

Build a Blog Website with Strapi & Vanilla Javascript

© L. Hernández, 2025

Setting up Strapi Backend

Creating Project Directory

First, we'll create a directory for our blog project.

This directory will contain project's Strapi backend folder and JS frontend folder.

```
mkdir blog-app && cd blog-app
```

Setting up Strapi Backend

Creating Strapi Project

Next, we'll create a new Strapi project using one of the following commands:

```
npx create-strapi@latest backend --quickstart
```

```
yarn create strapi-app@latest backend --quickstart
```

Setting up Strapi Backend

Creating Strapi Project

Now, we'll proceed with the installation and login to our Strapi account when prompted.

We'll have to signup instead if this is our first time using Strapi.

This will create a new Strapi project in the **blog-app** directory and install all necessary dependencies.

Setting up Strapi Backend

Creating Strapi Project

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
○ (base) juliet@Juliets-MacBook-Pro blog-app % npx create-strapi@latest backend --quickstart
Need to install the following packages:
create-strapi@5.3.0
Ok to proceed? (y) y
```

Strapi v5.3.0 🚀 Let's create your new project

We can't find any auth credentials in your Strapi config.

Create a **free account on Strapi Cloud** and benefit from:

- ✦ **Blazing-fast** ✦ deployment for your projects
- ✦ **Exclusive** ✦ access to resources to make your project successful
- An ✦ **Awesome** ✦ community and full enjoyment of Strapi's ecosystem

Start your 14-day free trial now!

? Please log in or sign up. (Use arrow keys)

> Login/Sign up

Skip

Setting up Strapi Backend

Creating Strapi Project

After creating our new Strapi project, we'll navigate to the project folder:

```
cd backend
```

Then, we'll start up the Strapi server (if we're not automatically directed to the Strapi panel on our browser) using one of the following commands:

```
npm run develop
```

```
yarn develop
```

Setting up Strapi Backend

Creating Strapi Project

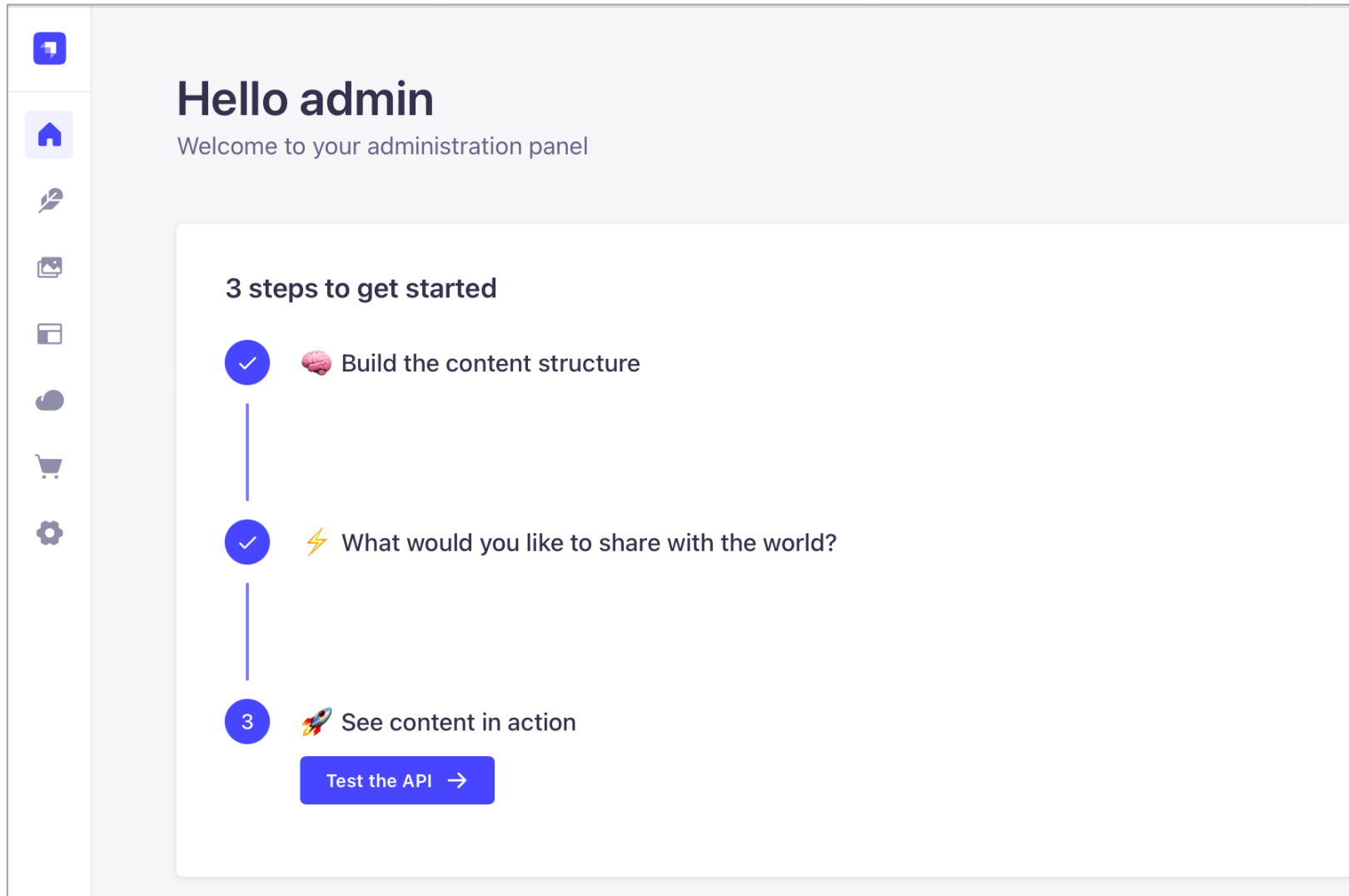
This will open up the Strapi admin dashboard of our newly created project at this URL:

`http://localhost:1337/admin`

We'll then input our details, including name, email, and password to access our admin panel.

Setting up Strapi Backend

Creating Strapi Project



Setting up Strapi Backend

Creating Strapi Project

This is where we'll create collections, manage content types, and configure API settings for our blog website.

Creating Strapi Collection Types

Create Category Collection Type

To get started, we'll first create content type for any collection by navigating to **Content-Type Builder** page.

We'll click "**Go to the Content type builder**" button on dashboard or click icon on sidebar navigation labelled "**Content-Type Builder**".

Creating Strapi Collection Types

Create Category Collection Type

We'll then create our first collection by clicking "**Create new collection type**" option.

A modal opens up where we'll be prompted to input our first collection name (**Category**), then hit "**Continue**".

Creating Strapi Collection Types

Create Category Collection Type

CT

Create a collection type

×

Configurations

A type for modeling data

BASIC SETTINGS

ADVANCED SETTINGS

Display name

Category

API ID (Singular)

category

The UID is used to generate the API routes and databases tables/collections

API ID (Plural)

categories

Cancel

Continue

Creating Strapi Collection Types

Create Category Collection Type

Next, we'll be asked to select our collection field type.

Choose field types, i.e, **Category** collection will have two fields: **name** and **slug**.

Select a field for your collection type

DEFAULT

CUSTOM



Text

Small or long text like title or description



Boolean

Yes or no, 1 or 0, true or false



Rich text (Blocks)

The new JSON-based rich text editor



JSON

Data in JSON format



Number

Numbers (integer, float, decimal)



Email

Email field with validations format



Date

A date picker with hours, minutes and seconds



Password

Password field with encryption



Media

Files like images, videos, etc



Enumeration

List of values, then pick one



Relation

Refers to a Collection Type



UID

Unique identifier



Rich text (Markdown)

The classic rich text editor

Creating Strapi Collection Types

Create Category Collection Type

name field will be a text (short text) type, so select 'Text' option and tick 'Short text' radio button.

slug field will be a 'UID' type, so we'll select 'UID' option.

Creating Strapi Collection Types

Create Category Collection Type

[← Back](#)

Category

Edit

+ Add another field

✓ Save

Build the data architecture of your content

☰ Configure the view

NAME	TYPE	
<div>Aa</div> name	Text	<div><div></div><div></div></div>
<div></div> slug	UID	<div><div></div><div></div></div>
<div></div> posts	Relation with <i>Post</i>	<div><div></div><div></div></div>

+ Add another field to this collection type

Creating Strapi Collection Types

Create Author Collection

Authors will manage all details about blog post authors.


We'll go to **Content-Types Builder** again, create a new collection type called "**Author**" and add fields for **name** (Text), **bio**(Text), and **bioImage** (Single Media).

Creating Strapi Collection Types

Create Author Collection

[← Back](#)


Author













 Edit

+ Add another field

✓ Save

Build the data architecture of your content

 Configure the view

NAME	TYPE	
<div> name</div>	Text	<div></div>
<div> bio</div>	Text	<div></div>
<div> bioImage</div>	Media	<div></div>
<div> posts</div>	Relation with <i>Post</i>	<div></div>

+

Add another field to this collection type

Creating Strapi Collection Types

Create Post Collection

Post collection holds the main content of our blog.

We will create a collection type called "**Post**" with fields for **title** (Text), **content** (Rich Text-Markddown), **excerpt** (Text), **featuredImage** (Media), and any other relevant fields like **slug** field (UID).

[← Back](#)

Post






















 Edit

[+ Add another field](#)

 Save

Build the data architecture of your content

 Configure the view

NAME	TYPE	
 title	Text	 
 content	Rich text (Blocks)	 
 slug	UID	 
 featuredImage	Media	 
 author	Relation with <i>Author</i>	 
 categories	Relation with <i>Category</i>	 
 excerpt	Text	 

[+ Add another field to this collection type](#)

Creating Strapi Collection Types

Author & Post Strapi Relation



To link authors with their blog posts, we need to create a relationship.

In the **Post** collection, we'll add a field of type **Relation**. Then we'll set it to relate one Author to many Posts.

This way, each post can only have one author, but an author can have multiple posts.

Creating Strapi Collection Types

Author & Post Strapi Relation

 Post / Author 







Edit Author

