Application Task for Full Stack Web-Developers

In this task, you will create a small application that reaches through all layers of our web stack: Front-end, back-end and finally the database.

Tech stack

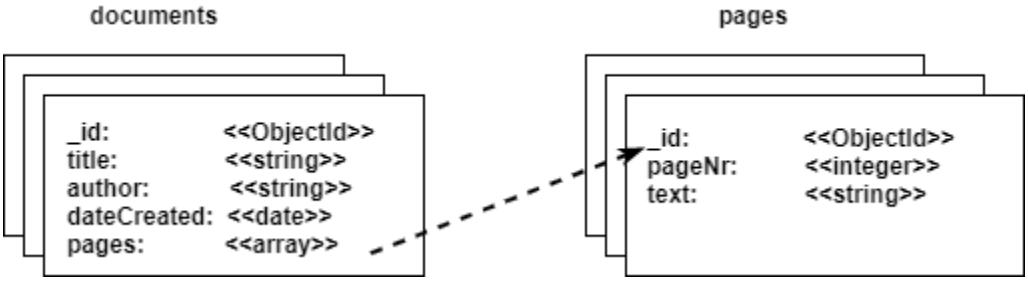
To solve the task, you are free to choose from the following frameworks:

- A modern component-based front-end, for example…
  - Angular
  - Vue
  - React
- A back-end that is based on Typescript (or JavaScript), for example…
  - Nest
  - Express
  - Node
- A document-based NoSQL-database
  - Preferably use the MongoDB access that we set up for you (see below)
  - Or set up your own database, on demand we can provide you a dump of the data

> In your daily work at Elevait, you will be using the MAN stack: MongoDB, Angular and Nest.
> We are always happy to see submissions with these frameworks, but it really is no must-have. 😉

Test database

For the task, a small dataset of documents and their pages with the following structure is required:



As mentioned above, you can set up your own database - just request the data dump and we will sent it to you. Alternatively, you can access and use our test database with the following credentials:

