venv → flaskapp

↳ templates

↳ statics

↳ Routing/urls } login application.

As a home work.

→ Pass word Generat

↳ flaskap

Flask Routing Concepts

https: localhost / hello.

@app.route("Link')

view function —— def func( ))

Introduction to routing

- **Routing** in Flask is the mechanism that maps URLs to view functions. When a client (like a web browser) makes a request to a specific URL, Flask determines which Python function should handle that request based on the defined routes.
- **Key Concepts:**
- **Route**: A URL pattern that Flask recognizes.
- **View Function**: A Python function that Flask calls when a specific route is requested.
- **URL**: The address clients use to access resources on the server.

Defining Routes

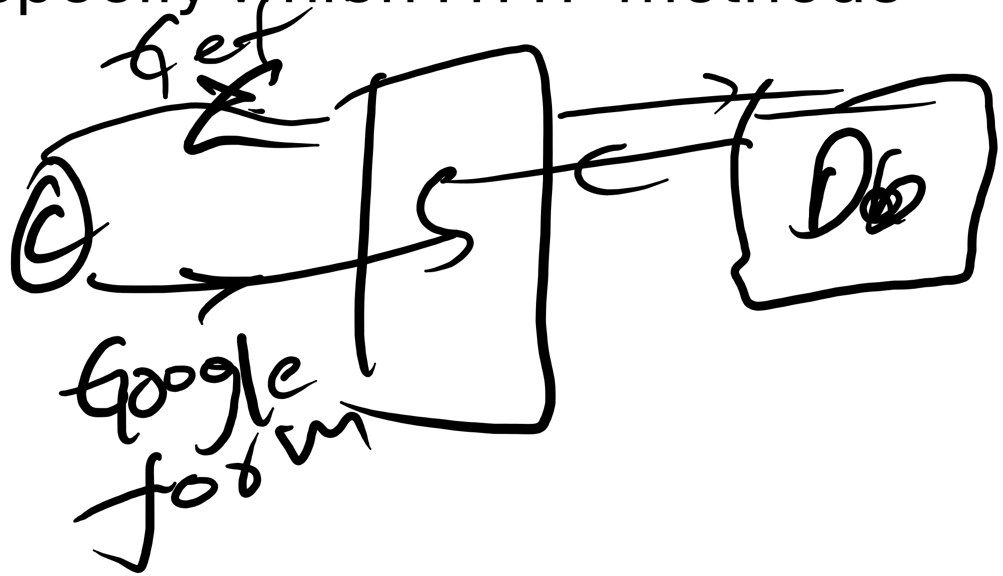
- In Flask, routes are typically defined using decorators. The `@app.route` decorator associates a URL with a view function.

→ Hyper text transfer protocol.

HTTP methods

Networking.

- HTTP methods define the action to be performed for a given resource. Flask routing allows you to specify which HTTP methods a route should respond to.
- **Common HTTP Methods:**
  - GET: Retrieve data from the server.
  - POST: Submit data to the server.
  - PUT: Update existing data.
  - DELETE: Remove data.
- **Default Method:**
  - If not specified, routes respond to GET requests by default.



Dynamic Routes and Variable Rules

- Dynamic routes allow parts of the URL to be variable, capturing values from the URL and passing them as arguments to the view function.

https:..../user?..=

- **Syntax:**

```
@app.route('/user/<username>')
def show_user(username):
    return f"User: {username}"
```

URL converters

type casting

int → string
int → float

- Flask provides **URL converters** to specify the type of variable parts in URLs. This ensures that the captured variables meet certain criteria before passing them to view functions.
- **Common Converters:**
- **`string`** (default): Accepts any text without a slash.
  - `<string:name>` or `<name>`
- **`int`**: Accepts positive integers.
  - `<int:id>`
- **`float`**: Accepts positive floating-point values.
  - `<float:price>`
- **`path`**: Like `string` but accepts slashes.
  - `<path:subpath>`
- **`uuid`**: Accepts UUID strings.
  - `<uuid:identifier>`

Route Defaults

- You can specify default values for parts of the URL, making certain parameters optional or providing fallback values.

- **syntax**

```
@app.route('/greet/', defaults={'name': 'Guest'})
@app.route('/greet/<name>')
def greet(name):
   return f"Hello, {name}!"
```

Example Usage:

- `/greet/` → "Hello, Guest!"
- `/greet/Alice` → "Hello, Alice!"

Route Endpoints

- An **endpoint** is a unique identifier for a route, usually the name of the view function. You can explicitly specify an endpoint name, which is useful when multiple routes share the same view function
- **Default Endpoint:**

```
@app.route('/home')
def home():
   return "Home Page"

# Endpoint: 'home'
```

- **Custom Endpoints**

```
@app.route('/start', endpoint='begin')
def home():
   return "Home Page"

# Endpoint: 'begin'
```

Url building

- The **url_for** function generates URLs for the specified view functions. This is beneficial for avoiding hardcoding URLs and makes your application more maintainable

- from flask import url_for

- @app.route('/dashboard')
- def dashboard():
-     return "Dashboard"

- @app.route('/go-to-dashboard')
- def go_to_dashboard():
-     return redirect(url_for('dashboard'))

Using Dynamic routes

- @app.route('/user/<username>')
- def user_profile(username):
-     return f"Profile of {username}"

- @app.route('/profile-link/<username>')
- def profile_link(username):
-     profile_url = url_for('user_profile', username=username)
-     return f"Link to profile: {profile_url}"

[HTML, CSS]

Basic application Building Login Page

- This Flask login application demonstrates basic user authentication and session management. It provides the following capabilities:
- **User Login**: Users can enter a username and password on the login page. Upon submission, the app checks the credentials against a predefined dictionary of users.
- **Session Management**: If authentication is successful, the username is stored in a session, allowing the user to access protected pages (e.g., the dashboard) without needing to log in again during the session.

login → session → logout

- **Dashboard Access**: After logging in, the user is redirected to a dashboard page, which is only accessible when the user is logged in. Unauthorized users are redirected back to the login page.

