

FINAL PROJECT MILESTONE SUMMARY

**Ujwal Chandrashekhar
Shreyas Pogal Naveen**

Group 2

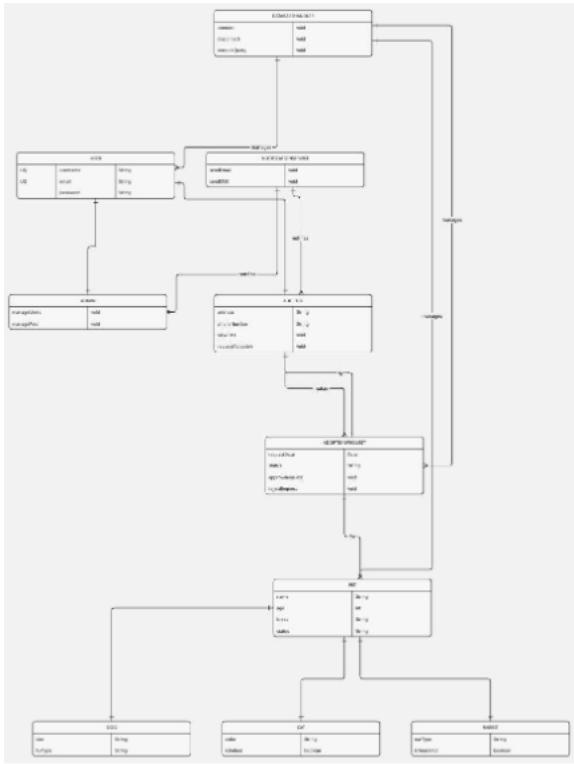
Project Topic: Pet Adoption Management System

Problem Statement

Many animal shelters face challenges in efficiently managing pet adoption processes. Adopters often find it difficult to search for pets, while shelter staff struggle to keep track of pet availability, adoption requests, and user details. This project aims to create a Pet Adoption Management System that allows shelters to add and manage pet listings and users to search for pets and apply for adoption online. The system will simplify the process through a user-friendly interface, streamline the approval workflow for staff, and improve adoption success rates by automating and organizing the adoption process.

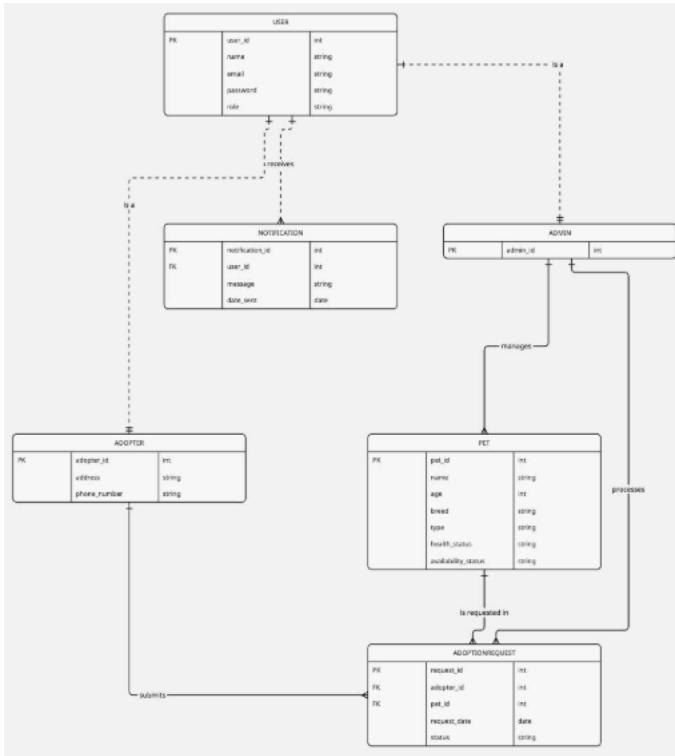
UML Diagram

The UML Class Diagram models the object-oriented structure of the Pet Adoption Management System. It outlines the key classes such as User, Admin, Adopter, Pet, AdoptionRequest, NotificationService, and DatabaseHandler. The diagram highlights inheritance relationships (e.g., Admin and Adopter are specialized forms of User) and clearly defines class attributes and methods. It also showcases class interactions such as how Adopters submit adoption requests, Admins manage pets, and NotificationService notifies users. This diagram emphasizes the use of core OOP principles like abstraction, inheritance, encapsulation, and method definition in line with Java development practices.



