Bluepipe

Table of contents:

Overview

What is refine?

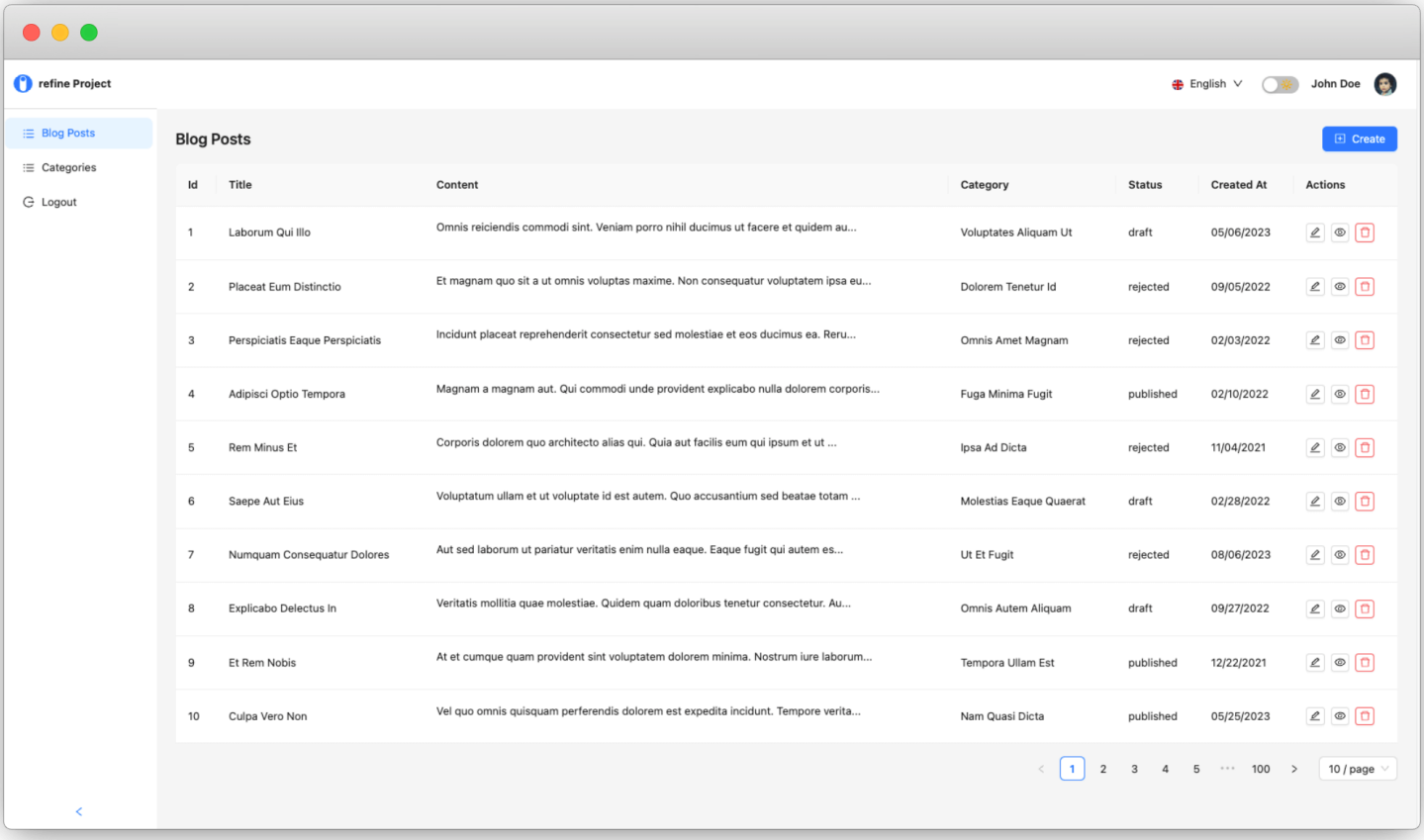
refine is a meta **React** framework that enables the rapid development of a wide range of web applications. From internal tools, admin panels, B2B apps and dashboards, it serves as a comprehensive solution for building any type of **CRUD** applications.

refine's internal hooks and components simplifies the development process and eliminates the repetitive tasks by providing industry-standard solutions for crucial aspects of a project, including **authentication**, **access control**, **routing**, **networking**, **state management**, and **i18n**.

Here's an overview of the refine structure:

App.tsx

```tsx
const App = () => (
    <Refine
        dataProvider={dataProvider}
        resources={[
            {
                name: "blog_posts",
                list: "/blog-posts",
                show: "/blog-posts/show/:id",
                create: "/blog-posts/create",
                edit: "/blog-posts/edit/:id",
            },
        ]}
    >
      /* ... */
    </Refine>
);
```



⚡ **refine** is *headless by design* and offers unlimited styling and customization possibilities, empowering developers to create tailored and fully functional applications that meet specific project requirements.

Utilizing integrated technologies you can efforlessly develop industry-standard CRUD applications with **refine**.

What do you mean by "headless" ?

Instead of being limited to a set of pre-styled components, **refine** provides collections of helper `hooks` , `components` and `providers` and more. Since business logic and UI are completely decoupled, you can customize UI without constraints.

