Content Delivery App

The Livingdocs Beta (beta.livingdocs.io) provides an API where articles and pages can be fetched. The json response contains the HTML of the rendered article and metadata. So far, customers integrate their frontend projects with our backend. Livingdocs does not provide a scalable layer to deliver full HTML pages over the web yet. This is what we are going to build for our new customer Bluewin.

Please build a simplified prototype to deliver web pages on top of the Livingdocs Beta API

Task

Simplest request cycle

Create a standalone node application that handles web requests (we favor express.js)

- On each request, fetch an article from the Livingdocs Beta API. To keep it simple, you can use a 1:1 mapping from the request URL to the API url (eg. delivery.com/public/publications/2871 calls api.com/public/publications/2871)
- The API only returns the HTML of the content. Please create a dummy template (html, body, dummy header, dummy footer) and render the article HTML to a placeholder in this layout. Serve a fully functional and rendered site to the user.

Improvements

Do as many as you have time for, we don't expect you to make it to the end. Follow the given order.

- Use the metadata from the API to dynamically fill the meta tags in the HTML head
- Make it easy to reuse your application for multiple customers. As they will have different layouts, API endpoints, etc. make them configurable
- Enable speaking urls on the delivery app. Make the mapping from delivery request to the Livingdocs API configurable (eg. delivery.com/sports/football/2871 maps to api.com/public/publications/2871)
- Support multiple layouts. For example pages in the category sports have layout A, but all other categories have layout B
- Create a dynamic navigation assume/fake an API endpoint that returns you a json tree structure with links to Livingdocs pages
- Make layouts dynamic. For example pages in the category sports have google ads, but all other categories have bing ads

Questions

We'd like to have a discussion on caching. No need to write down everything in detail, bullet points are sufficient.

- What do we need to do in order to scale the delivery layer to hundreds of requests per second?
- Describe a caching strategy on the delivery layer, also describe cache invalidation
- Describe the benefits but also the drawbacks of your chosen method and why it is the best solution in the circumstance

API endpoints

You can use the following API calls for your dummy:

Articles

- https://production-server.hosted.livingdocs.io/public/publications/2912
- https://production-server.hosted.livingdocs.io/public/publications/2913
- https://production-server.hosted.livingdocs.io/public/publications/2914
- https://production-server.hosted.livingdocs.io/public/publications/2915
- https://production-server.hosted.livingdocs.io/public/publications/2916

Pages

- https://production-server.hosted.livingdocs.io/public/publications/2918

For reference, you can request a rendered version (this is our non-scaling, non-cached dummy delivery that will be replaced by your code), by constructing a url as follows: https://production-server.hosted.livingdocs.io/articles/2918.html?project_id=319 - please note that *articles* is used in the url no matter if it's a page or an article. Just swap out the ID, project_id stays the same.

This dummy API is not very performant, please don't hammer it with requests