Universidad de La Laguna

Final Report to the project ‘Social medial website for pets’

WEB SYSTEMS AND TECHNOLOGIES

Petrov Oleh	alu0101688916@ull.edu.es
Rudenko Valeriia	alu0101688923@ull.edu.es

Context

1. Introduction

1.1 Brief Overview of the Project

In a world where pets are cherished members of our families, our Pet Social Media Platform is designed to be a dedicated space for pet lovers to connect, share delightful stories, and showcase adorable pictures of their furry companions.

1.2 Project Purpose

The primary purpose of this project is to establish a specialized social media platform focused on pets. From cute cat photos to heartwarming dog stories, our platform provides a tailored experience for pet owners to connect, share, and appreciate the joy that pets bring to their lives.

1.3 Project Goals

2. Pet-Centric Content: Create a platform where users can share images, stories, and experiences related to their pets.
3. Community Building: Foster a sense of community among pet lovers, encouraging interactions, and connections.
4. User-Friendly Experience: Design an intuitive and visually appealing interface to enhance user engagement.
5. Scalability: Ensure the platform is scalable to accommodate a growing user base and increased content.
6. Innovation: Integrate features that cater specifically to the pet-loving audience, keeping the platform fresh and engaging.

2. Methodology

In our project development methodology, we employed Pivotal Tracker as our primary project management and collaboration tool. Pivotal Tracker, designed to align with agile methodologies such as Scrum and Kanban, facilitated our iterative and incremental approach to software development. The tool allowed us to break down features into user stories, manage a backlog, and plan work in iterations or sprints.

Throughout the project, we utilized Pivotal Tracker to create tasks that needed completion. As we progressed, we marked tasks as "Finished" to signify completion, providing a clear overview of our accomplishments. This approach offered transparency into our progress and encouraged collaborative work.

It is worth noting that, in the past, our team had utilized Trello for project management. However, we found Pivotal Tracker to be more aligned with agile methodologies and provided features that supported our development workflow effectively. Despite the advantages of Pivotal Tracker, we encountered challenges in fully transitioning from Trello. There were instances where we inadvertently overlooked updating tasks in Pivotal Tracker, leading to occasional gaps in task tracking.

To mitigate this challenge, we occasionally resorted to holding meetings to discuss project updates instead of relying solely on Pivotal Tracker. However, we recognized the importance of diligently utilizing the tool to maintain a comprehensive and accurate record of our progress.

It is crucial to acknowledge that our use of Pivotal Tracker was not solely a matter of preference; it was a requirement assigned by our teacher. Despite occasional challenges, leveraging Pivotal Tracker proved beneficial for maintaining project transparency, fostering collaboration, and adhering to agile development principles.

3. Project technologies and application architecture

Our architecture consists of two main components: the frontend and the backend.

3.1 Frontend: React

The frontend is developed using React, providing a responsive and dynamic user interface. The decision to use React for the frontend was driven by its component-based architecture, which facilitates modularity and reusability. React's virtual DOM ensures efficient rendering, providing a seamless user experience. Additionally, the vibrant React ecosystem and community support make it an excellent choice for building interactive and engaging user interfaces.

3.2 Backend: Node.js and Express.js

Node.js was selected for the backend due to its non-blocking, event-driven architecture, making it well-suited for handling a large number of concurrent connections. Its use of JavaScript as the programming language allows for code reuse between the frontend and backend, streamlining development. Node.js, coupled with Express.js, provides a robust and scalable server foundation.

3.3 Database: MongoDB

MongoDB serves as the database for our project, offering a flexible and scalable document-oriented data model. This NoSQL database is suitable for storing diverse data types, making it ideal for handling the variety of information associated with pet stories, images, and user profiles.

3.4 External Libraries and Dependencies

- dotenv: Manages environment variables for secure configuration.
- jsonwebtoken: Facilitates secure user authentication with JWT.
- axios: Handles HTTP requests, particularly for downloading pet images.

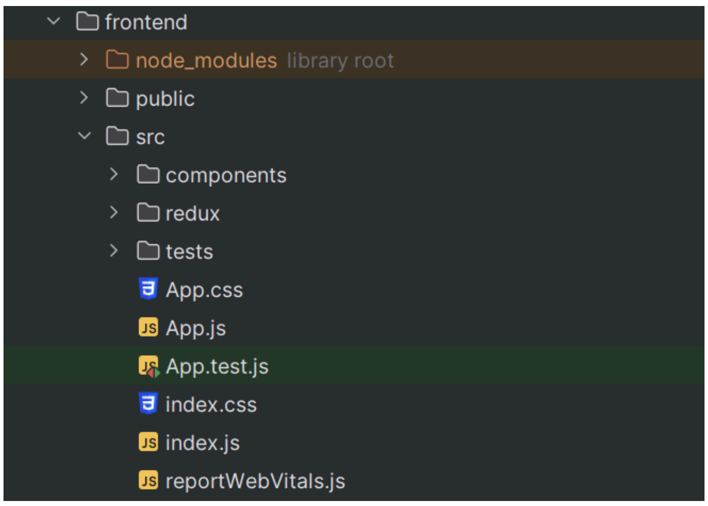
3.5 Project Structure

The project is organized into the following main directories:

- Frontend
    - components: Contains React components for different pages and features.

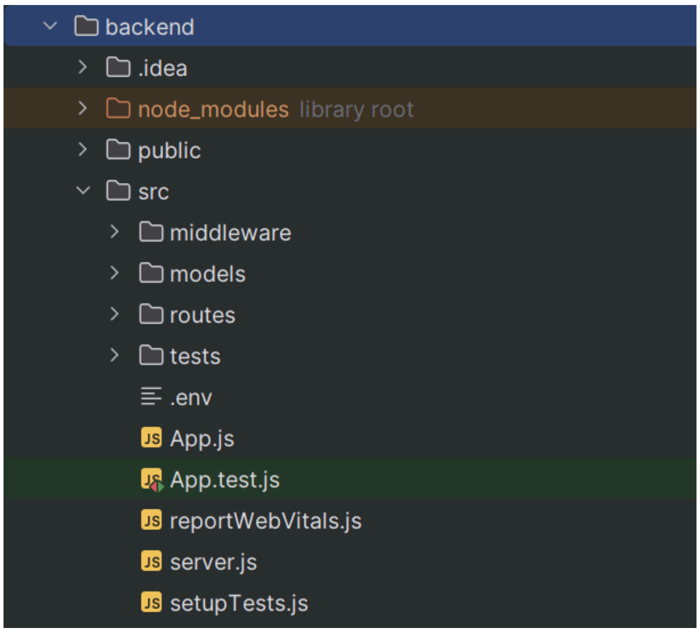
- styles: Houses styling files, ensuring a consistent and visually appealing design.

- Backend
    - routes: Defines the API routes for handling frontend requests.
    - models: Includes MongoDB schemas for user data, pet data, etc.
    - controllers: Manages the business logic for various functionalities.
- Other
    - config: Stores configuration files, including MongoDB connection settings and JWT secret.



```
frontend
  node_modules library root
  public
  src
    components
    redux
    tests
    App.css
    App.js
    App.test.js
    index.css
    index.js
    reportWebVitals.js
```

Figure 1. Frontend structure



```
backend
  .idea
  node_modules library root
  public
  src
    middleware
    models
    routes
    tests
    .env
    App.js
    App.test.js
    reportWebVitals.js
    server.js
    setupTests.js
```

Figure 2. Backend structure

4. Client-Server Interaction

The project involves a client-server architecture, where the client represents the React-based frontend, and the server comprises the Node.js backend. This architecture facilitates a seamless interaction between the user interface and the underlying database. The client, responsible for presenting the user interface and handling user actions, communicates with the server through a set of well-defined APIs.