It means, **refine** just works *seamlessly* with any *custom designs* or *UI frameworks*. Thanks to it's headless architecture, you can use popular CSS frameworks like TailwindCSS or even create your own styles from scratch.

refine also provides integrations with Ant Design, Material UI, Mantine, and Chakra UIto get you started quickly. These libraries are set of components which are nicely integrated with headless `@refinedev/core` package.

Headless in Routing

For the routing, refine's headless approach shines too. It doesn't tie you to a single routing method or library. Instead, it offers a simple routing interface with built-in integrations for popular libraries.

This means you can use refine seamlessly in different platforms like React Native, Electron, Next.js, Remix etc. without any extra steps for the setup.

Use cases

refine shines when it comes to *data-intensive* applications like *admin panels*, *dashboards* and *internal tools*.

Key Features

⚙️ Zero-config, **one-minute setup** with a **single CLI command**

🔌 Connectors for **15+ backend services** including REST API, GraphQL, NestJs CRUD, Airtable, Strapi, Strapi v4, Strapi GraphQL, Supabase, Hasura, Appwrite, Firebase, Nestjs-Query and Directus.

🌐 **SSR support** with **Next.js** or **Remix**

🔍 Auto-generation of **CRUD** UIs based on **your API data structure**

⚛️ Perfect **state management** & **mutations** with **React Query**

🔀 **Advanced routing** with any router library of your choice

🔐 Providers for seamless **authentication** and **access control** flows

⚡ Out-of-the-box support for **live / real-time applications**

📄 Easy **audit logs** & **document versioning**

💬 Support for any **i18n** framework

💪 Future-proof, **robust architecture**

💻 refine Devtools - dive deeper into your app and provide useful insights

✅ Full **test coverage**

Community

refine has a very friendly community and we are always happy to help you get started:

- 🌟 Apply for the Priority support program! You can apply to priority support program and receive assistance from the refine **core** team in your **private** channel.
- Join the Discord community! It is the easiest way to get help and ask questions to the community.
- Join the GitHub Discussions to ask anything about the refine project or give feedback; we would love to hear your thoughts!
- Learn how to contribute to the refine!
- Join our Guest Technical Writer Program and become a blog writer for **refine**.

Next Steps

👉 Continue with the Quickstart guide to setup and run your first **refine** project.

Quick Start Guide

refine works on any environment that can run **React** (incl. *Vite, Next.js, Remix, and CRA(Legacy)* etc.)

Although you could take the time to manually set up your environment and install the **refine** packages afterwards, the optimal way to get started with **refine** is using the Browser-based Scaffolder and **CLI-based Scaffolder**.

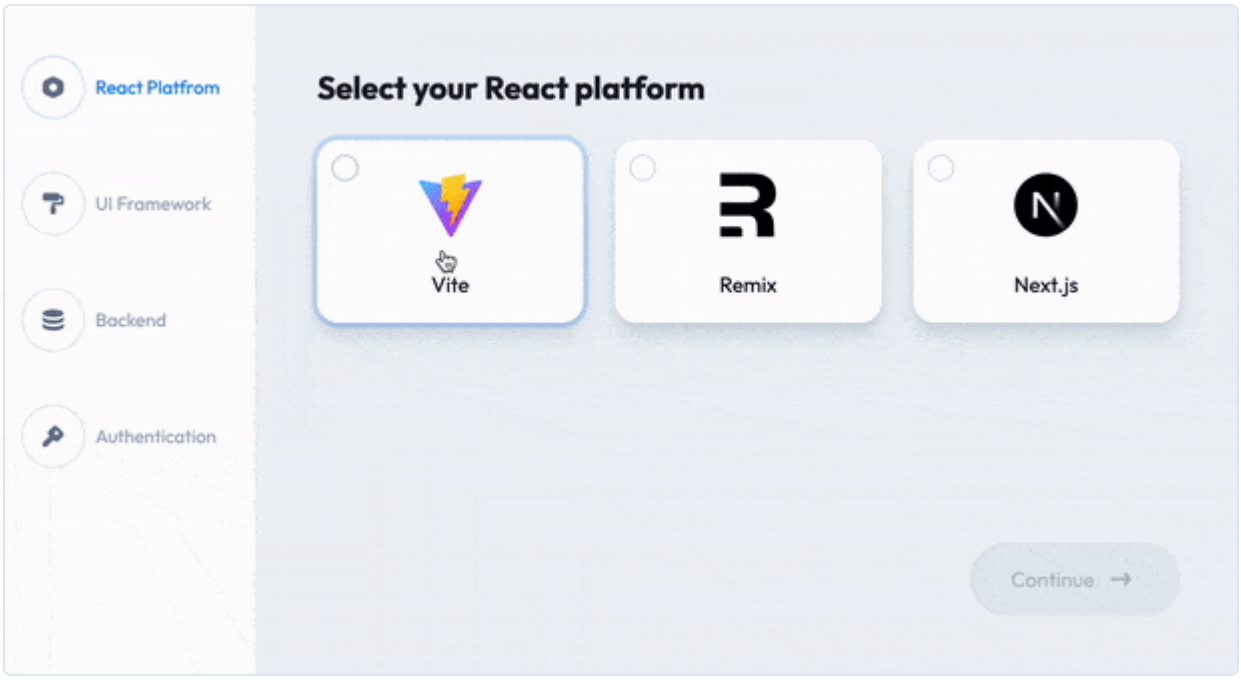
Choose one of the methods below to bootstrap a refine app:

With Browser-based | With CLI-based

This is an efficient tool that allows you to create refine app seamlessly in your browser.

