Technical Requirements Document - Antelope Data Dash

Project Overview

The purpose of this web application is to visualize data related to antelope species, allowing users to compare various species in an intuitive and interactive way. The application includes a data table, charts to display the antelope data, and antelope data fact cards, facilitating insights into species characteristics.

Scope

This TRD covers the requirements for the frontend and backend of the application. It outlines the functional and nonfunctional requirements, system architecture, and data handling.

System Overview

Frontend

Technologies Used: React.js, ChakraUI, Recharts
Components:

- **Data Table:** Displays a list of antelope species with key attributes (name, height, weight, horn type, continent)
- **Charts:** Visualize species data. Three charts are implemented:
  - A bar chart showing weight per species
  - A pie chart illustrating the distribution of species per continent
  - A scatter plot chart indicating correlation between weight and height
- **Cards:** Displays image and facts about each antelope species
- **User Interaction:** includes dropdown for filtering which chart to view, and tabs for filtering which component to view.

Backend

Technologies Used: Node.js, Express.js
Purpose: Serves the frontend with the antelope data and handles server side processing.
Endpoints: /api/antelopes: Returns the list of species in JSON format.

Functional Requirements

Frontend

1. **Data Table:**
   - Display all antelope species with columns for species name, weight, height, horn type, and continent
2. **Charts:**
   - Bar Chart: Displays weight per species
   - Pie Chart: Illustrates distribution of species per continent
   - Scatter Plot Chart: Indicates correlation between weight and height of species
3. **Interactivity:** Users can select which chart to view from a dropdown menu

Backend

1. **Data Serving:**
   - Serve the antelope data from the provided JSON dataset via an API endpoint.

Non Functional Requirements

Usability:

- Intuitive UI: Interface should be easy to navigate, with clear buttons, dropdowns, and data visualization components.
- Responsiveness: The application should be responsive and function well on different screen sizes.

System Architecture

Overview

The system consists of a frontend built with React.js and a backend developed using Node.js and Express.js. The frontend and backend communicate via RESTful API endpoints.

Frontend Architecture

Component-Based structure:

- The frontend is organized into a set of reusable components.

  Key Components:
  - `AntelopeTable.tsx`: Handles rendering the table that displays antelope data.
  - `BarChartComponent.tsx`: Manages the bar chart visualization.
  - `PieChartComponent.tsx`: Responsible for the pie chart visualization.
  - `ScatterPlotComponent.tsx`: Manages the scatter plot visualization.
  - `AntelopeView.tsx`: Renders the cards with antelope facts.
  - `Visualization.tsx`: Acts as the container that aggregates the different data components (table, charts) and manages switching between them based on user input.
  - `App.tsx`: Acts as the homepage container and manages switching views based on user input.

State Management:

- The application uses React’s state management (`useState`, `useEffect`) for handling component states and fetching data.
- A custom hook, `useFetchAntelopeData.ts`, is used to manage data fetching from the backend and ensure the data is available to the components as needed.

Backend Architecture

Server Structure:

- The backend is implemented using Node.js with Express.js framework, which handles incoming HTTP requests and serves the antelope data from `data.json`.

  Key Files:
  - `server.ts`: The main server file, which sets up the Express server, defines routes, and handles requests.
  - `data.json`: The static file containing the antelope species data, which is read and served to the frontend.

API Design:

- The primary endpoint `/api/antelopes` returns species data in JSON format.

Middleware:

- CORS: Implemented to handle cross-origin requests.

Interaction Between Frontend and Backend:

- The frontend makes asynchronous API calls to retrieve data from the backend, which is then rendered into tables, charts, and cards