**Full-Stack Developer Take-Home Project Documentation**

**Introduction**

This project involves building a simple content management platform using **Strapi** as the backend CMS (Content Management System) and **Next.js** as the frontend framework. The platform allows users to manage and display blog posts and videos. The main objectives include setting up Strapi for content management, implementing custom logic for calculating read times and video durations, and building a user-friendly interface using Next.js to display the content with features like search, pagination, and filtering.

**Project Overview**

* **Backend**: Strapi is used as the headless CMS to manage the content.
* **Frontend**: Next.js is used to fetch and display this content to users.

**Backend Setup with Strapi**

1. **Strapi Initialization and Collection Setup**

A screenshot of a computer

Description automatically generated

* + This image shows the terminal output when you run the Strapi development server.
  + **Key Points**:
    - Strapi is started and listens on http://localhost:1337.
    - Collections for Blogs and Videos are set up in the Strapi admin dashboard, allowing content management through a user-friendly interface.

1. **Strapi Admin Dashboard:**

**Blogs:** It calculated the estimated reading time for each blog post based on the length of the content. This helps users understand how long it might take to read a particular blog post.

A screenshot of a computer screen

Description automatically generated

**Videos:** Added functionality to automatically calculate and store the duration of the video. This ensures users are informed about how long a video is before they watch it.

A screenshot of a computer

Description automatically generated

* + This image shows the JSON response from the Strapi API both the video and Blogs.
  + **Key Points**:
    - The JSON object contains data fields for blogs, including Title, Slug, Publish\_Date, and Body.
    - It also shows pagination metadata, which is helpful for managing large sets of data.

**Frontend Setup with Next.js**

**Data Fetching**

* The frontend fetches data from the Strapi API, retrieving both blog and video collections to be displayed on a single page.

1. **Display Content**

A screenshot of a computer

Description automatically generated

* + This is the landing page of the Next.js app.
  + **Key Points**:
    - It provides links to navigate to the Blogs and Videos sections.
    - The page is clean and minimalistic, focusing on guiding users to the content sections.

**List View:**

* The main page displays a list of all available blogs and videos. Users can choose to view either blogs or videos by clicking the respective buttons.

**Details View:**

* When a user clicks on a blog post or video, they are redirected to a detail page where more information is displayed. For blogs, this includes the title and body; for videos, this includes the video player and description.

1. **Blogs List Page**

A screenshot of a computer

Description automatically generated

* + This image displays a list of blog entries.
  + **Key Points**:
    - Each blog entry shows the title, publish date, and estimated read time.
    - Clicking on a blog entry redirects to the blog's detail page.

1. **Blog Detail Page**

A screenshot of a computer

Description automatically generated

* + This image shows detailed content for a selected blog.
  + **Key Points**:
    - The page displays the full blog post, including sections like Introduction and Conclusion.
    - It provides an in-depth view of the blog, allowing users to read the complete article.

1. **Videos List Page**

A screenshot of a computer

Description automatically generated

* + This page lists all the videos available in the CMS.
  + **Key Points**:
    - Each video entry includes the title, publish date, and duration.
    - Users can click on a video to view more details or play the video.

1. **Video Detail Page**

A screenshot of a computer

Description automatically generated

* + This image shows the detailed view of a selected video.
  + **Key Points**:
    - The page includes the video player, allowing users to watch the video directly.
    - Additional information such as the video description is also displayed.

**Backend API Integration**

1. **API Calls and Responses**

A screenshot of a computer

Description automatically generated

* + This image demonstrates how Next.js fetches data from the Strapi API.
  + **Key Points**:
    - API endpoints for fetching blogs and videos are utilized in Next.js.
    - The data is structured in a JSON format, which is easy for the frontend to parse and display.

**Search Functionality**

* A search bar is provided to allow users to search through the blog and video titles. This feature filters the list of displayed content based on the user's input, helping them find specific posts or videos quickly.

A screenshot of a computer

Description automatically generated

* + Search for blogs:

A screenshot of a computer

Description automatically generated

* + Search for Videos:

A screenshot of a browser

Description automatically generated

**Pagination**

* Implemented pagination to manage the content list effectively, displaying a limited number of items per page. Users can navigate between pages to explore more content.  
    
  A screenshot of a computer

  Description automatically generated

**Filtering**

* Users can filter content by categories, helping them narrow down their choices based on their interests.
  + Filtering for Blogs:

A screenshot of a computer

Description automatically generated

* + Filtering for Videos:

A screenshot of a computer

Description automatically generated

**Conclusion**

This project demonstrates a typical modern web application setup using a headless CMS (Strapi) and a dynamic frontend (Next.js). By decoupling content management from content display, it offers flexibility and ease of maintenance, making it ideal for growing businesses looking to manage their content efficiently. The screenshots illustrate how backend data is structured and how the frontend displays this data to users, emphasizing the seamless integration between Strapi and Next.js.

**Future Enhancements**

Planned to implement in the future, providing a brief description for each:

**Planned Bonus Features**

* **Simple Caching Mechanism**: To improve response times and reduce server load, a caching strategy will be implemented for API requests.
* **Utilize GraphQL**: Planning to integrate GraphQL to optimize data retrieval by allowing clients to specify exactly what data is needed.
* **SEO Setup**: Aims to enhance the visibility of the web application in search engines through optimized meta tags and structured data.