You can choose the libraries and frameworks you want to work with, and the tool will generate a boilerplate code for you.

1. Navigate to scaffolder and select libraries and frameworks you want to work with by using interactive UI.
2. Download the generated project by clicking on the **"Build & Download"** button.



> **TIP**
>
> If you do not want to deal with the installation steps, you can use this pre-configured refine.new template →

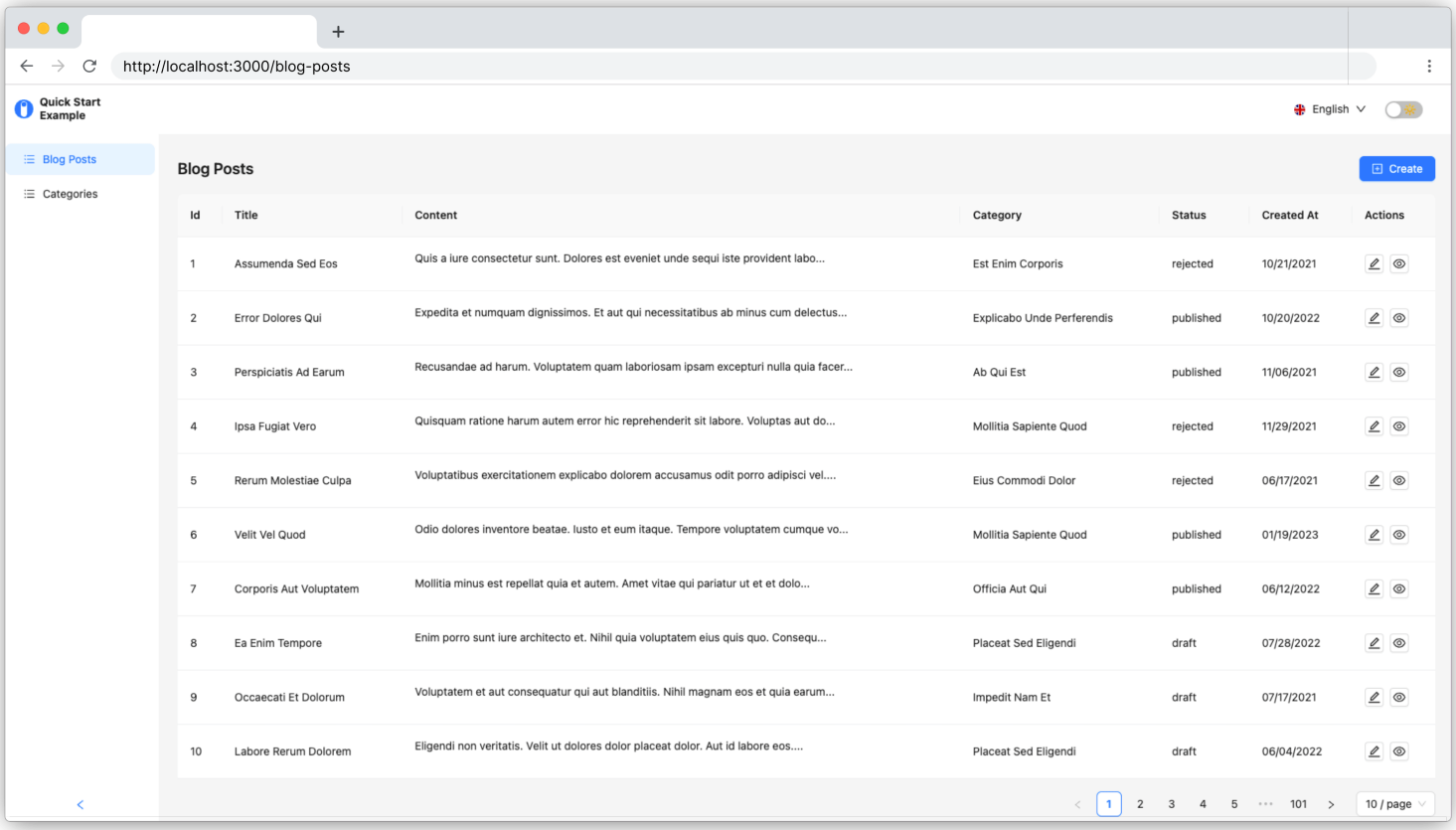
3. Next, navigate to the project folder, install the dependencies, and then start your project.

```
> npm install

> npm run dev
```

Now, you can access the refine application at http://localhost:3000.

You will see the output as a table populated with `blog_posts` and `category` data with filtering, sorting, and pagination features.



Next Steps

👉 See [real-life examples] built using **refine**

Contributing

We follow a code of conduct when participating in the community. Please read it before you make any contributions.

- If you plan to work on an issue, mention so in the issue page before you start working on it.
- If you plan to work on a new feature, create an issue and discuss it with other community members/maintainers.
- Ask for help in our community room.

