Jinja2 Templates

In this lesson, we will learn about dynamic templates and Jinja's templating engine.

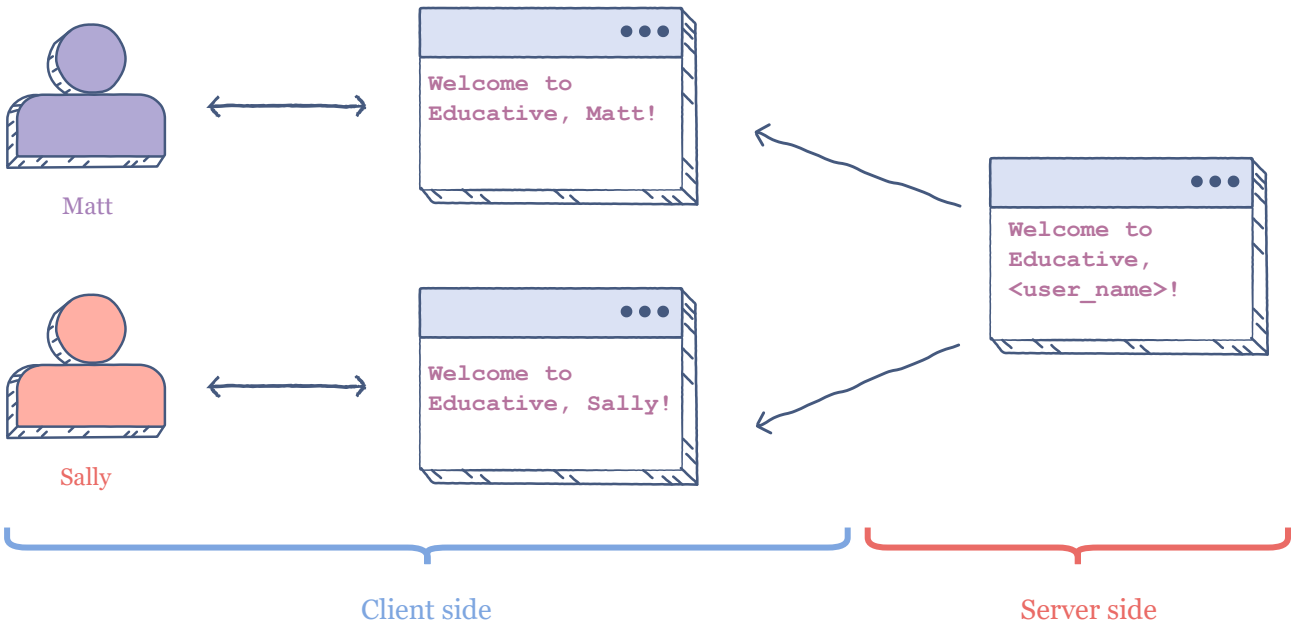
WE'LL COVER THE FOLLOWING

Introduction

Up until now, we have learned how to serve static `HTML` *templates* and other static files such as *images*, `CSS`, etc. In this lesson, we will take a look at the concept of **dynamic templates**. We will also learn how to serve dynamic content in our Flask application.

Dynamic templates

Consider that if we are making an application with multiple users, such as a social media application, each user will have a unique profile and unique information associated with them. It is the job of the web application to serve a unique template containing the content corresponding to the user logged in.



Example of a Dynamic Template

In the figure given above, we can observe that on the **server-side**, a generic template is placed containing a **variable rule**. When this template is rendered on the **client-side**, an appropriate value is placed instead of the rule. This new value is per the context of the application, i.e., *information regarding the currently logged-in user*. This kind of *dynamic behavior* of a template is called **dynamic templating**.

Many server-side technologies let us implement dynamic templating behavior. Flask has inbuilt support for a dynamic templating engine called **Jinja.**

Let's explore Jinja!

The Jinja Template Engine

Jinja is a template engine that lets us serve dynamic data to the template files. A Jinja template file is a *text* file that does not have a particular extension. We will be using the `.html` extension with the template files because they will also include `HTML` syntax.

Delimiters

Before we get started, let's explore some delimiters used in Jinja Syntax.

- `{% ... %}` is used for **statements**.

- `{{ ... }}` is used for **variables**.

- `{# ... #}` is used for **comments**.
- `# ... ##` is used for **line statements**.

In the next lesson, we will learn how to use Python variables inside Jinja!