Assignment: Virtual Pet Adoption Center

Objective

Build a simple yet visually appealing Virtual Pet Adoption Center where users can manage pets available for adoption. Each pet has attributes like name, species, age, and personality traits. Users can create, read, update, and delete pet profiles, as well as "adopt" pets (mark them as adopted). Pets also have dynamic moods that change based on how long they've been in the system.

You will use Node.js for the backend and React.js for the frontend. Complete this task over the weekend (Saturday and Sunday). UI design, responsiveness, and code structure optimization are critical for this assignment.

Task Sheet

1. Backend (Node.js)

- Build a RESTful API using Node.js to manage pets.
- Each pet should have:
    - id (auto-generated)
    - name (string)
    - species (string, e.g., "Dog", "Cat")
    - age (integer)
    - personality (string, e.g., "Friendly", "Shy")
    - mood (string, dynamically changes based on time in the system)
    - adopted (boolean, default false)
    - adoption_date (date, optional)

Endpoints to Implement:

1. Add a New Pet
    - URL: POST /pets
    - Input: JSON object with name, species, age, and personality.
    - Output: Created pet object with an auto-generated id.
2. View All Pets
    - URL: GET /pets
    - Output: List of all pets, including their current mood and adoption status.
3. View a Single Pet
    - URL: GET /pets/:id
    - Output: Pet object with the specified id.
4. Update a Pet's Profile
    - URL: PUT /pets/:id
    - Input: JSON object with updated fields (name, species, age, personality).
    - Output: Updated pet object.
5. Adopt a Pet
    - URL: PATCH /pets/:id/adopt
    - Action: Mark the pet as adopted (adopted = true) and set the adoption_date.
    - Output: Success message or updated pet object.
6. Delete a Pet
    - URL: DELETE /pets/:id

    - Output: Success message.
7. Filter Pets by Mood
    - URL: GET /pets/filter?mood=<mood>
    - Output: List of pets matching the specified mood.

Dynamic Mood Logic:

- Pets' moods change based on how long they've been in the system:
    - Less than 1 day: Happy
    - 1–3 days: Excited
    - More than 3 days: Sad

Code Structure Optimization for Node.js:

- Use a modular structure:
    - Separate files for routes, controllers, and models.
    - Example folder structure:

/backend

├── app.js // Main application file (initializes Express app)

├── server.js // Server setup (starts the server)

├── /routes // API routes

│ └── petRoutes.js

├── /controllers // Business logic

│ └── petController.js

├── /models // Data models

│ └── petModel.js

├── /services // Service layer for business logic

│ └── petService.js

└── /utils // Utility functions (e.g., mood logic)

└── moodLogic.js

2. Frontend (React.js)

- Build a simple React.js frontend to interact with the backend API.
- The interface should allow users to:
    - View all pets in a list or grid, showing their name, species, mood, and adoption status.

    - Add a new pet using a form.
    - Update a pet's profile (e.g., change its name or personality).
    - "Adopt" a pet by marking it as adopted.
    - Delete a pet from the system.
    - Filter pets by mood (e.g., "Show me all Happy pets").

Frontend Features:

- Use cards or a table to display pets.
- Use icons or colors to represent moods (e.g., green for Happy, red for Sad).
- Ensure the frontend communicates with the backend via API calls.

UI Design Requirements:

- Attractive and Responsive UI:
    - Use modern CSS frameworks like TailwindCSS or Bootstrap for styling.
    - Ensure the UI is responsive and works seamlessly on mobile, tablet, and desktop.
    - Use animations or transitions for actions like adopting a pet or deleting a pet (e.g., fade-out effect when a pet is deleted).
- File Structure for React:
    - Follow a clean and modular structure:

```
/frontend
  ├── public/         // Static assets (e.g., images)
  ├── src/
  │    ├── App.js    // Main component
  │    ├── index.js  // Entry point
  │    ├── /components
  │    │    ├── PetList.js       // Displays list of pets
  │    │    ├── PetCard.js       // Individual pet card
  │    │    ├── AddPetForm.js    // Form to add a new pet
  │    │    └── FilterBar.js     // Filters pets by mood
  │    ├── /pages
  │    │    └── HomePage.js      // Main page
  │    ├── /services
  │    │    └── api.js           // API calls
  │    ├── /styles
  │    │    └── global.css       // Global styles
  │    └── /utils
  │         └── helpers.js       // Helper functions
```

Optimized Code Practices for React:

- Use functional components with React Hooks (useState, useEffect).
- Centralize API calls in a single service file (api.js).
- Use reusable components for consistency (e.g., PetCard for displaying individual pets).
- Optimize performance by avoiding unnecessary re-renders (use React.memo or useCallback where needed).

3. Bonus Features (Optional)

- Pet Personality Quiz: Allow users to take a quiz to find a pet that matches their personality.
- Adoption Certificate: Generate a downloadable PDF certificate when a pet is adopted.
- Notifications: Notify users when a pet's mood changes to "Sad" after being in the system for too long.
- Animations: Add animations for mood changes or adoption actions (e.g., confetti when a pet is adopted).

Submission Guidelines

- Provide a GitHub repository with the complete source code.
- Include a README.md file with instructions on how to set up and run the application.
- If applicable, include screenshots or a video demo of the working application.

Timeline

- Saturday:
    - Set up the backend (Node.js) and implement all required API endpoints.
    - Test the backend thoroughly using tools like Postman.
    - Focus on optimizing the backend code structure.
- Sunday:
    - Build the frontend (React.js) and connect it to the backend.
    - Add styling and make the interface user-friendly.
    - Ensure the UI is responsive and attractive.
    - Test the full application and fix any bugs.

Evaluation Criteria

1. Code Quality:
    - Clean, readable, and well-organized code.
    - Proper separation of concerns in both backend and frontend.
2. Functionality:
    - All required features are implemented correctly.
3. Frontend Usability:
    - Intuitive and user-friendly interface.
    - Attractive and responsive design.
4. Performance Optimization:
    - Backend and frontend code is optimized for efficiency and scalability.
5. Bonus Features:
    - Creativity and effort in implementing optional features.

This assignment emphasizes UI design , responsiveness , and code structure optimization , ensuring that you deliver a polished and professional application. Good luck!