Ways to contribute

- **Stars on GitHub**: If you're a refine user and enjoy using our platform, don't forget to star it on GitHub! 🌟
- **Improve documentation**: Good documentation is imperative to the success of any project. You can make our documents the best they need to be by improving their quality or adding new ones.
- **Give feedback**: We're always looking for ways to make refine better, please share how you use refine, what features are missing and what is done good via GitHub Discussions or Discord.
- **Share refine**: Help us reach people. Share refine repository with everyone who can be interested.
- **Contribute to codebase**: your help is needed to make this project the best it can be! You could develop new features or fix existing issues - every contribution will be welcomed with great pleasure!

Commit convention

refine is a monorepo. For a monorepo, commit messages are essential to keep everything clear. We use conventional commits to keep our commit messages consistent and easy to understand.

```
<type>(optional scope): <description>
```

Examples:

- `feat: allow provided config object to extend other configs`

- `fix: array parsing issue when multiple spaces were contained in string`
- `docs: correct spelling of CHANGELOG`

Changeset

Changesets are designed to make your workflows easier, by allowing the person making contributions to make key decisions when they are making their contribution. Changesets hold two key bits of information: a version type (following semver), and change information to be added to a changelog.

Follow the steps below to create a changeset:

```
npm run changeset
```

After that you need to,

- select the package(s) you are modifying
- choose one of `major/patch/minor` according to your change
- add explanation about the changes

explanation should follow the same format with commit convention with some extra description:

```
feat: added x feature

Now with x feature, you can do y.
```

or

```
fix: issue with x.

We had an edge where it causes x issue to happen, now it's fixed.
```

and then you are done!

Running in development mode

`node` version 18 is required.

This project has multiple packages and uses Lerna to manage packages under `packages/` .

First, install dependencies:

```
npm install
```

From now on, depending on the packages you plan to work on, (they are located under `packages/` and `examples/` directories - see lerna.json) you will need to bootstrap them and start them in watch mode. Instead of running `lerna bootstrap` directly, read on to see how **refine** team handles it.

Refer to **lerna** docs to learn more about it. →

Refer to **lerna** docs to learn more about `lerna bootstrap`. →

Starting the packages you work in watch mode

Before running a package in development mode in the refine project, you need to first bootstrap and build all the packages.

Bootstrap & build all packages

```
npm run bootstrap:all
npm run build:all
```

Then, you can bootstrap the example you want to work on with the command below:

```
npm run bootstrap -- --scope refine-use-select-example
```

> You can add the packages you want to bootstrap with the `scope` filter.

At this point, all/required packages are bootstrapped. You can add the packages you want to run in development mode with the `scope` filter. This way, the `watch` mode will start working for the packages you specified.

In the command example below, we are running the core and antd packages in `development mode` along with the example.

```
npm run start -- --scope @refinedev/antd --scope refine-use-select-example
```

This command starts the example named `refine-use-select-example` in dev mode. The value of the flag `--scope` is the name that is defined in it's `package.json` file. Note that `--scope` flag should be used for every package that should be filtered. If you should start two packages:

Now all filtered packages are running in watch mode. They should re-compile when you make a change in any of them.

Starting documentation in watch mode

Our documentation is built with Docusaurus. To start it in development mode, run:

```
cd documentation
npm install
npm run dev:docs
# npm run dev:blog if you're working on blog posts
```

TIP

`dev:docs` and `dev:blog` scripts start a portion of the documentation and skips the unnecessary parts to speed up the development process such as type and props table generation, checklist generation, etc. If you want to start the documentation with all the features, you can use `npm run start` command.

DOCGEN PLUGIN AND PROPS TABLE

If you are working on type generation and props tables for specific packages, you can use `INCLUDED_PACKAGES` environment variable to run the scripts for only the packages you are working on by providing comma delimited list of package directories.

For example, if you are working on `@refinedev/antd` and `@refinedev/core` packages, which are located under `packages/antd` and `packages/core` directories, you can run the following command to generate type documentation for only these packages:

```
INCLUDED_PACKAGES=antd,core npm run start
```

To use `<PropsTable />` component, you should pass `module` prop as `string` to the component in form of `@refinedev/antd/MyComponent`.

```
<PropsTable module="@refinedev/antd/Create" />
```

Running tests

`npm run test` command runs tests for all packages. If you're working on a package (e.g. `/packages/core`), you can run tests only for that package:

```
cd packages/core
npm run test
```

Or you can do it for a specific file:

```
npm run test -- /src/hooks/export/index.spec.ts
```

Also, to run a specific file's tests in watch mode:

```
npm run test -- --watch /src/hooks/export/index.spec.ts
```

Get coverage report for that file:

```
npm run test -- --coverage /src/hooks/export/index.spec.ts
```

When you run the command that produces coverage report, go to `/coverage/lcov-report/index.html` file to see coverage results. If you run this command in `/packages/core` directory, then coverage report will be generated in `/packages/core/coverage`.

Please make sure you contribute well tested code.

Creating Live Previews in Documentation

We're using live previews powered with `react-live` to demonstrate our components and logic with `refine` running at full functionality. To create a live preview, you should add `live` property to your code blocks in markdown files.

> **TIP**
>
> You can use `import` statements to show them in the code block but they will be ignored when running the code. Instead all import statements related to **refine** will be converted into object destructures and prepended into code. This will allow you to do the import in the visible part of the code and also use them before the import statements. Check out Defined Scope section to learn more about the available packages and variables.

