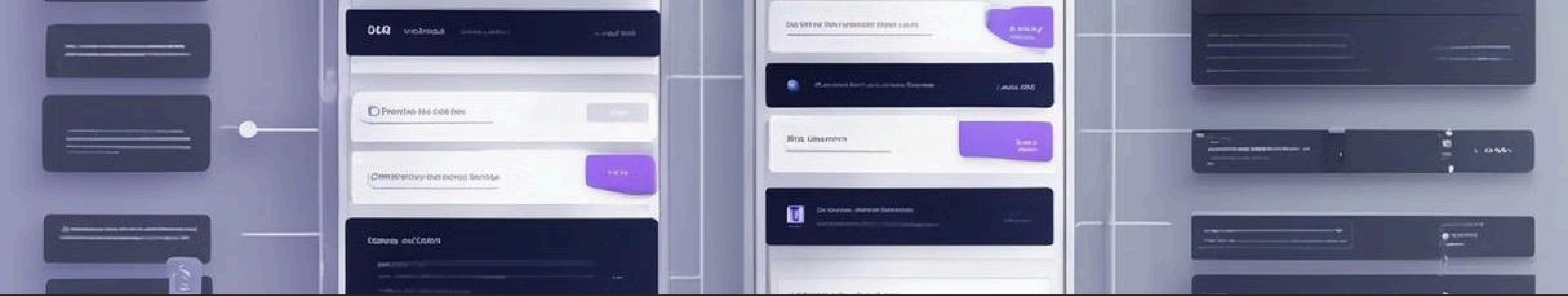
Understanding Flask Web Framework

Flask is a lightweight and versatile web framework for Python. It differs from other web frameworks in its simplicity and extensibility. While other frameworks may come with more built-in features and components, Flask allows developers to select and integrate only the components they need, making it a suitable choice for both small and large-scale applications.

S **by Shaik Lukhman**



Structure of a Flask Application

App.py

This file serves as the entry point for the Flask application, containing the configuration and setup related to the app's initialization and settings.

Templates Folder

Contains HTML templates for rendering dynamic content, allowing separation of presentation and business logic.

Static Folder

Holds static files such as CSS, JavaScript, and images, which are directly served to the client without processing by the application.

Installing Flask and Setting Up a Project

1 Installation

Install Flask using pip, the Python package manager, which ensures a smooth and hassle-free installation process.

2 Project Setup

Create a new directory for the project, set up a virtual environment, and install Flask. This isolates the project's dependencies from other Python projects.

Routing in Flask: Mapping URLs to Python Functions

URL Mapping

Routes in Flask are defined using Python decorators, which bind a URL to a function. This enables the execution of specific code when a particular URL is accessed.

Python Functions

Each route is associated with a Python function that handles the logic for the corresponding URL. This promotes clean and organized code architecture.

