SWE 573 SOFTWARE DEVELOPMENT PRACTICE

Fall, 2024

Project Final Deliverables

Name: Erkin Gönültaş
Student ID: 2022719228
Instructor: Asst. Prof. Suzan Üsküdarlı
Project Name: Lydian
Deployment URL: https://swe-573-erkin-gonultas.onrender.com
Git Repository URL: https://github.com/erkingonultas/SWE-573-Erkin-Gonultas
Git Tag Version URL: https://github.com/erkingonultas/SWE-573-Erkin-Gonultas/releases/tag/v0.9

HONOR CODE
Related to the submission of all the project deliverables for the Swe573 Fall 2024 semester project reported in this report, I, Erkin Gönültaş 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.
Erkin Gönültaş

Table of Contents

Overview

The Lydian project, developed as part of the SWE 573: Software Development Practices course, provided an invaluable opportunity to explore and master modern backend development practices. As a frontend and mobile developer, this project marked a significant step in expanding my expertise into backend technologies and full-stack application development.

Lessons Learned

Backend Development Fundamentals:

Working with Django and PostgreSQL offered hands-on experience with designing and implementing backend architectures, managing databases, and creating APIs. This deepened my understanding of how frontend applications communicate with server-side logic.

Containerization with Docker:

Learning to containerize the application enhanced my understanding of consistent deployment environments, making the development process more efficient and less error-prone.

Cloud Integration:

Integrating Firebase for media handling and deploying the project on Render taught me how to work with cloud services, improving scalability and optimizing performance.

Full-Stack Workflow:

Balancing frontend expertise with backend development provided a more holistic perspective on full-stack workflows, improving my ability to collaborate on cross-functional teams.

Growth Achieved

Broadened Skill Set: Transitioning from a predominantly frontend/mobile background, I gained confidence in backend development, database management, and deployment strategies.

Problem-Solving: The project encouraged critical thinking to overcome technical challenges, such as containerization issues, database optimization, and cloud integration.

Software Engineering Practices: Applying best practices in version control, agile methodologies, and clean coding principles reinforced my understanding of professional-grade software development.

Software Requirements Specification

1. User Accounts & Authentication

Requirement 1.1: Users must be able to create accounts using email, or anonymous guest login.

Requirement 1.2: Users must be able to log in, log out, and manage their profiles, including viewing posts and comments history.

Requirement 1.3: Account security features should include password reset, and privacy settings for personal information.

2. Posting & Object Submission

Requirement 2.1: Users must be able to create posts to upload images and descriptions of objects they want help identifying.

Requirement 2.2: Each post must allow users to provide a title, a detailed description, and tags (e.g., material, time period, location).

Requirement 2.3: Posts should allow up to 10 images (with image resizing and compression) and optional video support (via URL link or native upload).

Requirement 2.4: Users should have the ability to edit and delete their posts.

3. Comments & Discussion

Requirement 3.1: Users must be able to comment on posts, allowing for threaded discussions.

Requirement 3.2: Commenters should be able to include text, links, and images in their responses to help identify objects.

Requirement 3.3: Users must be able to upvote or downvote comments, with the highest-voted comments displayed at the top.

Requirement 3.4: Post authors should be able to mark a comment as the "best answer" to signify a resolved post.

4. Search & Tagging System

Requirement 4.1: Users must be able to search for objects based on keywords, tags, time periods, locations, and materials.

Requirement 4.2: Posts should include a tagging system (e.g., “metal tools,” “19th century,” “ceramics”) to help users categorize objects.

Requirement 4.3: Users should be able to filter posts based on tags, categories, and popularity (e.g., most discussed, most recent).

5. Expert Verification & Moderation

Requirement 5.1: Trusted users (experts and moderators) must have the ability to verify identifications and mark comments as “verified.”

Requirement 5.2: Moderators should have administrative access to delete inappropriate content, ban users, and manage community guidelines.

Requirement 5.3: An automated system should flag offensive or inappropriate language for review by moderators.

6. User Profiles & Contributions

Requirement 6.1: Each user profile should display their total posts, comments, and contributions to the community (e.g., upvotes received, verified identifications).

Requirement 6.2: Users should be able to earn badges or recognition based on contributions (e.g., "Top Contributor," "Artifact Expert").

Requirement 6.3: Profiles should allow users to track their favorite posts, follow other users, and receive notifications about activity on their posts.

7. Browsing & Content Discovery

Requirement 7.1: Users must be able to browse posts via multiple views, including "New Posts," "Most Upvoted," "Trending," and "Verified Identifications."

Requirement 7.2: A featured section should highlight popular or rare artifact discoveries, as well as expert-written articles.

Requirement 7.3: The site should feature a recommendations system that suggests similar posts based on user behavior (e.g., past searches or interactions).