> **INFORMATION**
>
> refine Live Previews has an independent package apart from the documentation and the previews are rendered through this package via iframe. `@refinedev/live-previews` runs on `3030` port by default and the fallback value for `LIVE_PREVIEW_URL` is set to `http://localhost:3030` for development purposes. If you want to run both the previews package and the documentation at the same time, use `npm run start:doc` command.

Properties

You can use the following properties to adjust your live blocks:

Property	Description	Default
`hideCode`	Adds title	false
`disableScroll`	Disables the scroll in the preview	false
`previewHeight`	Height of the preview	`400px`
`url`	URL to be shown in the header of the preview	`http://localhost:3000`

Hiding Boilerplate Code

There are two ways to hide/show code sections in your live code blocks; either you can wrap your visible code to `// visible-block-start` and `// visible-block-end` comments, or you can use `// hide-next-line`, `// hide-start` and `// hide-end` comments. You can use both of them in the same code block.

`// visible-block-start` and `// visible-block-end`

This wrapper can be used to show only the desired part of the code in the code block and will not affect the live preview. Copy code button will only copy the code inside this block. This is the recommended way to hide boilerplate/unwanted code.

`// hide-next-line` and `// hide-start` and `// hide-end`

These magic comments will hide the next line or the wrapped code block. You can use these to hide the code that you want to show in the code blocks. These will also not affect the live preview but those lines will be copied with the copy button. Use these to hide the required code pieces for the live preview but are out of scope for the example. Such as while showing how a property of a component works in live preview, you can hide the required but not relevant props.

Rendering the Preview

To render the live preview you should call the `render` function with your component. `render` function is specific to `react-live` and it will render the preview in the browser; therefore not

needed to be visible in the codeblock, it's recommended to leave the `render` part outside of the `// visible-block` wrapper.

Example Usage

```tsx live hideCode url=http://localhost:3000/posts/create
import { Refine } from "@refinedev/core";
import dataProvider from "@refinedev/simple-rest";
// You can use object destructuring to import the necessary functions and components wh
const { CreateButton } = RefineAntd;

interface ICategory {
    id: number;
    title: string;
}

interface IPost {
    id: number;
    title: string;
    content: string;
    status: "published" | "draft" | "rejected";
    category: { id: number };
}

// visible-block-start
// Import statements will be replaced with the object destructuring but visible code bl
import { Create, useForm, useSelect } from "@refinedev/antd";
import { Form, Input, Select } from "antd";

const PostCreate: React.FC = () => {
    const { formProps, saveButtonProps } = useForm<IPost>();

    const { selectProps: categorySelectProps } = useSelect<ICategory>({
        resource: "categories",
    });

    return (
        <Create saveButtonProps={saveButtonProps}>
            <Form {...formProps} layout="vertical">
                <Form.Item
                    label="Title"
                    name="title"
                    rules={[
                        {
                            required: true,
                        },
                    ]}
                >
```

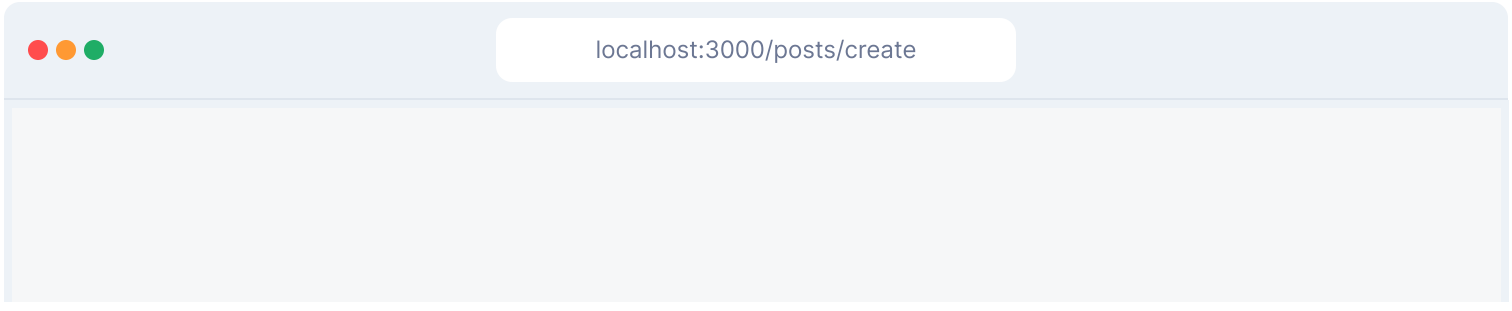
```
                    <Input />
                </Form.Item>
                <Form.Item
                    label="Category"
                    name={["category", "id"]}
                    rules={[
                        {
                            required: true,
                        },
                    ]}
                >
                    <Select {...categorySelectProps} />
                </Form.Item>
                <Form.Item
                    label="Status"
                    name="status"
                    rules={[
                        {
                            required: true,
                        },
                    ]}
                >
                    <Select
                        options={[
                            {
                                label: "Published",
                                value: "published",
                            },
                            {
                                label: "Draft",
                                value: "draft",
                            },
                            {
                                label: "Rejected",
                                value: "rejected",
                            },
                        ]}
                    />
                </Form.Item>
            </Form>
        </Create>
    );
};
```

```
  // visible-block-end

// We're setting the initial route to "/posts/create" to render the preview.
setInitialRoutes(["/posts/create"]);

  // This part is required to render the preview.
render(
    <ReactRouterDom.BrowserRouter>
        <Refine
            dataProvider={dataProvider("https://api.fake-rest.refine.dev")}
            resources={[
                {
                    name: "posts",
                    list: "/posts",
                    create: "/posts/create",
                },
            ]}
        >
            <ReactRouterDom.Routes>
                <ReactRouterDom.Route
                    path="/posts"
                    element={(
                        <div>
                            <p>This page is empty.</p>
                            <CreateButton />
                        </div>
                    )}
                />
                <ReactRouterDom.Route
                    path="/posts/create"
                    element={<PostCreate />}
                />
            </ReactRouterDom.Routes>
        </Refine>
    </ReactRouterDom.BrowserRouter>,
);
  ```;
```

### Result


