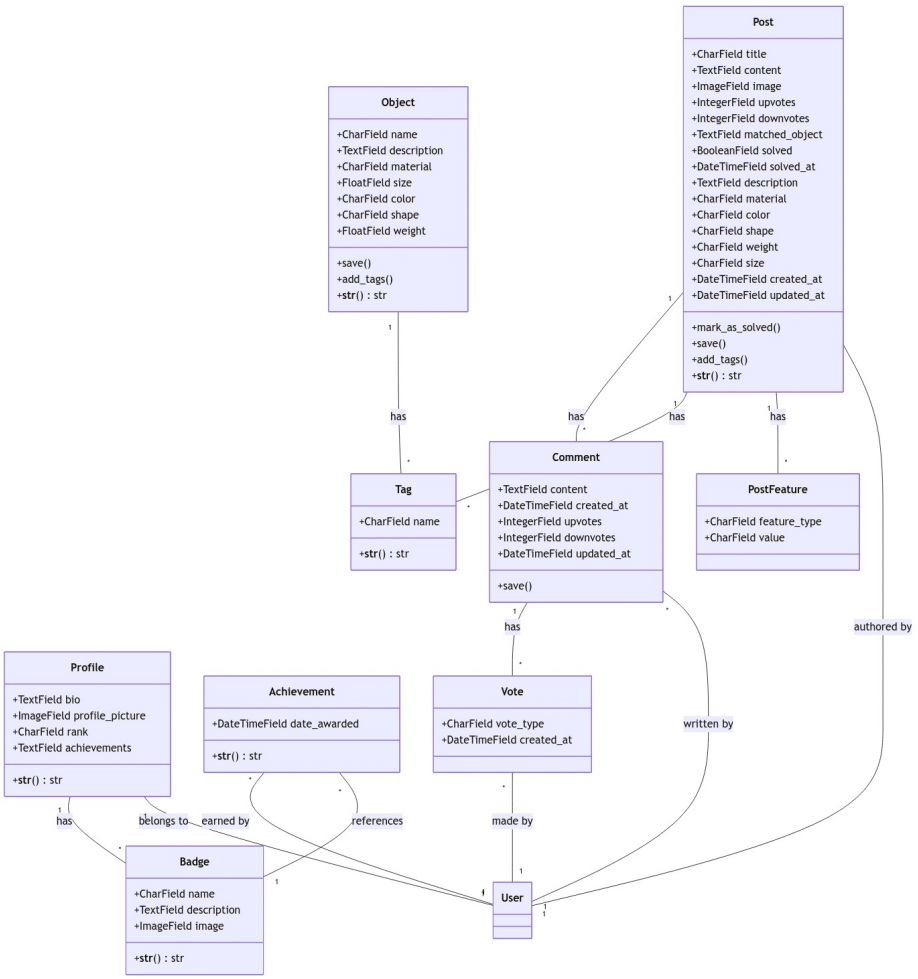

Figure 2: Class Diagram

3.4 Data Flow

3.4.1 User Actions and Data Processing

1. Registration and Login:
   - Users submit a registration form. The system validates input and creates a profile in the database.
   - Upon login, the system verifies credentials and grants access.
2. Creating Posts:

- Users submit a post form with details like title, description, and image.
- The backend processes the form, saves the post in the database, and uploads the image to Google Cloud Storage.
- Tags are added based on attributes such as material and color.

3. Searching for Objects:
   - Users input a search query or use filters to specify attributes.
   - The backend retrieves matching posts from the database and queries Wikidata for additional results.
   - Results are displayed with relevant metadata and images.
4. Commenting and Voting:
   - Users add comments or vote on existing ones.
   - The system updates the comment's vote count in real-time and stores user actions in the database.

