Node.js, Express, MongoDB, TypeScript, Frontend Assessment

Estimated time: 2.5 hours

You need to build a small part of the API for a blog, and a page that uses the API. The blog backend use a RESTful Web Service that can Create/Retrieve/Update/Delete articles.

Build the API using Node.js + Express, along with MongoDB (accessed via Mongoose) as the database.

Build the frontend using HTML + CSS + TS + Bootstrap and lists the articles in a card layout.

IMPORTANT NOTE: In the interest of time, it is suggested you complete an implementation of this task by building an Article model without comments embedded within first. Once you do that you can add comments and make necessary code changes.

The Model, API and Frontend application requirements are described below.

Model

The MongoDB database for this application has a single collection for articles. Since data is denormalized in a MongoDB database, every article document has the associated comments.

Article (with Comment) Model specifications

- _id: ObjectID, Auto-generated
- author: String [required]
- title: String [required]
- abstract: String [required]
- body: String [required]
- imageUrl: An image URL [String]
    - createdAt: Date, set automatically to the date the article was created on [required]
- comments: An array where every item is an embedded document with the following structure
    - commenter: String [required]. This is the name of person commenting on the article.
    - title: String
    - comment: String [required]
    - createdAt: Date, set automatically to the date the review was added on [required]

Notes:

1. You will need to define 2 schemas – one for Article, and one for Comment and use the Comment's schema within Article. **Refer**: https://mongoosejs.com/docs/subdocs.html
2. As suggested above, do not include comments in your initial implementation. Add comments to the model only once you have completed API implementation of Article without comments.

The requirements for the API follow.

API

Action	HTTP Verb	URL	What is sent in the request	HTTP Status Code (success)	What is received in the response
Retrieve all articles (comments not included)	GET	http://localhost:4201/articles	-	200	Array of articles (comments not present)
Retrieve all articles (comments included)	GET	http://localhost:4201/articles ?include=comments	-	200	Array of articles (comments present)
Retrieve article with given id	GET	http://localhost:4201/articles/1	-	200	Object with details of article with id = 1. Comments MUST be included.
Create a new article	POST	http://localhost:4201/articles	Object with details of new article (comments are not sent)	201	Object with details of new article (including id)
Update an article	PATCH	http://localhost:4201/articles/1	Object with new details for article with id = 1	200	Object with new details for article with id = 1
Add a comment	POST	http://localhost:4201/articles/1/ comments	Object with details of new comment	204	-
Delete an article with given id	DELETE	http://localhost:4201/articles/1	-	204	-

Note:
The tables show response codes in case of success only. The API will need to handle all error cases appropriately. Some of the popular codes in case of error are.
1. 400 – Bad Request. The data sent in the request was malformed.
2. 403 – Not Authorized
3. 404 – Resource not found
4. 409 – Duplicate resource exists and hence new resource was not created
5. 500 – Internal server error

The requirements for the frontend application follow.

FRONTEND APPLICATION

The frontend of this application is to be built using Bootstrap and consists of 2 pages.

1. Home page with a welcome message
2. An articles page which lists articles in a Bootstrap card layout – there should be 3 cards per row on large devices, 2 cards per row on medium devices and 1 card per row on small width screens. Every card this has the image of the article, and the author name, title, and the abstract of the article below it.

User should be able to switch between these 2 pages using a navigation menu.

Make sure to use TypeScript in the frontend application script that makes the Ajax call to fetch the articles. Compile it down to JavaScript which is then included in the HTML pages.