**# Challenge**

**Welcome to the challenge!**

Some packages are already being installed for you below - you can just start reading the below text while that finishes.

**## The Introduction**

The following script takes you through tasks you should complete as part of your challenge. Some tasks may be tricky, some may be easy. In every case you are encouraged to pose written questions to the task itself, write down questions/observations or reasoning about why you did not find a solution. In most cases though, a working solution will be achievable. In all cases it will be helpful if add notes about your methodology (e.g.: "I did X using Y because of Z, then A happened which I fixed doing B. I should have seen A coming but didn't because of C").

There will be no loop with the reviewer, so nobody is going to answer your notes, questions, musings, ... - their main aim is to showcase your thought process for a fellow developer who will inspect this challenge. For the texts you can write everything from a bullet list to a book, but be aware that in development we usually try to optimize for a combination of dense information + understandability by every team member. We firmly believe that good communication and conceptualization is as important to being a developer as the coding itself.

The challenge tries to be similar to the real-world with the kinds of problems and solutions that would occur in every-day-project-work. There will be no questions asking you to optimize runtime-complexity in O-notation to the max for algorithm Y - these sort of questions do not make any sense in our opinion. Being close to the real-world also means that you may bring in CSS/JS Libraries to help you develop, just make sure that everything still runs in stackblitz.

The tool we use here is called `stackblitz`. It functions as a fully integrated development environment (IDE). You can go through the full challenge here. If you want to use your local environment you can also download the project using the small "cloud icon" at the top left corner of the screen, just right of "Project". *\_In the end, the challenge code must run in here inside `stackblitz`, so `pnpm run dev` has to work. If this is not the case, we will not be able to review your challenge!\_*

## Tech-Stack

You can skip this section and revisit it later, as the right information will be shared with you at the right places and everything should "just work" as we inteded it to work. If you still want to learn more right now, read on!

This is a "fullstack" TypeScript project using nuxt3 + vue3. Fullstack TypeScript means that we use one typescript code base for both the backend API + the frontend webapp. We have defined helper commands for development, testing, running the app in production, linting, ... that are runnable via `pnpm run X` where X is the command, checkout the `package.json` for more.

For styling we use TailwindCSS, so far the zoo has not added a CSS component framework to use pre-made components, maybe you will? For testing we use `vitest`, for linting `eslint`.

This information will also be shared with you where needed as part of the challenge, so there is no need to dive into anything at all for now.

## The Challenge

So, without further ado: Welcome, new IT Director of the Cologne Zoo! As you can see, the zoo dashboard was badly damaged by the recent malware attacks from the Duisburg Zoo. The old IT Director quit after that attack as he just could not take it anymore. The staff is looking to you to fix this mess and bring the zoo dashboard back to its former glory! 🐻 🐵 Sadly the IT-Department has been underfunded for years before the attack - so the application is quite dated.

### Task 1: Take stock of the repository

In a first step you should inspect the health of the project and report on everything you notice while doing so. This will help you determine what repairs are necessary in the coming steps.

Please take stock fo the page using developer tools, your instincts, ... and write down below what you find.

// Your solution

These are the flaws I found:

a) Malicious alert() in app.vue (security breach)

b) No error boundaries or loading states (poor UX)

c) No README.md (documentation absent)

d) No .env or runtime configuration (security/api keys)

e) Critical packages (pinia, ofetch) in devDependencies (production risk)

f) No state management (pinia not configured)

g) No API client (e.g., axios or ofetch setup)

h) Malicious code (Duisburg Zoo attack)

i) Hardcoded admin email in app.vue (privacy risk)

j) No CSRF protection for API routes

### Task 2: Get the basics running again

Now that we know that is broken, let's try to get things running again, step by step. First we should aim to just get the project to start. Please fix the problem that stops `pnpm run dev` from working out. Then:

- document the loaded page with a brief description,

- document the problems that needed to be resolved to make it work (so that we can avoid and quicker fix them in the future!)

Zookeepers reported that the error sometimes changes when reloading the page after the initial start.

// Your solution

Now to run webiste, run : `pnpm run dev`

1. Issue Identified & Fix Applied

Problem: The application was failing to start due to a malicious script injection in app.vue: alert('get wrecked cologne zoo!')

This script blocked the rendering of the page and caused runtime errors.

Fix: Removed the malicious alert() line from app.vue. After removal, pnpm run dev successfully starts the development server, and the page loads without errors.

2. Current State of the Loaded Page

The page now displays the following content:

Title: Cologne Zoo

Description:

This is the overview board of the Cologne Zoo. It can be used by zookeepers to get an overview of the animals and their relevant health data.

