Functional requirements

The application composes of three main components: article list, article detail and article edit. It is safe to assume that the application will be used by only one client at a time.

Article list

The article list page is available at URL `./articles` and implements the following functionality:

- F01) User can view a paged list of articles by navigating to `./articles`. Each page contains at most 10 articles.
- F02) User can move to next/previous page by clicking `Next`/`Previous` buttons respectively.
- F03) When the user is on the first/last page `Previous`/`Next` button is hidden.
- F04) User can see the total number of pages on the screen.
- F05) For each article in the list there is an article name.
- F06) For each article in the list there is a link with the label `Show`. If a user clicks the link, they are navigated to the article view page - URL `./article/{id}`.
- F07) For each article in the list there is a link with the label `Edit`. If a user clicks the link, they are navigated to the article edit page - URL `./article-edit/{id}`.
- F08) For each article in the list there is a link with the label `Delete`. If a user clicks the link, the respective article is removed. The list of all articles is refreshed, and the current page is preserved unless the current page is the last page with no articles. In such a case, the page is changed to the last valid page. The delete operation must be handled using AJAX with HTTP DELETE request. The list update must be handled on client side using JavaScript, you are not allowed to perform full page reload.
- F09) There is a button with the label `Create article`.
- F10) If a user clicks `Create article` button a dialog is shown. The dialog contains a text input field with the label `Name`, which can be used to specify the name of a new article. The input field is required to be non-empty and the length is limited to 32 characters. The dialog contains buttons `Create` and `Cancel`.
    - F11) If a user clicks on the `Cancel` button, the dialog is closed.
    - F12) The `Create` button is enabled only if the text input for the article name is not empty.
    - F13) If a user clicks the `Create` button, a new article is created. The article is created with given name and empty body. Next, the user is redirected to the article edit for the new article.

Article detail

Article detail page is available at URL `./article/{id}`, where `{id}` is an article identifier. The page implements the following functionality:

- F14) User can view the article detail page by navigating to URL `./article/{id}`, where `{id}` is a valid article identifier.
- F15) If an article with given `{id}` does not exist, then the server returns an empty document with status code 404.
- F16) If the article exists, then the article's name and content, and buttons `Edit`, `Back to articles` are shown to the user.
    - F17) If a user presses the `Edit` button, then the user is redirected to the article edit page with URL `./article-edit/{id}`.
    - F18) If a user presses the `Back to articles` button, then the user is redirected back to the first page of the article list.

Article edit

Article edit page is available at URL `./article-edit/{id}`, where `{id}` is an article identifier. The page implements the following functionality:

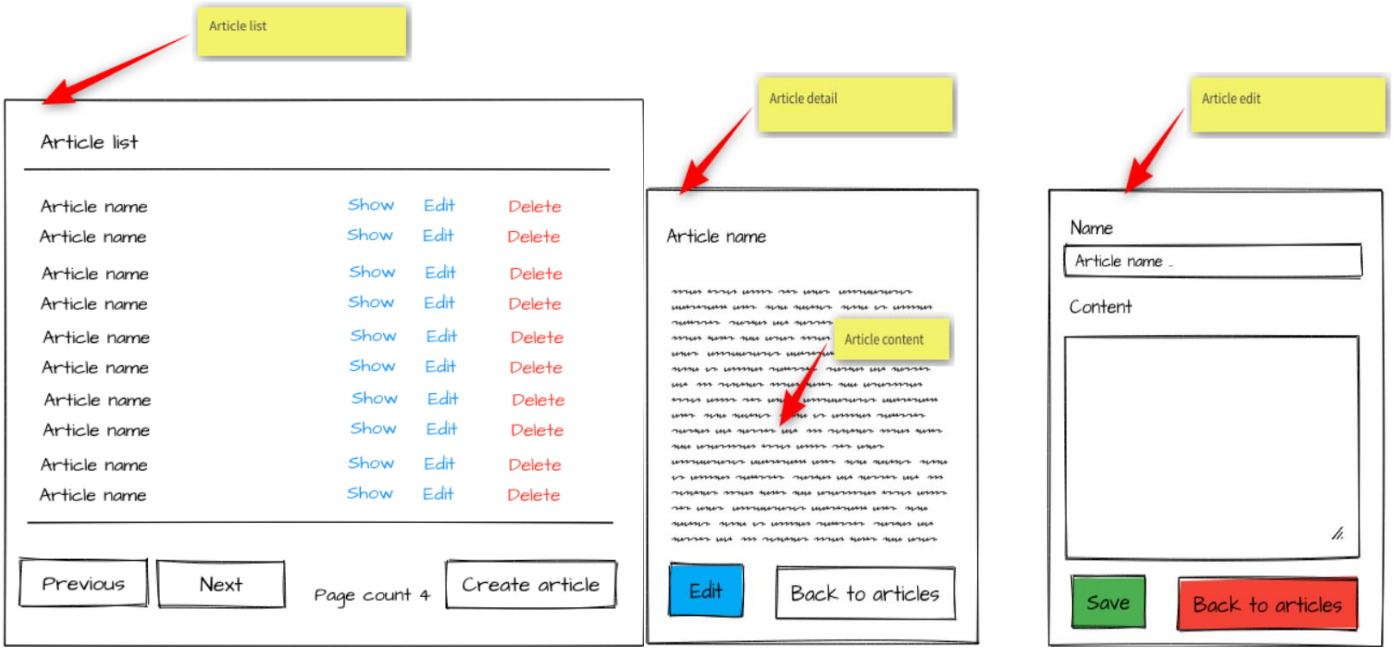
- F19) User can view the article edit page by navigating to URL `./article-edit/{id}`, where `{id}` is a valid article identifier.
- F20) If an article with given `{id}` does not exist, then the server returns an empty document with status code 404.
- F21) If an article exists, then edit form, button `Save` and button `Back to articles` are shown to the user.
- F22) If a user presses the `Back to articles` button, the user is redirected back to the first page of the article list.
- F23) The edit form contains text input for article name with label `Name` and a textarea for article content with label `Content`. The size of an article content is limited to 1024 characters. The size of an article name is limited to 32 characters.
- F24) User can use the `Save` button if and only if the text input (`Name`) is not empty. It the text is empty, then the button should be disabled or clicking the button must not lead to any action. In the second case (no action) the user should be notified.
- F25) If a user presses the `Save` button, all the changes are saved and the user is redirected back to the first page of the article list.

