

Understanding Flask: A Beginner's Guide to Your KrishnAI Project

Table of Contents

1. [What is Flask?](#what-is-flask)
2. [How Does Flask Work?](#how-does-flask-work)
3. [Key Components](#key-components)
4. [How Your Project Uses Flask](#how-your-project-uses-flask)
5. [Real Examples from Your Code](#real-examples-from-your-code)

What is Flask?

Simple Explanation

Flask is a **lightweight Python web framework** that helps you build websites and web applications. Think of it as a toolkit that gives you pre-built tools to handle common web tasks, so you don't have to build everything from scratch.

Analogy

Imagine building a restaurant:

- **Without Flask**: You have to build the kitchen, dining area, payment system, booking system, staff management - everything from scratch
- **With Flask**: The framework provides you with the kitchen equipment, tables, and a cash register system ready to use. You just need to customize them for your specific restaurant

What Problems Does Flask Solve?

- Receiving requests from users (when someone clicks a button on your website)
- Sending responses back (showing pages, data, or messages)
- Managing databases (storing and retrieving user information)
- Handling user login/logout
- Connecting different pages together

How Does Flask Work?

Basic Flow: Request-Response Cycle

```
User (Browser)  Your Server (Flask)

|  |
| 1. Makes a request |
| (clicks a button) |
|----->|
| 2. Flask processes
| the request
|  |
| 3. Executes code
| to handle it
|  |
| 4. Server sends back |
| a response |
|<-----|
|  |
| 5. Browser shows
| the response
|
```

Example in Plain English

1. **User clicks "Send Message" button** on your chat page
2. **Browser sends the message to the server** (Flask app running on your computer/server)
3. **Flask receives the request** and knows which function should handle it
4. **Flask runs that function** (e.g., saves message to database, gets AI response)
5. **Flask sends back the result** (the page with the new chat messages)
6. **Browser displays the page** to the user

Key Components

1. **Routes** (URLs/Paths)

Routes are like addresses to different pages in your website.

What it does: Tells Flask which function to run when someone visits a specific URL

****Real-world analogy**:** Like mail addresses - different mail goes to different addresses

```
@app.route("/chat") # This is the route (the address)

def chat(): # This function handles that address

    return render_template("chat.html")
```

When you visit `http://yoursite.com/chat`, Flask runs the `chat()` function.

2. ****HTTP Methods**** (GET and POST)

These are ways your browser can ask the server for something.

****GET**:** "Server, show me this page" (just viewing, not sending data)

****POST**:** "Server, here's some data, process it" (like submitting a form)

```
@app.route("/login", methods=["GET", "POST"])

def login():

    if request.method == "GET":

        return render_template("login.html") # Show the login form

    if request.method == "POST":

        # User submitted the form with username/password

        # Process the login here

        return redirect(url_for("dashboard"))
```

3. ****Templates**** (HTML Files)

Templates are HTML files that Flask fills in with data and sends to the user.

****What it does**:** Separates how your page looks (HTML) from your logic (Python)

****Location in your project**:** `templates/` folder

```
return render_template("chat.html", response=response, user_message=user_message)
```

This tells Flask: "Take the chat.html file, fill in the `response` and `user_message` variables, and send it to the user"

4. ****Database**** (SQLAlchemy)

This stores information permanently (user accounts, messages, etc.)

****What it does**:** Saves data so it persists even if the app restarts

****Tools you use**:**

- `db.session.add()` - Add something to save
- `db.session.commit()` - Actually save it to the database
- `.query()` - Retrieve data from the database

5. **Authentication** (Login/Logout)

Flask-Login handles user accounts and sessions.

What it does:

- Keeps track of who is logged in
- Protects pages so only logged-in users can see them
- Manages login and logout

```
@login_required # This means: only show this page to logged-in users

def chat():

    return render_template("chat.html")
```

How Your Project Uses Flask

Your project is **KrishnAI**, a chat application where users can:

1. Sign up and log in
2. Chat with an AI (Krishna)
3. Read daily shloka (verses)
4. Write reflections

Your Flask App Structure

```
app.py ← Main Flask application (the brain)

■■■ @app.route("/") ← Home page
■■■ @app.route("/signup") ← Sign up page
■■■ @app.route("/login") ← Login page
■■■ @app.route("/chat") ← Chat page
■■■ @app.route("/...") ← Other pages

models/

■■■ user.py ← Defines what a User is
■■■ chat.py ← Defines what a Chat message is
■■■ reflection.py ← Defines what a Reflection is
```

```
templates/

■■■ base.html ← Base template (header, navigation)
■■■ login.html ← Login page design
■■■ chat.html ← Chat page design
■■■ ... ← Other page designs

services/

■■■ groq_service.py ← Handles AI responses
■■■ shloka_service.py ← Handles daily shloka
```

Real Examples from Your Code

Example 1: Sign Up Process

****What happens:****

User fills in username, email, password and clicks "Sign Up"

****Code in app.py:****

```
@app.route("/signup", methods=["GET", "POST"])
def signup():
    if request.method == "GET":
        # Step 1: User visits /signup
        # Show them the signup form
        return render_template("signup.html")

    if request.method == "POST":
        # Step 2: User submitted the form with their data

        # Step 3: Create a new User object
        user = User(
            username=request.form["username"], # Get username from form
            email=request.form["email"] # Get email from form
        )

        # Step 4: Hash the password (for security)
        user.set_password(request.form["password"])

        # Step 5: Save to database
        db.session.add(user)
```

```
db.session.commit()

# Step 6: Redirect to login page
return redirect(url_for("login"))
```