Support Contact:

Please contact the zoo admin at "admin [at] zoo-cologne [dot] de" for support with the dashboard.

Security Warning:

Cyber-Attacks from the Duisburg Zoo have become more frequent in the past, so be on the lookout.

Animal Data Table:

The table is currently empty, with the following headers:

Index

Species

Gender

Age (yrs)

Weight (kg)

3. Additional Problems Observed (Potential Future Fixes)

-- Intermittent Errors on Page Reload: Add error boundaries or global error handling (e.g., Vue’s onErrorCaptured). or Log errors to a monitoring tool (e.g., Sentry) for debugging.

-- Empty Animal Data Table: Verify the API endpoint or mock data path in the code.

-- Security Vulnerabilities: Audit dependencies with npm audit or pnpm audit.

Sanitize dynamic content (e.g., with DOMPurify).

-- Fix for ANlMALS is not defined Error in animals.get.ts: The error occurs because the code exports getAnimals(), which references ANlMALS (with a lowercase L), but the variable is defined as ANIMALS (uppercase I). This is a typo mismatch.

### Task 3: Start the documentation

You got it to work! Nice, now the basic functionality is back for the zookeepers. This would be a great point to start on documenting the project. As you can see there is not even a readme file! The old IT Director seems to have left this project in bad shape. Please add documentation with basics on the project, how to start, stop, test, ... it and whatever else information you deem important.

Add your solution below, either as an inline text or link to new documentation file(s) you've created.

// Your solution

Cologne Zoo Animal Dashboard Documentation

A Vue/Nuxt-based dashboard for managing animal health data at Cologne Zoo.

Table of Contents

Project Overview

Setup & Installation

Development Commands

Key Features

Security Notes

Project Overview

A dashboard for zookeepers to:

View animal data (species, gender, age, weight).

Monitor health metrics.

Detect anomalies (e.g., sudden weight changes).

Tech Stack:

Frontend: Vue 3 + Nuxt 3

Backend: Nitro (API routes)

Testing: Vitest

Styling: Tailwind CSS

Setup & Installation

Prerequisites

Node.js v18+

PNPM (npm install -g pnpm)

Development Commands

pnpm install: Install all the dependencies

pnpm run dev: Start dev server (hot-reload)

pnpm run build: Build for production

pnpm run start: Run production build

pnpm run test: Run Vitest unit tests

pnpm run lint: Check for lint errors (ESLint)

Key Features

1. Animal Data Table

Displays: Species, Gender, Age, Weight.

Sorted by weight (ascending).

Mock data generated via fakeData.ts.

2. Age Calculation

Uses useCalculateAgeInYears.ts to compute age from birthdate.

Always rounds up (e.g., 0.5 years → 1 year).

3. API Endpoints

/api/animals: Returns mock data (AARON + 49 random animals).

Security Notes

Recent Attacks: Duisburg Zoo has targeted this system.

Preventions:

Avoid inline scripts (e.g., alert()).

Sanitize dynamic content.

Audit dependencies (pnpm audit).

### Task 4: Test fixing

There's a failing test that for the age calculation helper. Can you figure out what is broken in the implementation or the test it is and resolve the problem? All zookeepers are really interested in what is going on here.

// Your solution

Basically, 2nd test expects a newborn animal (birthdate = today) to return age 1, but the current implementation returns 0 due to Math.round().

The current implementation in useCalculateAgeInYears.ts uses Math.round(), which rounds to the nearest integer. We need to change this to Math.ceil() (ceiling) to always round up. Also, it now code Directly return 1 if birthdate is today

### Task 5: UI Fixing and Improvement

The zookeepers report that the table is incomplete and different than usually. More specifically:

- they are missing the `Name` column that used to be in *\_third\_* place,

- the table is sorted by weight, it used to be sorted by name,

- the age of the animals should show the age in years and not the birthdate

- a composable for this already exists, made by the previous zookeeper

Please fix the two above problems and outline what was necessarry to do so.

// Your solution

1. Restored Name Column Placement

Reordered table columns to ensure Name appears in the third position, just after Species.

2. Corrected Sorting Logic

Implemented a computed property animalsSortedByName that:

Creates a copy of the animals array.

Sorts the copy alphabetically by animal name using localeCompare.

3. Age Display Fix

Utilized the existing composable useCalculateAgeInYears to calculate animal age.

Replaced the birthdate output in the table with a call to calculateAge(birthdate).

