JENSEN yrkeshögskola
TRÄNING FÖR VERKLIGHETEN

***Course Name: Web Framework for JavaScript***
***Class: Frontend Developer Web Security***
***Term: VT 2024***

# DOG BOOK

**INTRODUCTION**

| | |
|---|---|
| Background description, question, delimitation and goals | Background: In this task you will build a web application with React. The application is intended as an admin tool for dog daycare staff to know which dogs are in the daycare and which of them are friends. A user can post a profile picture, short description, and a friend list for a dog. You will use previous knowledge in backend development as new knowledge in front-end development.<br><br>Goal: A social network for dogs<br><br>Limitation: It is allowed to use third party libraries/frameworks for this task. as long as React is used for the main task itself and JavaScript (ES6) is used. |
| | |
| Why should you do this? work? | Purpose:<br>• The student must independently learn to apply web frameworks i the development of applications<br>• The student should gain extended knowledge of Architecture patterns, for example Model-View-Controller (MVC)<br>• The student will acquire skills in Developing applications by using common architectural patterns, frameworks, libraries and design patterns<br>• The student must acquire skills in combining the front-end with back-end |
| What will you deliver? | The following must be delivered:<br>1. All project files 2. A README<br>file with instructions on how to start the application and testing<br>3. Video on review of application |

**YOUR PROJECT TASK**

| | |
|---|---|
| What will you do? **Node & Webpack / create-react-app /vite** | |

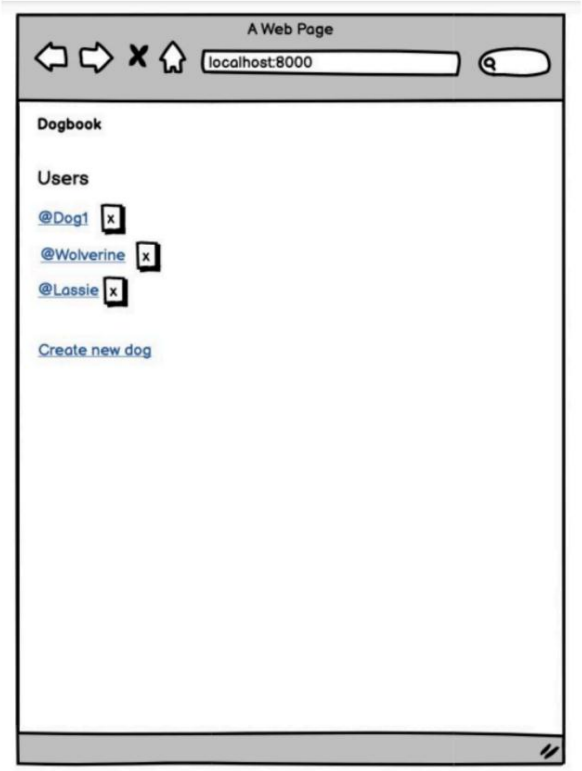
In this task, you can choose to either create your own build chain using Node, NPM and Webpack or use create-react-app/vite (which we have used during the lessons) if you want to focus more on React

**Structure**

The first part of the task is to create a structure for the application. In this task you do not need to think about responsive design and we only use 1 HTML document (index.html) which will be the entry point for your application. You will create a Single Page Application (SPA), so all HTML and CSS are created dynamically, in the same way as you have seen during the lessons.

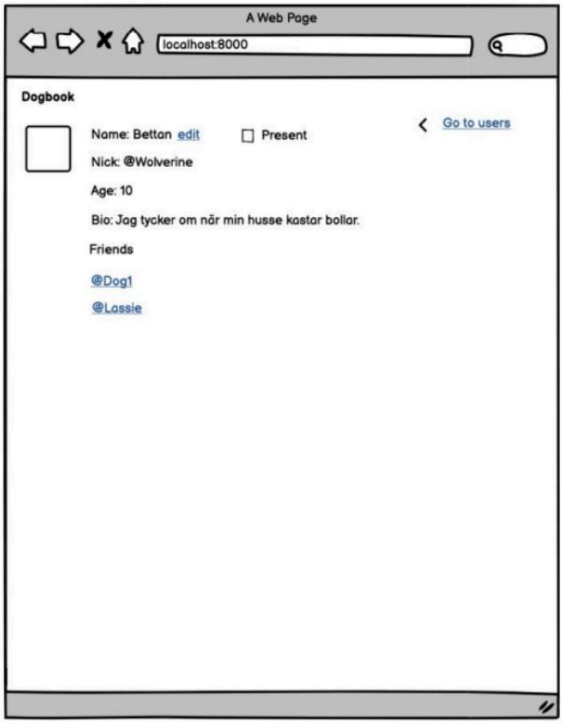
**Design specification**

The following design is just a suggestion of how it might look, you are free to style and include more information. The design presented can be considered a minimum level of what is required.




**Home page**

In this case, the user has reached the start page. All the dogs that are in the yard are listed here. In this list, all dogs have blue links, but in your application, those in the yard should be marked with green text, and the others with red text.
There are links to the dogs' profiles (ex: @Wolverine), a button to remove a dog (the button with 'x') and a link ("Create new dog") to create a new dog.