ER Diagram:

The Entity-Relationship (ER) Diagram represents the database structure of the Pet Adoption Management System. It includes entities such as User, Admin, Adopter, Pet, AdoptionRequest, and Notification, along with their primary keys, attributes, and foreign key relationships. The diagram captures how users receive notifications, adopters submit adoption requests, and admins manage and process pets. The use of "is-a" relationships denotes inheritance, while labeled associations like submits, manages, and processes define clear relational flows. This ER diagram provides a normalized view of how data is organized and related across the system.



Tech Stack

1. Backend: Java with Spring Boot
2. Frontend: React.js, HTML, CSS
3. Logging : CSV files for listing's CRUD updates
4. Database: MongoDB
5. IDE: Eclipse
6. Testing: Manual testing via console, UI and controller endpoints

Milestone 1: Planning and Core Design:

This phase focused on analyzing requirements and designing the system's architecture using UML and ER diagrams. Key object-oriented principles were outlined, and initial data flow, class relationships, and role-based access logic were defined.

Highlights:

- UML Diagram: Defined 'User', 'Admin', 'Adopter', 'Pet', 'AdoptionRequest', Inheritance and interaction relationships established.
- ER Diagram: Designed to show relationships among 'User', 'Pet', and 'AdoptionRequest' for future MongoDB schema.

- OOP Principles Defined:

- Abstraction using abstract Pet class
- Inheritance: 'Dog', 'Cat', 'Rabbit' from 'Pet'
- Polymorphism: Overloaded methods
- Encapsulation: private fields with getters/setters

This laid the groundwork for building a layered Java application with a strong foundation in object-oriented design.

Object-Oriented Concepts Implemented:

1. Abstraction:

The User, Item, and Listing classes act as **data models** that abstract real-world entities. Each class holds relevant properties and behavior related to its role in the system.

Example:

```
public class User {  
    private Long id;  
    private String username;  
    private String password;  
    private String role;  
  
    // Getters and Setters omitted for brevity  
}
```

2. Inheritance:

While classical inheritance is not deeply used in this milestone, the use of **interface extension** and Spring Boot's JpaRepository inheritance supports database operations.

Example:

```
public class Admin extends User {  
    public Admin(String username, String password) {  
        super(username, password, "Admin");  
    }  
}
```

This is a form of **interface-based inheritance**, allowing reuse of Spring's repository methods for entities.

3. Encapsulation:

All model classes use private fields with public getters and setters, protecting direct access to internal state.

Example:

```
public class Listing {  
    private Long id;  
    private String title;  
    private String description;  
    private String status;  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    // Other getters/setters...  
}
```

4. Polymorphism:

Polymorphism is demonstrated in the controller layer by how endpoints invoke services that return different model types (Item, Listing, etc.). The same controller method can return different data objects dynamically based on path or parameters.

Example:

```
public class Pet {  
    public String getDetails() {  
        return "General Pet";  
    }  
}  
  
public class Dog extends Pet {  
    @Override  
    public String getDetails() {  
        return "Dog-specific details";  
    }  
}  
  
@GetMapping("/items")  
public List<Item> getAllItems() {  
    return itemRepository.findAll();  
}
```

Different endpoints across ItemController, ListingController, and AuthController return or process different objects while following a uniform controller structure.

Milestone 2:

CLI Implementation and Layered Backend Milestone 2 focused on transitioning from a theoretical design to a functional CLI-based backend using Java. CSV files were used for persistence, and the system supported role-based login, pet management, adoption requests, and basic notification through console messages.

Functionalities Implemented:

- Adoption Requests: Adopters could submit requests by entering pet IDs. Admins could approve or reject.
- Notifications: Console outputs displayed the result of request decisions.
- File Handling: CRUD operations and request logs maintained in CSV files.
- Data Validation: Basic checks for ID matching and duplicate registration.

Backend Code Highlights Java model classes encapsulated user and pet data. Controllers handled logic like request routing and role filtering. Repositories were introduced using interfaces (though manual file operations were used instead of databases). This prepared the system for a Spring Boot upgrade in the next phase.