Templates in Flask for Dynamic HTML Content

1

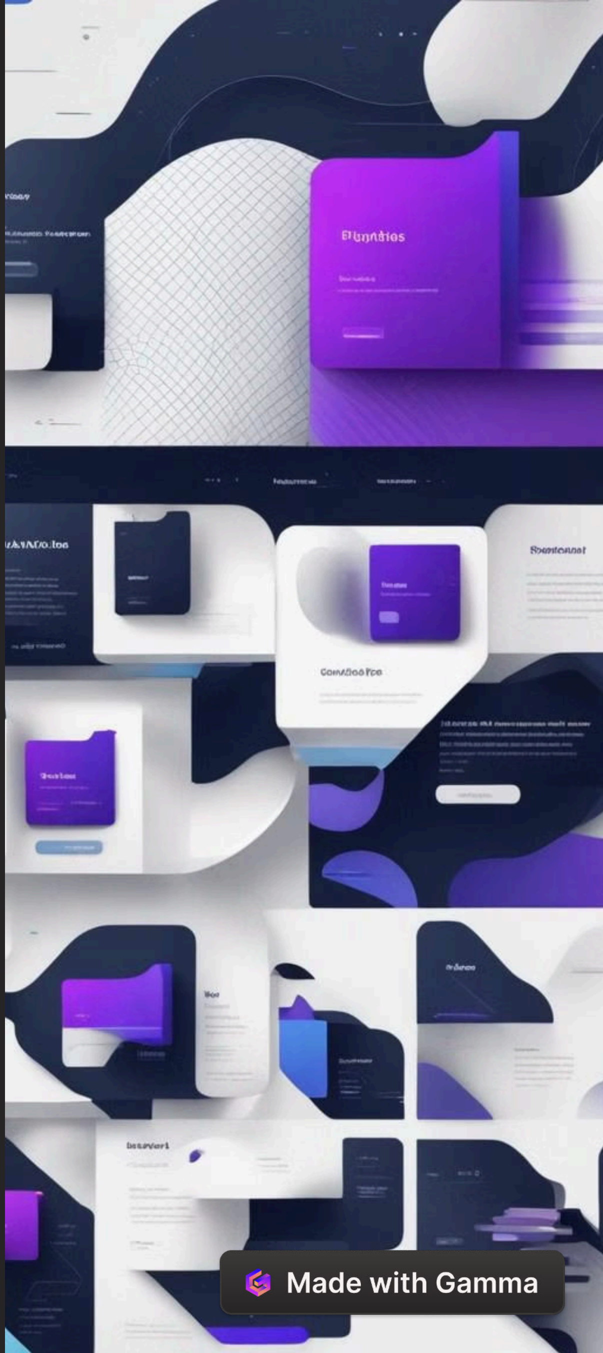
Data Integration

Templates allow the integration of dynamic data into HTML, enabling the generation of personalized output for each request.

2

Reusable Layouts

Template inheritance simplifies the management of a consistent layout across multiple pages by defining a base template with common elements.



Made with Gamma

Passing Variables from Flask Routes to Templates

1

Context Injection

Flask facilitates the injection of variables and data from routes into templates, expanding the range of content that can be displayed.

2

Rendering Process

The passing of variables enhances the rendering process, allowing for customized output based on the provided data.

Retrieving Form Data in a Flask Application

Data Submission

Flask provides mechanisms for capturing and processing form data submitted by users, enabling interactive and responsive applications.

Input Handling

Form data retrieval involves handling user input, validating the content, and executing appropriate actions based on the received data.

Jinja Templates in Flask: Advantages Over HTML

1

Dynamic Rendering

Jinja templates support the insertion of dynamic content and data, enabling the creation of interactive and adaptive web applications.

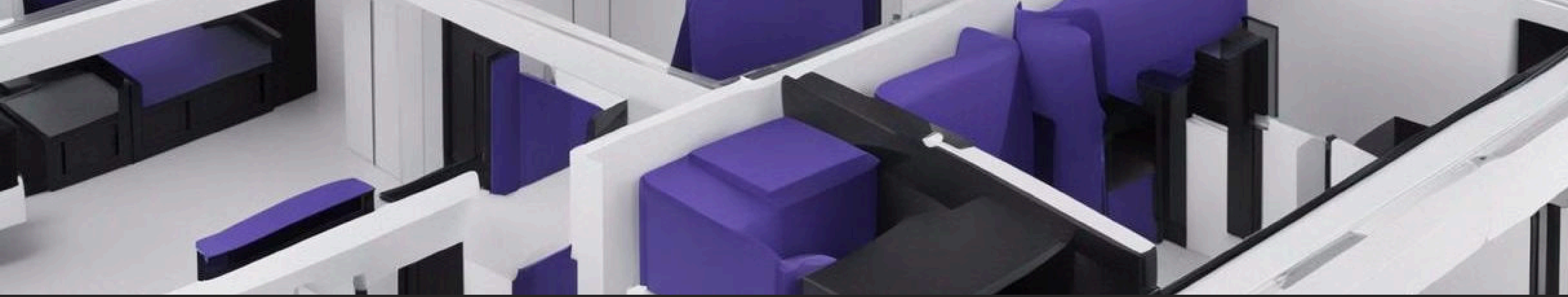
2

Reusable Components

Jinja's modular nature promotes code reusability, fostering a more efficient and maintainable development process.

Fetching Values from Templates and Arithmetic Calculations

Value Retrieval	Flask enables the extraction of specific values from templates, allowing seamless integration with backend logic.
Arithmetic Operations	Performing calculations within templates is supported, empowering dynamic and computation-rich web content.



Best Practices for Organizing and Structuring a Flask Project

Modular Design

Implement a modular structure to partition the codebase into manageable components, enhancing organization and maintainability.

Blueprints Implementation

Utilize blueprints to logically group related routes and views, fostering a robust and scalable application architecture.

Separation of Concerns

Emphasize the separation of presentation, business logic, and data access layers, promoting clarity and scalability in the project.