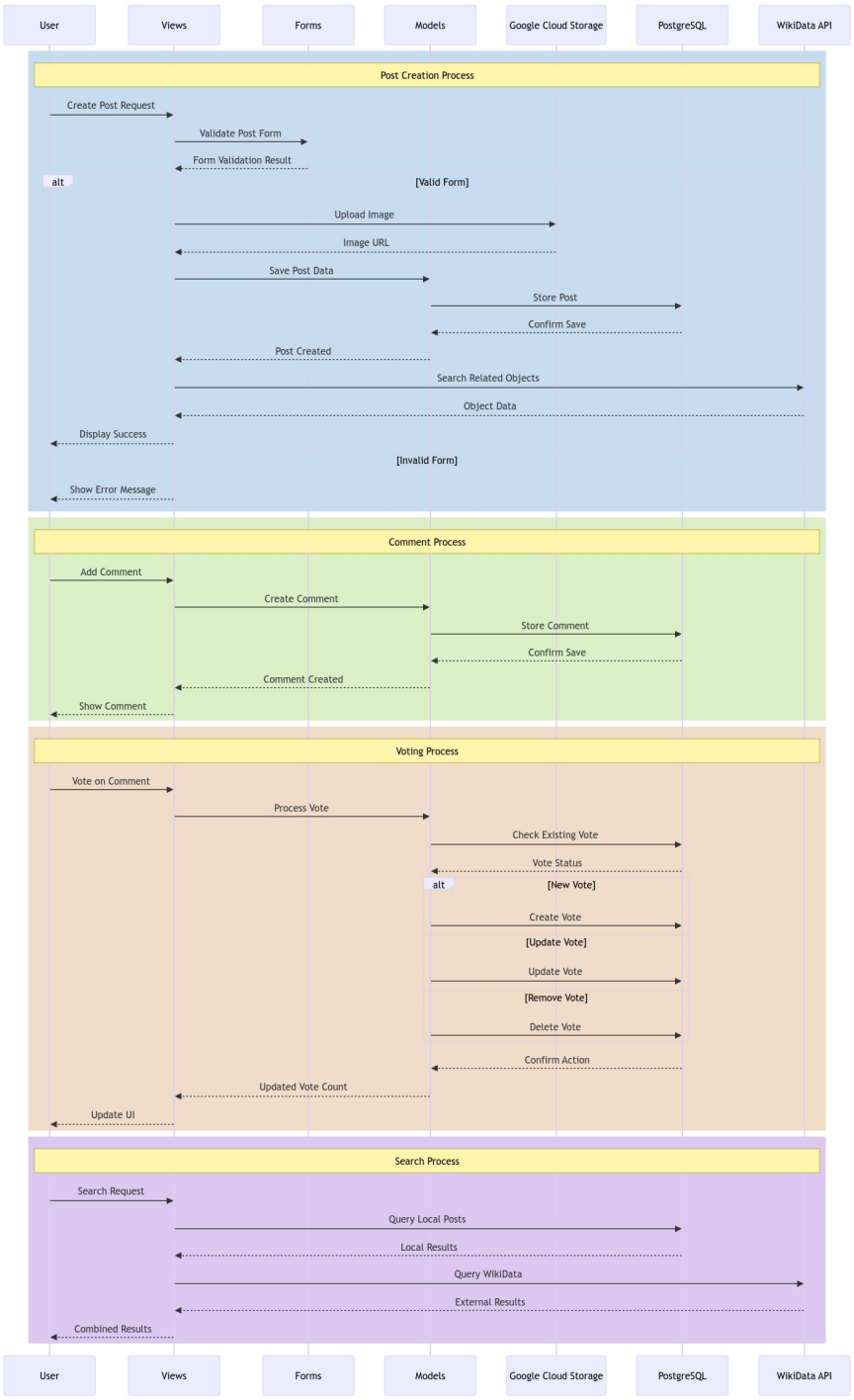

Figure 3: Sequence Diagram

3.5 Deployment and Configuration

3.5.1 Google Cloud Storage

- Media files are stored in a bucket with public read access.
- Files are uploaded through the Django backend using secure credentials.

3.5.2 Database Overview

- Database Type: PostgreSQL
- Hosting: AWS Relational Database Service (RDS)
- Scalability: AWS RDS provides features like automatic backups, multi-AZ deployments, and vertical scaling.
- Security: Encrypted connections and data at rest are enabled using AWS RDS's built-in security measures.

3.5.3 Security Settings

- CSRF protection is enabled for all forms.
- Sensitive data like secret keys and database credentials are stored in environment variables.

3.5.4 Debugging and Logging

- Debugging is enabled during development to log errors and file uploads.
- Logs include detailed information about file operations and server responses.

3.6 Key Technologies

- Backend: Django (Python).
- Frontend: HTML, CSS, JavaScript.
- Database: PostgreSQL.

- Media Storage: Google Cloud Storage.
- API Integration: Wikidata SPARQL API.
- Hosting: Google Cloud Run.