8. Notifications & Alerts

Requirement 8.1: Users must receive notifications for new comments, upvotes, and when their posts receive answers or are marked as resolved.

Requirement 8.2: Users should be able to manage their notification preferences (email, SMS, push notifications).

9. Data Privacy & Security

Requirement 9.1: All user data must be securely stored, with encryption for sensitive information (e.g., passwords, personal data).

Requirement 9.2: Users should have the ability to delete their account and associated data at any time.

Requirement 9.3: The platform must comply with data protection regulations, such as GDPR, ensuring users' privacy rights are respected.

10. Content Reporting & Feedback

Requirement 10.1: Users must be able to report inappropriate content, spam, or abusive behavior for review by moderators.

Requirement 10.2: A feedback system should be in place for users to suggest new features, report bugs, or share their experience with the platform.

Status of the Project

Feature	Overall Status	Details
Registration	Done	-
Login	Done	-
User Profile	Partially done	Users have a detailed profile model but don't have a page to display it.
Authorization	Done	
Supported Object Attributes	Done	Weight attribute could be like size attribute.
Sharing	Done	Users can share objects and delete their own contents.
Comments & Discussion	Done	
Basic Search	Done	
Advanced Search	Done	
Feed	Done	
Wikidata Tag Implementations	Done	
Expert Verification & Moderation	Not Available	
Data Privacy & Security	Done	User data is stored on a Postgresql database on the cloud that can be accessed only through cryptic keys.
Notifications	Not Available	Due to time constraints this feature is skipped.

Status of Deployment

The application is deployed on Render. The database is a Postgresql that is being served on Render. The media files are being served with a Firebase Storage. All of the images stored there. The Dockerization is handled by the server with the build commands.

Installation Instructions

To run locally:

- Make sure Docker Desktop is installed on your system.
- On your favorite terminal, get in the folder path named “LydianAPI”.
- In this path, run “docker compose up —build”.
- Wait until the process is complete. The app should be up and running.

Note: There can be an environment data error while setup process. If it occurs, please contact.

Test Account; username: “NathanDrake”, password: “qwerty123*”

User Guide

1. Authentication

Sign Up

1. Go to the sign-up page.
2. Enter your username, and password.
3. Click "Sign Up" to create an account.

Log In

1. Go to the log-in page.
2. Enter your username and password.
3. Click "Log In" to access the system.

Log Out

1. Click on the "Log Out" button in the navigation bar to log out.

2. Creating a Post

1. Navigate to the "Create Post" page.
2. Fill out the form:
3. Title and Description (required).
4. Upload an image (required).
5. Select attributes like Color, Shape, Material, etc., from the dropdown menus.
6. Enter Size (Height, Length, Depth) and select a unit (cm or inch).
7. To tag the post, search in Wikidata using the search bar below.

8. Click "Submit" to create the post.

3. Viewing Posts

1. The home page displays a list of the latest posts.
2. Click on a post to view its full details, including all attributes and comments.

4. Managing Posts

Deleting a Post

1. Post owners and admins can delete posts by clicking the "Delete" button on the post detail page.

5. Commenting

1. Navigate to the post detail page.
2. Add a comment in the comment form and submit.
3. Post owners and admins can delete comments.
4. Post owners can mark a comment as "resolved" to indicate the post's issue is resolved.

6. Tagging Posts

1. While creating or editing a post, search for tags via the Wikidata API.
2. Select a tag from the search results to associate it with the post.

7. Searching Posts

1. Click the "Search" button to open the search modal.
2. Use the dropdown fields to filter posts by attributes.
3. Submit the form to view search results.

Administrative Features

Post Deletion: Admins can delete any post.

Comment Deletion: Admins can delete any comment.

Testing

User Testing and Unit Tests

Due to time constraints, unit tests were not implemented for this project. Instead, user testing was conducted to ensure the application's functionality and usability. The testing process included:

Functionality Testing:

Verified that users could sign up, log in, and log out. Tested post creation, including adding images and attributes. Ensured the commenting system worked as expected, including deleting and resolving comments. Confirmed that search functionality provided accurate results based on selected attributes.

Usability Testing:

Evaluated the user interface for ease of navigation and clarity. Gathered feedback from testers (my brother) to identify and address any user experience issues. While comprehensive automated testing was not performed, the user testing provided sufficient validation for the core features. Future iterations of the project should include unit tests to ensure long-term reliability and maintainability.

References

1. “How to Deploy a Django App and Postgres Database to Render”, Pretty Printed, retrieved from https://www.youtube.com/watch?v=wczWm8j4v9w in 20 December 2024.
2. Render Documentation, retrieved from https://render.com/docs in 20 December 2024.
3. Django Documentation, retrieved from https://docs.djangoproject.com/en/5.1/ in 20 December 2024.