Upon loading the application, the client makes requests to the server to retrieve necessary data, such as user profiles, posts, and additional content. These requests are processed by the server, which interacts with the MongoDB database to fetch or store relevant information. The client and server communicate asynchronously, allowing for a dynamic and responsive user experience.

The server, implemented with Express.js, exposes RESTful APIs that the client utilizes to perform various actions, including user authentication, fetching and posting content, and managing user subscriptions. The client, built with React, then updates its state and view based on the data received from the server.

This client-server interaction is crucial for the real-time nature of the application, enabling users to seamlessly share and view pet-related content. Whether posting new additions, exploring profiles, or interacting with the subscription system, the client-server architecture ensures a smooth and efficient flow of information, contributing to an engaging and interactive user experience.

5. Database schema and models

5.1 User Model:

- Fields:
    - **username** (String, required): The username of the user.
    - **email** (String, required, unique): The email address of the user.
    - **password** (String, required): The hashed password of the user.
    - **avatar** (String): The filename of the user's avatar.

5.2 Addition Model:

- Fields:
    - **image** (String): The filename or URL of the image associated with the addition.
    - **label** (String, required): The label or title of the addition.
    - **text** (String, required): The textual content of the addition.

5.3 Comment Model:

- Fields:
    - **text** (String, required): The text content of the comment.
    - **Autor** (ObjectId, ref: 'User', required): The user who authored the comment.

- **post** (ObjectId, ref: 'Post', required): The post to which the comment is associated.

5.4 Pet Model:

- Fields:
    - **name** (String, required): The name of the pet.
    - **age** (Number): The age of the pet.
    - **breed** (String): The breed of the pet.
    - **type** (String): The type of the pet.

5.5 Post Model:

- Fields:
    - **image** (String): The filename or URL of the image associated with the post.
    - **label** (String, required): The label or title of the post.
    - **text** (String, required): The textual content of the post.
    - **author** (ObjectId, ref: 'User', required): The user who authored the post.

5.6 Subscription Model:

- Fields:
    - **subscribed** (ObjectId, ref: 'User', required): The user who is being subscribed to.
    - **subscriber** (ObjectId, ref: 'User', required): The user who is subscribing to another user.

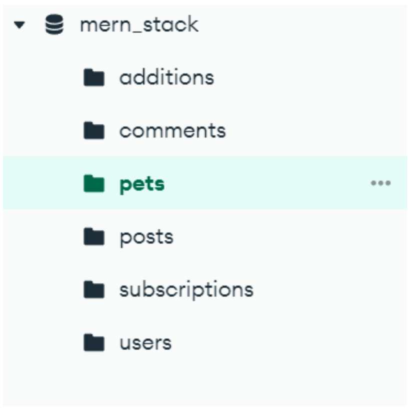


Figure 3. Database structure

5.7 Index Creation

Indexes have been created for each model to optimize database performance. The indexes ensure efficient retrieval and querying of data.

5.8 Data Generation

A script has been implemented to generate sample data for the application. This includes the creation of users, additions, pets, posts, and subscriptions. User avatars and image files associated with additions, pets, and posts are dynamically downloaded from LoremFlickr using the faker library.

5.9 Notes:

- Images for users, additions, pets, and posts are downloaded and stored in the backend/public/images directory.
- The getRandomUser function is a helper function used to randomly select a user, excluding a specified user if provided.

6. Authentication and Authorization

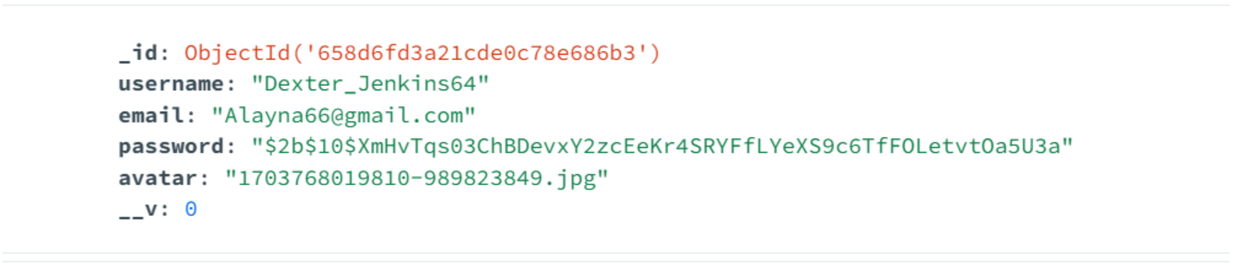
Authentication and authorization play a crucial role in ensuring secure and controlled access to the MERN stack project. The following sections elaborate on how these processes are implemented:

Authentication:

Authentication is the process of verifying the identity of users. In the MERN stack project, user authentication is achieved using a combination of email and password.

6.1 User Registration:

- When a user registers, their password is securely hashed using bcrypt before being stored in the database.
- A unique avatar image associated with each user is dynamically downloaded from LoremFlickr during the registration process.



```
_id: ObjectId('658d6fd3a21cde0c78e686b3')
username: "Dexter_Jenkins64"
email: "Alayna66@gmail.com"
password: "$2b$10$XmHvTqs03ChBDevxY2zcEeKr4SRYFfLYeXS9c6TfFOLetvtOa5U3a"
avatar: "1703768019810-989823849.jpg"
__v: 0
```

Figure 4. Hashed password

6.2 User Login:

- Users can log in by providing their email and password.
- Passwords are verified by comparing the hashed input with the stored hash.

6.3 JSON Web Tokens (JWT):

- Upon successful login, a JSON Web Token (JWT) is generated and sent to the client.

- This token is used to authenticate and authorize subsequent requests.

6.4 Authorization:

Authorization ensures that authenticated users have the appropriate permissions to access specific resources or perform certain actions.

6.5 Middleware for Authentication:

- Middleware functions are implemented to check the presence and validity of JWTs in incoming requests.
- If a valid token is found, the associated user information is added to the request object.

6.6 Authorization Checks:

- Authorization checks are implemented at various endpoints to verify if the user has the necessary permissions.
- For example, a user can only delete their posts, and only authenticated users can access certain routes.

6.7 Protected Routes:

- Certain routes, such as those related to user profiles or creating posts, are protected and require a valid JWT for access.

6.8 User Roles:

- User roles can be implemented to differentiate between regular users and administrators.
- Administrators might have additional privileges, such as managing user accounts or posts.

7. Security Considerations:

7.1 Secure Password Handling:

- Passwords are securely hashed using bcrypt, ensuring that even if the database is compromised, user passwords remain protected.

7.2 JWT Expiration:

- JWTs have a defined expiration time to enhance security. Users need to re-authenticate after the token expires.

7.3 HTTPS Usage:

- The use of HTTPS ensures secure communication between the client and the server, preventing data interception.

8. Pages overview

8.1 Home page

The Home page (Figure 5) is the central hub where users can discover and engage with popular posts within the community. It provides a visually appealing and organized layout to showcase a curated selection of posts, fostering community interaction.