4 Project Status

4.1 Overview of Current Progress

Significant progress has been made on the Object Finder project, but some requirements need to be added. Below is a summary of the current status of various components:

4.2 Completed Tasks

- System Setup:
  - The Django framework is installed and configured.
  - PostgreSQL database is hosted on AWS RDS and connected to the application.
  - Google Cloud Storage is set up for storing media files.
- User Management:
  - User registration and login functionalities are fully implemented.
  - Users can update their profiles, including adding a bio.
  - Adding reputation and ranking systems based on user activity and contributions.
  - Displaying badges for achievements like "Active Poster" or "Helpful Commenter."
- Post and Comment Features:
  - Users can create posts with images, titles, descriptions, and object attributes (e.g., material, color, size).
  - Users can comment on posts and upvote/downvote comments.
  - Post statuses ("Solved" or "Unsolved") can be updated.
- Database:
  - All required database tables are created, including Users, Posts, Comments, Votes, and Tags.

  - Relationships between tables are properly set up and tested.

- Search Functionality:
  - Basic search by keywords is implemented.
  - Advanced search filters (e.g., material, size, and color) are functional.
  - Integration with Wikidata is set up for retrieving additional object details.

- Deployment:
  - The application is deployed on Google Cloud Run and accessible via a public URL.
  - Static files are served through a content delivery network (CDN).

4.3 Tasks In Progress

- Improved Search Functionality:
  - Refining search algorithms for better relevance and accuracy.
  - Testing and optimizing SPARQL queries for faster results from Wikidata.

- Delete Post Functionality:
  - Implementing functionality to allow users to delete their own posts.
  - Ensuring that deleted posts are properly removed from the database.

- Delete Account Functionality:
  - Implementing the ability for users to delete their accounts.
  - Ensuring that deleted accounts do not remove posts involved in ongoing discussions.

4.4 Known Issues and Challenges

- Search Speed: SPARQL queries to Wikidata are slower than expected for large datasets.
- Image Uploads: Occasionally, large image files take longer to upload and process.
- Cross-Browser Compatibility: Some CSS issues on older browser versions need resolution.

4.5 Summary

The Object Finder project is progressing well, with key functionalities like user management, posting, commenting, and basic search already completed. The focus now is on enhancing user experience, improving search speed, and completing pending features like image-based search, delete post, and delete account functionalities.

5 Deployment Status

5.1 Current Deployment Details

The Object Finder platform has been successfully deployed and is accessible online. The deployment includes the following key configurations:

5.1.1 Deployed URL

- The platform is hosted on Google Cloud Run.
- Publicly accessible at: https://swe573finder-849897479442.us-central1.run.app

Is it Dockerized: Yes. **Docker HuB url:** https://hub.docker.com/r/ylmzunl/swe573finder

6 Installation Instructions

6.1 Overview

This guide explains how to set up the Object Finder platform on your local machine or server. You can install the project either manually or using Docker. Follow the steps below for the desired method.

6.2 Prerequisites

Before starting the installation, ensure the following tools are installed:

- Python 3.9 or higher
- PostgreSQL (for database setup)
- pip (Python package manager)
- Git (to clone the repository)
- Docker (if using Docker for deployment)
- Google Cloud SDK (for media storage integration)

6.3 Cloning the Repository

1. Open your terminal or command prompt.
2. Clone the project repository using Git:

   ```
   git clone https://github.com/ylmzunal/SWE573-Software-Development-Practice.git
   ```

3. Navigate to the project directory:

   ```
   cd SWE573-Software-Development-Practice
   ```

6.4 Manual Setup (Without Docker)

6.4.1 Setting Up the Virtual Environment

1. Create a virtual environment:

   ```
   python -m venv env
   ```

2. Activate the virtual environment:
   - On Windows:

     ```
     env\Scripts\activate
     ```

   - On macOS/Linux:

     ```
     source env/bin/activate
     ```

6.4.2 Installing Dependencies

1. Install all required Python packages:

```
pip install -r requirements.txt
```

6.4.3 Creating the Database

1. Start your PostgreSQL service and log in as a user with sufficient privileges.
2. Create a new database:

```
CREATE DATABASE objectfinderdb;
```

3. Note the database credentials (username, password, host, and port) for later use.

6.4.4 Configuring the Environment

1. Create a '.env' file in the project root directory with the following content:

```
DEBUG=True
SECRET_KEY=your_secret_key
DATABASE_URL=postgres://username:password@host:port/objectfinderdb
GOOGLE_APPLICATION_CREDENTIALS=path_to_your_google_service_account.json
```

Replace placeholders with your actual credentials and file paths.

