DOCUMENTATION

Foreword

To begin with, I want to say that every decision I made in this assignment was made in accordance with my experience or the best practices that I learned at the university. I will not thoroughly describe each of my thoughts, as then it will turn out to be a huge essay.

First thoughts

Reading the assignment, I immediately pay attention to the technologies that are required by the technical specification. I'm looking at html files that can be used directly in my application.

At first glance, everything is quite simple, since I have previously encountered the listed technologies to some extent or have a little experience with them.

I am setting up two projects, one backend on spring boot, the other frontend on angular.
I think that it’s not very good to keep everything in one project, and it’s better to separate the front from the back. The only thing that is confusing at the moment is Liquibase, I will need to figure out what it is and how to use it.

Throughout all project I will try my best to follow the principles of `Clean Code` to make the code more readable and reusable.

BackEnd (time spent ~3.5hr)

I initialize the latest version of the Spring framework along with the dependencies I need (h2, JPA, Liquibase, Spring Web e.t.c.) and configure the database to be h2 (add `application.yaml` where I specify the datasource to h2).

I am going to separate the code in layers and create an Entity that would be managed via our layers:

- **Controller layer** (expose functionality to external entities, in our case, our frontend. Works with service lower layer)
- **Service layer** (all the business logic, only works with lower layer, which is repository)
- **Repository layer** (responsible for storing and retrieving data)

In this way, we can ensure that each component is responsible for a certain logic, and we have better code readability.

I create a `Pet` model with fields:

- `Long id` Unique identifier needed to store objects in DB
  And rest of the field according to task:
- `String name` String to store name
- `Long code` Code is not int, because we suppose that the code is a long digit. Might have used String, as the code in different countries might contain characters, but for the simplicity kept this format.
- `String type` Type is String
- `String furColor` Fur Color is String
- `String country` Country is String

I use Lombok to take care of getters and setters.

Then create a `PetRepository` to store `Pet` objects.

Then create `PetService` which uses `PetRepository` to implement business logic with the model. Get all pets from DB, get pet from DB by id, add new pet to DB and e.t.c.

Then create `PetController` to expose the functionality of `PetService`. Add `GET`, `POST`, `PUT` methods.

The Api is exposed on `/api` (as it indicates in the url that the client/server is working with the api), whereas the REST Api for management Pets is `/api/pets`. I tried my best to follow the REST architecture.

I configure liquibase migrations. I create folders and files in `resources` folder and specify the *pet* table to be created and populated with records.

Also, I create custom exception to handle situations when Pet entity is not found in Database.

FrontEnd (time spent ~6hr)

I initialize the latest version of the Angular framework and install bootstrap for it to use across all the project.

Then I create a model interface for the `Pet` with the same fields to be used in frontend and the `PetService` with HttpClient to send requests, retrieve or send data to the endpoints from backend.

Then I create 3 main components according to the task:

- `pet-list`, which is a component, where user can view a list of all pets
- `create-pet`, which is a component, where user can add a new pet to the list
- `update-pet`, which is a component, where user can update some details of the certain pet

`create-pet` and `update-pet` are different components as they serve different tasks. Basic forms are used in the HTML. After submit I trigger the function to hit the endpoints with data via `PetService`.

The drawback I see right now in my realisation is that 3 fields (type, fur color and country) are realised as the `select` HTML element, so I need to initialise these structs to be able to properly use them and iterate over in the `option` tag and make a binding. Ideally, we should be able to retrieve and manipulate (add, remove, update) `type, fur color and country` from the backend and so these models have to be created in backend and then in the frontend should be created models with services accordingly, but for the simplicity of the application I created `select-model` to be a model for these 3 properties. I create the values for select option accordingly in `create-pet` and `update-pet` components.

Also, I considered using `DataTables` table plug-in (I used this thing a couple of times, and it comes in handy), which out of box enables sorting the table by all columns, pagination and searching.

Validation (time spent ~2hr)

All `Pet` model fields are mandatory.

The only validation that is done is that `name` must not be empty string (must contain at least 1 char) and be less than 50 symbols. `code` is a number with fixated length of 12 digits.

Angular

Create/update form fields validation is done with Angular built-in functionality. The validation is done specifically on two fields`name` and `code`, as with the `select` element we just need to make sure user has picked one of the options. The field validation is done after trying to submit the form. Invalid inputs turn red and error messages are shown under the fields.

Spring

Model validation is done with `Hibernate Validator` and annotations. The annotations used are `@NotBlank`, `@NotNull`, `Length` for the name, `@Min/@Max` for the code and `@Pattern` for the type, fur color and country, as I needed to check if provided string was in the select option list provided from frontend form and e.t.c.

Also, I create an `ApplicationExceptionHandler`, in package `advice`, which allows to easily process requests to different endpoints, validate Pet objects, and send the responses in JSON format.

Login and Users (time spent ~3hr)

- User1: username: `lisa` password:`sa`
- User2: username: `roberto` password:`sa`
- User3: username: `chad` password:`sa`

Spring

For the login part I needed to add new model `User` to the backend. I specify the foreign key in the `Pet` model, one user may have many pets.

Then I create `UserRepository` for storing User models, `UserService` for login logic and `UserController` for exposing the functionality to the endpoint. I also make a migration with liquibase for creating new table and populating it with 3 User account records. I do not encrypt password in the database for the simplicity of the application.

Angular

For the login I create new folder `authentication`, where all the piece of code related to auth will be stored.

I create `User` interface and `UserService` to hit the endpoint with the model created in the backend. Then I create `AuthenticationService`, where I implement the login and logout functionality. Logged-in user is stored in localStorage, so that user can stay authenticated the whole session. I use this service in `pet-list` component to draw pets owned only by the current user with one `ngIf` check. I also use `currentUserValue` to correctly update and create new pets, so that new pet is added with the current user and current user can modify only the pets he created previously.

There is also created `AuthGuardService`, so that the user will be redirected to the login page if not authenticated.