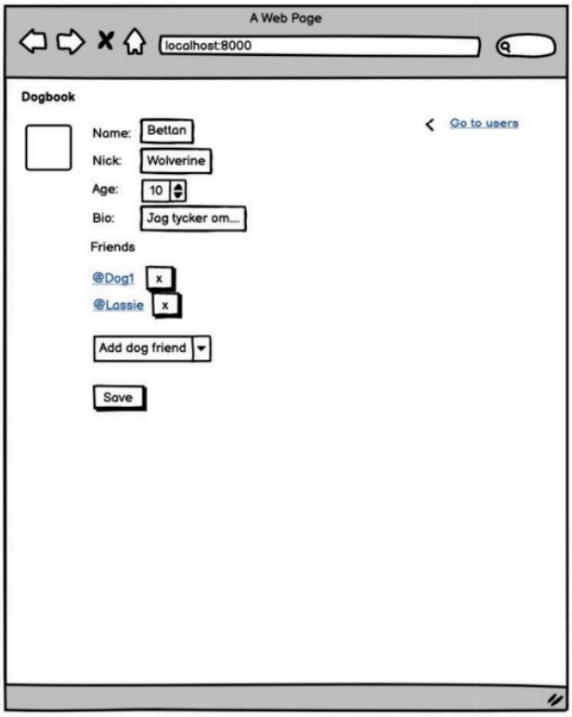



**Profile page**

In this case, the user has arrived at a dog's profile page. Here the user sees:

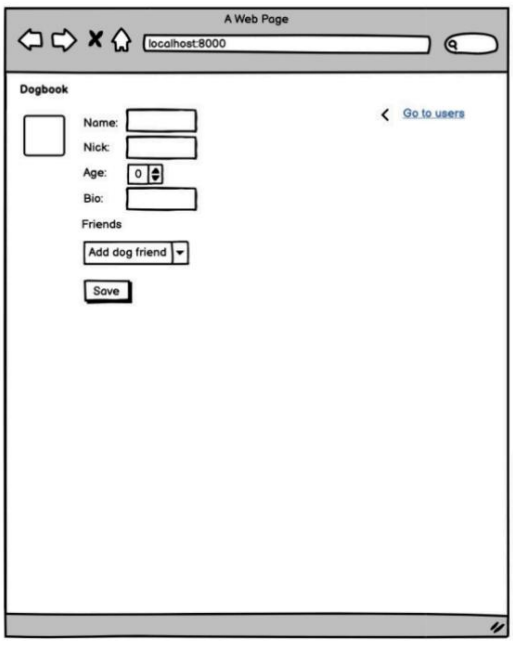
- Picture • Edit link
- Name
- Nickname (on the page) • Age
- Short description • Friends list • Checkbox for presence • Link back to start page

The image of the dog will be taken from an API (https://dog.ceo/api/breeds/image/random) . This API requires no key and returns an arbitrary image of a dog which serves our purposes of just having a static image of a dog since we don't have any real dog images




**Edit page**

In this case, the user has chosen to click on the "Edit" link on the profile page to edit information about the dog. Here the user can add and remove dogs from the profile's friend list. Keep in mind that when the user has removed a dog from the start page, this must also be reflected in all friend lists that the dog was in, i.e. they must be removed from all dogs' friend lists.




**Create page**

| | |
|---|---|
| | In this case, the user has clicked on “create new dog” on the home page. Here the user fills in information for a new dog.<br><br>**Save data**<br><br>To save data, you use a Node API (probably with Express, which you already know) and MongoDB as a database.<br><br>**JavaScript**<br><br>In order to successfully design the application with good structure and modularity, you should make separations of logic in the application with components and functions. You can use either Class components or Page 5 of 7 Functional components.<br>What you should think about is structuring the application so that it is easy for the reader of the code to follow the structure. You should preferably have the following component structure:<br><br>• App - Main component that displays the current view • Start - Home page view • Profile - Profile page view • Edit - Edit page view • Create - Create page view<br><br>Of course, this is just a suggestion, but it should be easy to follow the logical thread, and main components like these above (or those of your choice), should be separated into their own file. |
| How will you solve the task? | 1. Create a folder on your computer with the name "submission task 3".<br>2. Initialize a project in the create-react-app folder.<br>3. Create the following files: start.js, profile.js, edit.js and create.js.<br>4. Add React components with hardcoded html that represent the views in the spec 5. Start by adding logic, etc. 6. Use components where you see it is appropriate so you don't repeat yourself unnecessarily 7. Use console.log extensively for to validate your work at all times during the process. |

## SUBMISSION AND ACCOUNTING

| | |
|---|---|
| Submission | Submission takes place via LearnPoint **date *7/4 no later than 23:59***<br><br>• Submission of zip with project files<br><br>• README file with instructions for running and testing the application<br><br>• Video on review of application (max. 10 min) |

## ASSESSMENT AND FEEDBACK

| | |
|---|---|
| Assessment takes place against the following grading criteria: | The grading criteria for Pass and Pass with Distinction are:<br><br>**Passed** •<br>The student must have demonstrated knowledge of Component-Container thinking (architectural pattern) and how components cooperate in React with props • The student must have demonstrated knowledge of how React handles state and effects via the design pattern that React has built into the framework. •<br>The student must have demonstrated knowledge of how React's life cycle works. |

| | |
|---|---|
| | • The student must have demonstrated knowledge of testing and written tests with Jest (library) for at least 3 of the components<br>• The student must have used a self-developed backend API with Node that is used to save data.<br><br>**Well passed** •<br>The student must have demonstrated knowledge of how functional components and hooks work in React •<br>The student must have demonstrated knowledge of React Router (library) and with the help of this library manage navigation in an appropriate manner<br>• The student must have demonstrated knowledge of the Context API (design pattern) and used this appropriately to manage state in the application. • The<br>student must have demonstrated knowledge of end-to-end testing and written at least 1 end-to-end test for the application. |
| Feedback | Verbal feedback is given to the student at the reporting time.<br>The student receives written feedback via the learning platform LearnPoint **by 21/4 at the latest** |