6.4.5 Applying Migrations

1. Run the following commands to apply database migrations:

```
python manage.py makemigrations
python manage.py migrate
```

6.4.6 Collecting Static Files

1. Run the following command to collect static files:

   ```
   python manage.py collectstatic
   ```

6.4.7 Running the Application

1. Start the Django development server:

   ```
   python manage.py runserver
   ```

2. Open a browser and visit:

   ```
   http://127.0.0.1:8000
   ```

6.5 Setup Using Docker

6.5.1 Building the Docker Image

1. Ensure Docker is installed and running on your system.
2. Build the Docker image:

   ```
   docker build -t objectfinder .
   ```

6.5.2 Setting Up the Database in Docker

1. Use Docker to set up a PostgreSQL container:

   ```
   docker run --name postgresdb -e POSTGRES_USER=username \
   -e POSTGRES_PASSWORD=password -e POSTGRES_DB=objectfinderdb \
   -p 5432:5432 -d postgres
   ```

2. Update your ‘.env‘ file with the correct connection details for the Docker PostgreSQL instance:

   ```
   DATABASE_URL:postgres://username:password@localhost:5432/objectfinderdb
   ```

6.5.3 Connecting the Application to the Database

1. Apply database migrations:

```
docker exec -it container_name python manage.py makemigrations
docker exec -it container_name python manage.py migrate
```

6.5.4 Running the Application in Docker

1. Start the application container:

```
docker run -p 8000:8000 objectfinder
```

2. Open a browser and visit:

```
http://127.0.0.1:8000
```

6.6 Troubleshooting

- Ensure PostgreSQL or the PostgreSQL Docker container is running.
- Verify that all environment variables in the '.env' file are correctly configured.
- Check logs for errors:

```
docker logs container_name
```

- If you encounter permission errors, ensure your database user has sufficient privileges.

7 User Manual

7.1 Introduction

This guide explains how to use the system to find information about objects, create posts, comment, and interact with other users. Follow the steps below to get started.

7.2 Getting Started

1. Open a web browser and go to:

   `https://swe573finder-849897479442.us-central1.run.app`

2. You can browse posts without logging in. To create posts or comment, you must register or log in.

7.3 Using the Platform

7.3.1 Creating an Account

- Click the ”Register” button on the homepage.
- Fill in your username, email, and password.
- Click ”Submit” to create your account.

7.3.2 Logging In

- Click the ”Login” button on the homepage.
- Enter your username and password.
- Click ”Submit” to log in.

7.3.3 Creating a Post

- Log in to your account.
- Click ”Create New Post” on the homepage.
- Fill in the title and description of your post.
- Add object details like material, color, size, or weight.
- Upload an image if available.
- Click ”Submit” to publish your post.

7.3.4 Searching for Objects

- Use the search bar at the top of the homepage.
- Type keywords like ”metal object” or ”red toy.”
- Apply filters to narrow your search (e.g., by material, size, or color).
- Click ”Search” to view results.

7.3.5 Commenting on Posts

- Open a post you are interested in.
- Scroll down to the comments section.
- Type your comment in the text box and click ”Add Comment.”

7.3.6 Voting on Comments

- Upvote helpful comments by clicking the thumbs-up icon.
- Downvote unhelpful comments by clicking the thumbs-down icon.

7.3.7 Marking Posts as Solved

- If your question is answered, you can mark your post as ”Solved.”
- Open your post and click the ”Mark as Solved” button.

7.3.8 Editing Posts and Comments

- To edit a post, open it and click ”Edit.”
- To edit a comment, click ”edit comment” on the post page.

7.3.9 Managing Your Profile

- Click your username at the top of the page to view your profile.
- Add a bio or view your posts and comments.

7.4 Tips for Best Use

- Use detailed titles and descriptions for posts.
- Add clear images to help others understand your object.
- Use polite and respectful language when commenting.

7.5 Troubleshooting

- Cannot log in? Check your username and password.
- Page not loading? Refresh your browser or check your internet connection.