Milestone 3:

Full Stack Integration and Deployment Milestone 3 represented the final evolution of the system from a CLI and CSV-based application into a complete, production-ready web application. We implemented a RESTful backend using Spring Boot, persisted data with MongoDB, and developed a React.js frontend that interacts seamlessly with the backend through HTTP endpoints.

1. Backend Architecture The Java backend was built with Spring Boot and follows a strict Model–Repository–Controller (MRC) pattern. Models are simple POJOs with annotations like `@Document` for MongoDB. Controllers expose REST APIs for all system features, and repositories extend Spring Data interfaces to abstract away complex query logic.
2. Controllers implemented:
 - AuthController: Handles user login and registration.
 - ItemController: Manages creation and retrieval of items.

- ListingController: Admin operations for pet listings, adoption logic, approvals.

3. Models:

- User.java: Represents application users with fields for ID, name, email, password, and role.
- Item.java: Represents listing items and supplementary details.
- Listing.java: Core object for pet listings, includes details like type, status, adopter ID, and timestamps.

4. Repositories:

- Extend MongoRepository for each entity, e.g., UserRepository, ListingRepository.
- Allow CRUD operations without boilerplate using Spring Data's abstraction layer.

Maintained layered structure:

- a. Model: User, Listing, Item, Enum (UserRole, ListingStatus), BaseEntity (shared fields)
- b. Controller: Auth (Login/Register/Profile), Listing (CRUD, request, approval), Item (basic request flow)
- c. Service/Utility: ListingService (business logic), ListingCSVLogger (CSV logging), Exportable (CSV row interface)
- d. Data persistence via MongoDB (collections: users, listings, items) and CSV file (logs/listings_log.csv)
- e. Implemented Exportable interface for modular CSV row formatting (used in ListingCSVLogger)
- f. Added input validation & exception handling (e.g., missing fields, logging errors, user not found)
- g. Search/filter logic handled via service (get listings per user, show denied-to-public flow)
- h. Applied OOP principles:

1. Inheritance: User, Listing, Item extend BaseEntity (id, timestamps)
2. Polymorphism: Listing implements Exportable, used in logger polymorphically
3. Encapsulation: All models use private fields with public getters/setters
4. Interface: Exportable used for flexible CSV serialization

Frontend Implementation:

React.js The frontend of the system was implemented using React.js. We created a single-page application (SPA) with routing for login, registration, and dashboard views. The user experience varies based on role (Admin vs Adopter), enabling customized interactions for each user type.

1. React Components:
 - LoginPage – Collects user credentials and calls /login API.
 - SignupPage – Allows new users to register (role-based).
 - Dashboard – Main hub after login with different views for Admins/Users.
 - ListingsPage – Displays pet cards and adoption request buttons.
 - AdminPanel – Includes tools for approving/rejecting requests.
2. Axios used for API communication. The token-based logic could be added in future iterations for secure headers.

MongoDB and CSV Logging:

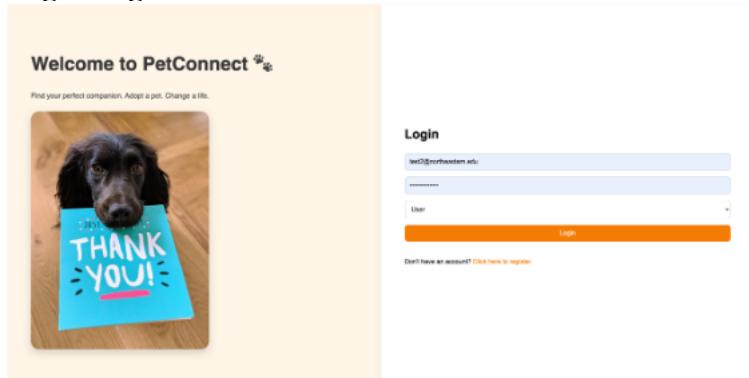
The application connects to MongoDB Atlas using a secure URI stored in the environment file. Spring Boot autoconfigures repositories to interface with MongoDB collections (users, listings). For traceability, each CRUD and adoption operation is also logged to listings_log.csv via a structured utility class that formats each action into an exportable row.

Final Application Workflow

- Users can register/login through the frontend.
- Admins add pet listings, view requests, and approve/deny them.
- Adopters browse pets, submit adoption requests.
- Listings dynamically update based on approval status.
- All activity is logged to CSV for audit trails.

Screenshots:

Login Page:



Welcome to PetConnect 

Find your perfect companion. Adopt a pet. Change a life.