Login → Dashboard

- **Logout Functionality**: The user can log out, clearing the session and redirecting them to the login page.
- **Feedback via Flash Messages**: The app provides real-time feedback, showing messages like "Login successful" or "Invalid credentials."

user pass → Dashboard → logout

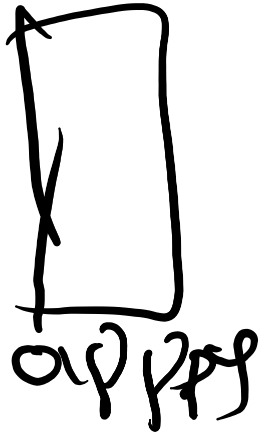
Structure of the application

Cred

- /flask_login_app
- /static → CSS files, Image, SVGs, Videos
- - style.css
- /templates → Jinja2.
- - login.html → templates
- - dashboard.html →
- app.py

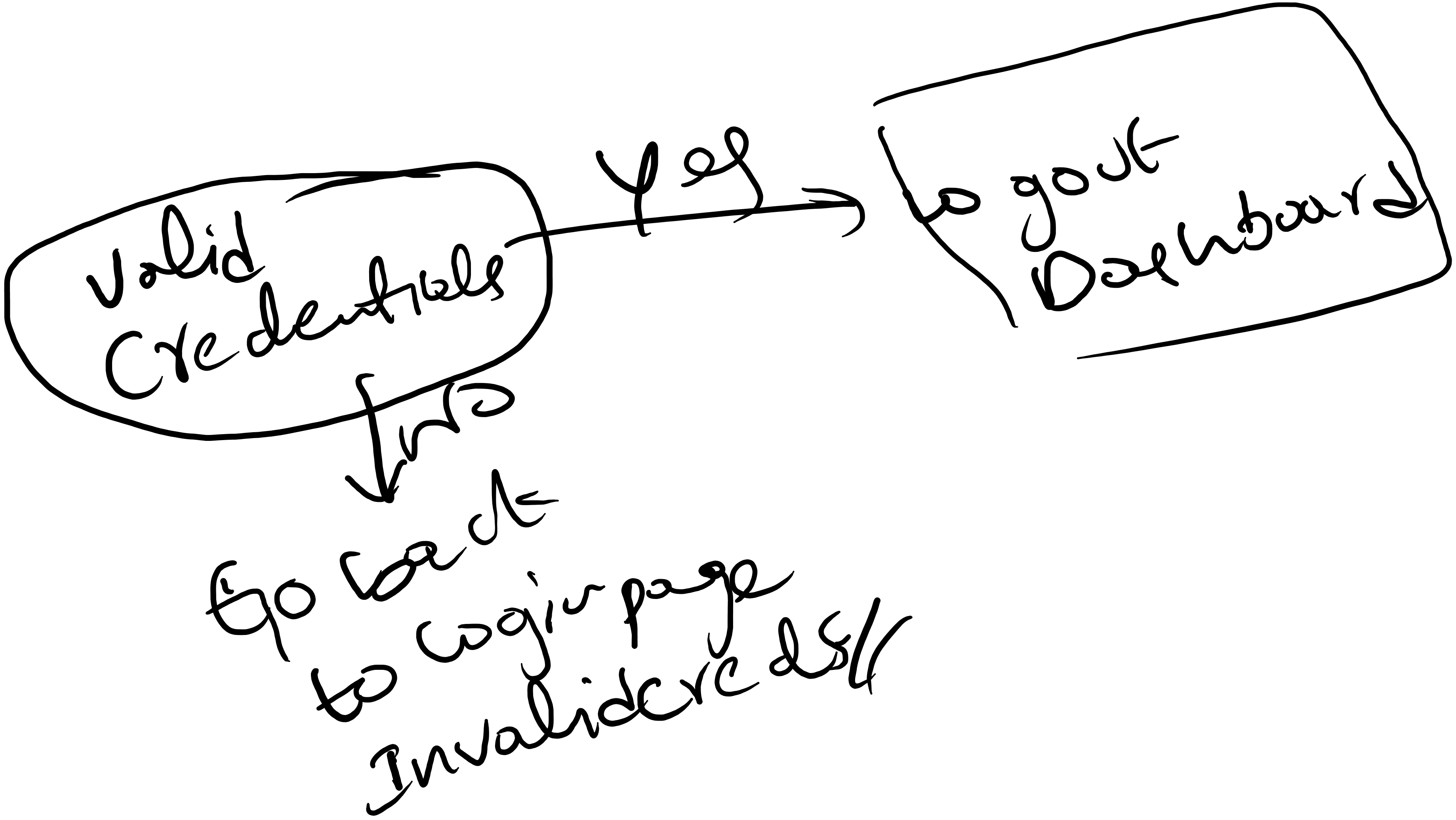
Not working with Any DB

Frontend



Backend user No. passw

DB


Valid Credentials
Yes
to your Dashboard
no
go back to login page Invalidcreds!!