****Step-by-step:****

1. User visits `http://yoursite.com/signup` (GET request)
2. Flask runs `signup()` function with `request.method == "GET"`
3. Flask shows the signup form (signup.html)
4. User fills in form and clicks "Sign Up" (POST request)
5. Flask runs `signup()` again, but now `request.method == "POST"`
6. Flask gets the form data: username, email, password
7. Creates a new User with that data
8. Saves it to the database
9. Redirects user to login page

Example 2: Chat with AI

****What happens:****

User sends a message, AI responds, message is saved

****Code in app.py:****

```
@app.route("/chat", methods=["GET", "POST"])
@login_required # Only logged-in users can access this
def chat():
    response = None
    user_message = None

    if request.method == "GET":
        # User visits the chat page
        # Show them the chat interface
        return render_template("chat.html", response=None, user_message=None)

    if request.method == "POST":
        # Step 1: User sent a message
        user_message = request.form["message"]

        # Step 2: Get AI response
```

```

response = get_krishna_response(user_message)

# Step 3: Save to database
chat = Chat(
    user_id=current_user.id, # Who sent it
    user_message=user_message, # What they said
    ai_response=response # What AI responded
)

db.session.add(chat)
db.session.commit()

# Step 4: Show the updated chat with the new message
return render_template(
    "chat.html",
    response=response,
    user_message=user_message
)

```

****Flow:****

```

1. User types message in chat.html
↓
2. Clicks Send → POST request to /chat
↓
3. Flask gets the message: request.form["message"]
↓
4. AI service processes it: get_krishna_response()
↓
5. Create a Chat object to record this
↓
6. Save to database: db.session.add() + db.session.commit()
↓
7. Re-render chat.html with the new message and response
↓
8. User sees the updated chat

```

Example 3: Login with Authentication

****What happens:****

User logs in, Flask verifies credentials, and protects pages

****Code in app.py:****

```
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "GET":
        return render_template("login.html")

    if request.method == "POST":
        # Step 1: Get username from form
        username = request.form["username"]

        # Step 2: Find user in database
        user = User.query.filter_by(username=username).first()

        # Step 3: Check if user exists AND password is correct
        if user and user.check_password(request.form["password"]):
            # Login is successful!
            login_user(user) # Flask-Login remembers this user
            return redirect(url_for("dashboard"))

        # If we reach here, login failed
        return render_template("login.html")
```

****Protected Pages:****

```
@app.route("/chat")
@login_required # Only logged-in users can see this
def chat():
    return render_template("chat.html")
```

The `@login_required` decorator means:

- If user is logged in → Show the chat page ✓
- If user is NOT logged in → Redirect to login page ✗

Example 4: Database Models (User, Chat, Reflection)

****user.py - Defines what a User is:****

```

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    email = db.Column(db.String(150), unique=True, nullable=False)
    password_hash = db.Column(db.String(200), nullable=False)

```

****In plain English:****

- Each user has an `id` (like a student ID number)
- Each user has a `username` (must be unique, like an email)
- Each user has an `email` (must be unique)
- Each user has a `password_hash` (encrypted password, for security)

****When you do:****

```

user = User(username="john", email="john@example.com")
db.session.add(user)
db.session.commit()

```

Flask creates a new row in the database with that user's information.

Key Concepts Summary

Concept	What It Does	Example from Your App
Route	Maps a URL to a function	<code>@app.route("/chat")`</code>
GET	Request to view something	User visits <code>/chat`</code> page
POST	Request to send data	User submits a message
Template	HTML file Flask fills with data	<code>render_template("chat.html")`</code>
Database	Stores information	User accounts, chat messages
Session	Remembers logged-in user	<code>current_user.id`</code>
Decorator	Special marker for functions	<code>@login_required`</code>

How Everything Works Together

Complete User Journey:

1. ****User visits your site****

- Browser: ``http://yoursite.com/signup``
 - Flask route: ``@app.route("/signup")``
2. ****User signs up****
- Fills form and clicks "Sign Up"
 - POST request to ``/signup``
 - Flask creates User in database
3. ****User logs in****
- Enters username and password
 - POST request to ``/login``
 - Flask checks password, calls ``login_user()``
4. ****User goes to chat****
- Visits ``http://yoursite.com/chat``
 - ``@login_required`` checks if logged in
 - If yes, shows chat.html
5. ****User sends message****
- Types message and clicks Send
 - POST request with message
 - Flask gets AI response
 - Saves to Chat database
 - Shows updated chat

Important Files in Your Project

File	Purpose
<code>`app.py`</code>	Main Flask application with all routes
<code>`models/user.py`</code>	Defines User structure for database
<code>`models/chat.py`</code>	Defines Chat message structure
<code>`models/reflection.py`</code>	Defines Reflection structure
<code>`templates/login.html`</code>	HTML for login page
<code>`templates/chat.html`</code>	HTML for chat page
<code>`services/groq_service.py`</code>	Connects to AI API

<code>`requirements.txt`</code>	List of Python packages needed
---------------------------------	--------------------------------

Conclusion

Flask is a simple but powerful framework that:

1. **Receives requests** from users (via URLs)
2. **Processes them** (runs Python code)
3. **Accesses the database** (saves/retrieves data)
4. **Sends back responses** (HTML pages, data)

Your KrishnAI project uses Flask to create a complete web application with user authentication, database storage, and AI integration - all with relatively simple code!

The magic is that Flask handles all the "boring" web stuff, letting you focus on what makes your app special (the AI chat, shloka, reflections, etc.).