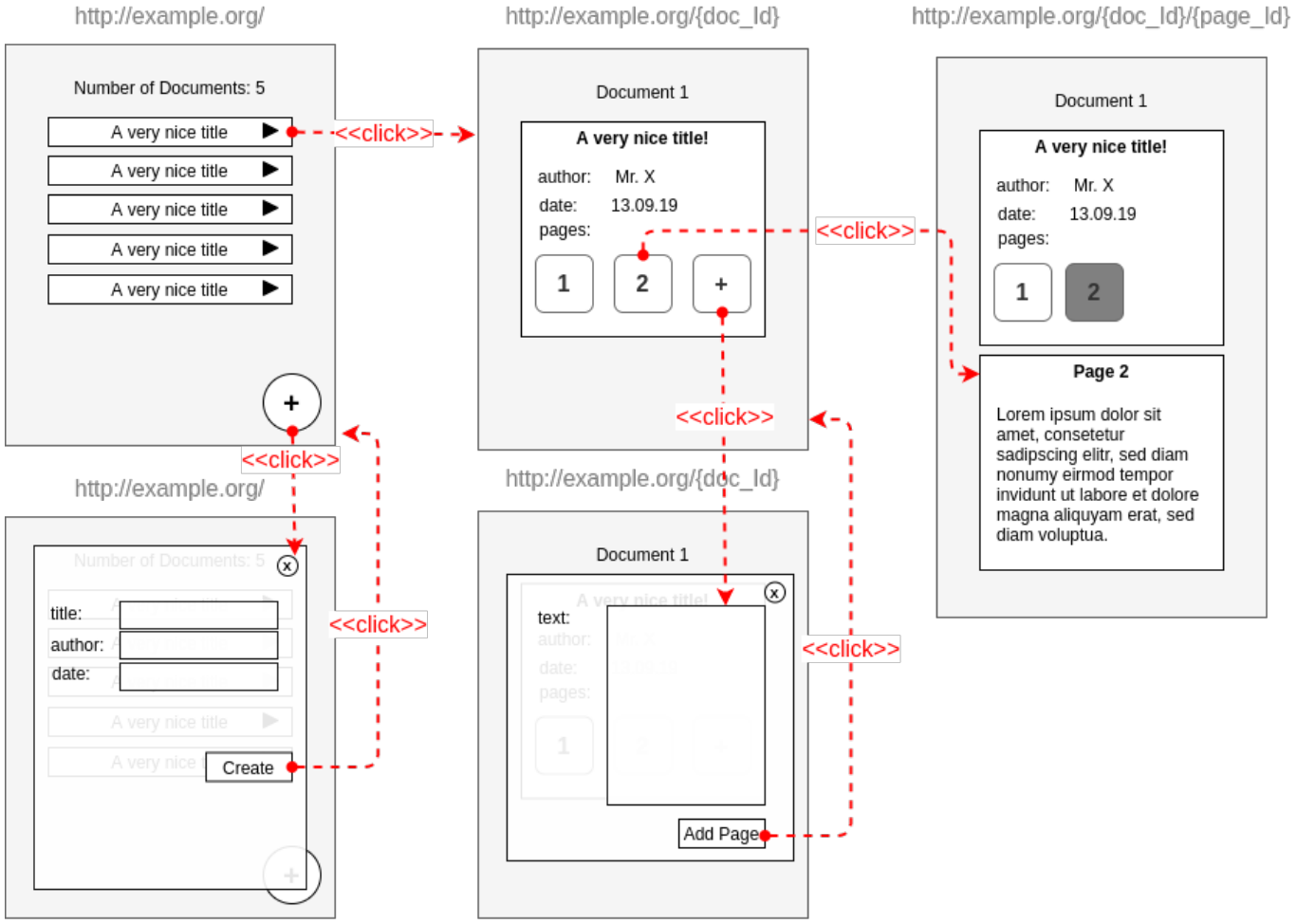
> **URL:** 94.130.203.236:27022
>
> **Name of Database:** doc_viewer
> ⚠️ use this DB as the authentication source
>
> **User:** doc_viewer
>
> **Password:** BJJjtTPyUV6XPd7x

Views

The application should contain basically two different views:

1. Initially, a user should see the *overview of all documents* stored in the collection **documents**.
   a. New documents may be created by clicking the plus button at the bottom of the list. When clicking it, a popup prompts to specify the title, author, and date. After creation, the new entry is shown in the list.
   b. After clicking on a document, the view should change to the document details view.
2. The *document details view* shows the meta-information of a document (title, author, and date).
   a. Each linked page from the **pages** collection should be visible here. When a page gets clicked, the view additionally shows its text and page number.
   b. Similar to the overview, it should be possible to create a new page for a document by clicking the plus button at the end of the list. Again, a popup appears and lets the user fill in the required info. After submitting a new page, it appears in the list.

The following mockups should clarify the intended structure:



Requirements

Must-haves

- [ ] A front-end that accesses documents and pages of a database through the back-end
  - [ ] Choose from the technologies mentioned above for each of the layers
  - [ ] If a framework gives you the choice: Prefer using Typescript over JavaScript
- [ ] Separate the code for documents and pages
  - [ ] Different back-end endpoints for accessing each database collection
  - [ ] Single front-end components with proper URL routing (see mockups)
- [ ] Handle edge cases in the implementation
  - [ ] Consider that the data may be incomplete (missing fields, etc.)
  - [ ] Behavior of the front-end if the back-end does not respond
- [ ] Use a public git repository (e.g. GitHub) to maintain your code and send its link to us

Optional

- [ ] Think about the separation of reusable components
- [ ] Add any kind of unit / e2e testing
- [ ] Implement a proper state management (Redux, Akita, …)
- [ ] Additional features for the documents / pages lists like

- [ ] Filtering
- [ ] Pagination
- [ ] Deletion
- [ ] Editing of entries