```

Defined Scope

Variable	Description
`React`	React 17
`RefineCore`	`@refinedev/core`
`RefineSimpleRest`	`@refinedev/simple-rest`
`RefineAntd`	`@refinedev/antd`
`RefineMui`	`@refinedev/mui`
`RefineMantine`	`@refinedev/mantine`
`RefineChakra`	`@refinedev/chakra-ui`
`RefineReactRouterV6`	`@refinedev/react-router-v6`
`RefineReactHookForm`	`@refinedev/react-hook-form`
`RefineReactTable`	`@refinedev/react-table`
`RefineAntdInferencer`	`@refinedev/inferencer/antd`
`RefineMuiInferencer`	`@refinedev/inferencer/mui`
`RefineMantineInferencer`	`@refinedev/inferencer/mantine`
`RefineChakraInferencer`	`@refinedev/inferencer/chakra-ui`
`LegacyRefineReactRouterV6`	`@refinedev/react-router-v6/legacy`

Variable	Description
`RefineReactRouterV6`	`@refinedev/react-router-v6`
`ReactRouterDom`	`react-router-dom`
`AntdCore`	`antd`
`MantineCore`	`@mantine/core`
`MantineHooks`	`@mantine/hooks`
`MantineForm`	`@mantine/form`
`MantineNotifications`	`@mantine/notifications`
`EmotionReact`	`@emotion/react`
`EmotionStyled`	`@emotion/styled`
`MuiLab`	`@mui/lab`
`MuiMaterial`	`@mui/material`
`MuiXDataGrid`	`@mui/x-data-grid`
`ChakraUI`	`@chakra-ui/react`
`ReactHookForm`	`react-hook-form`
`TanstackReactTable`	`@tanstack/react-table`
`RefineHeadlessDemo`	Predefined `<Refine/>` component with simple-rest and react-router-v6 props for easier use
`RefineMuiDemo`	Predefined `<Refine/>` component with Material UI, simple-rest and react-router-v6 props for easier use
`RefineAntdDemo`	Predefined `<Refine/>` component with Ant Design, simple-rest and react-router-v6 props for easier use
`setInitialRoutes`	For live previews, we use `MemoryRouter` from `react-router-v6` and to set the initial entries of the history, you can use this function.
`setRefineProps`	For live previews, you may need to set some props to `<Refine />` component that are unrelated to the code block you're writing. In those cases, you can use `setRefinProps` outside of the visible code block to set props or override the existing props.

TIP

Demo components are recommended to be used whenever possible to avoid unnecessary configuration at every code block. They are equipped with the `refine-react-router-v6` setup with `MemoryRouter`, `refine-simple-rest` data provider and the preferred UI Integration.

INFORMATION

`Refine` component from `RefineCore` has the default prop `reactQueryDevtoolConfig` set to `false` to disable the React Query Dev Tools since it doesn't work with the production version of the React.

INFORMATION

`setInitialRoutes` is a function to set the initial routes of the preview for `@refinedev/react-router-v6` using `MemoryRouter`. This function takes one argument `initialRoutes` which is an array of routes to be rendered initially. For example, if your component is rendered at `/posts/create`, you can pass `["/posts/create"]` as the argument.

TIP

Make sure you use `setInitialRoutes` function before rendering the `<Refine/>` component. Otherwise, `MemoryRouter` will not be able to set the initial routes. For some cases, you might find setting `initialRoutes` prop of demo `<Refine/>` (`RefineHeadlessDemo`, `RefineMuiDemo` and `RefineAntdDemo`) components easier. There's no difference between the two approaches.

TIP

`setRefineProps` is a function to set additional props to `<Refine />` or override existing props of `<Refine />`. Make sure you don't conflict with the props you set in the visible block while overriding; which may cause unwanted results.

Tutorial - Basics

5 minutes to learn the most important Docusaurus concepts.

Create a Page

Add Markdown or React files to src/pages to create a standalone page:

Create a Document

Documents are groups of pages connected through:

Create a Blog Post

Docusaurus creates a page for each blog post, but also a blog index page, a tag system, an RSS feed...

Markdown Features

Docusaurus supports Markdown and a few additional features.

Deploy your site

Docusaurus is a static-site-generator (also called Jamstack).

Congratulations!