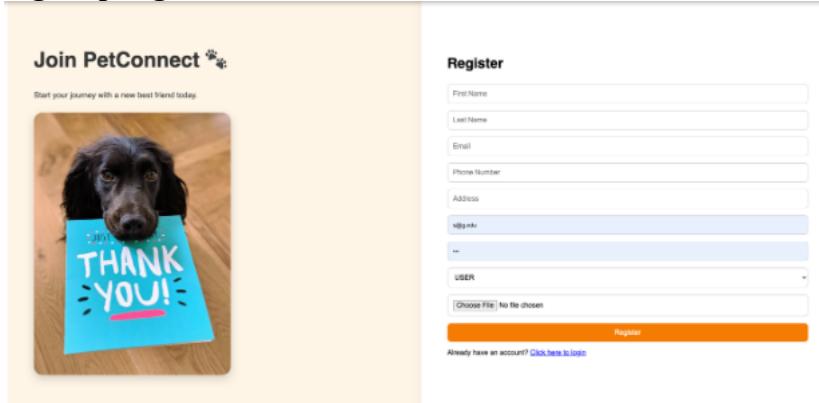


Login

User:

Don't have an account? [Click here to register](#)

Sign-Up Page:



Join PetConnect 

Start your journey with a new best friend today.



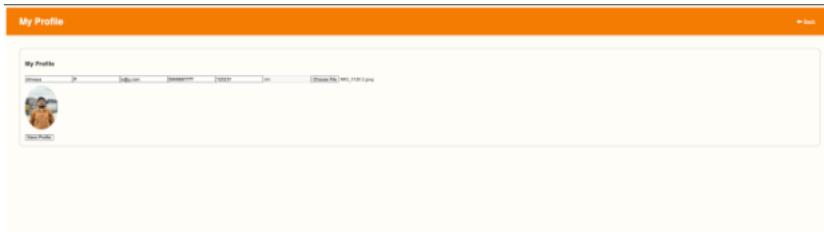
Register

First Name
 Last Name
 Email
 Phone Number
 Address
 [@petnkr](#)
...

Choose File: No file chosen

Already have an account? [Click here to login](#)

User Profile:



My Profile » Back

My Profile



Gender: Male Age: 2y Location: Boston, MA Zip: 02114 Photo:

Listings:

Welcome, srujan

Available Listings

Muffin
Age: 1 Year
Gender: Male
Cattery: None

[View Details](#)

My Profile Log Out

Admin:

Admin Dashboard

Add Listing

Title:

Description:

All Listings

Muffin
Age: 1 Year
Gender: Male
Cattery: None

[Details approved](#)

[Rejected by: srujan](#)

Moo
Age: 1 year
Gender: Female
Cattery: None

[Details approved](#)

[Rejected by: srujan](#)

My Profile Log Out

UserRequest:

Welcome, srujan

Available Listings

Muffin
Age: 1 Year
Gender: Male
Cattery: None

You have requested this item (Status: pending)

My Profile Log Out

Admin Approval:

The screenshot shows the Admin Dashboard interface. At the top, there's a form titled "Add Listing" with fields for "Title" and "Description". Below this, a section titled "All Listings" displays two items:

- Max**: A dog for adoption. Status: Pending. Photo: A black and white photo of a dog sitting.
- Leo**: A cat for adoption. Status: Pending. Photo: An orange tabby cat sitting.

For each listing, there are buttons for "Edit", "Delete", and "Approve".

User Approved:

The screenshot shows the User Profile page for a user named "shr". At the top, it says "Welcome, shr". Below that, a section titled "Available Listings" shows one item:

- Leo**: A cat for adoption. Status: Approved. Photo: An orange tabby cat sitting.

Below the listing, a message says "You have requested this item. It has been approved." There are "Edit" and "Delete" buttons at the bottom of the listing card.

Contributions

1. Ujwal Chandrashekhar – Backend development, User authentication
2. Shreyas Pogal Naveen – Frontend development, React.JS
3. Ujwal Chandrashekhar – Adoption Logic, workflow, Data Handling
4. Shreyas Pogal Naveen – Database connection Setup, Testing and debugging

GitHub Repository: https://github.com/CSYE6200-Object-Oriented-Design-Spr25/csye6200-spring-2025-final-project-final_project_group_2