Refers to a Collection Type

BASIC SETTINGSADVANCED SETTINGS

Post

Field name
author



Author ▼

Field name
posts

Author **has many** Posts



Cancel

+ Add another field

Finish

Creating Strapi Collection Types

Author & Post Strapi Relation

 Author / Posts 







Edit Posts

Refers to a Collection Type

BASIC SETTINGS ADVANCED SETTINGS

Author

Field name
posts



Post ▼

Field name
author

Author belongs to many Posts

Cancel

+ Add another field

Finish

Creating Strapi Collection Types

Category & Post Relation

Similarly, we will link categories to posts by adding a **Relation** field in the **Post** collection.

We'll set this to category to many posts, so each post can belong to many categories, and each category can relate to many posts.

Creating Strapi Collection Types

Category & Post Relation

[Category](#) / [Posts](#) ✕

Edit Posts

Refers to a Collection Type

[BASIC SETTINGS](#) [ADVANCED SETTINGS](#)

Category

Field name

posts

→

↔

↗

↘

⌘

↖

Post ▼

Field name

categories

Categories **has and belongs to many** Posts

Cancel

[+ Add another field](#) Finish

Creating Strapi Collection Types

Category & Post Relation

[Post](#) / Categories ✕

Edit Categories

Refers to a Collection Type

BASIC SETTINGSADVANCED SETTINGS

Post

Field name

categories

→

↔

↗

↘

⌘

↖

Category ▼

Field name

posts

Posts **has and belongs to many** Categories

Cancel

+ Add another field

Finish

Setting up Roles & Permission in Strapi

One important step in setting up a Strapi project is to configure roles and permissions to control who can access our blog data and grant public access to view and perform CRUD operations.


Setting up Roles & Permission in Strapi

To set up roles and permission to have public access for our front end to fetch posts, we'll navigate to **Settings → Users & Permissions Plugin → Roles → Public**. Click the 'edit' icon in **Public**.



Setting up Roles & Permission in Strapi

Roles

List of roles



+ Add new role

NAME	DESCRIPTION	USERS	
Authenticated	Default role given to authenticated user.	0 user	
Public	Default role given to unauthenticated user.	0 user	


Setting up Roles & Permission in Strapi


Toggle the Author, Category, and Post collections under "Permission" and tick **find** and **findOne** checkboxes for each of the collection.


[← Back](#) **Public**
Default role given to unauthenticated user.


Permissions

Only actions bound by a route are listed below.

Author
Define all allowed actions for the api::author plugin. 

Category
Define all allowed actions for the api::category plugin. 

Post
Define all allowed actions for the api::post plugin. 

POST  Select all

☐ create

☐ delete

☒ find

☒ findOne

☐ update

Accessing Strapi Endpoints

Strapi automatically generates REST API endpoints for each content type.

To access any API endpoint in Strapi, add the pluralized name of the collection type to the base URL.

Accessing Strapi Endpoints

In our case, for us to access all posts, we can use this endpoint to retrieve all posts in JSON format:

`http://localhost:1337/api/posts`

We can access this endpoint because we made the **/posts** endpoints public.

Accessing Strapi Endpoints

The reason we are getting an empty array JSON response for our endpoint is because we haven't added any post entry in our Strapi admin panel.

Adding entries in each collection

Let's add some entries in each of collection we created in our admin panel.

To do this, we'll navigate to **Content Manager** page and click "**Create new entry**" button.

Adding entries in each collection

[< Back](#)

Author

2 entries found

[+ Create new entry](#)

Filters

<input type="checkbox"/>	ID	NAME ▲	BIO	BIOIMAGE	STATUS	
<input type="checkbox"/>	5	Carlos Pérez	Especialista en DevOps y arquitectura cloud....		Published	...
<input type="checkbox"/>	7	María González	Desarrolladora frontend y escritora técnica c...		Published	...

Adding entries in each collection

[← Back](#)

Category

3 entries found

[+ Create new entry](#)

Filters

<input type="checkbox"/>	ID	SLUG	CREATEDAT	NAME ▲	STATUS	
<input type="checkbox"/>	5	buenas-practicas	Thursday, March 6, 2025 at 7:02 PM	Buenas Prácticas	Published	...
<input type="checkbox"/>	1	desarrollo-web	Thursday, March 6, 2025 at 7:00 PM	Desarrollo Web	Published	...
<input type="checkbox"/>	3	devops	Thursday, March 6, 2025 at 7:02 PM	DevOps	Published	...

Adding entries in each collection


[← Back](#)



Post

2 entries found

[+ Create new entry](#)

[Filters](#)



<input type="checkbox"/>	ID	TITLE ▲	SLUG	FEATUREDIMAGE	STATUS	
<input type="checkbox"/>	4	Automatización con Docker	docker		Published	...
<input type="checkbox"/>	1	Introducción a JavaScript Moderno	js_moderno		Published	...

JSON Response

If we check out API endpoints, we'll see entries we made in admin panel in JSON format:

`http://localhost:1337/api/categories`

`http://localhost:1337/api/authors`

`http://localhost:1337/api/posts`

JSON Response