You have just learned the basics of Docusaurus and made some changes to the initial template.

Create a Page

Add **Markdown or React** files to `src/pages` to create a **standalone page**:

- `src/pages/index.js` → `localhost:3000/`
- `src/pages/foo.md` → `localhost:3000/foo`
- `src/pages/foo/bar.js` → `localhost:3000/foo/bar`

Create your first React Page

Create a file at `src/pages/my-react-page.js`:

src/pages/my-react-page.js

```
import React from 'react';
import Layout from '@theme/Layout';

export default function MyReactPage() {
  return (
    <Layout>
      <h1>My React page</h1>
      <p>This is a React page</p>
    </Layout>
  );
}
```

A new page is now available at http://localhost:3000/my-react-page.

Create your first Markdown Page

Create a file at `src/pages/my-markdown-page.md`:

src/pages/my-markdown-page.md

```
# My Markdown page

This is a Markdown page
```

A new page is now available at http://localhost:3000/my-markdown-page.

Create a Document

Documents are **groups of pages** connected through:

- a **sidebar**
- **previous/next navigation**
- **versioning**

Create your first Doc

Create a Markdown file at `docs/hello.md`:

```
docs/hello.md
```

```
# Hello

This is my **first Docusaurus document**!
```

A new document is now available at http://localhost:3000/docs/hello.

Configure the Sidebar

Docusaurus automatically **creates a sidebar** from the `docs` folder.

Add metadata to customize the sidebar label and position:

```
docs/hello.md
```

```
---
sidebar_label: 'Hi!'
sidebar_position: 3
---

# Hello

This is my **first Docusaurus document**!
```

It is also possible to create your sidebar explicitly in `sidebars.js`:

sidebars.js

```
export default {
  tutorialSidebar: [
    'intro',
   'hello',
    {
      type: 'category',
      label: 'Tutorial',
      items: ['tutorial-basics/create-a-document'],
    },
  ],
};
```

Create a Blog Post

Docusaurus creates a **page for each blog post**, but also a **blog index page**, a **tag system**, an **RSS** feed...

Create your first Post

Create a file at `blog/2021-02-28-greetings.md`:

blog/2021-02-28-greetings.md

```
---
slug: greetings
title: Greetings!
authors:
  - name: Joel Marcey
    title: Co-creator of Docusaurus 1
    url: https://github.com/JoelMarcey
    image_url: https://github.com/JoelMarcey.png
  - name: Sébastien Lorber
    title: Docusaurus maintainer
    url: https://sebastienlorber.com
    image_url: https://github.com/slorber.png
tags: [greetings]
---

Congratulations, you have made your first post!

Feel free to play around and edit this post as much you like.
```

A new blog post is now available at http://localhost:3000/blog/greetings.

Markdown Features

Docusaurus supports Markdown and a few **additional features**.

Front Matter

Markdown documents have metadata at the top called Front Matter:

my-doc.md

```
---
id: my-doc-id
title: My document title
description: My document description
slug: /my-custom-url
---

## Markdown heading

Markdown text with [links](./hello.md)
```

Links

Regular Markdown links are supported, using url paths or relative file paths.

```
Let's see how to [Create a page](/create-a-page).
```

```
Let's see how to [Create a page](./create-a-page.md).
```

Result: Let's see how to Create a page.

Images

Regular Markdown images are supported.

You can use absolute paths to reference images in the static directory (`static/img/docusaurus.png`):

You can reference images relative to the current file as well. This is particularly useful to colocate images close to the Markdown files using them:

Code Blocks

Markdown code blocks are supported with Syntax highlighting.

````
```jsx title="src/components/HelloDocusaurus.js"
function HelloDocusaurus() {
    return (
        <h1>Hello, Docusaurus!</h1>
    )
}
```
````

src/components/HelloDocusaurus.js

```jsx
function HelloDocusaurus() {
  return <h1>Hello, Docusaurus!</h1>;
}
```

Admonitions

Docusaurus has a special syntax to create admonitions and callouts:

```
:::tip My tip

Use this awesome feature option

:::

:::danger Take care

This action is dangerous

:::
```

MY TIP

Use this awesome feature option

TAKE CARE

This action is dangerous

MDX and React Components

MDX can make your documentation more **interactive** and allows using any **React components inside Markdown**:

```
export const Highlight = ({children, color}) => (
  <span
    style={{
      backgroundColor: color,
      borderRadius: '20px',
      color: '#fff',
      padding: '10px',
      cursor: 'pointer',
    }}
    onClick={() => {
      alert(`You clicked the color ${color} with label ${children}`)
    }}>
    {children}
  </span>
);

This is <Highlight color="#25c2a0">Docusaurus green</Highlight> !

This is <Highlight color="#1877F2">Facebook blue</Highlight> !
```

This is Docusaurus green !

This is Facebook blue !

Deploy your site

Docusaurus is a **static-site-generator** (also called Jamstack).

It builds your site as simple **static HTML, JavaScript and CSS files**.

Build your site

Build your site **for production**:

```
npm run build
```

The static files are generated in the `build` folder.

Deploy your site

Test your production build locally:

```
npm run serve
```

The `build` folder is now served at http://localhost:3000/.

You can now deploy the `build` folder **almost anywhere** easily, **for free** or very small cost (read the Deployment Guide).