8 Test Results

8.1 User Acceptance Test

The following table summarizes the results of the User Acceptance Tests conducted for the Object Finder platform. Each test ensures that the system meets the users' expectations and functions as intended.
Test username: yilmazunal
Password : swe573573

Test Case	Description	Expected Result	Status
User Registration	Verify that new users can create an account with valid details.	The system creates a new account and redirects the user to the login page.	Passed
User Login	Ensure users can log in with valid credentials.	The system logs the user in and redirects them to the homepage.	Passed
Post Creation	Verify that users can create posts with details like title, description, and attributes.	Posts appear on the post page and are accessible to others.	Passed
Post Editing	Confirm that users can edit their own posts.	The selected post is edited.	Passed
Search Functionality	Ensure users can search posts using keywords and filters.	Search results match the provided keywords or filter criteria.	Passed
Commenting on Posts	Verify that users can add comments to posts.	Comments appear below the relevant post and are visible to others.	Passed
Voting on Comments	Check if users can upvote or downvote comments.	The vote count updates correctly after an upvote or downvote.	Passed
Profile Management	Verify that users can update their profiles with bio.	Updated profile information appears on the user's profile page.	Passed
Marking Posts as Solved	Confirm that users can mark their posts as "Solved."	Posts are updated with the "Solved" status.	Passed

Table 1: User Acceptance Test Results

8.1.1 Summary

- Total Test Cases: 10
- Tests Passed: 10
- Tests Failed: 0
- Pass Rate: 100%

All user acceptance tests were successful, confirming that the platform meets user expectations and performs as required.

8.2 Unit Test Results

The following unit tests were conducted to validate the core functionalities of the Object Finder platform. These tests cover user creation, post creation, comment creation, and the search functionality, ensuring the system behaves as expected.

Test Case	Description	Result
Create Test User	Tests if a test user can be created in the system with valid credentials.	Passed
Create Test Post	Tests if a post can be created by the test user with valid details, including attributes like material, color, and shape.	Passed
Create Test Comment	Tests if a comment can be added to the created post by the test user.	Passed
Search in Comments	Tests if posts can be found by searching for keywords in comments. The search should return posts related to the specified keyword in their comments.	Passed
Search in Posts	Tests if posts can be found by searching for keywords in their content. The search should return posts containing the specified keyword.	Passed
Empty Search	Tests how the system behaves when the search query is empty. It ensures no errors occur and the page loads successfully.	Passed
Combined Search	Tests if the system can search both posts and comments for a given keyword and return combined results.	Passed

Table 2: Unit Test Results

8.3 Unit Test Details

8.3.1 Test 1: Create Test User

- Description: A test user is created with the username ‘testuser‘ and password ‘12345‘.
- Expected Result: The user is successfully created and saved in the database.
- Actual Result: Passed.

8.3.2 Test 2: Create Test Post

- Description: A post is created by the test user with the title ‘Test Post‘ and attributes like material (‘metal‘), color (‘blue‘), and shape (‘round‘).
- Expected Result: The post is successfully created and associated with the test user.
- Actual Result: Passed.

8.3.3 Test 3: Create Test Comment

- Description: A comment is added to the test post by the test user with the content ‘Test comment with specific keyword‘.
- Expected Result: The comment is successfully created and linked to the post and user.
- Actual Result: Passed.

8.3.4 Test 4: Search in Comments

- Description: Tests if posts can be found by searching for keywords in comments.
- Expected Result: The post linked to the comment is displayed in the search results.
- Actual Result: Passed.

8.3.5 Test 5: Search in Posts

- Description: Tests if posts can be found by searching for keywords in their content.
- Expected Result: Posts containing the specified keywords are displayed in the search results.
- Actual Result: Passed.

8.3.6 Test 6: Empty Search

- Description: Tests how the system behaves when the search query is empty.
- Expected Result: The page loads successfully without errors.
- Actual Result: Passed.

8.3.7 Test 7: Combined Search

- Description: Tests if the system can search in both posts and comments for a given keyword.
- Expected Result: Both posts and comments matching the keyword are displayed in the search results.
- Actual Result: Passed.

8.4 Summary

- Total Test Cases: 7
- Tests Passed: 7
- Tests Failed: 0
- Pass Rate: 100%

All unit tests were successfully executed, verifying that the core functionalities of user creation, post creation, comment creation, and search work as expected.

References

[1] Simon Bennett, Steve McRobb, and Ray Farmer. *Object-oriented systems analysis and design using UML*. McGraw Hill Higher Education, 2005.

[2] Plantuml developers. Plantuml. https://plantuml.com, 2024.

[3] PyGraphviz developers. Pygraphviz. https://pygraphviz.github.io, 2024.