Key Features:

Popular Posts Display:

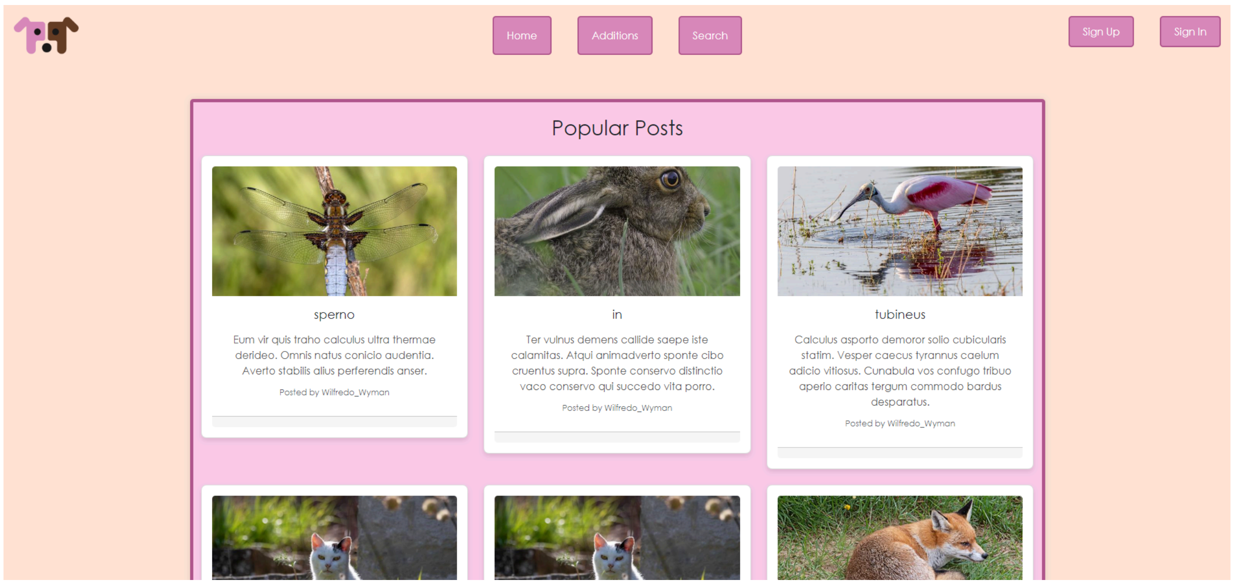
- The Home page prominently showcases popular posts, each presented in a card format.
- Posts include images, labels, and text descriptions, creating an engaging visual experience.

Author Information:

- Usernames of post authors are displayed, providing attribution and a sense of community connection.

Comments Section:

- Users can view comments associated with each post directly on the Home page.
- Comment sections offer a space for community members to express thoughts and engage in discussions.



Popular Posts

sperno

Eum vir quis traho calculus ultra thermae derideo. Omnis natus conicio audentia. Averto stabilis alius perferendis anser.

Posted by Wilfredo_Wyman

in

Ter vulnus demens callide saepe iste calamitas. Atqui animadverto sponte cibo cruentus supra. Sponte conservo distinctio vaco conservo qui succedo vita porro.

Posted by Wilfredo_Wyman

tubineus

Calculus asporto demoror solio cubicularis statim. Vesper caecus tyrannus caelum adicio vitiosus. Cunabula vos confugo tribuo aperio caritas tergum commodo bardus desparatus.

Posted by Wilfredo_Wyman

Figure 5. Home page

8.2 Profile page

The Profile page (Figure 6) provides users with the ability to manage and customize their personal information and avatar within the community. This page acts as a central location for users to view and update details associated with their profile.

Key Features:

User Information Display:

- The page displays essential user information, including full name, age, breed, and pet type.
- Users can upload or change their avatar, providing a personalized touch to their profile.

Edit Mode:

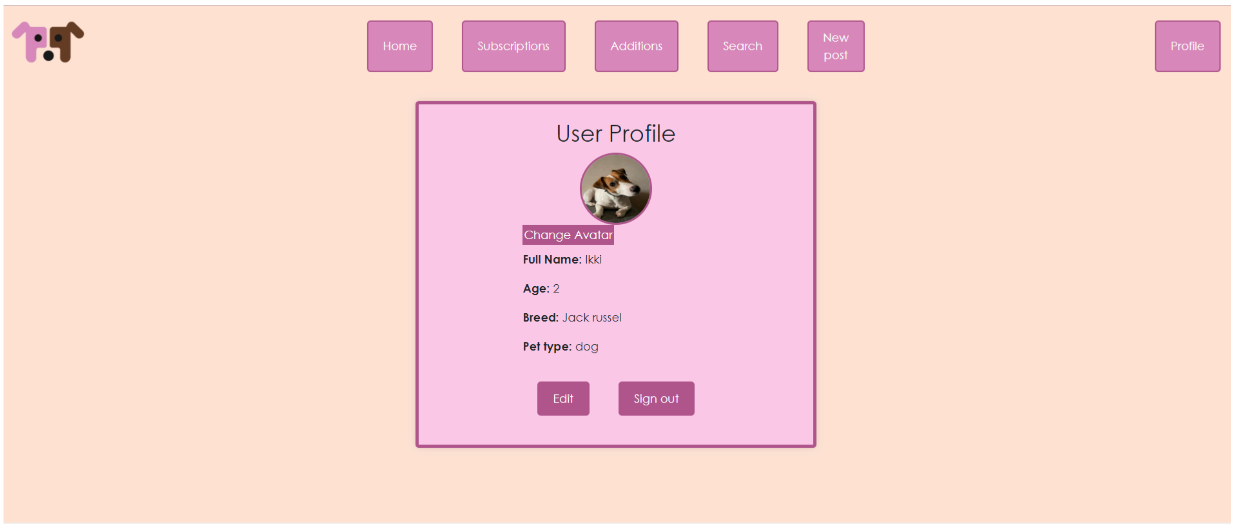
- Users can switch to an edit mode, enabling them to modify their profile information.
- In edit mode, input fields become editable, allowing users to make changes.

Save Changes:

- Users can save their modifications by clicking the "Save" button while in edit mode.
- Save functionality sends the updated data to the server, ensuring profile information is up-to-date.

Sign Out:

- A "Sign Out" button allows users to securely log out from their account.



User Profile

Full Name: Ikki

Age: 2

Breed: Jack russel

Pet type: dog

Figure 6. Profile page

8.3 Search page

The Search page (Figure 7) enhances user interaction by providing a feature to search for other users based on their usernames. Users can easily find and navigate to specific profiles within the community.

Key Features:

User Search Functionality:

- The primary feature of the Search page is to allow users to search for other community members by their usernames.
- Users can enter a nickname in the search input field, triggering real-time filtering of user results.

Real-time Filtering:

- As users type in the search input field, the list of displayed users is dynamically updated to match the entered query.
- The real-time filtering ensures a responsive and intuitive search experience.

Clickable User Results:

- Search results are presented in a list format, with each username displayed as a clickable button.
- Clicking on a username button redirects the user to the profile page of the selected user, providing a quick way to view detailed information.