Congratulations!

You have just learned the **basics of Docusaurus** and made some changes to the **initial template**.

Docusaurus has **much more to offer**!

Have **5 more minutes**? Take a look at versioning and i18n.

Anything **unclear** or **buggy** in this tutorial? Please report it!

What's next?

- Read the official documentation
- Modify your site configuration with `docusaurus.config.js`
- Add navbar and footer items with `themeConfig`
- Add a custom Design and Layout
- Add a search bar
- Find inspirations in the Docusaurus showcase
- Get involved in the Docusaurus Community

Tutorial - Extras

Manage Docs Versions

Docusaurus can manage multiple versions of your docs.

Translate your site

Let's translate docs/intro.md to French.

Manage Docs Versions

Docusaurus can manage multiple versions of your docs.

Create a docs version

Release a version 1.0 of your project:

```
npm run docusaurus docs:version 1.0
```

The `docs` folder is copied into `versioned_docs/version-1.0` and `versions.json` is created.

Your docs now have 2 versions:

- `1.0` at `http://localhost:3000/docs/` for the version 1.0 docs
- `current` at `http://localhost:3000/docs/next/` for the **upcoming, unreleased docs**

Add a Version Dropdown

To navigate seamlessly across versions, add a version dropdown.

Modify the `docusaurus.config.js` file:

docusaurus.config.js

```
export default {
  themeConfig: {
    navbar: {
      items: [
      {
        type: 'docsVersionDropdown',
      },
      ],
    },
  },
};
```

The docs version dropdown appears in your navbar:

Update an existing version

It is possible to edit versioned docs in their respective folder:

- `versioned_docs/version-1.0/hello.md` updates `http://localhost:3000/docs/hello`
- `docs/hello.md` updates `http://localhost:3000/docs/next/hello`

Translate your site

Let's translate `docs/intro.md` to French.

Configure i18n

Modify `docusaurus.config.js` to add support for the `fr` locale:

docusaurus.config.js

```
export default {
  i18n: {
    defaultLocale: 'en',
    locales: ['en', 'fr'],
  },
};
```

Translate a doc

Copy the `docs/intro.md` file to the `i18n/fr` folder:

```
mkdir -p i18n/fr/docusaurus-plugin-content-docs/current/

cp docs/intro.md i18n/fr/docusaurus-plugin-content-docs/current/intro.md
```

Translate `i18n/fr/docusaurus-plugin-content-docs/current/intro.md` in French.

Start your localized site

Start your site on the French locale:

```
npm run start -- --locale fr
```

Your localized site is accessible at http://localhost:3000/fr/ and the `Getting Started` page is translated.

> **CAUTION**
>
> In development, you can only use one locale at a time.

Add a Locale Dropdown

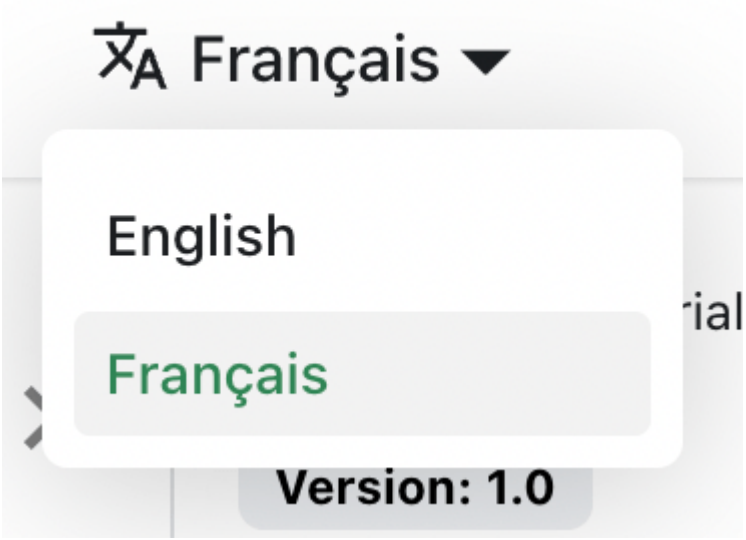
To navigate seamlessly across languages, add a locale dropdown.

Modify the `docusaurus.config.js` file:

docusaurus.config.js

```
export default {
  themeConfig: {
    navbar: {
      items: [
      {
        type: 'localeDropdown',
      },
      ],
    },
  },
};
```

The locale dropdown now appears in your navbar:



Build your localized site

Build your site for a specific locale:

```
npm run build -- --locale fr
```

Or build your site to include all the locales at once:

```
npm run build
```

Overview

refine is licensed under the MIT Licence. It only requires the preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- ✅ Commercial use
- ✅ Modification
- ✅ Distribution
- ✅ Private use

Limitations

- ❌ Liability
- ❌ Warranty

Licence

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Testing

refine's components and hooks are made from small pieces of code. Each component or hook is designed to be testable and work independently of each other.

So, you don't need unit testing, because **refine** is already tested by its maintainers. However, you can write unit tests in your own code (helper, definitions, etc.).

We strongly recommend that you write end-to-end tests of your application. **refine** used the cypress framework as an example. You are free to write tests with any framework you want.

Example

View Source Code

```
>_  Run on your local

npm create refine-app@latest -- --example with-cypress
```