Nonfunctional requirements

- N01) Application must not use any third-party libraries or code/style snippets. You are allowed to draw inspiration from any publicly available sources. You must be sole author of all the code.
  Failure to explain any part of the code is sufficient reason for not accepting the solution.
- N02) Articles are stored in SQL database.
- N03) Database access configuration is stored in a standalone file called `db_config.php` created from the following template.
```
<?php
$db_config = [
  'server'   => 'localhost',
  'login'    => '<my-username>',
  'password' => '<my-password>',
  'database' => 'stud_<my-username>',
];
```
- N04) Article detail page is rendered completely on server-side. JavaScript should not be used to communicate with the server.
- N05) Article edit detail page is rendered completely on server-side. JavaScript should not be used to communicate with the server.
- N06) Article list page is server-side rendered and contains all articles. Pagination and dialog are handled on a client-side using JavaScript.
- N07) PHP application must utilize the front-controller design pattern. This means that you will need to redirect HTTP requests to your `index.php` file. You can do this using `.htaccess` with the following content (replace skoda with your username):
```
RewriteEngine On
RewriteBase /~skoda/cms/

Options -MultiViews

RewriteCond %{REQUEST_URI} !index\.php
RewriteRule ^([-a-zA-Z0-9_/]+)$
index.php?%{QUERY_STRING}&page=$1 [L]
```
  You should place the file next to your `index.php` file. In addition, routing and dispatching must not happen in a same function.
- N08) All the HTML pages must be properly styled using CSS, inspired by given mockups.



- N09) PHP application must utilize the model-view-controller / model-view-presenter design pattern. The view is not allowed to communicate with model directly. There must be a separate controller for each endpoint (./article/, /article-edit/{id}, ...).
- N10) If you plan to employ any type of transpilation, like Typescript, let us know in advance, so we can specify the rules.
- N11) All static (non-generated) JavaScript and CSS styles must be in their respective files. You can put generated JavaScript and CSS styles into PHP files.

Data

Data should be stored in the MySQL database. Database schema must contain table `articles`, where all the articles are stored. The `articles` table must contain columns `name` and `content`, which store article name and content respectively. You should prepare some test data to support smooth presentation of your application.

External resources

Any resources on the internet, namely StackOverflow, Github Copilot, or Chat GPT, can only be used to solve minor problems, not to solve your problem directly. Thus, searching/prompting for class and function implementations and architectural composition is forbidden. Searching for feature documentation, examples, and bugs is fine.

Although many tasks are solvable by today's AI, the restrictions ensure that you understand the core of web application development, which is necessary for more complex problems where skilled programmers are still needed. The rule of thumb is that you use AI, not the

other way around. As a result, you must be able to fully explain, and provide reasoning, behind all of the code. Failure to do so will result in failing the test.

You must be able to demonstrate individual parts of your solution by providing examples which are not part of your source code.

Details

Make sure your implementation fulfils the following conditions:

- Valid HTML and CSS.
- Systematically handle insertion of values to HTML output (templates) to prevent script-injection attacks.
- Systematically handle querying of the database to prevent SQL injection. Use reasonable abstractions for the data layer.
- The used forms appropriately validate inputs using HTML5 validation attributes.
- All user inputs and data should be validated or appropriately sanitized.
- Errors are displayed to the user in a sensible way.
- Reasonable code-style and function/variable naming.