```
{
  "data": [
    {
      "id": 13,
      "documentId": "a6i4b71wt6p5lg0ip4f4unqv",
      "Name": "Database",
      "slug": "database",
      "createdAt": "2024-11-07T12:26:23.909Z",
      "updatedAt": "2024-11-07T12:57:01.568Z",
      "publishedAt": "2024-11-07T12:57:01.572Z"
    },
    {
      "id": 14,
      "documentId": "mf9pomf42831wf57wt5o2n95",
      "Name": "Frameworks",
      "slug": "frameworks",
      "createdAt": "2024-11-07T12:26:06.582Z",
      "updatedAt": "2024-11-07T12:57:11.000Z",
      "publishedAt": "2024-11-07T12:57:11.004Z"
    },
  ],
}
```

Setting up HTML & Javascript Frontend

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8"/>
  <meta content="width=device-width, initial-scale=1.0" name="viewport"/>
  <title>Posts con Autor y Categorías</title>
  <!-- Cargar Tailwind CSS -->
  <link href="/tailwind.min.css" rel="stylesheet">
</head>
<body class="bg-gray-100 p-8">
<h1 class="text-3xl font-bold mb-6">Listado de Posts</h1>
<!-- Contenedor de posts en grid -->
<div class="grid grid-cols-1 md:grid-cols-2 gap-6" id="posts-container"></div>
...
```