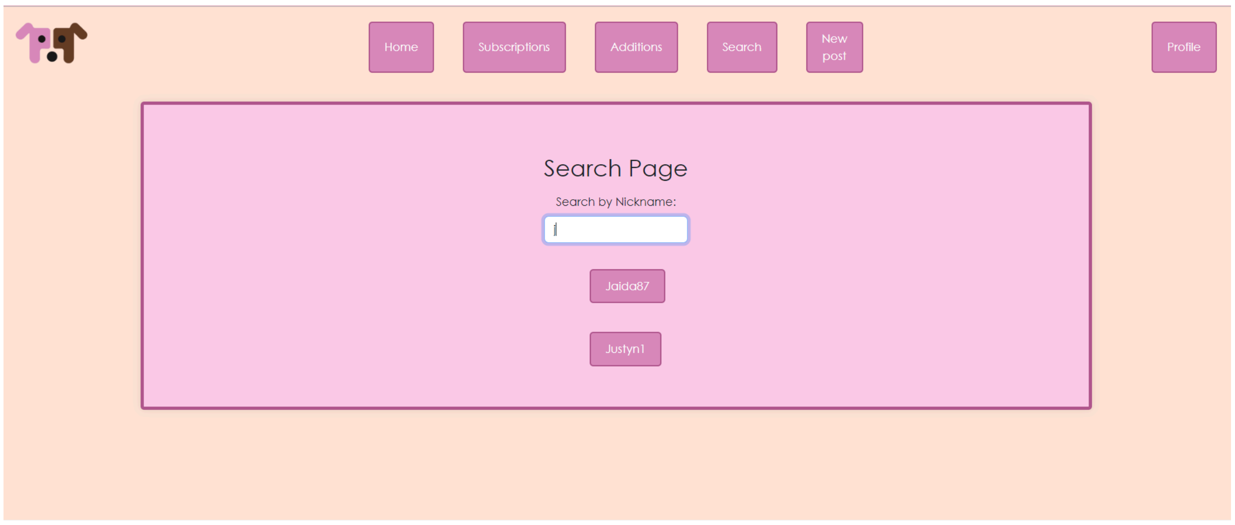


Figure 7. Search page

8.4 User Profile page

The User Profile page (Figure 8) provides detailed information about a specific user within the community. Users can view the profile details, and posts, and choose to follow or unfollow the displayed user.

Key Features:

Profile Information:

- The page displays the user's avatar, name, pet type, breed, and age.
- Users can quickly identify and learn more about the specific user.

Follow/Unfollow Functionality:

- Users can choose to follow or unfollow the displayed user, influencing their subscription status.
- The button dynamically changes based on the user's current subscription status.

User Posts:

- The page showcases a list of posts made by the user, including images, labels, and text.
- Users can explore the content created by the user.

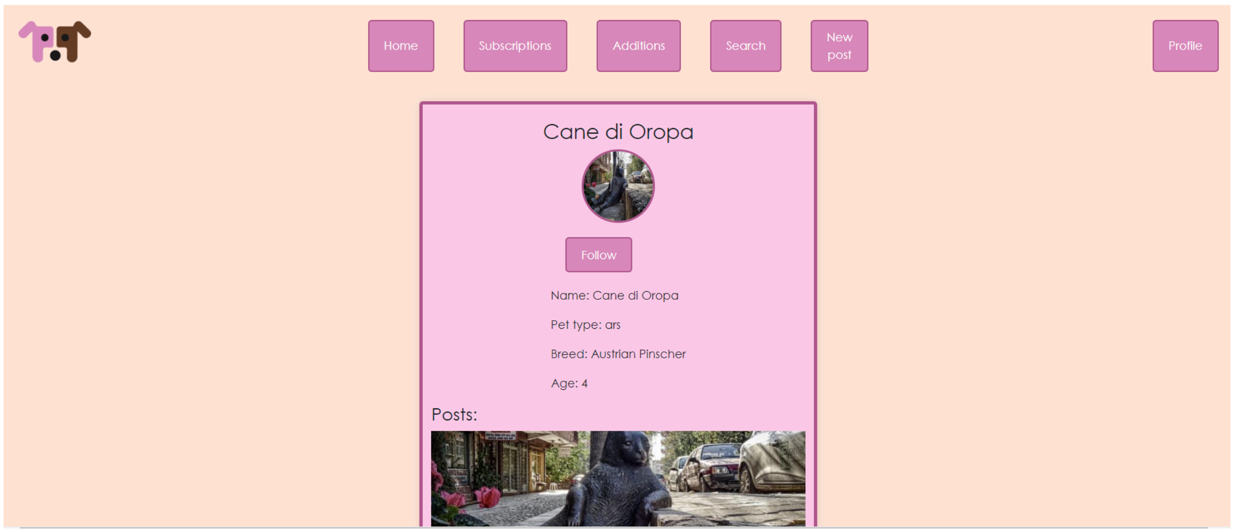
Follow/Unfollow Logic:

Follow Button:

- If the user is not already following the displayed user, the Follow button is visible.
- Clicking the Follow button initiates the follow process and updates the button to Unfollow upon success.

Unfollow Button:

- If the user is already following the displayed user, the Unfollow button is visible.
- Clicking the Unfollow button initiates the unfollow process and updates the button to Follow upon success.



Cane di Oropa

Name: Cane di Oropa

Pet type: ars

Breed: Austrian Pinscher

Age: 4

Figure 8. User profile page

8.5 Subscription Posts Page

The Subscription Posts page (Figure 9) allows users to view and engage with posts from accounts they are subscribed to. Users can explore the content, post comments, and interact with the community.

Key Features:

Subscription Posts Display:

- The page showcases posts from users whom the logged-in user is subscribed to.
- Each post includes an image, label, text, and the author's username.

Commenting Functionality:

- Users can post comments on each subscription post, fostering engagement and community interaction.

Comment Input Section:

- Users can input comments for each post through dedicated input sections.

- Comment input fields are specific to each post, allowing separate interactions for different posts.

Commenting Logic:

Comment Input Fields:

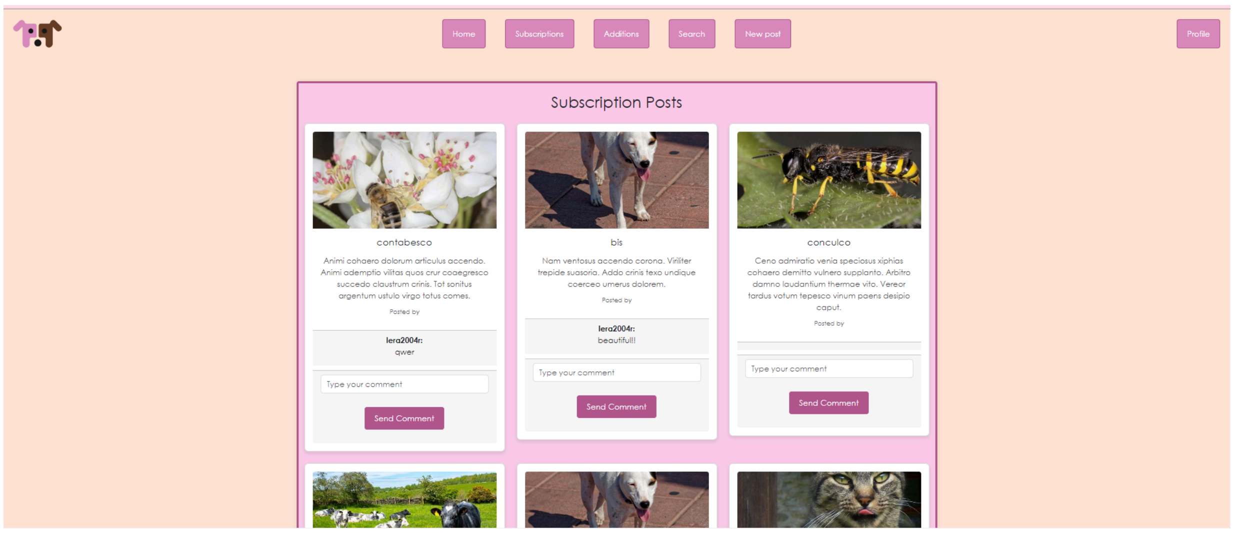
- Dedicated comment input fields are provided for each post.
- Users can type comments specific to the post they are engaging with.

