Mini Router Documentation

The Mini Router is a simple Express.js router designed to handle basic CRUD (Create, Read, Update, Delete) operations for a blog post API. It utilizes the `express.Router()` to define and manage various endpoints for creating, retrieving, updating, and deleting blog posts.

1. GET

baseURL /post/:postId

- Description: Retrieves a single blog post by its postId.
- Response: Returns the post details as a JSON object.
- Example Response:

```json
{
  "id": 98148,
  "title": "Hassan_Nadeem",
  "text": "I Have Done My Assignment"
}
```

2. GET

baseURL /posts

- Description: Retrieves all blog posts stored in the database.
- Response: Returns an array of all blog posts in JSON format.
- Example Response:

```json
[
  {
    "id": "9bMgsP7F",
    "title": "Sample Title 1",
    "text": "This is the content of the first post."
  },
  {
    "id": "2Z4XJDkP",
    "title": "Sample Title 2",
    "text": "This is the content of the second post."
  }
]
```

3. POST

baseURL /post

- Description: Creates a new blog post with the provided title and text.
- Request Body: The request must include a JSON object with "title" and "text" properties.

```json
{
  "title": "New Post Title",
  "text": "The content of the new blog post."
}
```

- Response: Returns a success message with the creation date.
- Example Response: `post Created at 2023-07-31`

4. PUT

baseURL/post/:postId

- Description: Updates an existing blog post with the given postId.
- Request Parameters:
  - `postId`: The unique identifier of the post to update.
- Request Body: The request must include a JSON object with "title" and "text" properties to update the post.

```json
{
  "title": "Updated Post Title",
  "text": "The updated content of the blog post."
}
```

- Response: Returns a success message if the update is successful.

- Example Response: `Post Updated successfully`

5. DELETE

baseURL /post/:postId

- Description: Deletes an existing blog post with the given postId.

- Request Parameters:

  - `postId`: The unique identifier of the post to delete.

- Response: Returns a success message if the post is deleted successfully.

- Example Response: `Post deleted Successfully`

Important Notes

1. The server should be stateless, but in this implementation, the `posts` array stores the blog post data, which is not ideal for a production environment. A real-world application would use a database to store and manage the data.

2. The current implementation uses the `mongodb` package to interact with a MongoDB database. It is assumed that the MongoDB connection has been established beforehand, and the `client` object is accessible from the `mongodb.mjs` file.

3. The `nanoid` package is used to generate unique identifiers for each blog post. The unique identifiers (`id`) are used to perform CRUD operations on specific posts.

4. The code contains a few minor issues, such as typos and inconsistencies (e.g., `tittle` instead of `title`). Ensure to review and fix these issues before deploying to production.