Setting up HTML & Javascript Frontend

...

<!-- Template para un post -->

<template id="post-template">

 <div class="bg-white rounded shadow p-4">

 <h2 class="text-2xl font-bold mb-2 post-title"></h2>

 <p class="text-gray-700 mb-4 post-excerpt"></p>

 <!-- Contenedor para categorías -->

 <div class="post-categories mb-4"></div>

 ...

</div>

</template>

...

Setting up HTML & Javascript Frontend

...

<!-- Template para un post -->

<template id="post-template">

 <div class="bg-white rounded shadow p-4">

 ...

 <div class="flex ">

 <div>

 <p class="font-semibold post-author-name"></p>

 <p class="text-sm text-gray-500 post-author-bio"></p>

 </div>

 </div>

 </div>

</template>

...

Setting up HTML & Javascript Frontend

...

```
<!-- Template para un post -->  
<template id="post-template">
```

...

```
</template>
```

```
<!-- Cargar JS para descarga de datos -->  
<script src="app.js"></script>  
</body>  
</html>
```

Setting up HTML & Javascript Frontend

```
document.addEventListener("DOMContentLoaded", async () => {  
  
  const postsContainer = document.getElementById("posts-container");  
  const postTemplate = document.getElementById("post-template");  
  
  const serverUrl = "http://localhost:1337";  
  const apiUrl = serverUrl + "/api";  
  
  try {  
  
    // Obtener los autores con sus imágenes  
    const authorsUrl = apiUrl + "/authors?populate=*";  
    const authorsResponse = await fetch(authorsUrl);  
    const authorsData = await authorsResponse.json();  
  
    ...  
  }  
}
```

Setting up HTML & Javascript Frontend

...

```
const authorsMap = {};  
authorsData.data.forEach(author => {  
  authorsMap[author.id] = author;  
});
```

// Obtener los posts con sus relaciones

```
const postsUrl = apiUrl + "/posts?populate=*";  
const postsResponse = await fetch(postsUrl);  
const postsData = await postsResponse.json();
```

...

Setting up HTML & Javascript Frontend

...

```
postsData.data.forEach(post => {  
  
    // Obtener el autor completo usando el diccionario  
    const fullAuthor = authorsMap[post.author.id] || post.author;  
  
    // Construir la URL de la imagen destacada del post  
    let featuredImageUrl = "";  
    if (post.featuredImage && post.featuredImage.url) {  
        featuredImageUrl = serverUrl + post.featuredImage.url;  
    }  
    ...  
}
```

Setting up HTML & Javascript Frontend

...

```
postsData.data.forEach(post => {
```

...

```
// Obtener la imagen del autor (biolmage)
```

```
let authorImageUrl = "";
```

```
if (fullAuthor.biolmage && fullAuthor.biolmage.url) {  
    authorImageUrl = serverUrl + fullAuthor.biolmage.url;  
}
```

```
// Clonar el template y rellenar los datos
```

```
const clone = postTemplate.content.cloneNode(true);
```

```
clone.querySelector(".post-title").textContent = post.title;
```

...

Setting up HTML & Javascript Frontend

...

```
postsData.data.forEach(post => {
```

...

```
  if (featuredImageUrl) {  
    const imgEl = clone.querySelector(".post-image");  
    imgEl.src = featuredImageUrl;  
    imgEl.alt = post.title;  
  } else {  
    clone.querySelector(".post-image").remove();  
  }
```

```
  clone.querySelector(".post-excerpt").textContent = post.excerpt;
```

...

Setting up HTML & Javascript Frontend

...

```
postsData.data.forEach(post => {
```

...

```
// Renderizar las categorías del post
```

```
const catContainer = clone.querySelector(".post-categories");
```

```
if (post.categories && post.categories.length > 0) {
```

```
// Ordenar las categorías alfabéticamente por el nombre
```

```
const sortedCategories =
```

```
    post.categories.sort((a, b) => a.name.localeCompare(b.name));
```

```
let categoriesHtml = "";
```

...

Setting up HTML & Javascript Frontend

...

```
postsData.data.forEach(post => {
```

...

```
if (post.categories && post.categories.length > 0) {
```

...

```
    sortedCategories.forEach(category => {  
        categoriesHtml += '<span class="...">${category.name}</span>';  
    });
```

```
    catContainer.innerHTML = categoriesHtml;
```

```
} else {
```

```
    catContainer.remove();
```

```
}
```

...

Setting up HTML & Javascript Frontend

...

```
postsData.data.forEach(post => {
```

...

```
// Datos del autor
```

```
const authorImgEl = clone.querySelector(".post-author-image");
```

```
authorImgEl.src = authorImageUrl;
```

```
authorImgEl.alt = fullAuthor.name;
```

```
clone.querySelector(".post-author-name").textContent = fullAuthor.name;
```

```
clone.querySelector(".post-author-bio").textContent = fullAuthor.bio;
```

```
postsContainer.appendChild(clone);
```

```
});
```

```
} catch (error) { ... }
```

```
});
```