Send Comment Button:

- The "Send Comment" button allows users to submit their comments for a particular post.

Comment Display:

- Comments are displayed below each post, organized by post ID.
- Users can view comments made by others, contributing to a sense of community.



Subscription Posts

contabesco

Animi cohaero dolorum articulus accendo. Animi ademptio vilitas quos crur coaegresco succedo claustrum crinis. Tot sonitus argentum ustulo virgo totus comes.

Posted by

lera2004r:
qwer

bis

Nam ventosus accendo corona. Viriliter trepide suasoria. Addo crinis texo undique coerceo umerus dolorem.

Posted by

lera2004r:
beautiful!!

conculco

Ceno admiratio venia speciosus xiphias cohaero demitto vulnero supplanto. Arbitro damno laudantium thermae vito. Vereor tardus votum tepesco vinum paens desipio caput.

Posted by

Figure 9. Subscription posts

8.6 Additions page

The Additions page (Figure 10) serves as a platform for users to create and share posts related to lost pets or pets seeking a loving family. This feature facilitates community engagement, encouraging users to contribute valuable information about pets in need. Below is an overview of the key components and functionalities of the Additions page:

Key Features:

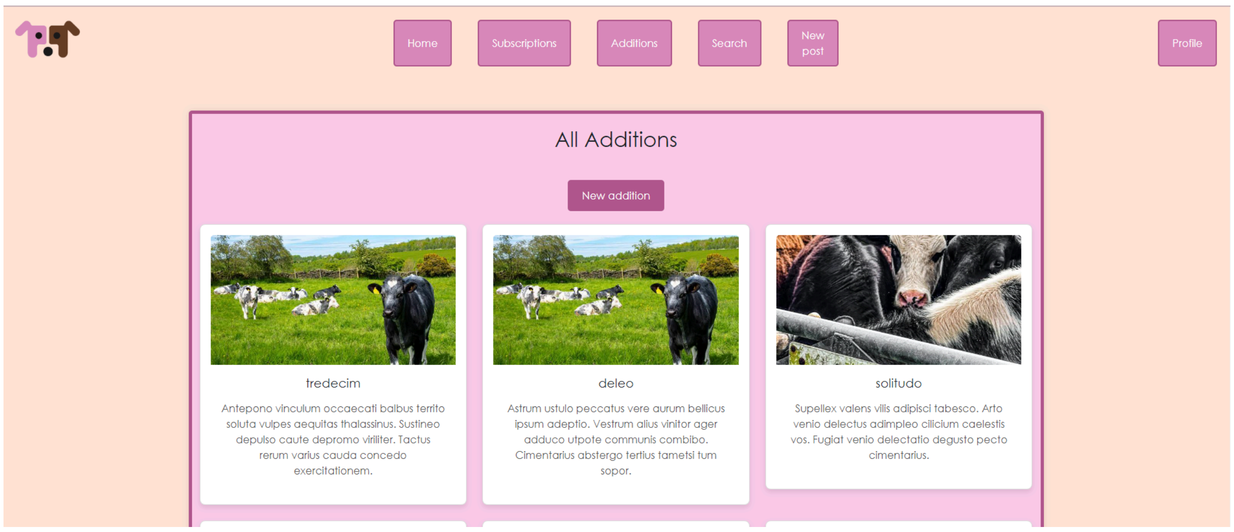
Create New Addition:

- Users can create new additions by providing essential information about a pet.
- This includes uploading an image, adding a label, and providing descriptive text about the pet's situation.

Display of Existing Additions:

- The page displays existing additions created by users in a visually appealing format.

- Each addition card includes an image, label, and descriptive text, allowing users to easily comprehend the information.



tredecim

Antepono vinculum occaecati balbus territo soluta vulpes aequitas thalassinus. Sustineo depulso caute depromo viriliter. Tactus rerum varius cauda concedo exercitationem.

deleo

Astrum ustulo peccatus vere aurum bellicus ipsum adeptio. Vestrum alius vinitor ager adduco utpote communis combibo. Cimentarius abstergo tertius tametsi tum sopor.

solitudo

Supellex valens vilis adipisci tabesco. Arto venio delectus adimpleo cilicium caelestis vos. Fugiat venio delectatio degusto pecto cimentarius.

Figure 10. Additions page

8.7 New Post Creation Page

The New Post page (Figure 11) enables users to create and share new posts about pets. Users can upload an image, provide a label (e.g., pet's name), and add descriptive text to create a post. Below is an overview of the key components and functionalities of the New Post page:

Key Features:

Image Upload:

- Users can upload an image of the pet they want to create a post for. The system accepts only image files.

Label:

- Users can enter a label for the pet, typically the pet's name. This provides a quick identification for viewers.

Text Description:

- A text area allows users to provide additional information or details about the pet. This could include information about the pet's situation, characteristics, or any other relevant details.

Post Creation Button:

- Clicking the "Create Post" button triggers the process of creating and submitting the post.

Feedback Message:

- After attempting to create a post, the page displays a feedback message to inform users about the success or failure of the post-creation process.

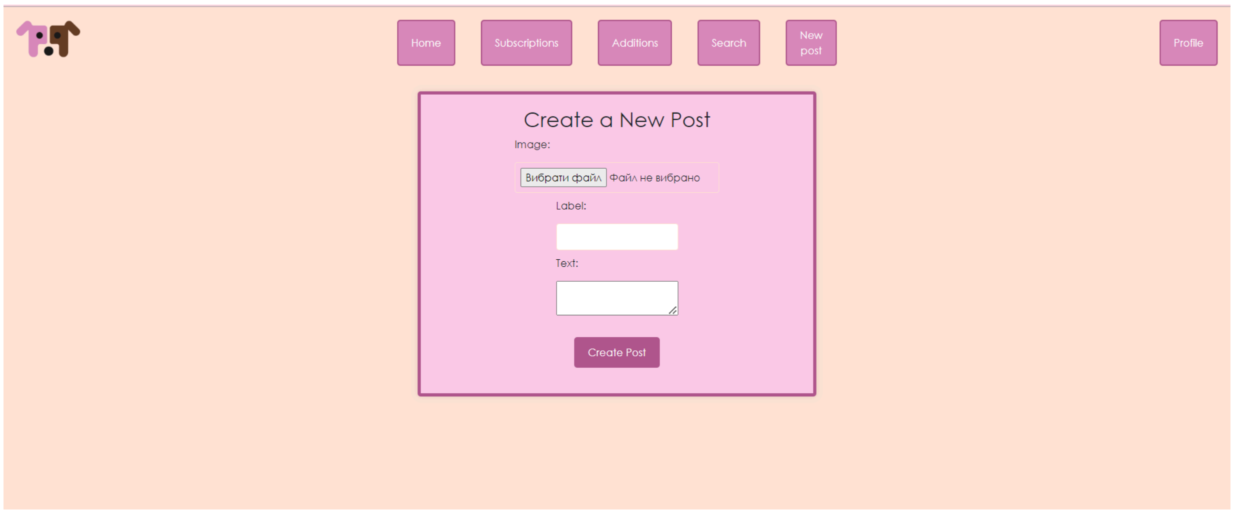


Figure 11. Create post page

8.8 New Addition Creation Page

The Add Addition page (Figure 12) provides users with the ability to contribute new additions related to pets, such as information about lost pets, pets in search of a family, or any other relevant content. Below is an overview of the key components and functionalities of the Add Addition page:

Key Features:

Image Upload:

- Users can upload an image related to the addition they want to create. The system accepts only image files.

Label:

- Users can enter a label for the addition. The label provides a quick identification for viewers and typically includes information about the addition.

Text Description:

- A text area allows users to provide additional information or details about the addition. This could include information about the pet's situation, location, or any other relevant details.

Add Addition Button:

- Clicking the "Add Addition" button triggers the process of creating and submitting the addition.

Feedback Message:

- After attempting to add an addition, the page displays a feedback message to inform users about the success or failure of the addition creation process.

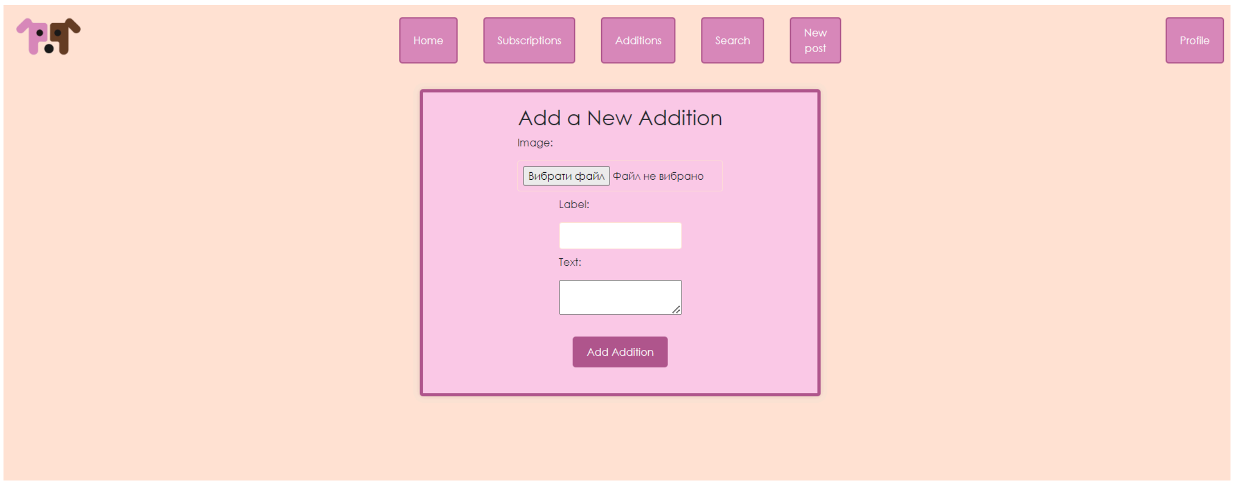


Figure 12. New addition page

8.9 Navigation Bar

The Navigation component(Figure 13) serves as the header or navigation bar for the application. It provides users with quick access to different sections and features of the platform. Below is an overview of the key components and functionalities of the Navigation bar:

Key Features:

Logo:

- The navigation bar features a logo, providing a visual identity for the application.

Navigation Buttons:

- Home Button: Navigates users to the home page.
- Subscriptions Button: Appears if the user is authenticated and the token is valid, leading to the Subscriptions page.
- Additions Button: Navigates users to the Additions page.
- Search Button: Directs users to the Search page.

- New Post Button: Appears if the user is authenticated and the token is valid, leading to the New Post creation page.
- Profile Button: Appears if the user is authenticated and the token is valid, leading to the Profile page.
- Sign Up and Sign In Buttons: Displayed if the user is not authenticated or the token is not valid.

Dynamic Visibility:

- Buttons related to user-specific actions (Subscriptions, New Post, Profile) dynamically appear based on the user's authentication status and the validity of the token.

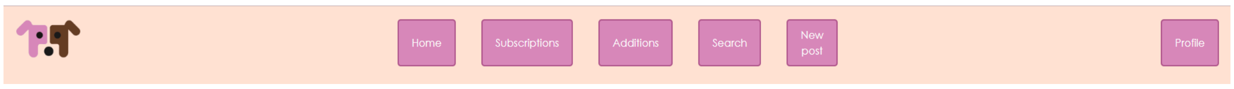


Figure 13. Navigation bar

9. How to run a website

To run the website, follow the steps outlined below. Make sure you have Node.js and MongoDB installed on your system.

Install MongoDB:

If MongoDB is not already installed on your system, download and install it from the official MongoDB website: MongoDB Download

Install Node.js Packages:

Navigate to the project's root directory using the terminal or command prompt and execute the following commands to install the necessary Node.js packages:

npm install

Populate the Database:

Run the script filldatabase.js to populate the MongoDB database with initial data:

node ./filldatabase.js

Install Frontend Dependencies:

Change directory to the 'frontend' folder:

cd frontend

Install frontend dependencies:

npm install

Install Backend Dependencies:

Change directory to the 'backend' folder:

cd ../backend

Install backend dependencies:

npm install

Start the Frontend Server:

While still in the 'frontend' folder, start the frontend development server:

npm start

Start the Backend Server:

Switch back to the 'backend' folder and start the backend server:

cd ../backend

npm start

10. Diagrams

10.1 Sequence diagram

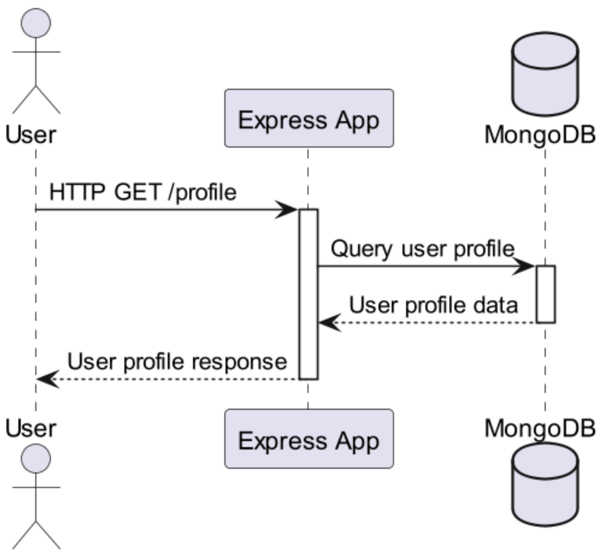


Figure 14. Sequence diagram

The provided sequence diagram (Figure 14) illustrates the interaction between an actor (User), an Express application (App), and a MongoDB database for the scenario of a user requesting their profile information.