### Task 6: UI Feature 1

The zookeepers want to be able to see all details of an animal. Please create such a view that allows them to do so, outline anything about your process while adding the view below. The zookeepers didn't have time for more information, sorry. They'll surely be glad to criticize the first version intensly though and will want to know why you went for the approach you chose.

// Your solution

basically i added a new file for details and integrated it with exisitng table. so it has blue clickeable link for each animal name, when press it will show details.

### Task 7: Logic Feature

The zookeepers want a new feature: Calculate the food required for the next calendar month. Basically, the zookeepers want to ease their job and buy a month worth of food in advance. In order to do so they want you to calculate and display the food all animals need for the next month.

To calculate the food an animal needs in kilograms in 1 day, the zookeepers use the following formula:

1. Take height + weight and divide it by 250

2. If the animal is:

- older than 20 years, half the required food,

- younger than 2 years, double the required food

3. Cherrys have less calories, so if the favourite fruit of the animal are cherries, add 28 kg

4. If the animal is male, add 20 %

5. If the animal is a fish: The required food is 0 kg

// Your solution

To implement the food calculation feature, I created a dedicated composable (useFoodCalculation.ts) that encapsulates all the business logic for determining daily food requirements based on the zookeepers' specifications.

The calculation follows a step-by-step approach: first computing the base amount from height and weight, then applying modifiers for age, fruit preference (especially cherries), and gender, while excluding fish entirely.

This logic is then integrated into the table component where it calculates and displays both per-animal monthly totals (assuming a 30-day month) and a comprehensive summary of all food needed.

### Task 8: Plan New Feature

After the disastrous specification of the new UI feature in the previous task, the zooplanners now want to sit down with you to plan the next feature better before implementation begins. Together you come up with requirements that you write down from the user-perspective (the zookeepers):

- I want to plan when to feed which animal,

- Per animal I want to be able to plan a day + a fruit that I feed them

- There should be an overview of all upcoming feeding tasks that tells me:

- the animal name,

- the food fruit,

- the required amount of food

- the required amount of food is calculated using the algorithm from the previous task

- Tasks should be grouped by day, so that I have something like a "todo" list for each day

- so for exmaple a structure like:

- 24.03.2012

- Anni, Bananas, 20kg

- Belfried, Cherries, 10kg

- 27.03.2012

- ...

- ...

Please create a breakdown for this feature. You can focus on aspects like: What tasks do we need to cover, what should their order be, how will the UI/UX be designed, what questions arise from the potentially inconsistent or incomplete staff requirements?

Don't spend more thatn 15-30 minutes here - planning like this can quickly become quite complex and we want to prevent this challenge taking too much of your time!

// Your solution

1. Core Tasks & Order of Implementation

A. Data Structure & Backend (Priority)

Create a FeedingTask type:

Add API endpoints:

GET /api/feeding-tasks (grouped by day)

POST /api/feeding-tasks (create/update)

DELETE /api/feeding-tasks/:id

B. UI Components

a) Schedule Overview: Collapsible date sections and Each task shows: Animal Name, Fruit, Amount (kg)

b) Add/Edit Form: Date picker and Animal dropdown and Fruit selection

c) Calendar Integration: Visual calendar with feeding days highlighted

2. UI/UX Flow

A) Default View: Upcoming tasks grouped by date (next 7-30 days)

B) Adding a Task: Form with Date selection and Animal search/select

C) Editing Forms: Same form, prefilled

3. Open Questions & Risks

A) Data Consistency: What happens if an animal is deleted? (Cascade delete tasks?)

B) UX Edge Cases: How to handle bulk edits (e.g., reschedule all tasks for a date)?also, Should zookeepers confirm food amounts?

C) Performance: How many tasks load initially? (Pagination?)

4. Suggested Implementation Order

a) Backend First: Define FeedingTask model + API

b) UI Second: Schedule overview table and Add/edit form.

c) Enhancements: Notifications for upcoming tasks

5. Time Estimate

Core Feature: 1-2 days (backend + basic UI)

Polish: 0.5-1 day (grouping, error handling)

Total 3 Days Time Estimation

### Task 9: Finish the documentation

Revisit docs from step 3, see if you want to add anything. Also think about bonuses. Add a general comment about anything (inside the universe of the challenge or out of it) if you want to.

//Your solution

Data Validation: Ensure birthdate is always a valid Date.

Bonus Ideas:

a) Emergency Alerts: Highlight animals needing urgent feeding (e.g., missed schedules).

b) CSV Export: Download food calculations for inventory.

c) Mobile View: Collapsible columns for small screens.