Actor/User: Represents an external user interacting with the system.

Express App (App): Represents your Express.js application.

MongoDB: Represents the MongoDB database.

User -> App: HTTP GET /profile: Denotes the user making an HTTP GET request to the '/profile' endpoint of the Express App. This is a request to retrieve the user's profile.

activate App: Activates the App, indicating that it is processing the incoming request.

App -> MongoDB: Query user profile: Shows the App sending a query to MongoDB to retrieve the user's profile information.

activate MongoDB: Activates the MongoDB database, indicating that it is processing the query.

MongoDB --> App: User profile data: Indicates MongoDB sending the user's profile data back to the App.

deactivate MongoDB: Deactivate MongoDB, signifying the completion of the query.

App --> User: User profile response: Shows the App sends the user's profile data as a response to the initial HTTP request.

deactivate App: Deactivate the App, signifying the completion of processing the request.

10.2 Class diagram

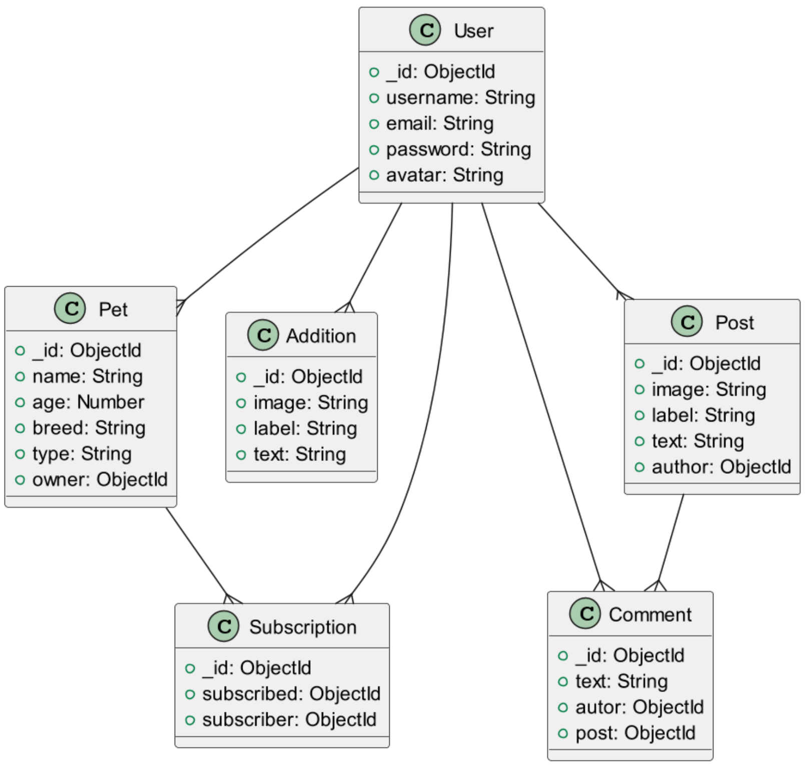


Figure 15. Class diagram

In the presented PlantUML class diagram, the fundamental entities and their relationships within the system are outlined. The classes represent core data structures, and the associations between them depict key interactions. Each class encapsulates attributes relevant to its purpose in the system.

The User class represents a user entity with attributes such as user ID, username, email, password, and avatar. This class serves as a central entity, establishing associations with various other classes. Users can have associated pets, subscriptions, comments, posts, and additions.

The Pet class encapsulates information about pets, including their ID, name, age, breed, type, and the ID of their owner (a User). The relationships show that a User can be associated with multiple pets, and a Subscription links Users and Pets.

The Addition class captures data related to additional content, featuring an ID, image, label, and text. Users can be associated with additions, as indicated by the relationship.

The Comment class represents comments on posts, containing an ID, text, and references to the author (a User) and the post being commented on. Users can be associated with comments, and there's a relationship between Posts and Comments.

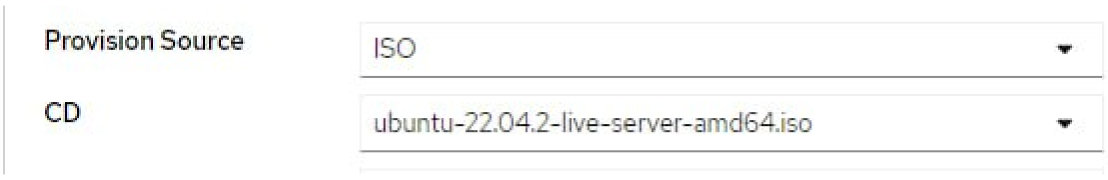
The Post class models posts made by users, including attributes such as ID, image, label, text, and the ID of the authoring user. Users can be associated with posts, and there's a relationship between Posts and Comments.

Lastly, the Subscription class captures the relationships between subscribing and subscribed users. It includes an ID, the ID of the user being subscribed to, and the ID of the subscribing user.

11. Continuous integration/deployment

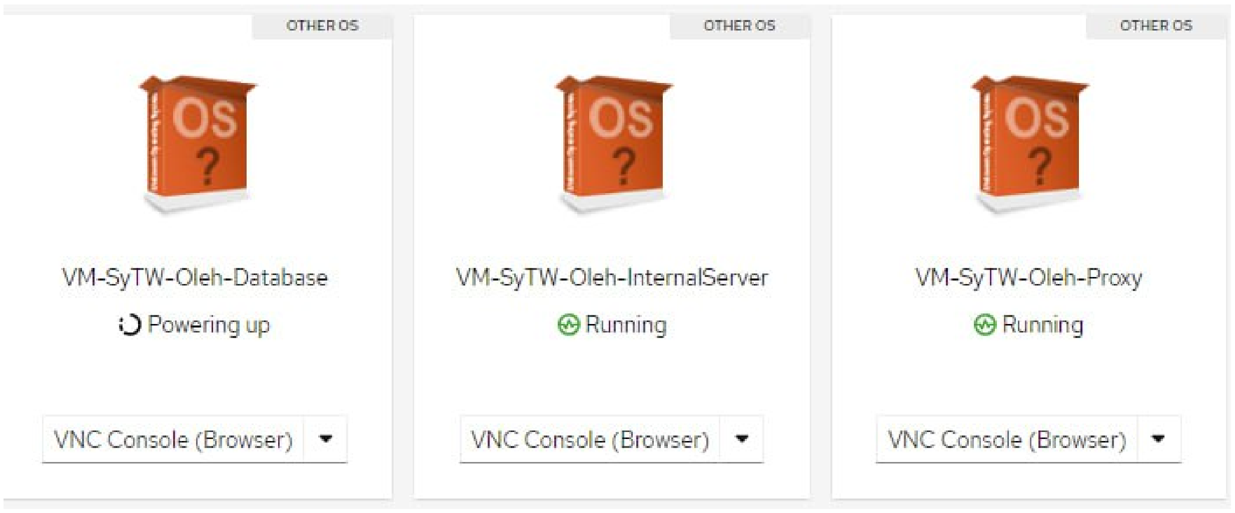
Continuous Integration (CI) and Continuous Deployment (CD) are integral components of our development pipeline, ensuring the seamless and efficient delivery of our website. Leveraging the Infrastructure as a Service (IaaS) provided by ULL, we deployed our system on Ubuntu 22.04.2. To enhance scalability and maintain separation of concerns, we implemented a three-tier architecture, utilizing virtual machines for distinct purposes. The first virtual machine houses our MongoDB database, providing a robust and flexible storage solution for our pet-centric data. The second VM functions as our internal server, hosting the

Node.js backend powered by Express.js, while the third VM acts as a proxy, handling external requests and routing them to the appropriate components. Apache 2 serves as our web server, ensuring reliable communication between the front end and back end. This infrastructure allows for efficient scaling, improved resource management, and secure deployment of our pet-friendly social media platform. The adoption of CD on this infrastructure facilitates the automatic testing, integration, and deployment of updates, ensuring a streamlined development process and consistent delivery of features to our users.



Provision Source	ISO
CD	ubuntu-22.04.2-live-server-amd64.iso

Figure 16. Vm



OTHER OS	OTHER OS	OTHER OS
VM-SyTW-Oleh-Database	VM-SyTW-Oleh-InternalServer	VM-SyTW-Oleh-Proxy
Powering up	Running	Running
VNC Console (Browser)	VNC Console (Browser)	VNC Console (Browser)

Figure 17. Created virtual machines

12. Testing

Our comprehensive testing strategy for authentication involved using Supertest and Chai to evaluate the functionality of our user authentication system rigorously. The set of tests covered various scenarios to ensure the system's reliability, security, and user-friendliness.

1. Sign Up a New User:

- Created a new user account to validate the successful registration process.
- Verified the response included a token for subsequent authentication.

2. Avoid Duplicate Email Sign Up:

- Tested prevention of duplicate email registrations.
- Verified the system correctly responded with a 400 status and an appropriate error message.

3. Sign In an Existing User:

- Authenticated an existing user to ensure a seamless login process.
- Checked for a successful response containing a valid token.

4. Avoid Incorrect Sign-In:

- Tested the system's ability to reject login attempts with incorrect credentials.
- Validated the expected 401 status and an error message for incorrect email or password.

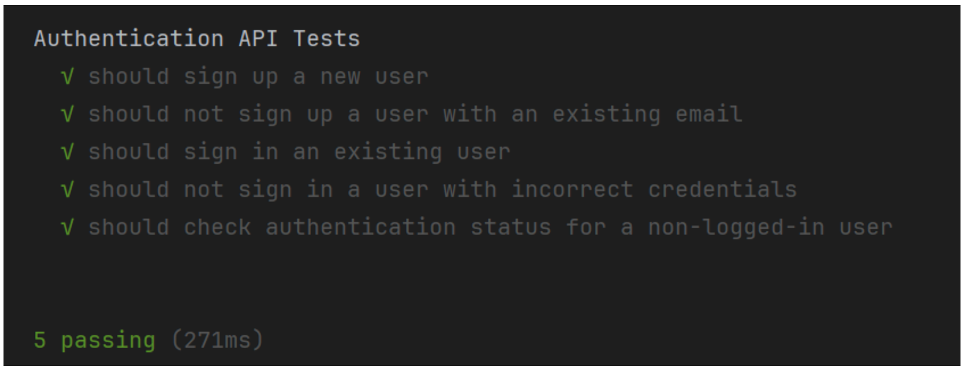
5. Check Authentication Status for Non-Logged-In User:

- Verified the system's response when checking authentication status without logging in.
- Confirmed the expected 401 status, 'false' for authentication, and a null user.

6. Post-Test Cleanup:

- Executed essential cleanup tasks after testing, ensuring the removal of test user data.
- Maintained the integrity and efficiency of the system post-testing.

These tests collectively fortified our authentication system, contributing to a secure and reliable user experience on our platform.



```
Authentication API Tests
  √ should sign up a new user
  √ should not sign up a user with an existing email
  √ should sign in an existing user
  √ should not sign in a user with incorrect credentials
  √ should check authentication status for a non-logged-in user


5 passing (271ms)
```

Figure 18. Tests

13. Conclusion:

In concluding our journey in developing the Social Media Website for Pets, we reflect on the collaborative effort and dedication that went into creating a platform that celebrates the joy of pets. The project's overarching goal was to provide a tailored space for pet enthusiasts to connect, share, and build a vibrant community. Through an iterative and agile development approach, we successfully implemented a feature-rich website with functionalities ranging from user profiles to dynamic content creation. Utilizing Pivotal Tracker for project management ensured transparency and alignment with agile principles. Our chosen technologies, including React for the front end, Node.js for the backend, and MongoDB for the database, laid a robust foundation for scalability and responsiveness.

14. Improvements:

While proud of our accomplishments, we acknowledge areas for improvement. The transition from Trello to Pivotal Tracker posed occasional challenges in task tracking, leading to supplementary meetings for updates. In future projects, we would explore enhancing task-tracking mechanisms to avoid lapses in communication. Additionally, the deployment on ULL's IaaS demonstrated scalability but could benefit from optimizations in resource allocation for enhanced efficiency. Further integration of automated testing in our CI/CD pipeline would fortify the stability of our platform. User feedback and analytics will guide future refinements, ensuring our Social Media Website for Pets continues to evolve into a thriving community hub. The commitment to continuous improvement remains a cornerstone, driving us to refine, optimize, and expand the platform in response to user needs and technological advancements.

Time estimations

Task	Estimated time	Real-time
Project organization	2	4
HTML code for pages	4	6
Css code for pages	3	2
NavBar implementation	2	2
Database creation and configuration	4	4
Routes setup and configuration	8	7
Server creation	3	3
Implementing authorization	4	6
Password hashing and JWT tokens	4	6
Debugging	12	10
Refactoring	5	10
Implementing all functionality	20	50

Reports (Moodle tasks)	7	12
Models implementation	5	7
Final report/documentation	2	5
Total code	76	111
TOTAL	85	128

Figure 16. estimated time against real time

It is the number of hours that we spent on these tasks, but the implementation of the full site took longer.

References

MDN Web Docs - https://developer.mozilla.org/

- Comprehensive documentation on web technologies, including HTML, CSS, and JavaScript.

W3Schools - https://www.w3schools.com/

- Tutorials and references for web development technologies.

Stack Overflow - https://stackoverflow.com/

- Community-driven platform for asking and answering programming-related questions.

Node.js and Express:

Node.js Official Documentation - https://nodejs.org/en/docs/

- Documentation for Node.js, the JavaScript runtime used for server-side development.

Express.js Official Documentation - https://expressjs.com/

- Documentation for the Express.js web application framework.

MongoDB:

MongoDB Documentation - https://docs.mongodb.com/

- Official documentation for MongoDB, the NoSQL database used in the project.

React:

React Documentation - https://reactjs.org/docs/getting-started.html

- Official documentation for React, the JavaScript library used for building user interfaces.

Authentication and Authorization:

JWT (JSON Web Tokens) Documentation - https://jwt.io/introduction/

- Information on JSON Web Tokens, commonly used for authentication.

Additional:

Multer Documentation - https://www.npmjs.com/package/multer

- Documentation for Multer, a middleware for handling multipart/form-data (used for file uploads).

Axios Documentation - https://axios-http.com/docs/intro

- Documentation for Axios, a promise-based HTTP client (commonly used in React applications).

Mongoose Documentation - https://mongoosejs.com/docs/

- Documentation for Mongoose, an ODM (Object Data Modeling) library for MongoDB and Node.js.